UNIVERSIDAD AUTÓNOMA DE MADRID

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

TESIS DOCTORAL

INCORPORACIÓN DE LA USABILIDAD EN EL PROCESO DE DESARROLLO OPEN SOURCE SOFTWARE

Autor:

JOHN WILMAR CASTRO LLANOS

Directoras:

SILVIA TERESITA ACUÑA

NATALIA JURISTO

Madrid, Diciembre de 2014

A mi amada esposa,
Yanedt, por todo su apoyo incondicional
y ánimo constante.

AGRADECIMIENTOS

La realización de la presente tesis doctoral ha requerido de sacrificios, de perseverancia, de afrontar serios quebrantos de salud y de un gran esfuerzo a lo largo de todo este tiempo, y constituye el paso final para alcanzar mi sueño de ser Doctor en Informática.

En primer lugar, quiero agradecer de un modo especial a las Directoras de la Tesis: Silvia Teresita Acuña Castillo -Profesora de la Universidad Autónoma de Madrid- y Natalia Juristo Juzgado -Catedrática de la Universidad Politécnica de Madrid-. Gracias por su orientación, por la magnífica capacidad de análisis y visión demostrada, por sus inestimables sugerencias/críticas constructivas, por su paciencia y por la ayuda inquebrantable recibida. Su apoyo incondicional, sus palabras de ánimo, sus consejos y total implicación han sido esenciales para concluir este trabajo.

Agradecer también las facilidades brindadas por Rob Weir y Christian Foltin, administradores de los proyectos *OpenOffice Writer* y *FreeMind* respectivamente. La validación de la viabilidad del marco de integración propuesto en esta investigación no hubiese sido posible sin su participación y recursos brindados.

No menos importante ha sido la colaboración brindada por los usuarios de *OpenOffice Writer* y *FreeMind*, gracias por el esfuerzo que realizaron para planificar una hora de trabajo y poder trabajar juntos a pesar de la diferencia horaria. Gracias por el interés en participar y por las sugerencias dadas. También agradecer a amigos y familiares, así como a todos los estudiantes de grado y postgrado por participar en la validación de esta investigación. Gracias a Javier por su colaboración en una parte importante de la tesis.

En el contexto privado, agradecer enormemente todo el apoyo incondicional recibido por mi Esposa desde que decidí realizar mis estudios de tercer ciclo en el extranjero, hasta su materialización en las presentes páginas. Finalmente, agradecer a todas y cada una de las personas que de una u otra manera han hecho posible este logro. Un agradecimiento especial a mi amigo Jaidermes por todo el apoyo recibido desde que iniciamos esta aventura de ser investigadores.

RESUMEN

Contexto: Debido al crecimiento de los usuarios de aplicaciones *open source software* (OSS) que no son desarrolladores y a que las empresas y organizaciones cada vez mas están usando aplicaciones OSS, surge la necesidad y el interés por desarrollar OSS usable. Sin embargo, solo unos pocos proyectos OSS están comenzando a incorporar unas pocas técnicas de usabilidad y la mayoría de ellas requieren contextos de desarrollo que la construcción de OSS no satisface. No se conoce exactamente cómo es posible permitir el uso de técnicas de usabilidad en los desarrollos OSS. Además, no está claro cuáles técnicas de usabilidad aplicar en cada actividad y cómo utilizar las técnicas de usabilidad en los desarrollos OSS.

Objetivo: Este trabajo determina cuáles son las condiciones desfavorables que impiden el uso de técnicas de usabilidad en los desarrollos OSS y analiza qué tipos y cuáles transformaciones son necesarias para poder facilitar su uso en este tipo de proyectos. Además, analizamos cuáles técnicas pueden ser usadas en OSS gracias a las transformaciones y validamos la viabilidad de incorporar técnicas de usabilidad en los desarrollos OSS.

Método de Investigación: Hemos estudiado –a través de la literatura– el proceso de desarrollo OSS y las técnicas de usabilidad usadas ocasionalmente por la comunidad OSS. Para el análisis de las técnicas hemos utilizado un catálogo existente de técnicas recomendadas por el área de la Interacción Persona Ordenador (IPO) para mejorar la usabilidad. Posteriormente, hemos analizado las condiciones de las técnicas que dificultan su uso en OSS y hemos estudiado las técnicas usadas por OSS para determinar cómo han sido incorporadas en sus desarrollos. A través de dos casos de estudio hemos validado la viabilidad de nuestra propuesta de incorporación de técnicas de usabilidad, participando como voluntarios en dos proyectos OSS reales: *OpenOffice Writer* y *FreeMind*.

Resultados: Hemos identificado varias razones para la baja usabilidad de las aplicaciones OSS. Los proyectos de desarrollo OSS están empezando a adoptar técnicas de usabilidad. Algunas técnicas de usabilidad están siendo adoptadas según lo prescrito por la IPO. Sin embargo, la mayoría de técnicas de usabilidad no pueden ser incorporadas en los desarrollos OSS. Hemos identificado tres grupos de condiciones desfavorables que impiden tal incorporación: participación de un experto en usabilidad; participación de usuarios; y complejidad de aplicación (varios pasos para su ejecución o preparación previa o necesitan de cierta información inicial). Hemos observado que algunas de las técnicas han sido incorporadas por OSS gracias a ciertas transformaciones. Estas transformaciones pueden ser o bien una sola o una combinación de varias.

Conclusiones: La principal contribución de esta tesis es la propuesta de un marco que permite la integración de determinadas técnicas de usabilidad en los desarrollos OSS. Dicho marco está conformado por todas las posibles transformaciones que deben sufrir las técnicas de usabilidad para poder ser aplicadas en OSS. Luego de identificar y analizar cuáles técnicas de usabilidad están siendo incorporadas en los desarrollos OSS, nos hemos dado cuenta que es posible generalizar muchas de las transformaciones de modo que otras técnicas (otros proyectos OSS) se puedan beneficiar de ellas. No todas las técnicas de usabilidad pueden sufrir todas las

adaptaciones. La transformación depende de la idiosincrasia de la técnica. Por tal razón, el marco también propone para cada técnica y sus características intrínsecas y condiciones desfavorables asociadas, qué adaptaciones pueden ser realizadas a la misma. Con el marco que proponemos cualquier persona que quiera aplicar una técnica de usabilidad en OSS puede seleccionar la técnica que necesita dependiendo de su desarrollo y de su proyecto y aplicar alguna de las transformaciones sugeridas. De esta manera, se logra que una técnica que antes no se podía aplicar directamente en OSS, porque no se adaptaba a las circunstancias de los desarrollos OSS, pueda hacerlo gracias a nuestro marco. Por tanto, es posible incorporar técnicas de usabilidad con transformaciones en los desarrollos OSS. Esta incorporación cuenta con un punto muy positivo: el entusiasmo tanto de los administradores del proyecto como de los usuarios OSS.

Palabras clave: Proceso de desarrollo, Interacción Persona Ordenador, Open Source Software, Usabilidad, Técnicas de Usabilidad.

ABSTRACT

Context: Because the number of non-developer *open source software* (OSS) users is growing and more and more companies and organizations are using OSS applications, there is a need and interest in developing usable OSS. However, only a few OSS projects are starting to adopt a small number of usability techniques, and OSS construction does not conform to the development contexts required by most techniques. It is not exactly known how to enhance the use of usability techniques in OSS developments. Additionally, it is unclear which usability techniques to apply in each activity and how to apply usability techniques in OSS development projects.

Objective: This research determines the conditions that are adverse to the use of usability techniques in OSS development projects and analyses what type of and which adaptations are necessary to enhance their use in this type of projects. Additionally, it analyses which techniques can be used in OSS thanks to adaptations and validates the feasibility of adopting usability techniques in OSS developments.

Research Method: A literature review examined the OSS development process and usability techniques used occasionally by the OSS community. The techniques were analysed from an existing catalogue of techniques recommended by the field of human-computer interaction (HCI) to improve usability. The conditions that pose an obstacle to the use of the techniques in OSS were then examined, and the techniques used by OSS were studied to determine how they have been adopted in OSS development projects. Two case studies were used to validate the feasibility of the proposal for adopting usability techniques, as part of which team members participated as volunteers in two real OSS projects: *OpenOffice Writer* and *FreeMind*.

Results: Several reasons for the low usability of OSS applications were identified. OSS development projects are starting to adopt usability techniques. Some usability techniques are being adopted as prescribed by HCI. However, most usability techniques cannot be adopted in OSS development projects. Three groups of adverse conditions preventing their adoption were identified: participation of a usability expert, participation of users, and complexity of application (several steps, previous preparation or preliminary information are required for technique execution). It was found that some techniques have been adopted by OSS thanks to certain adaptations. These adaptations can consist of one or more changes.

Conclusions: The main contribution of this thesis is the proposal of a framework for integrating some usability techniques into OSS developments. This framework is composed of all the possible adaptations that usability techniques should undergo for application in OSS. After identifying and analysing which usability techniques are being adopted in OSS developments, it was found that many of these adaptations can be generalized to other techniques (other OSS projects). Not all usability techniques can be adapted. Adaptation depends on the technique's idiosyncrasy. On this ground, the framework also proposes how each technique can be adapted based on its inherent features and associated adverse conditions. Using the proposed framework, anyone who wants to apply a usability technique in OSS can select the technique that they require depending on their development project

and apply any of the suggested adaptations. Thanks to this framework, techniques that were not previously directly applicable to OSS because they did not conform to OSS development conditions can now be employed. Therefore, it is possible to adopt usability techniques with adaptations in OSS development projects. OSS project managers and users are enthusiastic about the adoption of usability techniques, which is a very positive point.

Keywords: Development Process, Human Computer Interaction, Open Source Software, Usability, Usability Techniques.

Índice

CAPÍTULO 1

INTRODUCCIÓN

El trabajo de investigación que se presenta en este documento se enmarca en las áreas del proceso de desarrollo *open source software* (OSS) y usabilidad, y plantea el problema de la integración de técnicas de usabilidad en el proceso de desarrollo OSS. En primer lugar, se introducen las áreas de investigación para, a continuación, tratar la existencia y relevancia del problema que el presente trabajo afronta. Finalmente, se presenta la aproximación a la solución propuesta, la estructura del trabajo y se detallan las contribuciones y publicaciones derivadas de esta investigación.

1.1. Áreas de Investigación

1.1.1. El Proceso de Desarrollo OSS

El proceso de desarrollo OSS es un tema importante de estudio por parte de profesionales y de la comunidad científica [65][67][102][111] gracias al éxito de proyectos OSS como Linux [191], Apache [125] y FreeBSD [57]. Uno de los trabajos de investigación más populares ha sido el realizado por Raymond [155]. En su trabajo, Raymond distingue entre dos diferentes tipos de estilo de desarrollo de software. El primero, es el desarrollo OSS que es comparable a un bazar, donde cualquier persona tiene el derecho a participar y contribuir. El segundo, es el desarrollo de software tradicional, similar a una catedral donde un arquitecto lidera un pequeño grupo de trabajadores calificados y especializados, con horarios y responsabilidades precisas. Raymond [155] afirma que el estilo bazar crea un ambiente democrático donde los colaboradores pueden discutir las mejores soluciones para el código fuente de manera eficiente, ya que cada desarrollador es un usuario.

El desarrollo OSS ofrece un modelo alternativo al tradicional, donde el software es desarrollado por o para una entidad corporativa. En el modelo de desarrollo OSS, una diversidad de desarrolladores llevan a cabo el desarrollo y distribución del código fuente del producto. Esto permite una mejora incremental del software por parte de otros o el desarrollo de productos complementarios que perfectamente pueden inter-operar con distintos productos OSS.

En el desarrollo OSS el equipo de trabajo está distribuido por todo el mundo. Los desarrolladores OSS no suelen encontrarse cara a cara. Más bien, la comunidad de desarrollo de cualquier proyecto OSS se centra alrededor de un sitio Web y la comunicación se lleva a cabo a través de chats, foros de discusión y lista de correos. Los proyectos OSS no tienen restricciones de tiempo y no existe un mecanismo que obligue a implementar una funcionalidad determinada. El mayor interés en la gestión de un proyecto OSS consiste en que las contribuciones de los desarrolladores (en forma de código fuente) vayan acorde con el

producto y aquellas que no lo estén sean descartadas [181]. En el proceso de desarrollo OSS los requisitos son inciertos y pueden cambiar constantemente. Los desarrolladores eligen en qué desean trabajar y cuándo. La elección se basa en lo que ellos consideran necesita el producto, en sus intereses personales o que suponga un reto desde el punto de vista técnico. Por tanto, los requisitos no son asignados a los desarrolladores, son ellos quienes los seleccionan [181].

Los miembros de la comunidad OSS juegan diferentes roles, tales como desarrollador principal, propietario del módulo, desarrollador, administrador del repositorio de código fuente, revisor o usuario final. Muchos de los miembros que contribuyen en proyectos OSS lo hacen como voluntarios, es decir sin remuneración; otros son pagados por sus empleadores, pero no directamente por el proyecto. Como resultado de esto, el reclutar y retener nuevos miembros es un factor crítico de éxito para un proyecto OSS [54]. Los miembros de la comunidad OSS contribuyen, por ejemplo, con programas, scripts de ejecución, revisión de código o comentarios. Las contribuciones son realizadas en los sitios Web de cada comunidad y la comunicación de las actualizaciones de contenido se realiza a través de foros de discusión en línea, grupos de noticias y correo electrónico. Los errores encontrados por cualquier miembro de la comunidad son reportados a través de grupos de noticias, en las páginas Web destinadas para el reporte de los mismos o en los sistemas de reportes de errores (por ejemplo, Bugzilla) [170].

1.1.2. Usabilidad en el Proceso de Desarrollo OSS

La usabilidad es uno de los atributos de calidad clave en el desarrollo de software [71]. En los últimos años, OSS se ha convertido en un componente importante dentro de la informática. Tiene un papel relevante en las tecnologías de la información tanto en los negocios como en la educación y el gobierno [171], y ha creado una industria de servicios de miles de millones de dólares para compañías como Red Hat, Novell e IBM [167].

La creciente importancia de OSS en los últimos años ha llevado a los investigadores a estudiar cómo los procesos OSS difieren de los procesos de la Ingeniería del Software (IS) tradicional. Estas investigaciones incluyen el estudio de muchos aspectos del desarrollo del código OSS tales como la motivación de sus participantes [86], los repositorios de código fuente [126], su forma de gestión [141] y sus requisitos [172]. Un grupo reducido de trabajos se ha centrado en cuestiones de usabilidad [35][136][195]. La presente investigación está motivada por la premisa de que la usabilidad es un área débil en OSS. Además, hay pocas comparaciones entre la usabilidad en OSS y las aplicaciones propietarias [136]. Los principales argumentos para afirmar que la usabilidad puede ser un motivo de preocupación para OSS se pueden resumir de la siguiente manera:

- Los desarrolladores no son típicos usuarios finales y construyen software para sí mismos.
- Los expertos de usabilidad no se involucran en proyectos OSS.
- Los proyectos OSS carecen de recursos para ocuparse de la usabilidad.
- El (re)diseño de la interfaz no puede ser susceptible de los mismos enfoques empleados en el (re)diseño de una funcionalidad.

Muchos proyectos OSS se han desarrollado sin una participación comercial significativa. Si bien, esto demuestra la posibilidad de escribir software de calidad fuera de un entorno comercial, tales proyectos tienen una desventaja importante de recursos en comparación con sus rivales propietarios. Sin embargo, como las aplicaciones OSS han ganado importancia, las empresas han comenzado a colaborar con proyectos OSS y han proporcionado importantes recursos a algunos de ellos. Por ejemplo, Sun Microsystems ha contribuido a GNOME [79] a

través de pruebas de usabilidad basadas en laboratorio [183] y en el desarrollo de un conjunto de pautas de usabilidad para las interfaces de usuario [35].

Uno de los enfoques para la participación comercial en el desarrollo de interfaces es simplemente una capa de interfaz propietaria sobre una base OSS, por ejemplo, el sistema operativo OS X de Apple. Un enfoque relacionado consiste en crear una capa superior que enmascare un aspecto de la funcionalidad OSS que se perciba ampliamente como compleja, por ejemplo la instalación de Linux. Sin embargo, estos proyectos son las excepciones y la mayoría se ven seriamente limitados por la falta de recursos y conocimientos de usabilidad.

El panorama general es de una comunidad que sólo recientemente comienza a relacionarse con los conceptos de diseño de la experiencia del usuario y la ingeniería de usabilidad que han sido comunes en otras áreas de conocimiento durante más de una década [138]. La comunidad OSS tiene características que son inusuales en el mundo de la Interacción Persona-Ordenador (IPO): distribución geográfica mundial, una visión del mundo centrado en el código, falta de recursos y una cultura que puede ser algo ajena a los desarrolladores de la interacción. Sin embargo, se pueden encontrar aspectos que propicien el manejo de la usabilidad tales como un grupo grande de usuarios que suministran una retroalimentación importante en aspectos como la interacción con el sistema, facilidad de uso, satisfacción, etc.

Raymond [155] fue uno de los primeros en describir algunas de las peculiaridades que caracterizan el desarrollo OSS, incluida la práctica de "liberar versiones rápidamente y a menudo", y la idea de que los desarrolladores se "rascan su propia comezón" escribiendo el software que necesitan ellos mismos. Éstas han sido citadas como las principales razones de porqué OSS es criticado por estar centrado en el desarrollo y del porqué tiene una pobre usabilidad [11][113][129][136].

Poco después del artículo de Raymond [155], se formaron proyectos de usabilidad OSS -no está claro si como una respuesta a su artículo, o como una conciencia natural de la comunidad OSS-. Hay varios proyectos de usabilidad [15][17][18][19][20], cuyos objetivos son mejorar la usabilidad en OSS mediante la educación de los desarrolladores sobre prácticas de usabilidad y ejecución de estudios de usabilidad en OSS. También, muchas empresas centradas en OSS tales como Canonical, Novell, Red Hat y Sun Microsystems han contratado a diseñadores e ingenieros de usabilidad para ayudar a mejorar la usabilidad de su software.

En la última década, la comunidad OSS ha prestado atención a la mejora de sus prácticas de usabilidad [35][75][131][136][157][189]. Siguiendo esta tendencia, algunos investigadores han identificado los desafíos inherentes en el tratamiento de la usabilidad en el desarrollo OSS. Por ejemplo, la distribución y la naturaleza voluntaria del desarrollo OSS dificultan la participación en los métodos de diseño integral, ya que los desarrolladores tienden a trabajar en los componentes de las aplicaciones de forma aislada [29][35][136].

Por una parte, debido a estos retos, la comunidad OSS está empezando a incorporar poco a poco algunas técnicas para abordar los problemas de usabilidad. Por ejemplo, Nichols y Twidale [137] reportan que algunos proyectos realizan la práctica "design-by-blog", donde los posibles diseños de ciertas funcionalidades son publicados en los blogs personales de los desarrolladores, con el objetivo de solicitar la opinión de los usuarios del software. Otra práctica consiste en realizar capturas de pantalla o capturas de pantalla con notas, para apoyar las discusiones por correo electrónico y para ser anexadas en los reportes de errores [195]. Por otra parte, los expertos en usabilidad lentamente se están integrando en los proyectos y son en parte responsables de las prácticas de usabilidad que se están incorporando en los desarrollos OSS y de la creación de una infraestructura de usabilidad especial (por ejemplo, wikis, listas

de correo) [28]. Sin embargo, falta una incorporación adecuada de las técnicas de usabilidad adaptadas al modo de desarrollo OSS [44][47][89][188].

1.2. Problema de Investigación

A medida que la base de usuarios de la comunidad OSS ha crecido más allá de los desarrolladores de software, se ha incrementado el interés en la creación de software bien diseñado y usable [35][75][136]. Las primeras contribuciones a los esfuerzos de usabilidad en OSS fueron impulsadas por empresas como Sun y Novell, que establecieron directrices y pautas para las interfaces de usuario [35] y las pruebas de usabilidad [75]. Las directrices resultantes han demostrado ser muy útiles, pues constituyen una referencia de peso para algunas cuestiones de diseño de interfaces, como por ejemplo las guías de usabilidad desarrolladas por Sun para el proyecto OSS GNOME [35][137].

Mientras que las contribuciones empresariales han jugado un papel importante en el comienzo de los esfuerzos de la usabilidad en la comunidad OSS, se han identificado una serie de problemas en la incorporación de métodos y técnicas de usabilidad en proyectos individuales. Estos desafíos surgen debido a la forma en que se desarrolla OSS, así como a la cultura centrada en el desarrollo [188].

El diseño de la arquitectura de OSS generalmente es altamente modular. Sin embargo, la usabilidad tiene que considerarse durante todo el proceso de desarrollo de una aplicación. Como consecuencia, puede ser difícil para los desarrolladores realizar pequeñas mejoras incrementales en la usabilidad de un software, un problema importante para gran parte del software desarrollado por voluntarios en su tiempo libre [28][35][136]. Del mismo modo, la falta de una infraestructura dedicada a las actividades de usabilidad y de artefactos de diseño, hace que sea difícil coordinar el trabajo de usabilidad en entornos de desarrollo distribuidos [28][135]. Por último, ya que esta cultura basada en el mérito siempre ha valorado el código fuente como su principal activo, en ocasiones resulta difícil para los profesionales expertos en usabilidad unirse y hacer contribuciones que se perciban como valiosas [28][136].

Al contrario de lo que habitualmente se piensa, la usabilidad no solo radica en la apariencia de la interfaz de usuario, sino en cómo el usuario interactúa con el sistema y para definirlo se utilizan cinco atributos básicos: facilidad de aprendizaje, eficiencia, recuerdo en el tiempo por parte del usuario, tasa de errores y satisfacción [92]. Para mejorar estos cinco atributos que definen la usabilidad, en los sistemas software debe mejorarse el proceso de desarrollo OSS, incorporando actividades y técnicas que permitan alcanzar o mejorar el nivel de usabilidad deseado en los productos software desarrollados.

Para estudiar las técnicas de usabilidad que OSS está utilizando, algunos investigadores han realizado estudios empíricos [23] y otros han analizado trabajos publicados en la literatura [114]. Algunos de los investigadores que han estudiado las técnicas usadas en OSS afirman que la comunidad OSS ha creado nuevas técnicas para mejorar la usabilidad de sus aplicaciones [137][188], pero un análisis detallado de la "nueva" técnica arroja que dicha técnica no es una creación de la comunidad OSS. Realmente lo que ha ocurrido es que la técnica de la IPO ha sufrido transformaciones haciéndola parecer como una técnica completamente nueva.

No hemos encontrado en la literatura investigaciones que identifiquen los aspectos clave que deben ser tenidos en cuenta para permitir el uso de técnicas de usabilidad en los desarrollos OSS, ni tampoco que establezcan cómo deben ser incorporadas tales técnicas. Más aún, afirmaciones como la mencionada en el párrafo anterior [137][188] dificultan la comprensión del uso de técnicas de usabilidad en OSS. El problema que se aborda en esta tesis doctoral

consiste en analizar cómo incorporar técnicas de usabilidad en el proceso de desarrollo OSS, teniendo en cuenta las características e idiosincrasias propias del proceso de desarrollo OSS.

Para la incorporación de las técnicas de usabilidad en el proceso de desarrollo OSS, primero es necesario comprender cómo la comunidad OSS desarrolla software y cuáles son las principales actividades de su proceso de desarrollo. Debido a que no existe un modelo o *framework* ampliamente aceptado que defina cómo se desarrolla en la práctica OSS [170], es necesario que en nuestra investigación abordemos la tarea de identificar las actividades que comprende el proceso de desarrollo OSS.

Este trabajo de investigación presenta las principales actividades que conforman el proceso de desarrollo OSS. Posteriormente, se estudian las técnicas de usabilidad (de un catálogo previamente establecido), con el objetivo de determinar las barreras que presentan estas técnicas cuando se pretenden incorporar en los desarrollos OSS. Esta incorporación debe entenderse como las adaptaciones que requieren las técnicas para poder ser incorporadas en los desarrollos OSS. Finalmente, se proponen qué técnicas de usabilidad podrían ser incorporadas en OSS y qué adaptaciones debe ser consideradas para tal fin.

1.3. Visión General de la Solución

El presente trabajo de investigación tiene tres objetivos principales. En primer lugar, analizar y proponer un mecanismo que permita la incorporación de técnicas de usabilidad en el proceso de desarrollo OSS. En segundo lugar, proponer un conjunto de técnicas de usabilidad que podrían ser incorporadas en los desarrollos OSS. En tercer lugar, validar la viabilidad del mecanismo propuesto que permite la incorporación de ciertas técnicas de usabilidad en los proyectos OSS. El análisis de cómo permitir la incorporación de técnicas de usabilidad incluye el estudio de cuáles técnicas han sido incorporadas por OSS y cómo han logrado dicha incorporación. Como resultado de este estudio, se obtiene un conjunto de condiciones desfavorables que impiden, debido a las características particulares del proceso de desarrollo OSS, que ciertas técnicas no puedan ser incorporadas en OSS tal como prescribe el área de la IPO. Por último, proponemos determinadas transformaciones a las técnicas para sortear cada una de las condiciones desfavorables.

Para alcanzar estos objetivos, se estudia tanto el proceso de desarrollo OSS como las técnicas incorporadas por la comunidad OSS. El estudio del proceso de desarrollo OSS es clave para conocer cómo esta comunidad desarrolla software y determinar qué consideraciones deben tenerse en cuenta al momento de usar técnicas de usabilidad en los proyectos OSS. El estudio de las técnicas de usabilidad incorporadas por OSS permite determinar qué técnicas y cómo han sido incorporadas. Ambos estudios se realizan a través de la literatura publicada al respecto. Para analizar dicha literatura, se llevan a cabo dos *Systematic Mapping Study* (SMS), uno por estudio.

Posteriormente, se analiza cuáles son las características de las técnicas de usabilidad que suponen impedimentos para su aplicación en los desarrollos OSS. A fin de determinar tales impedimentos, se tienen en cuenta lo prescrito por la IPO para cada una de las técnicas, así como también el modo de desarrollo que sigue la comunidad OSS. Para disponer de una guía de cuales técnicas debían ser estudiadas, se toma como referencia el catálogo recopilado por investigadores del área de la IS. Cada una de las técnicas del catálogo se analiza a partir de las referencias de la literatura de la IPO correspondientes.

Cada una de las técnicas de usabilidad incorporadas por OSS se estudia con el objetivo de determinar cómo han sido incorporadas en sus desarrollos, es decir, cómo han sorteado los impedimentos identificados. Las técnicas que tienen en común un impedimento se agrupan y se asocian al impedimento las soluciones correspondientes dadas por la comunidad OSS. Estas soluciones se proporcionan en forma de adaptaciones a las técnicas. Como resultado de este análisis, en este trabajo, se obtiene una clasificación de las técnicas de acuerdo a cada una de las adaptaciones propuestas.

Una vez se determinan tanto las adaptaciones realizadas por la comunidad OSS a las técnicas como así también los impedimentos que las motivan, se procede a analizar cuáles de las técnicas no incorporadas por OSS pueden ser modificadas con las adaptaciones propuestas. En este análisis se consideran tanto los impedimentos identificados para cada una de estas técnicas como si la adaptación a realizar sobre la técnica no va en contra de su esencia ni en contra de la filosofía de trabajo OSS.

1.4. Estructura del Trabajo

Este trabajo de investigación presenta la incorporación de la usabilidad en el proceso de desarrollo OSS y ha sido dividido en los siguientes capítulos:

- El primer capítulo introduce el trabajo de investigación y es el presente capítulo.
- La revisión del estado del arte referente al problema de investigación planteado se presenta en el Capítulo 2.
- Tanto el planteamiento del problema como la aproximación a la solución son presentados en el Capítulo 3.
- Los diferentes métodos de investigación utilizados para resolver el problema de investigación son discutidos en el Capítulo 4.
- En el Capítulo 5 se analiza desde una perspectiva amplia el estado actual de la usabilidad en la comunidad OSS. En concreto, investigamos: por qué la usabilidad de las aplicaciones OSS es baja; si la comunidad está tomando medidas para mejorar la usabilidad de OSS; qué técnicas de usabilidad se están adoptando en los desarrollos OSS; y cómo se están adoptando e integrando técnicas de usabilidad en el proceso de desarrollo OSS.
- En el Capítulo 6 se determinan cuáles son las condiciones desfavorables que impiden el uso de las técnicas de usabilidad en los desarrollos OSS y se analizan qué tipos de transformaciones son necesarias para poder incorporarlas. Además, se analizan cuáles técnicas pueden ser usadas en OSS gracias a las transformaciones propuestas.
- En el Capítulo 7 se presenta la aplicación de la solución propuesta y la evaluación de los resultados obtenidos.
- Finalmente, en el Capítulo 8 se detallan las conclusiones obtenidas de la realización del presente trabajo de investigación y las futuras líneas de investigación.

- Tras la bibliografía consultada en la realización de esta investigación, en los anexos se incluyen los estudios primarios sobre el proceso de desarrollo OSS (Anexo A), los estudios primarios sobre usabilidad en el proceso de desarrollo OSS (Anexo B), la descripción de los modelos de procesos de desarrollo OSS encontrados en la literatura (Anexo C), el listado de los problemas de usabilidad identificados por cada estudio primario (Anexo D), el listado de técnicas de usabilidad incorporadas por OSS según la literatura (Anexo E) y finalmente, el catálogo de técnicas de la IPO (Anexo F).

1.5. Contribuciones y Publicaciones Derivadas

La Tabla 1.1 presenta para cada una de las tareas realizadas en la presente investigación las contribuciones y publicaciones derivadas. Para cada una de las contribuciones se especifica tanto la publicación derivada (detallada después de la tabla) como el estado de la publicación y el evento/revista dónde fué presentado. Los estados de la publicación pueden ser: (P) publicado, (E) enviado y en espera de respuesta y (E-C) en construcción.

Tabla 1.1: Contribuciones y Publicaciones Derivadas de la Investigación.

Tarea	Contribución/Resultados	Publ.	Est.	Dónde
Extensión de una Técnica de Usabilidad para su Incorporación en la Actividad de Análisis del Proceso de IS	Esta tarea nos ha permitido verificar la viabilidad de modificar una técnica de la IPO, con el objetivo de que alcance los estándares de sistematización de la IS para permitir así su incorporación en las actividades del proceso de desarrollo OSS. Para ello, hemos sistematizado y formalizado la técnica Personas para construir una versión modificada de la técnica, que hemos incorporado en la actividad de requisitos. Los resultados de la investigación contribuyen en la dirección de incorporar el conocimiento de la IPO en la práctica habitual de la IS, emplean- do la técnica Personas como un puente que acerque ambas disciplinas. Esta in- vestigación constituye el primer acerca- miento a cómo poder incorporar técnicas de usabilidad en un proceso de desarrollo.	WI-1	P	I-USED
		CI-1	P	ENC
		CI-2	P	INTERACCION
		RIO-1	P	Revista Faz
		RIO-2	P	Revista Biblos-e
		CN-1	P	JISBD
		CI-3	P	CCGIDIS
		JCR-1	P	IST
Estudio de la Literatura sobre el Proceso de Desarrollo OSS	Identificación de los trabajos de investigación publicados en la literatura (en total 22) que describen las actividades que conforman el proceso de desarrollo OSS.	CN-2	P	JISBD
		CI-4	P	EASE

Tabla 1.1: Contribuciones y Publicaciones Derivadas de la Investigación (continuación).

Tarea	Contribución/Resultados	Publ.	Est.	Dónde
Análisis del Proceso de Desarrollo OSS	Definición de un mapa del proceso de desarrollo OSS obtenido a partir de la revisión de la literatura. Como resultado de esta revisión, se han extraído un total de 25 modelos de proceso.		E-C	
	Análisis de las diferencias y similitudes existentes entre el proceso de desarrollo OSS y el proceso de desarrollo comercial.	CN-3	P	JISBD
		CI-5	P	PROFES
		CL-1	P	JIISIC
Estudio de la Literatura sobre la Usabilidad en OSS	Identificación y clasificación de los trabajos de investigación publicados en la literatura (en total 46) que estudian diferentes aspectos de la usabilidad en OSS.		E-C	
	Análisis del estado de la usabilidad en los desarrollos OSS. Particularmente, se ha dado respuesta a las siguientes preguntas: ¿porqué la usabilidad de OSS es baja?, ¿la comunidad OSS está tomando medidas para mejorar la usabilidad de OSS?, ¿cuáles técnicas de usabilidad están siendo incorporadas en los desarrollos OSS? y ¿cómo estas técnicas de usabilidad están siendo adoptadas e integradas en el proceso de desarrollo OSS?	EJCR-1	E	TSE
Análisis de las Técnicas de Usabilidad Aplicadas en OSS	Identificación de las técnicas de usabilidad incorporadas en algunos proyectos OSS estudiados en la literatura.	EJCR-1	E	TSE
	Clasificación de las técnicas adoptadas por OSS de acuerdo al tipo de transformación que han sufrido para poder ser incorporadas. Estas transformaciones dependen del contexto y recursos de los proyectos OSS.	EJCR-1	E	TSE
	Identificación de las técnicas de usabilidad reinterpretadas, es decir, las técnicas que parecen creaciones de la comunidad OSS, pero que después de un profundo análisis y de su comparación con las técnicas de usabilidad existentes, nos ha permitido descubrir que las transformaciones que han sufrido la "disfrazan", haciendo que a primera vista parezca una técnica "propia" de OSS cuando no es así, porque comparten su esencia con alguna técnica de la IPO.	EJCR-1	E	TSE

Tabla 1.1: Contribuciones y Publicaciones Derivadas de la Investigación (continuación).

Tarea	Contribución/Resultados	Publ.	Est.	Dónde
Elaboración del Marco para la Adopción de Técnicas IPO en Desarrollos OSS	Análisis en profundidad de las técnicas de usabilidad incorporadas en algunos proyectos OSS, con el objetivo de determinar cuáles son las razones que motivan las transformaciones realizadas a las mismas.	EJCR-2	E	TSE
	Propuesta de un conjunto de transformaciones "inspirada" en las transformaciones realizadas por parte de algunos proyectos OSS a ciertas técnicas de usabilidad.	EJCR-2	E	TSE
	Propuesta de una nueva adaptación para dar solución al hecho de que ciertas técnicas de usabilidad requieren contar previamente con cierta información obtenida por unos canales prescritos por la IPO que no pueden ser seguidos en OSS porque, por ejemplo, en esta comunidad no es posible realizar entrevistas contextuales.	EJCR-2	E	TSE
	Creación de familias de transformaciones a partir del estudio de las realizadas por algunos proyectos OSS.	EJCR-2	E	TSE
	Con base en la generalización de las adaptaciones, se han analizado todas las técnicas de usabilidad (según el catálogo tomado como base) para determinar cuáles podrían ser incorporadas en OSS. Según nuestro análisis, se ha encontrado que cerca del 80 % de las técnicas de usabilidad podrían ser incorporadas en OSS con transformaciones ligeras. Sin embargo, sólo cerca del 40 % de las técnicas podrían ser reinterpretadas. Se ha identificado que las técnicas de usabilidad candidatas a ser reinterpretadas, son todas aquellas que requieran de la participación de varios usuarios físicamente reunidos.	EJCR-2	E	TSE
	Mapeo de las técnicas de usabilidad existentes (de acuerdo al catálogo tomado como base) según los tipos de transformaciones que pueden tener. Como resultado de este mapeo, obtenemos un marco general que permite la integración de técnicas de usabilidad en los desarrollos OSS.	EJCR-2	E	TSE

Tabla 1.1: Contribuciones y Publicaciones Derivadas de la Investigación (continuación).

Tarea	Contribución/Resultados	Publ.	Est.	Dónde
Evaluación de la Viabilidad del Marco para la Adopción de Técnicas IPO en Desarrollos OSS	Participación como voluntario en el proyecto OSS "FreeMind" con el objetivo de evaluar el marco de integración propuesto. "FreeMind" es un proyecto pequeño pero con un largo recorrido dentro de la comunidad OSS.	EJCR-3	E-C	IST
	Participación en el proyecto "OpenOffice Writer" con el objetivo de evaluar el marco de integración propuesto en un proyecto OSS grande y popular.	EJCR-3	E-C	IST
	Los resultados obtenidos con la incorporación de técnicas en los dos proyectos OSS han sido satisfactorios. Desde un comienzo los administradores del proyecto se han mostrado entusiastas con la idea de aplicar técnicas de usabilidad. Algunos usuarios se han mostrado muy interesados en participar. Incluso algunos usuarios agradecían este tipo de iniciativas.	EJCR-3	E-C	IST
	A partir de la participación en los dos proyectos anteriores, se han identificado las dificultades y facilidades a la hora de participar como voluntarios para trabajar en temas de usabilidad en una comunidad OSS.	EJCR-3	E-C	IST

El siguiente es el listado de las publicaciones derivadas del presente trabajo de investigación. Todas estas publicaciones están directamente relacionadas con la presente tesis doctoral. Como se observa en la Tabla 1.1 algunas publicaciones han sido enviadas y estamos a la espera de respuesta por parte de las revistas.

Revistas Internacionales con Factor de Impacto

- (JCR-1) Silvia T. Acuña, **John W. Castro** y Natalia Juristo. (2012). A HCI Technique for Improving Requirements Elicitation. *Information and Software Technology*, vol. 54(12), pp. 1357-1375. ISSN: 0950-5849.
  Índice de Calidad: **Factor de Impacto 1.522, Q1**; 1 cita obtenida en Web of Science (Thomson Reuters); 3 citas obtenidas en ScienceDirect; 7 citas obtenidas en Google Scholar.
  Relación con la Tesis: El contenido de este artículo está relacionado con la sección 6.6.

Revistas Iberoamericanas

- (RIO-1) **John W. Castro** y Silvia T. Acuña. (2009). Integrando la Técnica Personas en la Actividad de Análisis de Requisitos. *Revista Faz No.3*, Online. http://www.revistafaz.org/index_n3.html, pp. 78-94. ISSN: 0718-526X.
  Índice de Calidad: Revista que cuenta con comité científico internacional y contiene solo artículos de investigación científica; Artículo sometido a evaluación externa por pares; 100 % de artículos de autores no vinculados con la institución editora.
  Relación con la Tesis: El contenido de este artículo está relacionado con la sección 6.6.

- (RIO-2) Silvia T. Acuña y **John W. Castro**. (2011). Extensión de la Técnica Personas de la Disciplina Interacción Persona-Ordenador y su Incorporación en las Actividades de Requisitos de la Ingeniería del Software. *Revista Biblos-e UAM*, Online. http://hdl.handle.net/10486/6915, pp. 1-26. Universidad Autónoma de Madrid, Madrid, España.
  Relación con la Tesis: El contenido de este artículo está relacionado con la sección 6.6.

Capítulo de Libro

- (CL-1) **John W. Castro** y Silvia T. Acuña. (2014). Diferencias y Similitudes de las Actividades de Requisitos en los Procesos de Desarrollo de Software Tradicional y Open Source. In *Ingeniería de Software e Ingeniería del Conocimiento: Dos Disciplinas Interrelacionadas*, Capítulo XX, pp. 341-362. Sello Editorial Universidad de Medellín, Medellín (Colombia). ISBN: 978-958-8815-31-2.
  Índice de Calidad: Capítulo sometido a evaluación externa por pares.
  Relación con la Tesis: Este trabajo está descrito en el anexo C.

Congresos Internacionales

- (CI-1) **John W. Castro**, Silvia T. Acuña y Natalia Juristo. (2008). Integrating the Personas Technique into the Requirements Analysis Activity. In *Proceedings of the Ninth Mexican International Conference on Computer Science* (ENC'08). IEEE Computer Society. Mexicali, Baja California (México), pp. 104-112. ISBN: 978-0-7695-3439-8.
  Índice de Calidad: Artículo sometido a evaluación externa por pares; 1 cita obtenida en IEEE Xplore; 1 cita obtenida en Web of Science (Thomson Reuters); 15 citas obtenidas en Google Scholar.
  Relación con la Tesis: El contenido de este artículo está relacionado con la sección 6.6.

- (CI-2) **John W. Castro**, Silvia T. Acuña y José A. Macías. (2009). Application of Personas Technique as a Bridge to Bring the Disciplines of HCI and SE. In *Proceedings of the X International Conference on Human Computer Interaction* (INTERACCION'09). Universidad Pompeu Fabra, Barcelona (Spain), pp. 1-10. ISBN: 978-84-692-5005-1.
  Índice de Calidad: Artículo sometido a evaluación externa por pares.
  Relación con la Tesis: El contenido de este artículo está relacionado con la sección 6.6.

- (CI-3) **John W. Castro** y Silvia T. Acuña. (2011). Extension of Personas Technique for the Requirements Stage. In *Proceedings of the 1st International Symposium on Communicability, Computer Graphics and Innovative Design for Interactive Systems* (CCGIDIS'11). Universidad de Córdoba, Córdoba (Spain), pp. 102-111. ISBN: 978-88-96471-09-8.
  Índice de Calidad: Artículo sometido a evaluación externa por pares.
  Relación con la Tesis: El contenido de este artículo está relacionado con la sección 6.6.

- (CI-4) Silvia T. Acuña, **John W. Castro**, Oscar Dieste y Natalia Juristo. (2012). A Systematic Mapping Study on the Open Source Software Development Process. In *Proceedings of the 16th International Conference on Evaluation & Assessment in Software Engineering* (EASE'12). Universidad de Castilla-La Mancha, Ciudad Real (Spain), pp. 42-46. ISBN: 978-1-84919-541-6.
  Índice de Calidad: **Core A**; 1 cita obtenida en Web of Science (Thomson Reuters); 3 citas obtenidas en Google Scholar.
  Relación con la Tesis: Este artículo está relacionado con las secciones 2.1 y 2.3.

- (CI-5) **John W. Castro** y Silvia T. Acuña. (2012). Differences between Traditional and Open Source Development Activities. In O. Dieste, A. Jedlitschka, and N. Juristo (Eds), *Proceedings of the 13th International Conference on Product-Focused Software Process Improvement* (PROFES'12), Lecture Notes in Computer Science, volume 7343, pp. 131-144. Springer, Madrid (Spain). ISBN: 978-3-642-31062-1.
  Índice de Calidad: **Core B**; 3 citas obtenidas en Google Scholar.
  Relación con la Tesis: Este trabajo está relacionado con el anexo C.

Workshop Internacionales

- (WI-1) **John W. Castro**, Silvia T. Acuña y Natalia Juristo. (2008). Enriching Requirements Analysis with the Personas Technique. In *Proceedings of the International Workshop on Interplay between Usability Evaluation and Software Development* (I-USED'08). University of Leicester, Pisa (Italia), pp. 13-17. ISSN: 1613-0073.
  Índice de Calidad: Artículo sometido a evaluación externa por pares; 4 citas obtenidas en Google Scholar.
  Relación con la Tesis: El contenido de este artículo está relacionado con la sección 6.6.

Congresos Nacionales

- (CN-1) **John W. Castro** y Silvia T. Acuña. (2011). Mejora de la Educción de Requisitos Mediante la Técnica Personas de IPO. In *Proceedings of the XVI Jornadas de Ingeniería del Software y Bases de Datos* (JISBD'11). Universidade da Coruña, A Coruña (España), pp. 375-388. ISBN: 978-84-9749-486-1.
  Índice de Calidad: Artículo sometido a evaluación externa por pares.
  Relación con la Tesis: El contenido de este artículo está relacionado con la sección 6.6.

- (CN-2) **John W. Castro** y Silvia T. Acuña. (2011). Comparativa de Selección de Estudios Primarios en una Revisión Sistemática. In *Proceedings of the XVI Jornadas de Ingeniería del Software y Bases de Datos* (JISBD'11). Universidade da Coruña, A Coruña (España), pp. 319-332. ISBN: 978-84-9749-486-1.
  Índice de Calidad: Artículo sometido a evaluación externa por pares.
  Relación con la Tesis: El contenido de este artículo está relacionado en las secciones 2.1 y 2.3.

- (CN-3) **John W. Castro**, Silvia T. Acuña y Oscar Dieste. (2012). Diferencias entre las Actividades de Mantenimiento en los Procesos de Desarrollo Tradicional y Open Source. In *Proceedings of the XVII Jornadas de Ingeniería del Software y Bases de Datos* (JISBD'12). Universidad de Almería, Almería (España), pp. 651-664. ISBN: 978-84-15487-28-9.
  Índice de Calidad: Artículo sometido a evaluación externa por pares.
  Relación con la Tesis: Este artículo está relacionado con el anexo C.

Enviados a Revistas con Factor de Impacto

- (EJCR-1) **John W. Castro**, Silvia T. Acuña y Natalia Juristo. (2014). Usability in Open Source Software Development. *IEEE Transactions on Software Engineering*, pp. 1-27.
  Relación con la Tesis: El contenido de este artículo está relacionado con el Capítulo 5.
  Resultados: El editor de la revista ha comentado lo siguiente: "*Thank you for an impressive submission covering the literature review of usability in open source community. All reviewers agreed that this is timely and substantial contribution. However, they all pointed out a similar set of fundamental issues that may be impossible to address within the scope of a major revision. You may resubmit your paper, but it will be treated as a NEW submission and given a new log number. If you choose to resubmit your paper please refer to this original log number (TSE-2014-02-0032), and we will include your previous manuscript's history in it's files and forward the necessary information to the Editor-in-Chief and Associate Editor. The manuscript will then undergo a new review process...*"

- (EJCR-2) **John W. Castro**, Silvia T. Acuña y Natalia Juristo. (2014). Enhancing the Use of Usability Techniques in Open Source Developments. *IEEE Transactions on Software Engineering*, pp. 1-26.
  Relación con la Tesis: Este artículo está relacionado con el Capítulo 6.

- (EJCR-3) **John W. Castro**, Silvia T. Acuña y Natalia Juristo. (2014). Modifying Usability Techniques for their Application in Open Source Software: Two Case Studies. *Information and Software Technology*, pp. 1-20.
  Relación con la Tesis: Este trabajo está descrito en el Capítulo 7.

CAPÍTULO 2

ESTADO DE LA CUESTIÓN

La revisión de la literatura permite encontrar y analizar las publicaciones relacionadas con las áreas en las cuales se quiere investigar y es usualmente el primer paso para iniciar un trabajo de investigación. Es este capítulo se presenta la revisión de las publicaciones tanto de los modelos de proceso de desarrollo OSS existentes, así como también de las publicaciones relacionadas con la usabilidad en OSS. Para ello, se ha aplicado un proceso de revisión conocido en inglés como *Systematic Mapping Study* (SMS). Un SMS es una metodología que consiste en investigar la literatura sobre un área de interés particular, con el objetivo de determinar la naturaleza, el alcance y la cantidad de estudios primarios publicados [147]. Los SMS categorizan los estudios primarios, mostrando una visión sintetizada del área de investigación que está siendo considerada.

2.1. Modelos de Proceso de Desarrollo OSS

Esta sección describe el SMS realizado para responder a la pregunta de investigación:

RQ: ¿Qué actividades conforman los modelos de proceso OSS?

El proceso SMS se inició con la identificación de las palabras clave y las cadenas de búsqueda construidas a partir de la cuestión de investigación. Se realizó inicialmente una búsqueda tradicional, a partir de la cual se obtuvieron algunos artículos que fueron estudiados para determinar los términos de búsqueda más apropiados para el SMS. Estos términos de búsqueda fueron validados y completados por dos expertos investigadores en el área de la IS. Las cadenas de búsqueda empleadas fueron:

- OS-SPM: **O**pen **S**ource AND **S**oftware **P**rocess **M**odel
- OS-SDP: **O**pen **S**ource AND **S**oftware **D**evelopment **P**rocess
- OS-DP: **O**pen **S**ource AND **D**evelopment **P**rocess
- FS-SPM: **F**ree **S**ource AND **S**oftware **P**rocess **M**odel
- FS-SDP: **F**ree **S**ource AND **S**oftware **D**evelopment **P**rocess
- FS-DP: **F**ree **S**ource AND **D**evelopment **P**rocess

Las bases de datos (BBDD) electrónicas usadas en el SMS fueron: IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect y Scopus. Cada una de las seis cadenas de búsqueda definidas fue aplicada a cada una de las BBDD anteriores. Durante la búsqueda se fijó como fecha límite de publicación de los artículos el 31 de marzo de 2010. Para determinar los estudios primarios que tienen relación con nuestra pregunta de investigación, se utilizaron los siguientes criterios de inclusión y exclusión.

Criterios de inclusión:

- El título del artículo debía contener las palabras open source o free source; **OR**
- Las palabras clave hacían alusión al proceso de desarrollo OSS; **OR**
- El resumen hacía alusión a alguna cuestión sobre el proceso de desarrollo OSS; **OR**
- El artículo describe el proceso de desarrollo OSS; **OR**
- El artículo presenta las actividades del proceso de desarrollo OSS; **OR**
- El artículo presenta las actividades del proceso de desarrollo de software free source; **OR**
- El artículo discute el proceso de desarrollo seguido en un proyecto particular OSS; **OR**
- El artículo presenta una propuesta del proceso de desarrollo OSS.

Criterios de exclusión:

- El artículo no discute el proceso de desarrollo OSS; **OR**
- El artículo no describe las actividades del proceso de desarrollo de software open source; **OR**
- El artículo no describe las actividades del proceso de desarrollo de software free source.

Las búsquedas se realizaron en el siguiente orden: *IEEE Xplore*, *ACM Digital Library*, *SpringerLink*, *ScienceDirect* y *Scopus*. La Tabla 2.1 presenta para cada una de las BBDD electrónicas consideradas los campos donde fueron aplicados las cadenas de búsqueda definidas previamente. Los campos disponibles para realizar la búsqueda no eran siempre los mismos porque dependían de las opciones de cada una de las BBDD. Posteriormente, se realizó una búsqueda en cada uno de los campos disponibles según la BBDD empleando las seis cadenas de búsqueda anteriores.

Tabla 2.1: Campos de Búsqueda Empleados en cada BBDD.

BBDD	Campos de Búsqueda
IEEE Xplore	"Abstract", "Publication Title", **"Index Terms"**
ACM Digital Library	**"Abstract"**, "Title", "Abstract OR Title"
SpringerLink	"Abstract", "Title", "All Text", **"All Text OR Abstract"**
ScienceDirect	"Abstract", "Title", "Keywords", **"Abstract OR Title OR Keywords"**
Scopus	**"All Title OR Abstract OR Keywords"**

Con base en el número de registros obtenidos por el campo de búsqueda en cada BBDD se ha definido el campo que sería usado en la búsqueda final. La decisión ha sido tomada en función del número total de registros obtenidos y el porcentaje de diferencia entre cada campo de búsqueda. Por ejemplo, en la base de datos ACM Digital Library el número total de registros obtenidos por los campos *"Abstract"* y *"Abstract OR Title"* eran valores muy cercanos, mientras

que el número total de registros obtenidos por el campo *"Title"* eran muy pocos comparados con los otros dos (*"Abstract"* y *"Abstract OR Title"*). Así, se descarta el campo *"Title"* debido al escaso número de registros que aporta. Finalmente, como la diferencia entre los campos *"Abstract"* y *"Abstract OR Title"* es muy reducida, se decide tomar únicamente el campo de búsqueda que proporciona más registros (*"Abstract"*). En la Tabla 2.1 se puede apreciar en negrita el campo de búsqueda seleccionado para cada BBDD.

La estrategia de selección de estudios primarios está representada en la Figura 2.1. En primer lugar, una vez definidos las cadenas y campos de búsqueda para cada BBDD (Tabla 2.1), se procedió a realizar las búsquedas. El conjunto de artículos resultado de la búsqueda ha sido denominado "artículos *Encontrados*". Los artículos *Encontrados* fueron revisados mediante el examen del título, palabras clave y resumen (estos dos últimos cuando estaban disponibles). Aquellos artículos que podrían contener información acerca del proceso de desarrollo OSS fueron incluidos en el grupo de "artículos *Preseleccionados*". Cuando se completó el grupo final de artículos *Preseleccionados*, se eliminaron los artículos duplicados entre cada BBDD (es decir, entre los diferentes términos de búsqueda de la misma BBDD) y luego se eliminaron los duplicados entre todas las BBDD. El grupo de artículos resultantes ha sido denominado como "artículos *Preseleccionados Diferentes*".

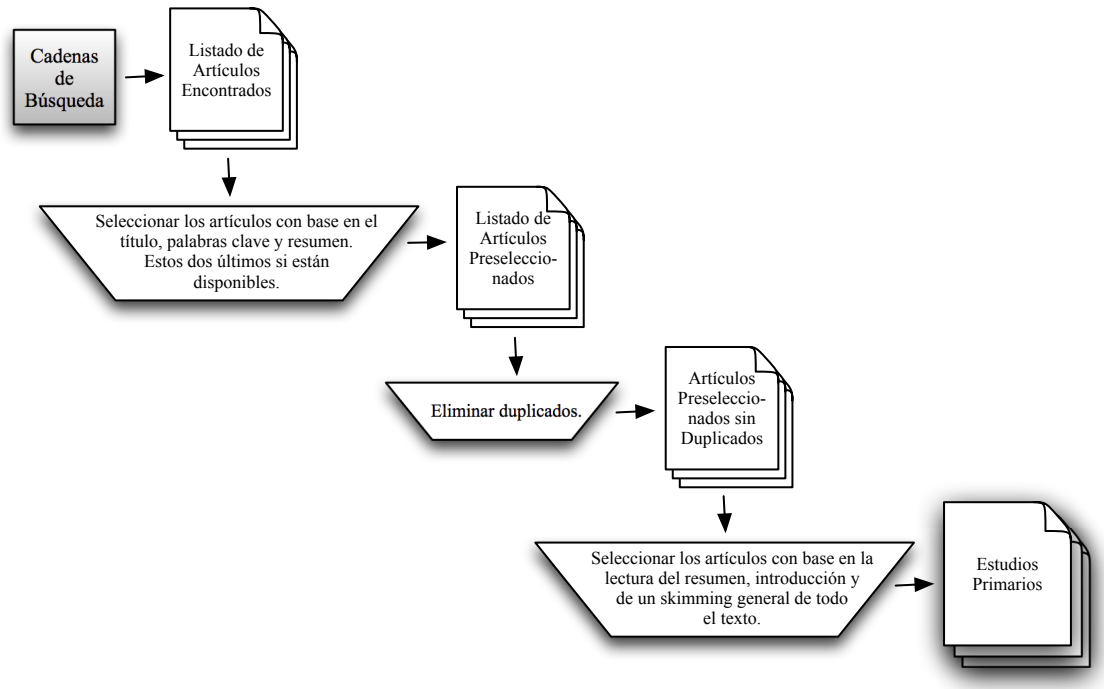


Figura 2.1: Estrategia de Selección de Estudios Primarios.

Es importante mencionar que cuando se encontraban artículos duplicados, se dejaba siempre la primera ocurrencia del artículo y se eliminaban los demás. Así, a medida que se eliminaban duplicados entre todas las BBDD, el número de artículos iba disminuyendo. Es decir, el artículo siempre se conservaba en la primera BBDD donde se encontraba. De este modo, si el orden de las búsquedas fuese cambiado, entonces el número de artículos *Preseleccionados Diferentes* que aporta cada BBDD podría ser distinto. Para cada uno de los artículos pertenecientes al grupo *Preseleccionados Diferentes* se ha leído el resumen, la introducción y se ha realizado un *skimming* (esto es, una lectura rápida) general para determinar si el artículo describía o no el proceso de desarrollo OSS. Finalmente, el nuevo grupo obtenido con estos artículos ha sido denominado *Estudios Primarios*.

Las Tablas 2.2, 2.3, 2.4, 2.5 y 2.6 muestran para cada BBDD el número de artículos en cada uno de los grupos (*Encontrados*, *Preseleccionados*, *Preseleccionados Diferentes* y *Estudios Primarios*) obtenidos durante el proceso de selección de estudios primarios. Por ejemplo, en la Tabla 2.3 se puede apreciar que en la BBDD ACM Digital Library para el término de búsqueda OS-SPM, el número total de artículos que conforman el grupo de *Preseleccionados* ha sido de 215, mientras que para el término de búsqueda OS-SDP el número total de artículos para el mismo grupo ha sido de 166. Finalmente, el número total de artículos del grupo *Preseleccionados* de la BBDD ACM Digital Library para todos los términos de búsqueda ha sido de 616.

Tabla 2.2: Número Total de Artículos Obtenidos en la BBDD IEEE Xplore.

Term. Búsq.	**Encontrados**	**Preseleccionados**	**Preseleccionados Diferentes**	**Estudios Primarios**
OS-SPM	71	30	30	0
OS-SDP	105	58	47	4
OS-DP	115	51	2	0
FS-SPM	46	5	2	0
FS-SDP	25	13	8	1
FS-DP	25	6	0	0
TOTAL	**387**	**163**	**89**	**5**

Tabla 2.3: Número Total de Artículos Obtenidos en la BBDD ACM Digital Library.

Term. Búsq.	**Encontrados**	**Preseleccionados**	**Preseleccionados Diferentes**	**Estudios Primarios**
OS-SPM	1.342	215	200	4
OS-SDP	1.507	166	42	2
OS-DP	1.354	117	17	0
FS-SPM	674	26	3	0
FS-SDP	675	43	14	0
FS-DP	568	49	8	0
TOTAL	**6.120**	**616**	**284**	**6**

Tabla 2.4: Número Total de Artículos Obtenidos en la BBDD SpringerLink.

Term. Búsq.	**Encontrados**	**Preseleccionados**	**Preseleccionados Diferentes**	**Estudios Primarios**
OS-SPM	14	5	5	0
OS-SDP	425	53	50	0
OS-DP	1.709	94	65	1
FS-SPM	1	1	0	0
FS-SDP	70	22	6	0
FS-DP	240	56	21	0
TOTAL	**2.459**	**231**	**147**	**1**

Tabla 2.5: Número Total de Artículos Obtenidos en la BBDD ScienceDirect.

Term. Búsq.	Encontrados	Preseleccionados	Preseleccionados Diferentes	Estudios Primarios
OS-SPM	26	8	8	0
OS-SDP	30	12	8	0
OS-DP	47	1	0	0
FS-SPM	37	3	3	0
FS-SDP	29	3	2	0
FS-DP	30	3	0	0
TOTAL	**199**	**30**	**21**	**0**

Tabla 2.6: Número Total de Artículos Obtenidos en la BBDD Scopus.

Term. Búsq.	Encontrados	Preseleccionados	Preseleccionados Diferentes	Estudios Primarios
OS-SPM	342	58	27	2
OS-SDP	534	79	37	0
OS-DP	397	15	13	0
FS-SPM	591	2	1	0
FS-SDP	839	1	1	0
FS-DP	401	25	1	0
TOTAL	**3.104**	**180**	**80**	**2**

La Tabla 2.7 presenta un resumen para cada BBDD del número de artículos obtenidos al aplicar las seis cadenas de búsqueda, así como el número de artículos preseleccionados. Los artículos preseleccionados son todos aquellos estudios que cumplen con los criterios de inclusión/exclusión, pero aplicados únicamente sobre el título y las palabras clave. Esta estrategia nos ha permitido filtrar rápidamente el resultado de las búsquedas, al reducir de 12.269 a 621 el número de artículos que evaluar en detalle (solo un 5,1 % del total). Este conjunto de preseleccionados diferentes no contiene duplicados.

Tabla 2.7: Número Total de Artículos Obtenidos en cada BBDD.

BBDD	Encontrados	Preselecc.	Preseleccionados Diferentes	Estudios Primarios
IEEE Xplore	387	163	89	5
ACM Digital Library	6.120	616	284	6
SpringerLink	2.459	231	147	1
ScienceDirect	199	30	21	0
Scopus	3.104	180	80	2
TOTAL	**12.269**	**1.220**	**621**	**14**

La búsqueda ha dado como resultado un total de 14 estudios primarios. A medida que se iban leyendo cada uno de estos estudios primarios se revisaban sus referencias buscando artículos que podían resultar relevantes para la investigación. Como resultado de esta revisión se encontraron 8 nuevos estudios primarios, que aunque no se encontraban en ninguna de las BBDD donde se realizó la búsqueda, sí eran relevantes. Finalmente, se obtuvieron un total de 22 estudios primarios. Se ha asignado un código alfanumérico a cada uno de estos estudios para facilitar su referencia en el presente capítulo.

Los estudios primarios seleccionados se muestran en el Anexo A, mientras que una visión sintética de los mismos se proporciona en la Figura 2.2. Esta figura consiste básicamente en dos gráficos de dispersión XY con burbujas en las intersecciones de cada categoría (lado izquierdo). Las categorías están determinadas por el año de publicación del estudio primario, tipo (revistas, conferencias, etc.) y si analiza o no un proyecto OSS en particular. El tamaño de cada burbuja está determinado por el número de estudios primarios que se han clasificado como pertenecientes al par de categorías correspondientes a las coordenadas de la burbuja. El lado derecho de la Figura 2.2 muestra el número de estudios primarios por año de publicación.

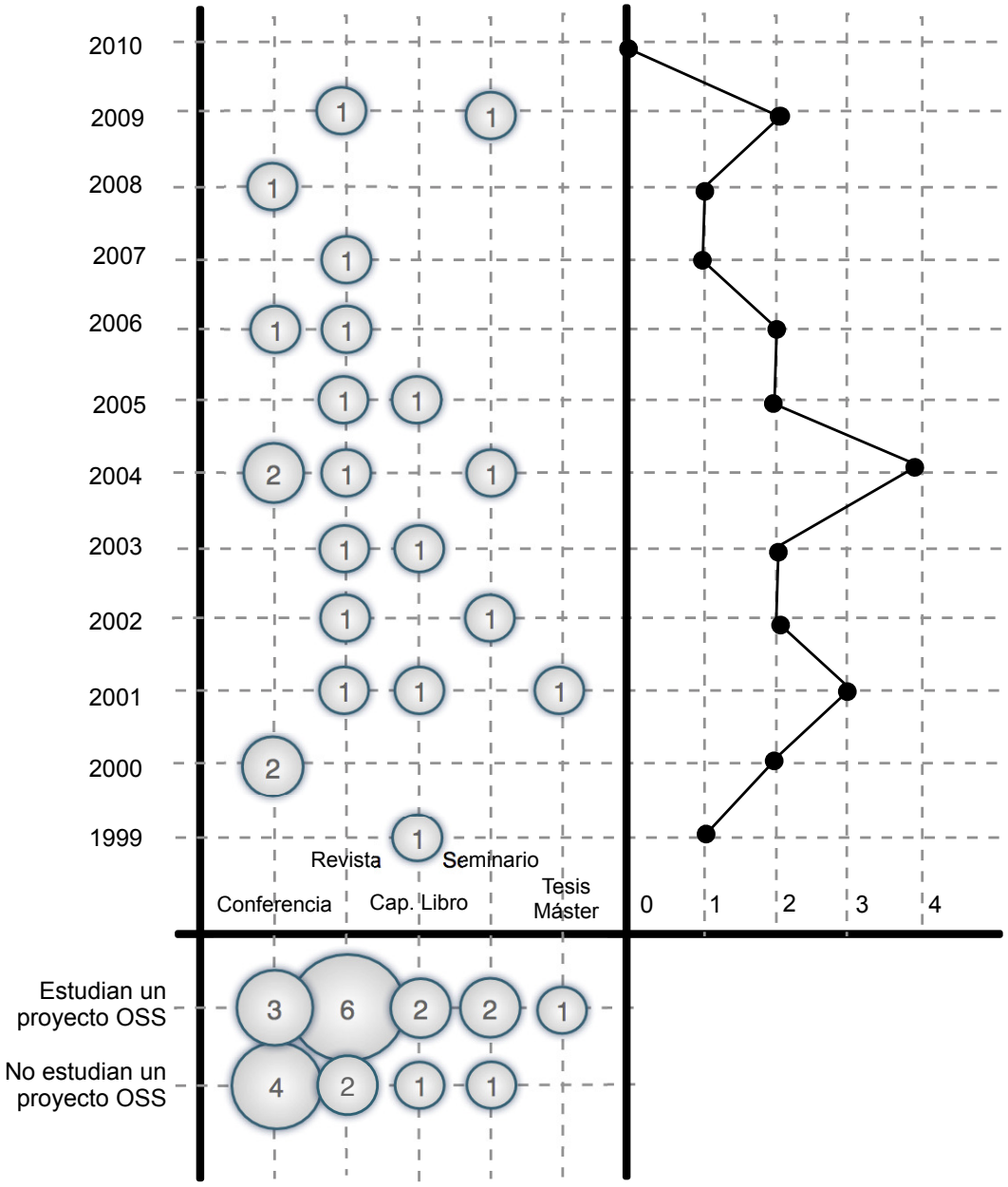


Figura 2.2: Mapeo con la Distribución de Estudios Primarios según Estudian o No un Proyecto OSS en Particular, Incluyendo Tipo y Año de Publicación.

Como se observa en la Figura 2.2, hay dos periodos de tiempo (1999-2001 y 2003-2004) con un crecimiento de trabajos de investigación, reflejando un importante interés en comprender el proceso de desarrollo seguido por la comunidad OSS. Posteriormente, hay un descenso en el número de trabajos publicados hasta el 2007, estabilizándose en los siguientes años. En cuanto a los tipos de publicación, los estudios primarios seleccionados han sido publicados principalmente en revistas y conferencias (65 %), aunque también se encuentran capítulos de libro, seminarios y tesis de máster, sobre todo en la primera mitad (2002-2006) del periodo examinado. El hecho de que en años recientes las publicaciones sean realizadas en medios especializados y sometidos a revisión de pares, sugiere una progresiva madurez del área de OSS.

Finalmente, es relevante indicar que el 60 % de las publicaciones describen el proceso de desarrollo OSS seguido en un proyecto OSS concreto. Esto es importante porque las actividades descritas por estos trabajos reflejan la realidad de cómo se lleva en la práctica el desarrollo OSS. Los proyectos OSS más populares para su estudio son "FreeBSD", "Mozilla" y "Apache Server" los cuales han sido estudiados en 7 de los 23 trabajos (30 %).

Hemos realizado una categorización de los estudios primarios según los tipos de actividades. Para ello, sólo consideramos cinco grupos de actividades basados en el Standard International IEEE 1074:2006 [94]: *Exploración de Conceptos*, *Requisitos Software*, *Diseño*, *Mantenimiento* y *Evaluación*. Para *Requisitos Software* hemos considerado el SWEBOK [187]. Esta categorización permite ubicar el proceso de desarrollo OSS en relación con el tradicional. Además, permite conocer qué actividades del proceso de desarrollo tradicional son útiles para la comunidad OSS. La utilidad se puede inferir del número de estudios que incluyen una actividad determinada.

Para realizar la categorización descrita anteriormente, se leyeron los estudios primarios prestando especial atención a las secciones donde se describía el proceso de desarrollo seguido por la comunidad OSS. Posteriormente, de la descripción del proceso de desarrollo OSS se extrajeron el nombre y descripción de cada actividad. Según la descripción de cada actividad, se realizaba un emparejamiento con su actividad análoga en el Standard International IEEE 1074:2006 [94], teniendo en cuenta el objetivo de la actividad pero no los métodos y técnicas aplicados. A continuación, se listarán para cada una de las cinco actividades (*Exploración de Conceptos*, *Requisitos Software*, *Diseño*, *Mantenimiento* y *Evaluación*) los estudios primarios que las consideran.

- **Exploración de Conceptos**

  La Figura 2.3 presenta los estudios primarios que cuentan con actividades que han sido emparejadas con el grupo *Exploración de Conceptos* del Standard International IEEE 1074:2006 [94]. Se hace referencia a los estudios primarios utilizando el código alfanumérico asignado a cada uno en el Anexo A. Este grupo consta de las actividades: *Identificar Ideas o Necesidades*, *Formular Posibles Enfoques*, *Realizar Estudios de Viabilidad* y *Refinar y Finalizar la Idea o Necesidad*. Los rectángulos con esquinas redondeadas y sombreados en gris, corresponden a las actividades del grupo *Exploración de Conceptos* donde hay estudios primarios. Todas las actividades del grupo *Exploración de Conceptos* cuentan con estudios primarios asociados. Aunque en las actividades *Realizar Estudios de Viabilidad* y *Refinar y Finalizar la Idea o Necesidad* solo hay un estudio primario en cada una de ellas, lo que indica que estas actividades no son frecuentes en el proceso de desarrollo OSS. Los estudios primarios subrayados son artículos que estudian un proyecto OSS en particular. Como se aprecia en la Figura 2.3, sólo los proyectos OSS estudiados por Ezeala y colegas [SP4] y Raymond [SP14] cuentan con actividades emparejadas en *Identificar Ideas o Necesidades*, *Formular Posibles Enfoques* y *Refinar y Finalizar la Idea o Necesidad*.

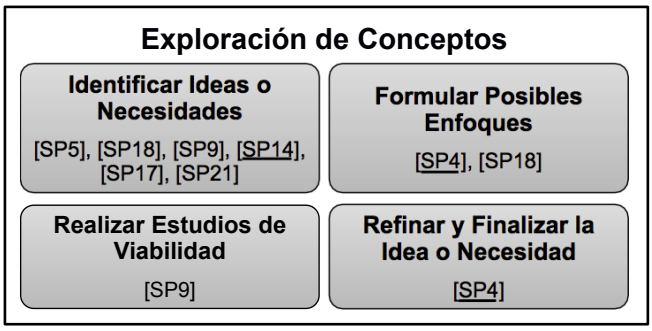


Figura 2.3: Estudios Primarios con Actividades que se Corresponden con Exploración de Conceptos.

- **Requisitos Software**

  En la Figura 2.4 se ilustran los estudios primarios que cuentan con actividades que han sido emparejadas con el grupo *Requisitos Software*. Las actividades con linea discontinua corresponden a las actividades del SWEBOK [187] que hemos adicionado para facilitar la categorización. Las actividades que tienen una mayor cantidad de estudios primarios son *Educción de Requisitos*, *Especificación de Requisitos* y *Negociación de Requisitos*. Para las actividades *Clasificación de Requisitos*, *Modelado Conceptual*, *Definir Requisitos de Interfaz* y *Priorizar e Integrar los Requisitos Software* solo un estudio primario, en cada caso, fue emparejado con estas actividades, indicando la poca presencia de estas actividades dentro del proceso de desarrollo seguido por la comunidad OSS. La actividad *Validación de Requisitos* no sería realizada en los proyectos OSS. Como ilustra la Figura 2.4, la mitad de los estudios primarios que se emparejaron en el grupo *Requisitos Software* pertenecen a estudios primarios que examinan un proyecto OSS particular.

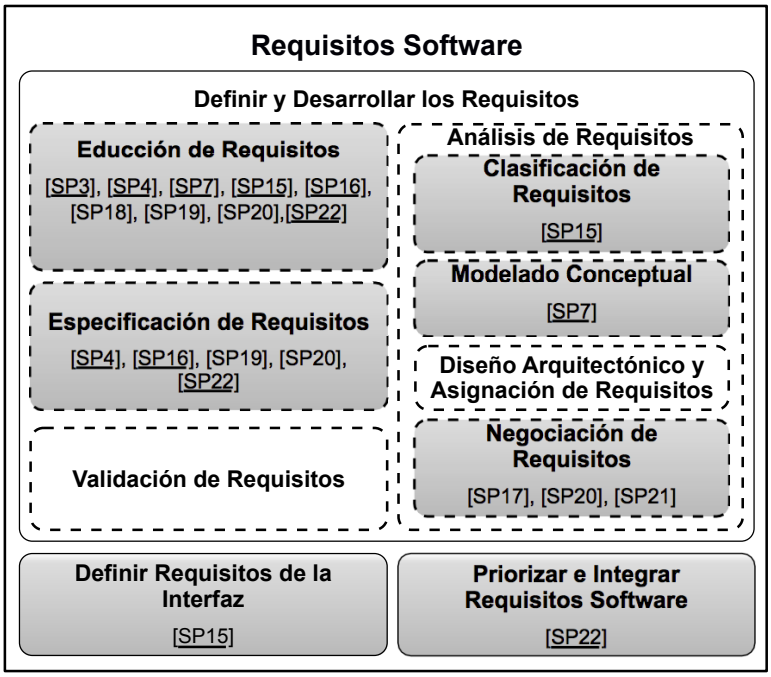


Figura 2.4: Estudios Primarios con Actividades que se Corresponden con Requisitos Software.

- **Diseño**

  La Figura 2.5 ilustra los estudios primarios que referencian actividades que han sido emparejadas con el grupo de actividades de *Diseño*. Este grupo contiene las actividades: *Realizar el Diseño Arquitectónico*, *Diseño de la Base de Datos*, *Diseño de las Interfaces* y *Realizar el Diseño Detallado*. Como se aprecia en la Figura 2.5, sólo las actividades *Realizar el Diseño Arquitectónico* y *Realizar el Diseño Detallado* cuentan con estudios primarios. Aunque esta última solo con un estudio primario. Lo anterior refleja como en el proceso de desarrollo OSS no existen actividades análogas al *Diseño de la Base de Datos*, y *Realizar el Diseño Detallado* llevadas a cabo en el desarrollo de software tradicional. Los estudios primarios subrayados en la Figura 2.5 corresponden a aquellos que estudian un proyecto OSS en particular, siendo mayoría, y reflejando la importancia de la actividad *Realizar el Diseño Arquitectónico* en proyectos OSS concretos.

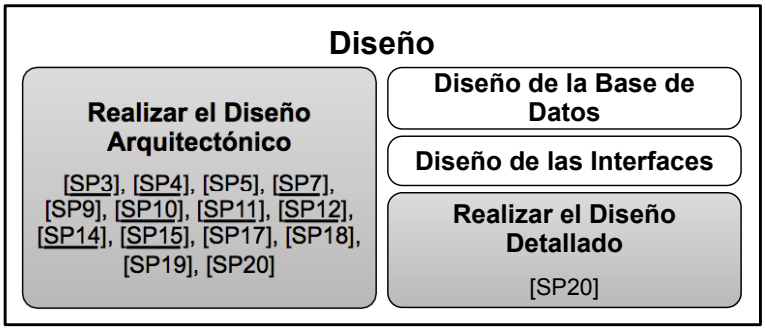


Figura 2.5: Estudios Primarios con Actividades que se Corresponden con Diseño.

- **Mantenimiento**

  En la Figura 2.6 se ilustran los estudios primarios que cuentan con actividades que han sido emparejadas con el grupo de actividades de *Mantenimiento*. Este grupo está conformado por las actividades *Identificar las Necesidades de Mejora del Software*, *Implementar el Método de Reporte de Problemas* y *Reaplicar el Proceso del Ciclo de Vida del Proceso Software SPLCP (del inglés, Software Process Life Cycle Process)*. Las actividades que cuentan con una mayor cantidad de estudios primarios son *Identificar las Necesidades de Mejora del Software* e *Implementar el Método de Reporte de Problemas*. Como se aprecia en la Figura 2.6, la mayor parte de los estudios primarios corresponde a aquellos que estudian un proyecto OSS en particular, reflejando la importancia del grupo de actividades de *Mantenimiento* para la comunidad OSS.

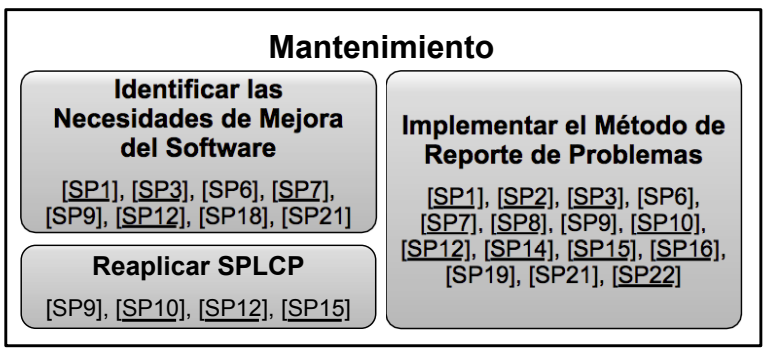


Figura 2.6: Estudios Primarios con Actividades que se Corresponden con Mantenimiento.

- **Evaluación**

  La Figura 2.7 presenta los estudios primarios que referencian actividades que han sido emparejadas con el grupo de actividades *Evaluación*. Este grupo está conformado por las actividades *Realizar Revisiones*, *Crear Datos de Prueba*, *Reportar los Resultados de la Evaluación*, *Confirmar Acreditación de Seguridad*, *Crear Matriz de Trazabilidad*, *Realizar Auditorías*, *Desarrollar Procedimientos de Prueba* y *Ejecutar Pruebas*. La mitad de estas actividades han sido emparejadas con estudios primarios que tienen actividades análogas a: *Realizar Revisiones*, *Ejecutar Pruebas*, *Desarrollar Procedimientos de Prueba* y *Reportar los Resultados de la Evaluación*. Aunque estas dos últimas actividades solo tienen asociado un estudio primario cada una. La actividad con mayor número de estudios primarios asociados es *Realizar Revisiones*, lo que refleja la importancia de las revisiones de pares en el proceso de desarrollo OSS.

  Como se observa en la Figura 2.7 las actividades del proceso de desarrollo tradicional pertenecientes al grupo de actividades de *Evaluación* más útiles para el proceso de desarrollo seguido por la comunidad OSS son *Realizar Revisiones* y *Ejecutar Pruebas*. La mayoría de los estudios primarios que se emparejaron en el grupo de actividades de *Evaluación* pertenecen a estudios primarios que estudian un proyecto OSS particular.

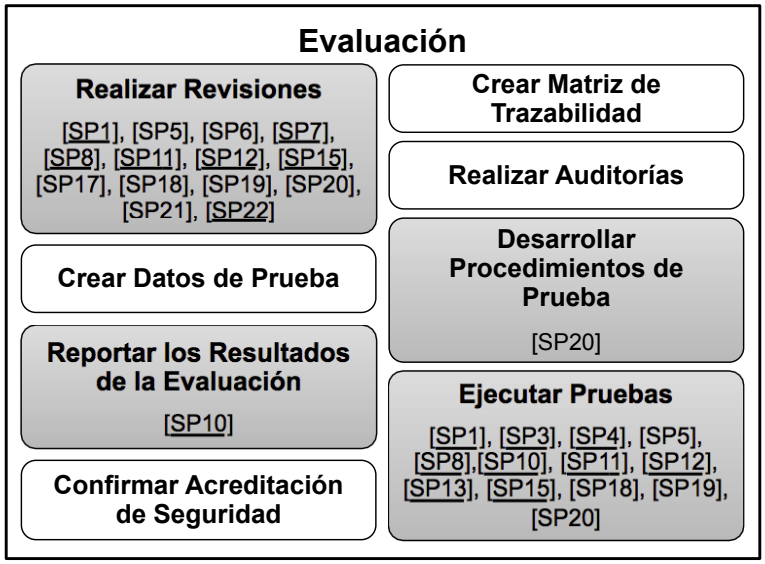


Figura 2.7: Estudios Primarios con Actividades que se Corresponden con Evaluación.

A partir del SMS realizado, se encontraron un total de 22 estudios primarios, publicados principalmente en revistas y conferencias. En el Anexo C describimos cada uno de los modelos de proceso identificados en la literatura. El 60 % de los estudios primarios estudian un proyecto OSS en particular. Los proyectos OSS más populares para su estudio son "FreeBSD", "Mozilla" y "Apache Server". Los estudios primarios que cuentan con actividades emparejadas con la mayoría de las cinco actividades del proceso tradicional consideradas son aquellos que estudian proyectos OSS, reflejando la importancia de estudiar los mismos.

2.2. Usabilidad en OSS

El objetivo de esta sección es describir el SMS realizado para responder a la pregunta de investigación:

RQ: ¿Cuál es el estado actual de la usabilidad en los desarrollos OSS?

El proceso SMS se inició con la identificación de las palabras clave y las cadenas de búsqueda construidas a partir de la cuestión de investigación. Se realizó inicialmente una búsqueda tradicional, a partir de la cual se obtuvieron algunos artículos que fueron estudiados para determinar las cadenas de búsqueda más apropiadas. Estas cadenas fueron validadas y completadas por dos expertos investigadores en las áreas de la IS y de la IPO. Las cadenas de búsqueda empleadas fueron:

- OS-U: **O**pen **S**ource AND **U**sability
- FS-U: **F**ree **S**ource AND **U**sability

Las siguientes fueron las BBDD electrónicas usadas: *ScienceDirect*, *IEEE Xplore*, *ACM Digital Library*, *SpringerLink*, *Scopus* y *FLOSShub*. Cada una de las dos cadenas de búsqueda definidas fue aplicada a cada una de las seis BBDD seleccionadas. La Tabla 2.8 presenta para cada una de estas BBDD electrónicas los campos empleados en la búsqueda. Los campos empleados no fueron siempre los mismos, ya que dependían de las opciones que brindaban cada una de las BBDD.

Tabla 2.8: Campos de Búsqueda Empleados en cada BBDD.

BBDD	Campos de Búsqueda
ScienceDirect	"Abstract OR Title OR Keywords"
IEEE Xplore	"Index Terms"
ACM Digital Library	"Abstract"
SpringerLink	"Title & Abstract"
Scopus	'Article Title OR Abstract OR Keywords"
FLOSShub	"Will All of the Words"

Durante la búsqueda se fijó como fecha límite de publicación de los artículos el 09 de agosto de 2013. Para determinar los estudios primarios relevantes a nuestra pregunta de investigación, se utilizaron los siguientes criterios de inclusión y exclusión.

Criterios de inclusión:

- El título del artículo debe contener las palabras *'open source'* o *'free source'*; **AND**
- El título del artículo debe contener la palabra *'usability'*; **OR**
- Las palabras clave hacían alusión a algún aspecto relacionado con la usabilidad en OSS; **OR**
- El resumen mencionaba alguna cuestión sobre la usabilidad en OSS.

Criterios de exclusión:

- El artículo no presenta ningún aspecto relacionado con la usabilidad en *'open source'*; **OR**
- El artículo no presenta ningún aspecto relacionado con la usabilidad en *'free source'*.

La estrategia de selección de estudios primarios es análoga a la que representa la Figura 2.1. En primer lugar, una vez definidos las cadenas y campos de búsqueda para cada BBDD, se procedió a realizar las búsquedas. El conjunto de artículos resultado de la búsqueda ha sido denominado "artículos *Encontrados*". Los artículos *Encontrados* fueron revisados mediante el examen del título, keywords y abstract (estos dos últimos cuando estaban disponibles). Aquellos artículos que podrían contener información acerca de la usabilidad en *open source* fueron incluidos en el grupo de "artículos *Preseleccionados*". Cuando se completó el grupo final de artículos *Preseleccionados*, se eliminaron aquellos que se encontraban duplicados entre cada BBDD (es decir, entre los diferentes términos de búsqueda de la misma BBDD) y luego se eliminaron los duplicados entre todas las BBDD. El grupo de artículos resultantes fue denominado como "artículos *Preseleccionados Diferentes*".

Es importante mencionar que cuando se encontraban artículos duplicados se dejaba siempre la primera ocurrencia del artículo, y se eliminaban los demás. Así, a medida que se eliminaban duplicados entre todas las BBDD, el número de artículos iba disminuyendo en las BBDD de donde se eliminaban los duplicados. Es decir, el artículo siempre se conservaba en la primera BBDD donde se encontraba. De este modo, si el orden de las búsquedas fuera cambiado, entonces el número de artículos *Preseleccionados Diferentes* que aporta cada BBDD podría ser distinto.

Para cada uno de los artículos pertenecientes al grupo *Preseleccionados Diferentes*, se leyó el resumen, la introducción y se realizó un *skimming* (esto es, una lectura rápida del artículo) general para determinar si analizaba algún aspecto relacionado con la usabilidad en OSS. El nuevo grupo obtenido con estos artículos se denominó *Estudios Primarios*.

Las Tablas 2.9, 2.10, 2.11, 2.12, 2.13 y 2.14 presentan para cada BBDD el número de artículos en cada uno de los grupos (*Encontrados*, *Preseleccionados*, *Preseleccionados Diferentes* y *Estudios Primarios*) obtenidos durante el proceso de búsqueda. Por ejemplo, en la Tabla 2.11 se puede apreciar que en la BBDD ACM Digital Library para el término de búsqueda OS-U, el número total de artículos que conforman el grupo de *Preseleccionados* fue de 38, mientras que para el término de búsqueda FS-U el número total de artículos para el mismo grupo fue de 15. Finalmente, el número total de artículos del grupo *Preseleccionados* de la BBDD ACM Digital Library para todos los términos de búsqueda fue de 53.

Tabla 2.9: Número Total de Artículos Obtenidos en la BBDD ScienceDirect.

Term. Búsq.	**Encontrados**	**Preseleccionados**	**Preseleccionados Diferentes**	**Estudios Primarios**
OS-U	13	1	1	1
FS-U	11	2	2	0
TOTAL	**24**	**3**	**3**	**1**

Tabla 2.10: Número Total de Artículos Obtenidos en la BBDD IEEE Xplore.

Term. Búsq.	**Encontrados**	**Preseleccionados**	**Preseleccionados Diferentes**	**Estudios Primarios**
OS-U	30	8	8	3
FS-U	9	3	1	0
TOTAL	**39**	**11**	**9**	**3**

Tabla 2.11: Número Total de Artículos Obtenidos en la BBDD ACM Digital Library.

Term. Búsq.	Encontrados	Preseleccionados	Preseleccionados Diferentes	Estudios Primarios
OS-U	146	38	32	6
FS-U	79	15	7	0
TOTAL	**225**	**53**	**39**	**6**

Tabla 2.12: Número Total de Artículos Obtenidos en la BBDD SpringerLink.

Term. Búsq.	Encontrados	Preseleccionados	Preseleccionados Diferentes	Estudios Primarios
OS-U	45	10	9	6
FS-U	8	0	0	0
TOTAL	**53**	**10**	**9**	**6**

Tabla 2.13: Número Total de Artículos Obtenidos en la BBDD Scopus.

Term. Búsq.	Encontrados	Preseleccionados	Preseleccionados Diferentes	Estudios Primarios
OS-U	257	49	27	13
FS-U	78	22	10	0
TOTAL	**335**	**71**	**37**	**13**

Tabla 2.14: Número Total de Artículos Obtenidos en la BBDD FLOSShub.

Term. Búsq.	Encontrados	Preseleccionados	Preseleccionados Diferentes	Estudios Primarios
OS-U	16	15	15	4
FS-U	5	5	0	0
TOTAL	**21**	**20**	**15**	**4**

Resumiendo, la Tabla 2.15 muestra para cada BBDD el número de artículos obtenidos en cada uno de los grupos. Se obtuvieron un total de 33 estudios primarios. Para algunos de los *Estudios Primarios* se leyó la bibliografía en busca de otros artículos que pudieran resultar relevantes para la investigación. Como resultado de esta revisión se encontraron 13 nuevos estudios primarios, que aunque no se encontraban en ninguna de las BBDD donde se realizó la búsqueda, sí eran relevantes. Finalmente, se obtuvieron un total de 46 estudios primarios. Se ha asignado un código alfanumérico a cada uno de estos estudios para facilitar su referencia en el presente capítulo.

Tabla 2.15: Número Total de Artículos Obtenidos en cada BBDD.

BBDD	Encontrados	Preselecc.	Preseleccionados Diferentes	Estudios Primarios
ScienceDirect	24	3	3	1
IEEE Xplore	39	11	9	3
ACM Digital Library	225	53	39	6
SpringerLink	53	10	9	6
Scopus	335	71	37	13
FLOSShub	21	20	15	4
TOTAL	**697**	**168**	**112**	**33**

Todos los estudios primarios seleccionados se muestran en el Anexo B, mientras que una visión sintética de los mismos se proporciona en la Figura 2.8. Esta figura consiste básicamente en dos gráficos de dispersión XY con burbujas en las intersecciones de cada categoría (lado izquierdo de la Figura 2.8). Las categorías vienen determinadas por el año de publicación del estudio primario, su tipo (revistas, conferencias, etc.) y la temática estudiada (problemas de usabilidad, percepción de la usabilidad o técnicas de usabilidad incorporadas por OSS). El tamaño de cada burbuja está determinado por el número de estudios primarios que se han clasificado como pertenecientes a las categorías correspondientes a las coordenadas de la burbuja. El lado derecho de la Figura 2.8 muestra el número de estudios primarios por año de publicación.

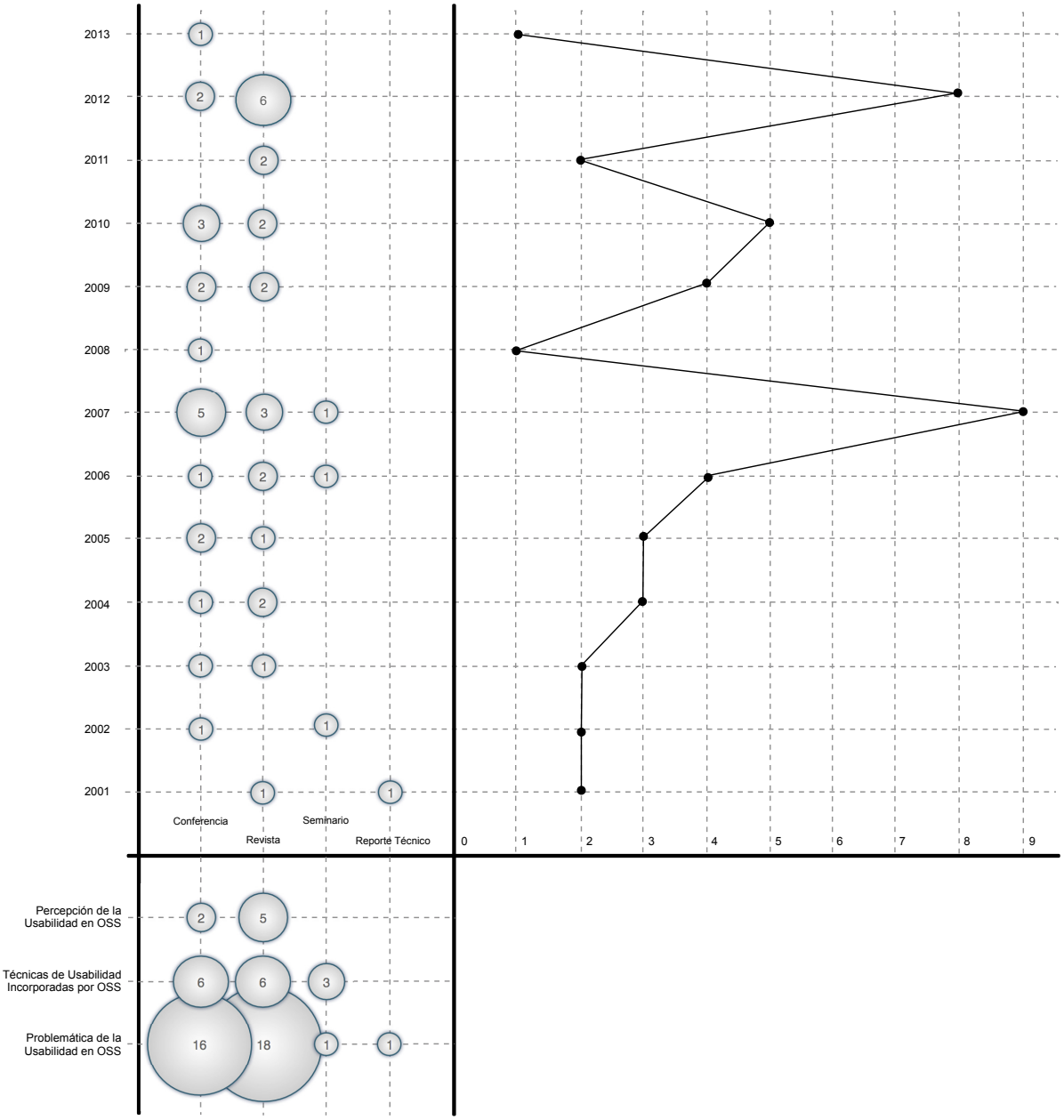


Figura 2.8: Mapeo con la Distribución de Estudios Primarios según Estudian la Problemática, Percepción o Técnicas de Usabilidad Incorporadas por OSS, Incluyendo Tipo y Año de Publicación.

Como se observa en la Figura 2.8, hay tres periodos de tiempo (2005-2007, 2008-2010 y 2011-2012) con un crecimiento de trabajos de investigación en el área de la usabilidad en OSS, reflejando un importante interés en este tema durante la última década. En cuanto a los tipos de publicación, la mayoría de los estudios primarios seleccionados han sido publicados casi en su totalidad en conferencias y revistas (91,3 %). El hecho de que en años recientes las publicaciones sean realizadas en conferencias y revistas (sometidas a revisión de pares) sugiere un interés creciente en el área de la usabilidad en el proceso de desarrollo OSS.

Luego de realizar una lectura detallada de cada uno de los estudios primarios, se identificaron tres temas recurrentes en la literatura: los problemas de la usabilidad en OSS, la percepción que tiene esta comunidad de la usabilidad y las técnicas de usabilidad adoptadas en los desarrollos OSS. La Figura 2.9 ilustra, en un diagrama de Venn, todos los estudios primarios clasificados según el tema estudiado.

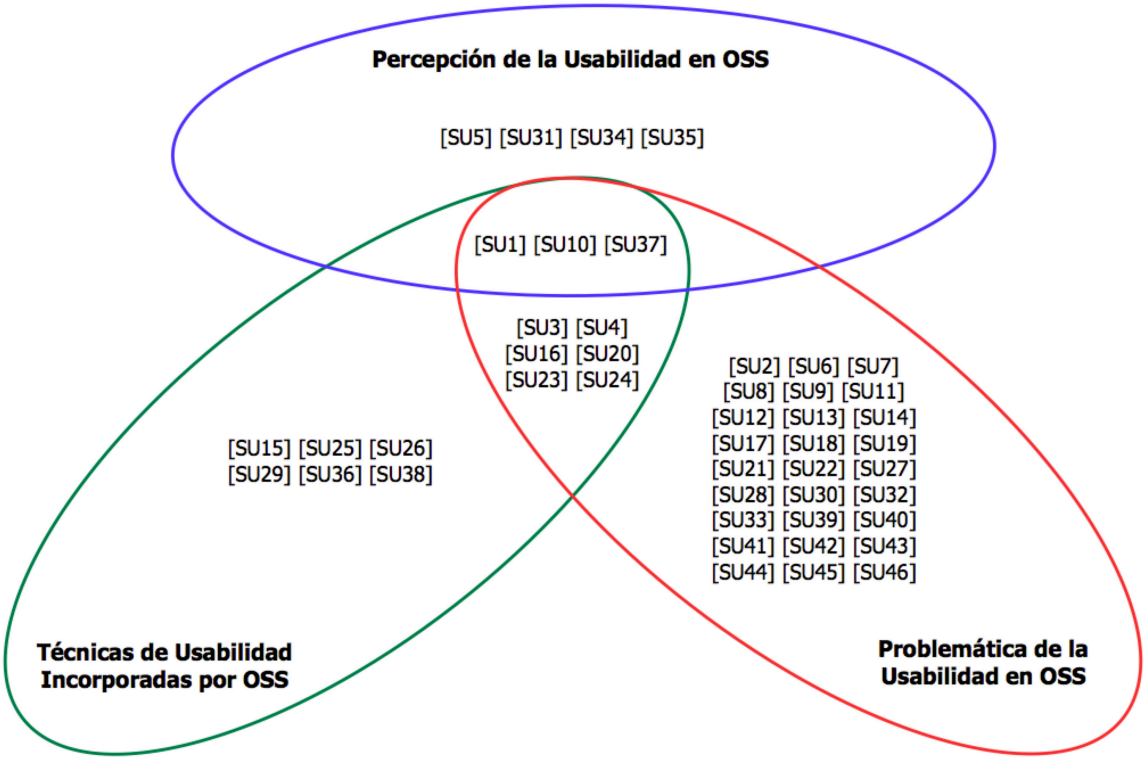


Figura 2.9: Distribución de Estudios Primarios según su Temática.

Como se observa en la Figura 2.9, algunos estudios primarios tratan más de un tema y unos pocos incluso los tres, como por ejemplo [SU1] y [SU10]. A continuación, se discutirán los principales aspectos de cada uno de los tres temas identificados en la literatura.

- **Problemática de la Usabilidad en OSS**

  La temática que ha sido estudiada por el mayor número de estudios primarios es la relacionada con los problemas de usabilidad que enfrentan los desarrollos OSS. Hemos identificado unos pocos trabajos que dedican una sección explícita a los problemas de usabilidad (por ejemplo [SU17][SU22][SU23]), pero otros tratan en general la usabilidad en OSS (por ejemplo [SU3][SU8][SU16]). En este último caso, hemos tenido que darnos cuenta cuáles son los problemas. En el Anexo D se presenta el detalle de los problemas que hemos extraído de la literatura. Como se puede observar en el Anexo D son muchos los problemas de usabilidad, así que los hemos dividido por estudio primario. Describiremos a continuación los principales problemas.

Los desarrolladores OSS construyen software para si mismos, es decir, escriben el software que necesitan ellos mismos e implementan las funcionalidades que desean usar. Además, el público objetivo de las aplicaciones OSS no está bien definido (porque no tienen que vender a un grupo específico de usuarios). El resultado es que las aplicaciones OSS están hechas por desarrolladores para desarrolladores, y sus interfaces no podrían ser utilizadas por usuarios novatos o sin conocimientos técnicos, aunque estas mismas interfaces sean perfectamente adecuadas para los desarrolladores [SU17][SU22].

Los desarrolladores comparten una visión común de la funcionalidad de la aplicación que están desarrollando, pero a menudo tienen una visión muy pobre de la interfaz de usuario. Aspectos importantes de usabilidad son pasados por alto hasta que es demasiado tarde, a menos de que el fundador del proyecto OSS tenga sólidos conocimientos del diseño de la interacción [SU22]. Algunos desarrolladores se sienten orgullosos de crear productos sofisticados, pero con interfaces difíciles de aprender. Por lo general, estos productos tienen muchas opciones de configuración que pueden ser desconcertantes para usuarios novatos o sin conocimientos técnicos [SU22].

Para los desarrolladores es muy importante contar con el respeto de sus compañeros y tener el reto de resolver problemas difíciles. Adicionar una funcionalidad u optimizar código proporcionan oportunidades para mostrar sus talentos como hacker a otros hackers. Si los participantes en la comunidad OSS perciben que realizar mejoras a la usabilidad no supone un reto, es menos exigente o simplemente menos interesante, entonces es menos probable que elijan trabajar en esta área, pues el trabajo en la comunidad OSS es voluntario, y los desarrolladores eligen trabajar en lo que ellos quieren [SU22].

Los proyectos OSS carecen de recursos para ocuparse de la usabilidad. Los proyectos OSS trabajan con pequeños presupuestos, debido a su carácter voluntario. Por tal razón, emplear expertos externos, como por ejemplo diseñadores gráficos, no es posible. Así mismo, tener laboratorios de usabilidad y realizar experimentos detallados a gran escala, no es económicamente viable para muchos proyectos OSS [SU22].

La arquitectura de las aplicaciones OSS, generalmente, cuenta con un diseño muy modular. Sin embargo, la usabilidad pasa por todo el proceso de desarrollo de una aplicación. Como consecuencia, puede ser difícil para los desarrolladores realizar pequeñas mejoras incrementales en la usabilidad de un software [SU2][SU3][SU22].

El proceso de liberar rápidamente y a menudo, que caracteriza la comunidad OSS, puede ocasionar que se incluyan funcionalidades poco útiles y cuya única contribución sea "hinchar" el software. Cuando se libera una nueva versión con una mejor interfaz, los *early adopters* pueden resistirse a adaptarse a la nueva interfaz. Aunque la nueva interfaz sea más fácil de aprender y usar que la anterior, el aprendizaje de la versión antigua es una inversión perdida. Por tal razón, los desarrolladores se ven tentados a mantener varias interfaces heredadas y coordinadas con la última versión. Esto agrada a los usuarios más veteranos, crea más oportunidades para el desarrollo, y mantiene las contribuciones de las interfaces más antiguas en la última versión, pero suma complejidad al software [SU21].

El reporte de errores convencional usado en la comunidad OSS no es suficiente para discutir un problema o un error de usabilidad. Una secuencia lineal en el tiempo de comentarios puede ser adecuada para casos donde hay pocos comentarios o cuando los comentarios documentan un flujo de trabajo sencillo. Sin embargo, el proceso de describir y analizar algunos errores de usabilidad puede ser más complejo y controvertido [SU39]. Además, no todos los errores de usabilidad pueden ser explicados textualmente y aunque el desarrollador solicite, a quien reporta el error, una captura de pantalla, es posible que esto no sea suficiente. Hace falta una discusión *face-to-face* que ayude al desarrollador a

entender el problema, y en la comunidad OSS estos encuentros son una excepción más que una regla [SU39].

- **Percepción de la Usabilidad en OSS**

  Unos pocos trabajos de investigación estudian la percepción que tiene la comunidad OSS de la usabilidad. Todos estos trabajos son recientes, tienen menos de 5 años de ser publicados, a excepción del estudio realizado por Andreasen et al. [SU1] en el año 2006. La juventud de estos trabajos refleja el reciente interés que se ha despertado en la comunidad científica por conocer la percepción de los miembros OSS sobre la usabilidad.

  En el estudio empírico realizado por Andreasen et al. [SU1], la mayoría de los encuestados (más del 80 %) considera la importancia de la usabilidad como "alta", "muy alta" o "extremadamente alta". Dos de los desarrolladores OSS entrevistados, afirmaron que la usabilidad tenía una importancia extremadamente alta. Sin embargo, uno de ellos, explicó que algunos desarrolladores ven la usabilidad como una tarea trivial que no es interesante, ni estimulante intelectualmente. Entre los encuestados existe una confusión sobre la definición de usabilidad. Aunque los contribuyentes deseaban realizar software fácil de usar, las definiciones del término "usabilidad" en el cuestionario variaron constantemente.

  En el estudio realizado por Terry et al. [SU37], los autores realizaron entrevistas a miembros de proyectos OSS. Los entrevistados fueron libres de definir el concepto de usabilidad de la forma en que quisieran, por lo que algunos de ellos citaron varios conceptos en sus definiciones. Estas definiciones de usabilidad abarcaron todo el rango de definiciones que se encuentran comúnmente en los libros de texto de la IPO, y demostraron que la comunidad OSS, en conjunto, tiene una buena noción del concepto de usabilidad [SU37].

- **Técnicas de Usabilidad Incorporadas por OSS**

  Existen unos pocos trabajos de investigación que han estudiado las técnicas de usabilidad que la comunidad OSS ha incorporado en sus desarrollos. En el Anexo E se presentan las técnicas adoptadas de la IPO incorporadas por OSS que hemos identificado en los estudios primarios. En este anexo, el listado de las técnicas es dividido por estudio primario.

  Las técnicas de usabilidad que la comunidad OSS está comenzando a incorporar las hemos agrupado en tres categorías. En la primera, hemos incluido las técnicas incorporadas puras, es decir, sin ningún tipo de adaptación. Estas técnicas se han incorporado en su mayoría porque algunos proyectos OSS tienen los recursos necesarios para aplicarlas (por ejemplo, expertos en usabilidad), gracias al apoyo de empresas. En la segunda, incluimos las técnicas que han sido incorporadas con alguna adaptación. En la tercera, hemos incluido las técnicas que unos pocos trabajos de investigación [SU3][SU23][SU26][SU37] consideran como propias de OSS, porque surgen de la filosofía colaborativa y distribuida de la comunidad. Este grupo está compuesto por cuatro técnicas: *design-by-blog*, *design-workshops*, *open content projects* y *seasons of usability*. A continuación, describiremos brevemente cada una de estas técnicas.

  La técnica *design-by-blog* permite que cualquier miembro de la comunidad (incluyendo los usuarios no desarrolladores) puedan analizar y discutir, a través de blogs, los diseños de la interfaz de usuario y de nuevas funcionalidades. Un diseño orientado al blog está compuesto por una descripción del problema, los objetivos y restricciones o criterios del diseño, capturas de pantalla del diseño propuesto y diferentes comentarios sobre el mismo [SU23][SU37].

Algunos proyectos OSS, especialmente aquellos que tienen el apoyo de empresas como Sun Microsystems, llevan a cabo conferencias con cierta frecuencia (por ejemplo, anualmente). En estas conferencias se reúnen desarrolladores y expertos en usabilidad. Estas reuniones son conocidas como *design-workshops*. En estas reuniones, los expertos en usabilidad aconsejan a los desarrolladores sobre los diferentes aspectos de usabilidad, por ejemplo, si la aplicación debe incluir o no ciertas características [SU3][SU37].

La técnica *open content projects* surge en un proyecto OSS dedicado a la creación de un visualizador 3D [SU37]. Esta técnica involucra la construcción de un producto creativo, como por ejemplo, un videojuego. En esta construcción participan desarrolladores OSS y usuarios finales de la aplicación. El objetivo de esta técnica es descubrir problemas y mejoras de usabilidad. Los problemas son descubiertos a medida que la aplicación es usada por los desarrolladores y los usuarios.

Las *seasons of usability* son una serie de proyectos, generalmente patrocinados, para fomentar que estudiantes de usabilidad y de diseño de la interacción participen en proyectos OSS. Durante un período de tiempo que oscila entre tres a seis meses hasta los dos años, los estudiantes trabajan junto con un mentor experto en usabilidad y con los desarrolladores principales del proyecto OSS con el fin de mejorar la usabilidad [SU37].

Las técnicas de usabilidad más populares dentro de la comunidad OSS corresponden a técnicas para la evaluación de la usabilidad. Ejemplos de estas técnicas son: evaluación heurística [SU10][SU27] e inspección de conformidad con estándares [SU1][SU3][SU10] [SU23][SU27][SU37].

De las técnicas de test de usabilidad la que tiene un mayor uso dentro de la comunidad OSS es la evaluación por control remoto [SU1][SU37]. Otras técnicas muy usadas en los proyectos OSS son foros [SU37] y prototipos de papel [SU10]. También podemos encontrar en este grupo las técnicas para realizar test de usabilidad: pensar en voz alta [SU37] y test de usabilidad en laboratorio [SU29]. Es importante mencionar que esta última técnica solo ha sido aplicada en pocos proyectos OSS que cuentan con el apoyo de empresas, puesto que la mayoría de proyectos no tienen los recursos necesarios para instalar o contratar un laboratorio de usabilidad.

2.3. Conclusiones de la Revisión del Estado de la Cuestión

Las actividades del proceso de desarrollo de software tradicional con mayor presencia en el proceso de desarrollo seguido por la comunidad OSS son: *Identificar Ideas o Necesidades*, *Educción de Requisitos*, *Realizar el Diseño Arquitectónico*, *Identificar las Necesidades de Mejora del Software*, *Implementar el Método de Reporte de Problemas*, *Realizar Revisiones* y *Ejecutar Pruebas*. No existe ningún estudio primario que describa un proceso de desarrollo OSS con alguna actividad que no haya sido posible mapear en los grupos de actividades del proceso tradicional considerados en este análisis. Los únicos grupos de actividades del proceso tradicional que cuentan con estudios primarios en todas las actividades que lo conforman, son *Exploración de Conceptos* y *Mantenimiento*. Sin embargo, de estos dos grupos, la mayor concentración de estudios primarios se encuentra en *Mantenimiento*.

El grupo de actividades *Evaluación* cuenta con las dos actividades con mayor presencia de estudios primarios son: *Realizar Revisiones* y *Ejecutar Pruebas*. Esto evidencia el papel fundamental que desempeñan los propios usuarios de las aplicaciones OSS como reporteros de errores y probadores de versiones beta. Estas dos actividades concentran poco más del 80 % de los estudios primarios, reflejando la importancia que tienen para el proceso de desarrollo seguido por la comunidad OSS. Los principales esfuerzos de la comunidad OSS se centran en el mantenimiento y evaluación del software (*Realizar Revisiones* y *Ejecutar Pruebas*).

A pesar de que en la literatura existen trabajos de investigación que estudian el proceso de desarrollo OSS, no existe un proceso estandarizado, así que para poder determinar cómo incorporar la usabilidad en este proceso necesitamos, como un primer paso en nuestra investigación, estudiar cómo la comunidad OSS desarrolla software. El objetivo de este trabajo consiste en obtener, a partir de la literatura que hemos identificado, el proceso de desarrollo OSS como un factor común de las actividades descritas en la literatura.

Con respecto a la usabilidad, hemos identificado que existe una variedad de trabajos de investigación que tratan principalmente tres aspectos relacionados con la usabilidad en la comunidad OSS: la problemática de la usabilidad en los desarrollos OSS, la percepción de la usabilidad y las técnicas de usabilidad incorporadas. Sin embargo, no existe en la literatura un trabajo de investigación que estudie de manera conjunta estos tres aspectos y reporte el estado actual de la usabilidad en la comunidad OSS. Por tanto, se hace necesario realizar tal estudio como siguiente paso para identificar cómo lograr incorporar la usabilidad en los desarrollos OSS.

Incorporar la usabilidad en el proceso de desarrollo OSS parece ser más complejo que en el desarrollo comercial, debido a que algunas de las características de la comunidad OSS como: (i) su cultura centrada en el desarrollo de funcionalidades, (ii) distribución geográfica mundial, (iii) falta de recursos y (iv) una cultura que puede ser algo ajena al diseño de la interacción, hacen de la incorporación de técnicas de usabilidad una labor difícil, porque la mayoría de las técnicas de la IPO no están pensadas para el tipo de entorno en el que desarrolla OSS.

Hemos observado que cada día la comunidad OSS es más consciente de la importancia de la usabilidad de sus aplicaciones. Ahora mismo, esta comunidad ha comenzando a incorporar algunas técnicas, en su mayoría técnicas para la evaluación de la usabilidad. Han sido pocas las técnicas relacionadas con el análisis de requisitos y el diseño incorporadas por la comunidad. Parece que algunas técnicas de usabilidad han sido transformadas para poder ser incorporadas en los desarrollos OSS.

Como conclusión del estudio de la literatura relevante para la presente investigación, podemos afirmar que ninguno de los trabajos de investigación presenta una propuesta general de cómo integrar técnicas de usabilidad en el proceso de desarrollo OSS, considerando para ello sus características particulares, su filosofía e idiosincrasia y manteniendo la esencia de las técnicas de usabilidad. Por tanto, el problema de cómo incorporar la usabilidad en OSS no ha sido resuelto, así que hace falta una mayor investigación al respecto.

CAPÍTULO 3

PLANTEAMIENTO DEL PROBLEMA

Una vez realizado el estudio del estado de la cuestión, podemos ya establecer con detalle en este capítulo cómo se va a enfocar el problema y cómo se plantea su solución.

3.1. Definición del Problema

En la última década la usabilidad ha despertado la atención de la comunidad OSS, debido al crecimiento de los usuarios no-desarrolladores de sus aplicaciones. La usabilidad es un atributo de calidad que tiene un impacto decisivo en la satisfacción de los usuarios y por tanto determina el éxito o fracaso de un sistema software.

Por una parte, en el área de la IPO existen técnicas de usabilidad cuya finalidad principal es la obtención de software usable. Sin embargo, se aplican en el marco de métodos IPO, y no en el proceso de desarrollo OSS. Por otra parte, el proceso de desarrollo OSS está centrado en el código fuente y por tanto en el desarrollo de funcionalidades y tiene ciertas características (por ejemplo, desarrolladores y usuarios distribuidos por todo el mundo) que impiden que muchas de las técnicas de usabilidad de la IPO puedan ser incorporadas.

Podemos resumir el problema a resolver como la necesidad de encontrar la manera de incorporar técnicas de usabilidad en los desarrollos OSS. De tal forma, que quien necesite aplicar una técnica de usabilidad, conozca si es posible aplicarla en OSS y cómo hacerlo, es decir, qué adaptaciones deben ser realizadas a la misma. Las principales dificultades para afrontar dicho problema son:

- No existe un modelo de proceso de desarrollo OSS ampliamente aceptado. Conocer el modo de desarrollo OSS es fundamental para identificar qué aspectos deben ser tenidos en cuenta al momento de incorporar técnicas de usabilidad.
- El modo de desarrollo OSS y la idiosincrasia de su comunidad presentan ciertos desafíos para incorporar técnicas de usabilidad. Además, estas técnicas no están pensadas para ser incorporadas en el proceso de desarrollo OSS.

En las secciones siguientes se detalla la problemática de cada uno de los puntos anteriores.

3.1.1. Carencia de un Modelo de Proceso de Desarrollo OSS

A medida que las aplicaciones OSS han ganado popularidad gracias a proyectos exitosos (por ejemplo, Mozilla), ha crecido el interés por parte de la comunidad científica de conocer más

acerca de su proceso de desarrollo. La comprensión del contexto, estructura y actividades de los procesos de desarrollo OSS que se encuentran en la práctica ha sido y sigue siendo un problema difícil de resolver [173].

Conocer el proceso de desarrollo OSS puede ayudar a que futuros voluntarios participen en los proyectos OSS al conocer cómo se desarrolla software en esta comunidad, favoreciendo la creación y evolución de nuevos proyectos exitosos [177]. La importancia de contar con una descripción del proceso radica en que todas las actividades de desarrollo software, que involucra tanto a las personas como a la tecnología, puedan ser coordinadas. Esta coordinación es posible porque los ingenieros de proceso tienen la oportunidad de discutir colectivamente y administrar las dependencias entre personas, procesos y tecnologías [121].

Los modelos de proceso software y metodologías actuales no tratan adecuadamente la cuestión del desarrollo OSS. La mayoría de los modelos de procesos software publicados se basan en la suposición de que el equipo de desarrollo de software se encuentra en un sitio centralizado, los requisitos pueden obtenerse tempranamente y que el equipo del proyecto puede ser administrado de cerca. Sin embargo, ninguno de estos supuestos se mantiene en el caso del desarrollo OSS [181].

Algunos autores han demostrado que los procesos OSS son diferentes en muchos aspectos de los procesos software tradicionales [148][173][190]. Otros autores como Fuggetta [76] y Godfrey y Tu [80], sin embargo, afirman que esto no es así. Estos autores sugieren que el modelo de desarrollo OSS no es realmente una nueva descripción del proceso; es solo una visión alternativa de las actividades de la IS aplicada a los modelos de desarrollo tradicional. En cualquier caso, podemos fácilmente encontrar numerosos ejemplos de sistemas OSS exitosos, como los estudiados por Mockus et al. [125][126]. Por tanto, aunque diferente del modelo estándar, el proceso de desarrollo seguido por la comunidad OSS da lugar a productos de calidad reconocidos internacionalmente. Por tanto, realmente ¿puede afirmarse que los procesos OSS difieren del modelo tradicional? Entonces, es preciso identificar las diferencias y similitudes existentes entre el proceso de desarrollo seguido por la comunidad OSS y el comercial, para profundizar en la comprensión del proceso de desarrollo OSS.

Sin embargo, no existe un modelo de proceso de desarrollo OSS ampliamente aceptado que defina cómo se desarrolla en la práctica OSS. Según Scacchi, el punto de partida para entender el proceso de desarrollo de este tipo de software es el de investigar los desarrollos OSS en las diferentes comunidades [30]. Existen trabajos de investigación que han descrito el proceso de desarrollo seguido en algunos proyectos OSS populares. Por ejemplo, como se ha mencionado, el trabajo de Raymond [155] describe el modelo de proceso de un proyecto OSS inspirado en el proceso de desarrollo de Linux. A este modelo lo nombró *Bazar* y lo contrapuso al modelo de procesos de la IS tradicional, al que etiquetó como *Catedral*. Raymond se limita a describir este modelo mediante una serie de cláusulas sencillas, en las que no incluye ninguna metodología específica ni rigurosa, ya que esto iría en contra del mismo modelo *Bazar* [155]. Por tanto, para encontrar la manera de incorporar técnicas de usabilidad en los desarrollos OSS es preciso inicialmente comprender su proceso de desarrollo.

3.1.2. Desafíos del Modo de Desarrollo OSS para la Usabilidad

Tradicionalmente las aplicaciones OSS fueron inicialmente desarrolladas para ser usadas por los mismos desarrolladores que las implementaban. Sin embargo, poco a poco el perfil de los usuarios de las aplicaciones OSS ha ido cambiando [89][95][160]. Actualmente, es común encontrar a personas sin conocimientos técnicos y de programación usando aplicaciones OSS. Además, muchas organizaciones dependen total o parcialmente de aplicaciones OSS [44]. Por estas razones, se ha despertado el interés por la usabilidad de tales aplicaciones. Algunos

investigadores han estudiado diferentes aspectos de la usabilidad en OSS, como por ejemplo los problemas que enfrentan los proyectos OSS cuando intentan incorporar técnicas de usabilidad [38][139].

La construcción de software usable no es una tarea fácil, para conseguirlo es necesario disponer y gestionar equipos de trabajo interdisciplinarios, donde participan, diseñadores gráficos y expertos en usabilidad, entre otros. Si bien en el desarrollo de software comercial hay una conciencia de la importancia de la usabilidad y de sus beneficios, aún quedan problemas por resolver siendo un campo de investigación abierto. Para la comunidad OSS incorporar la usabilidad supone un reto aún mayor, debido a las características de su proceso de desarrollo, como por ejemplo la poca participación de expertos en usabilidad, la distribución geográfica de sus miembros y una cultura centrada en el desarrollo de funcionalidades. El entorno distribuido de los proyectos OSS dificulta tener una muestra representativa de usuarios para realizar actividades relacionadas con la usabilidad [160]. Además, las técnicas de la IPO no están pensadas para el tipo de entorno en el que se desarrolla OSS [114][136][137][195].

La literatura sobre la usabilidad en OSS está compuesta por un conjunto de publicaciones desacopladas que estudian diferentes aspectos. Sin embargo, no existe un trabajo que estudie como un todo esta literatura y reporte cuál es el estado actual de la usabilidad. Unos pocos trabajos de investigación [44][47][89][188] han sugerido reconceptualizar las técnicas de la IPO para que se ajusten mejor a la cultura de la comunidad OSS y a su sistema de valores, pero no detallan cómo realizar tal reconceptualización, ni tampoco a cuáles técnicas. Por tanto, es preciso analizar cuales técnicas de usabilidad podrían ser incorporadas en los desarrollos OSS y cómo realizar tal incorporación.

3.2. Aproximación a la Solución

El proceso de desarrollo OSS se ha convertido en un tema importante de estudio por parte de profesionales y de la comunidad científica [65][67][102][111] gracias al éxito de proyectos OSS como Linux [191], Apache [125] y FreeBSD [57]. Existen trabajos de investigación que incluyen el estudio de muchos aspectos del desarrollo OSS tales como la motivación de sus participantes [87], los repositorios de código fuente [126], su forma de gestión [141] y sus requisitos [172]. Sin embargo, no existe un modelo de proceso de desarrollo OSS ampliamente aceptado, que defina cómo se desarrolla software en las comunidades OSS [170].

Hemos definido un mapa del modelo de proceso de desarrollo OSS obtenido a partir del estudio de la literatura y hemos estudiado las diferencias y similitudes existentes entre este proceso y el proceso tradicional. El estudio del proceso de desarrollo OSS fue clave para conocer cómo esta comunidad desarrolla software y determinar qué consideraciones deben tenerse al incorporar técnicas de usabilidad en los proyectos OSS.

A pesar de que la usabilidad es un concepto relativamente nuevo para las comunidades OSS, éstos están empezando a interesarse por ella [35]. Aunque aún las investigaciones sobre la incorporación de la usabilidad en el desarrollo OSS son escasas, existen trabajos de investigación que han estudiado las técnicas de usabilidad que están siendo incorporadas en los desarrollos OSS [137][188]. Unos pocos investigadores han realizado estudios empíricos [23] y otros han estudiado la literatura [114] con el objetivo de descubrir las técnicas de usabilidad que están siendo usadas en OSS. Sin embargo, no existe en la literatura un trabajo de investigación que considere desde una perspectiva más amplia, el estado actual de la usabilidad en la comunidad OSS.

El presente trabajo ha investigado diferentes temas del estado de la usabilidad en los desarrollos OSS: ¿porqué la usabilidad de las aplicaciones OSS es baja?; ¿la comunidad OSS está tomando medidas para mejorar la usabilidad de OSS?; ¿cuáles técnicas de usabilidad están siendo incorporadas en los desarrollos OSS?; y ¿cómo estas técnicas están siendo adoptadas e integradas en el proceso de desarrollo OSS? Este análisis ha permitido identificar los aspectos clave que deben ser tenidos en cuenta para permitir la incorporación de técnicas de usabilidad en los desarrollos OSS. Esta investigación propone un marco general que describe cómo realizar adaptaciones a las técnicas de usabilidad para poder ser incorporadas en OSS. La viabilidad de esta propuesta se ha evaluado incorporando algunas técnicas de usabilidad en dos proyectos reales OSS (OpenOffice Writer y FreeMind). A continuación, se amplía la descripción de estas contribuciones.

Hemos identificado diferentes razones para la pobre usabilidad de las aplicaciones OSS. Hemos divido estas razones en dos grupos. El primer grupo está relacionado con el modo en que la comunidad OSS desarrolla el software. Por ejemplo, los desarrolladores OSS suelen desarrollar software para satisfacer sus propias necesidades, lo que significa que el desarrollador y el usuario es a menudo la misma persona, y el concepto de usuario está diluido; OSS es desarrollado en ambientes distribuidos, usando canales de comunicación basados en texto (por ejemplo emails, foros de discusión), que obstaculizan el diseño de interfaces de usuario visuales. El segundo grupo está relacionado con el hecho de que los proyectos OSS no siguen las recomendaciones para mejorar la usabilidad (por ejemplo, heurísticas para el diseño de la interfaz de usuario) y a los problemas que surgen cuando deciden adoptar tales recomendaciones (por ejemplo, los desarrolladores no utilizan las heurísticas previamente definidas para el diseño de las interfaces de usuario).

A través del estudio de la literatura relacionada con la usabilidad en OSS, hemos encontrado que la comunidad OSS ha pasado, en un periodo relativamente corto de tiempo, de una concepción errónea de lo qué es la usabilidad a un buen entendimiento del término. La comunidad OSS ahora comparte la opinión de que la usabilidad es un aspecto importante del desarrollo de software. Por lo tanto, están las puertas abiertas para aplicar técnicas de usabilidad en los desarrollos OSS, lo que aporta relevancia a esta investigación.

Debido al crecimiento de los usuarios de aplicaciones OSS que no son desarrolladores y a que las empresas y organizaciones cada vez más están usando aplicaciones OSS, surge la necesidad y el interés por desarrollar OSS usable [35][44][75][137][159][160][161]. La comunidad OSS está empezando a incorporar una serie de técnicas de usabilidad en su proceso de desarrollo, principalmente gracias al apoyo de algunas compañías como Canonical, Novell, Red Hat, y Sun Microsystems.

Hemos identificado que la incorporación de algunas de las técnicas de usabilidad ha ayudado a la comunidad OSS a resolver algunos de los problemas de usabilidad a los que se enfrenta. Por ejemplo, (i) algunos proyectos han adoptado las técnicas perfiles de usuario y Personas para ayudar a los desarrolladores a determinar el perfil de los usuarios no-desarrolladores de sus aplicaciones, y (ii) la comunidad OSS ha sacado partido de sus usuarios no-desarrolladores para que realicen labores de inspección de las interfaces de usuario porque existen pocos expertos en usabilidad participando en proyectos OSS.

Luego de realizar varias iteraciones a través de un análisis minucioso de las técnicas de usabilidad que OSS está comenzando a incorporar, hemos logrado identificar tres categorías que las definen completamente. En la primera, hemos incluido los principios básicos de usabilidad y las técnicas que han sido incorporadas puras, es decir, sin ningún tipo de adaptación. En la segunda, incluimos las técnicas de usabilidad que han necesitado de ligeras

modificaciones para poder ser aplicadas en OSS. En la tercera, hemos incluido las técnicas que o son adaptadas en profundidad o no pueden aplicarse en OSS. Un análisis profundo de estas técnicas y su comparación con cada una de las técnicas de usabilidad existentes nos ha permitido descubrir que tales técnicas han sufrido adaptaciones mayores que las "disfrazan". Es decir, a primera vista, parecen una técnica "propia" de OSS, cuando en realidad no lo es, porque comparten su esencia con alguna técnica de la IPO. En este caso, lo que ha ocurrido es que la técnica ha sido reinterpretada.

La comunidad OSS se ha visto en la necesidad de adaptar las técnicas de usabilidad para poder aplicarlas en sus desarrollos. Hemos identificado las condiciones desfavorables que dan origen a las ligeras adaptaciones. Es decir, estas adaptaciones surgen para sortear las condiciones desfavorables que impiden que las técnicas de usabilidad sean incorporadas por OSS. Para cada una de las técnicas de usabilidad que han sido adaptadas, hemos analizado cuáles son las condiciones desfavorables asociadas a cada adaptación. Una vez finalizado este análisis, hemos agrupado para cada condición desfavorable las distintas adaptaciones realizadas. Como resultado de esta agrupación, hemos identificado que para una misma condición desfavorable se han realizado distintas adaptaciones. Estas adaptaciones dependen del contexto de los proyectos OSS donde han sido incorporadas las técnicas. Por tanto, una misma condición desfavorable ha sido superada de varias formas distintas (adaptaciones). Estas distintas formas las hemos clasificado en familias de adaptaciones. Para el caso de las técnicas reinterpretadas, solo hasta cuando hemos sido capaces de descubrir cuál es la esencia de estas supuestas nuevas técnicas, nos hemos dado cuenta con qué técnica original de la IPO se correspondía. Es en este punto donde hemos podido identificar las familias de adaptaciones.

La participación de los usuarios es fundamental para muchas de las técnicas de la usabilidad. Este hecho supone un gran impedimento para aplicar estas técnicas en los proyectos OSS, debido a que los usuarios se encuentran distribuidos por todo el mundo. Hemos encontrado que este impedimento tiene la mayor familia de adaptaciones diferentes, en total cinco: (i) los usuarios participan cuando se imparten tutoriales o capacitaciones sobre la aplicación OSS o cuando se enseña el mismo en talleres, (ii) los usuarios son familiares y amigos de los desarrolladores o son conseguidos durante las conferencias que realizan algunos proyectos OSS, (iii) los usuarios asisten a las conferencias no solo como participantes pasivos de la misma sino también activamente, por ejemplo impartiendo talleres a otros participantes, (iv) los usuarios son reemplazados por los desarrolladores y (v) los usuarios participan remotamente a través de foros o emails, realizando pruebas al software antes de su liberación, opinando a través de las herramientas para el reporte de errores, realizando comentarios en una wiki o permitiendo al desarrollador acceder remotamente a su ordenador.

Hemos logrado percibir que las condiciones desfavorables pueden ser clasificadas completamente en tres grandes grupos. En primer lugar, algunas técnicas de usabilidad requieren de un experto en usabilidad (la mayoría de proyectos OSS no cuenta con la participación de expertos). En segundo lugar, ciertas técnicas requieren de la participación de los usuarios o que varios de ellos se encuentren físicamente reunidos (los usuarios de OSS están geográficamente distribuidos por todo el mundo). En tercer lugar, algunas técnicas requieren de varios pasos para su ejecución, una preparación previa o necesitan de cierta información inicial (el trabajo en la comunidad OSS es completamente voluntario y realizado en el tiempo libre de sus miembros).

La principal contribución de esta tesis es la propuesta de un marco conformado por todas las posibles adaptaciones, tanto ligeras como de mayor complejidad, que deben sufrir las técnicas de usabilidad para poder ser aplicadas en OSS. Este marco está compuesto por dos piezas fundamentales. Tras construir el mapa del estado de la usabilidad en los desarrollos

OSS e identificar las técnicas de usabilidad que los proyectos OSS están incorporando, nos hemos dado cuenta que es posible generalizar muchas de las adaptaciones de modo que otras técnicas (otros proyectos OSS) se puedan beneficiar de ellas. Esta generalización constituye la primera pieza de nuestro marco. No todas las técnicas de usabilidad pueden sufrir todas las adaptaciones. La adaptación depende de la idiosincrasia de la técnica. Por tal razón, el marco también propone para cada técnica y sus características intrínsecas y condiciones desfavorables asociadas, qué adaptaciones pueden ser realizadas a la misma. Esta asociación técnica-adaptación constituye la segunda pieza del marco propuesto.

Con el marco que proponemos cualquier persona que quiera aplicar una técnica de usabilidad en OSS puede elegir la técnica que necesita dependiendo de su desarrollo y de su proyecto y aplicar alguna de las adaptaciones sugeridas. De esta manera, logrará que una técnica que antes no se podía aplicar directamente en OSS, porque no se adapta a las circunstancias de los desarrollos OSS, pueda hacerlo gracias a nuestro marco.

Con el objetivo de determinar cuáles técnicas podrían ser incorporadas en OSS, gracias a las adaptaciones propuestas en el marco, hemos realizado un análisis de todas las técnicas de usabilidad (según el catálogo tomado como base). En este análisis hemos considerado lo que prescribe la IPO para cada una de las técnicas, el escenario típico en el que desarrolla la comunidad OSS y las condiciones desfavorables de cada técnica. Según nuestro análisis, se ha encontrado que cerca del 80 % de las técnicas de usabilidad podrían ser incorporadas en OSS con adaptaciones ligeras. Sin embargo, sólo cerca del 40 % de las técnicas podrían ser reinterpretadas. Hemos logrado caracterizar las técnicas de usabilidad candidatas a ser reinterpretadas, como todas aquellas técnicas que requieren de la participación de varios usuarios físicamente reunidos.

Hemos evaluado la viabilidad del marco propuesto en dos proyectos OSS reales. Para llevar a cabo esta evaluación hemos participado como voluntarios en los proyectos OSS: *OpenOffice Writer* y *FreeMind*. El primero es grande y muy popular dentro de la comunidad OSS. Mientras que el segundo proyecto es pequeño aunque con un largo recorrido. Antes de iniciar con la aplicación de las técnicas de usabilidad pensamos que los administradores de ambos proyectos contaban con una lista de los emails de los usuarios y con una clasificación de los mismos, es decir, usuarios representativos de la aplicación. Sin embargo, solo uno de los proyectos tenía la lista de los emails de los usuarios y en ninguno de los dos proyectos tenían identificados quienes eran sus usuarios representativos. La falta de conocimiento, por parte de los desarrolladores OSS, del perfil de sus usuarios es un problema que ya ha sido identificado en otros trabajos de investigación [35][132][137]. Nuestro trabajo contribuye aportando evidencia empírica de ello, y proponemos una adaptación, con base en nuestro marco de integración, de la técnica perfiles de usuario para solucionar dicho problema.

Los resultados obtenidos al utilizar el marco para incorporar técnicas de usabilidad son satisfactorios. Desde un comienzo los administradores de los proyectos se han mostrado entusiastas con la idea de aplicar técnicas de usabilidad. Los administradores de ambos proyectos siempre han estado dispuestos a colaborar con lo que hemos necesitado en cada momento. Algunos de los usuarios se han mostrado muy interesados en participar en la aplicación de las técnicas, desde el primer instante en que fueron contactados. Incluso, algunos usuarios daban su agradecimiento por este tipo de iniciativas y otros se mostraban interesados por conocer sobre las técnicas de usabilidad y cómo se llevarían a cabo. Por tanto, la incorporación de técnicas de usabilidad en OSS contará con un punto muy positivo: el entusiasmo tanto de los jefes de proyecto como de los participantes.

Como resultado de la aplicación de las técnicas, hemos encontrado que el número de usuarios que participan es mayor cuando su participación no requiere invertir mucho tiempo. Si su participación exige, por ejemplo la instalación de software adicional o permitir el acceso remoto a su ordenador, el número de usuarios interesados en participar disminuye considerablemente. Es importante mencionar que siempre se encontraron usuarios comprometidos y dispuestos a participar en las diferentes actividades realizadas, sin importar por ejemplo las exigencias de tiempo.

CAPÍTULO 4

MÉTODO DE INVESTIGACIÓN

El presente capítulo presenta los diferentes métodos de investigación utilizados para resolver el problema de investigación tratado en esta tesis doctoral. Básicamente, hemos empleado tres métodos de investigación. El primero, es la revisión sistemática de literatura. Concretamente, utilizamos como método el SMS. Hemos utilizado el SMS para conocer la literatura relacionada sobre las dos áreas que abarca nuestra investigación, el proceso de desarrollo OSS y la usabilidad en OSS. Para cada una de estas áreas hemos realizado un SMS. Los detalles de los dos SMS realizados han sido presentados en el Capítulo 2. El segundo, es el método clásico de análisis de síntesis utilizado en el análisis de las técnicas de usabilidad y en la solución propuesta, es decir, en el marco general de integración de técnicas. Este método tiene tanto componentes de pensamiento inductivo como deductivo. El tercero, es el estudio de casos que hemos utilizado para validar la viabilidad de nuestro marco de integración de técnicas de usabilidad en los desarrollos OSS. Dicho método tiene un fuerte componente empírico.

Comprender el proceso de desarrollo OSS es vital en nuestra investigación para identificar qué aspectos deben ser considerados en el momento de incorporar técnicas de usabilidad en los proyectos OSS. Hemos realizado un SMS [147] para conocer las actividades que son parte del proceso de desarrollo seguido por la comunidad OSS. La búsqueda de la literatura la hemos realizado en las principales BBDD de artículos científicos: *ScienceDirect*, *IEEE Xplore*, *ACM Digital Library*, *SpringerLink* y *Scopus*. El detalle de este SMS se encuentra en la sección 2.1.

Luego de estudiar todos los estudios primarios hemos obtenido, para cada uno, el nombre y la descripción de las actividades que forman parte del proceso de desarrollo OSS. Según la descripción de cada actividad, realizamos un emparejamiento con la actividad análoga del estándar IEEE 1074 [94], teniendo en cuenta para ello el objetivo de la actividad del proceso OSS. El estándar IEEE 1074 [94] es nuestro modelo de proceso de línea base ya que ha influenciado muchos procesos de desarrollo. Como resultado del emparejamiento obtuvimos un mapa del proceso de desarrollo OSS.

Con el objeto de estudiar el estado de la usabilidad en los desarrollos OSS, hemos realizado un SMS [147]. Esta revisión ha analizado las publicaciones existentes en revistas, actas de conferencias y workshops relacionadas con la usabilidad en OSS. Para ello hemos realizado una búsqueda en las principales BBDD de artículos científicos: *ScienceDirect*, *IEEE Xplore*, *ACM Digital Library*, *SpringerLink* y *Scopus*. Además, realizamos una búsqueda en *FLOSShub*[1] que incluye una BBDD de artículos. El detalle de este SMS se encuentra en la sección 2.2.

[1]Un portal especializado en recursos de investigación para OSS.

Para el análisis de las técnicas de usabilidad adoptadas en los desarrollos OSS hemos utilizado un catálogo de técnicas (ver Anexo F) recopilado por investigadores en el área de la IS [68], debido a la gran diversidad de técnicas en la IPO. Hemos usado este catálogo como una guía para analizar cuáles técnicas ha adoptado OSS en sus proyectos de desarrollo. El análisis de estas técnicas ha sido realizado comparando el modo en que OSS usa la técnica con la prescripción original de la misma establecida por la IPO.

Para analizar cómo permitir el uso de técnicas de usabilidad en los desarrollos OSS, hemos estudiado tanto su proceso de desarrollo así como también las técnicas incorporadas en algunos proyectos OSS. El estudio del proceso de desarrollo OSS nos permitió conocer cómo esta comunidad desarrolla software y determinar qué consideraciones deben tenerse en cuenta al momento de usar técnicas de usabilidad en los proyectos OSS. El estudio de las técnicas de usabilidad incorporadas por OSS nos permitió determinar qué técnicas y cómo éstas han sido incorporadas. Ambos estudios, han sido realizados revisando la literatura. Para obtener dicha literatura, hemos llevado a cabo dos SMS (uno por cada temática).

Posteriormente, hemos analizado cuáles son las características de las técnicas de usabilidad que impiden su aplicación en los desarrollos OSS. Para determinar tales impedimentos, hemos tenido en cuenta lo prescrito por la IPO para cada una de las técnicas así como el modo de desarrollo que sigue la comunidad OSS. Para tener una guía de cuales técnicas de usabilidad debíamos considerar en nuestra investigación, hemos tomado como referencia el catálogo recopilado por investigadores del área de IS [68]. Para determinar la esencia de las técnicas hemos estudiado cada una a partir de las referencias literarias de la IPO correspondientes [50][52][92][120][138][149][180].

Hemos estudiado cada una de las técnicas de usabilidad incorporadas por OSS, con el objetivo de determinar cómo han logrado incorporarlas, es decir, cómo han sorteado los impedimentos identificados. Las técnicas que tienen en común un impedimento han sido agrupadas y hemos asociado al impedimento las distintas soluciones dadas en algunos proyectos OSS. Estas soluciones suponen adaptaciones a las técnicas. Como resultado de este análisis, hemos confeccionado una clasificación de las técnicas de la IPO de acuerdo a cada una de las adaptaciones propuestas.

Una vez hemos tenido claro tanto las adaptaciones realizadas a las técnicas como los impedimentos que las motivan, procedimos a analizar cuáles de las técnicas no usadas aún en desarrollos OSS podrían ser modificadas con las adaptaciones. En este análisis, hemos necesitado considerar no sólo los impedimentos identificados para cada una de estas técnicas, sino también si la adaptación a realizar sobre la técnica viola su esencia, así como también que la técnica adaptada resultante no viola la filosofía de trabajo OSS.

La validación de la viabilidad de nuestro marco de integración de técnicas de usabilidad en los desarrollos OSS hace necesario que participemos como voluntarios en proyectos OSS. Esta participación implica ser parte de la comunidad OSS como un miembro más. El método de investigación idóneo para llevar a cabo esta validación es el estudio de casos, para lo cual hemos seguido los lineamientos de Runenson y Höst [168]. Este método de investigación es utilizado cuando el fenómeno bajo investigación (en este caso, la incorporación de técnicas con adaptaciones) se estudian dentro de su contexto real (en nuestro caso, proyectos OSS).

Hemos validado nuestro marco de integración en dos proyectos OSS reales: uno grande y popular -*OpenOffice Writer*- y el otro pequeño -*FreeMind*-, por lo que han sido dos los casos de estudio realizados (uno por cada proyecto). El objetivo de los dos casos de estudio es el mismo, identificar si es posible incorporar técnicas de usabilidad con adaptaciones en proyectos OSS y determinar qué aspectos de ésta comunidad favorecen o dificultan dicha incorporación.

El conocimiento adquirido con el estudio del proceso de desarrollo OSS, nos ha permitido saber cuáles son los pasos a seguir para participar como voluntarios en proyectos OSS. El primer paso fue contactar con los administradores del proyecto para comentar nuestro interés en incorporar técnicas de usabilidad en tales proyectos. Posteriormente contactamos con usuarios de los dos proyectos OSS para preguntar quiénes estaban interesados en participar en la aplicación de técnicas de usabilidad. Trabajamos con dos grupos de usuarios. En el primer grupo, estaban usuarios de las aplicaciones que son amigos y familiares nuestros. Estos usuarios fueron contactados a través de diferentes medios, principalmente por teléfono y por correo electrónico. En el segundo grupo (el más grande de los dos), encontramos los usuarios que están distribuidos geográficamente por todo el mundo. Este grupo de usuarios fue contactado a través de foros de discusión y por correo electrónico.

Una vez obtenidos los usuarios interesados, procedimos con la aplicación de las técnicas de usabilidad: perfiles de usuario, observación directa e información post-test. Para poder aplicar estas técnicas en los desarrollos OSS fue necesario realizar adaptaciones a las mismas. Estas adaptaciones fueron realizadas según nuestro marco de integración. En la aplicación de las técnicas de usabilidad participaron diferentes personas: (i) los administradores de los proyectos OSS, (ii) los usuarios de las aplicaciones OSS, (iii) un estudiante de grado, y (iv) el autor del presente trabajo de investigación.

La aplicación del método de investigación estudio de casos nos ha permitido verificar que el marco de integración de técnicas de usabilidad propuesto en este trabajo de investigación es viable y nos ha permitido adquirir un profundo conocimiento de cuáles son los pasos a seguir cuando se pretende aplicar técnicas de usabilidad con adaptaciones en los desarrollos OSS. Además, hemos podido identificar cuáles son las principales dificultades y ventajas que se tienen cuando se aplican técnicas de usabilidad en una comunidad de desarrollo OSS.

CAPÍTULO 5

USABILIDAD EN EL PROCESO DE DESARROLLO OPEN SOURCE SOFTWARE

En este capítulo presentamos el estado actual de la usabilidad en la comunidad OSS desde una amplia perspectiva. Iniciamos analizando las razones por las que, según la literatura, la usabilidad de las aplicaciones OSS es baja. Luego, estudiamos la percepción que tiene la comunidad OSS de la usabilidad. Posteriormente, presentamos el catálogo de técnicas de la IPO que hemos utilizado como nuestra línea base para investigar las técnicas de usabilidad que están siendo usadas por la comunidad OSS. Una vez discutido el catálogo, reportamos el uso de técnicas de la IPO en los desarrollos OSS, y analizamos y comparamos las técnicas de usabilidad que OSS está adoptando en su proceso de desarrollo con la prescripción original dada por la IPO. Finalmente, discutimos nuestros resultados y presentamos las conclusiones del estado actual de la usabilidad en la comunidad OSS.

5.1. Porqué el Open Source Software tiene una Menor Usabilidad que el Software Comercial

Existen varias razones explicadas en la literatura para la baja usabilidad de OSS. Hemos dividido estas razones en dos grandes grupos. El primero está relacionado con las diferencias del desarrollo entre OSS y el software comercial. El segundo tiene que ver con el hecho de que los proyectos OSS o bien no siguen las recomendaciones sobre cómo mejorar la usabilidad o se enfrentan a una serie de problemas si adoptan estas prácticas.

5.1.1. Nivel de Usabilidad de OSS

La baja usabilidad de OSS ha sido reconocida por varios autores [35][44][75][89][114] [136][146][163][188][205][206]. Sin embargo, no en todas las aplicaciones o componentes desarrollados por la comunidad open source la usabilidad es baja. Zefferer y Krnjic [204] evalúan la usabilidad de dos componentes open source (MOCCA Local y MOCCA Online) que facilitan la integración de tarjetas inteligentes en aplicaciones e-Government en Austria. Los resultados obtenidos muestran que ambos componentes básicamente cumplen los requisitos de usabilidad dados, aunque se identificaron algunos problemas de usabilidad menores que probablemente se resolverían proporcionando información más detallada sobre el proceso de instalación de los componentes. En el estudio piloto realizado por Braccini et al. [39] se analiza la percepción de una muestra de 59 usuarios al usar *Moodle* (plataforma OSS de e-learning) durante un periodo de seis meses. Los participantes fueron clasificados en dos grupos de

usuarios: principiantes (76,3 % del total) y avanzados (23,7 % del total). Los resultados de este estudio muestran que ambos grupos de usuarios están satisfechos de su experiencia con el software. Sin embargo, en un estudio más reciente De Porto y De Moraes [56] investigan si los problemas de usabilidad obstaculizan el uso que los profesores hacen de *Moodle*. De las entrevistas a los profesionales que trabajan en educación a distancia, los autores encontraron con que había discrepancia sobre la usabilidad de *Moodle*, existiendo un número similar de referencias a una buena usabilidad como a la existencia de problemas de usabilidad.

A pesar de un par de casos excepcionales como los reportados por Zefferer y Krnjic [204] y Braccini et al. [39], la baja usabilidad del OSS parece un hecho. En el estudio empírico realizado por Raza et al. [163], el 60 % de los encuestados (usuarios no-desarrolladores) afirmó que la baja usabilidad es el principal obstáculo que deben superar las aplicaciones OSS para que los usuarios migren del software comercial. Raza y Capretz [158] y Raza et al. [162] concluyen que el nivel de usabilidad y sus problemas relacionados deben tratarse más a fondo en los proyectos OSS. Algunos autores [23][61] consideran que las aplicaciones OSS tienen una usabilidad especialmente baja para usuarios novatos y usuarios no-desarrolladores, comparada con la usabilidad de aplicaciones análogas en el software comercial.

En la literatura, sin embargo, existen pocos estudios rigurosos que realicen comparaciones que aporten evidencias empíricas de que OSS es menos usable que el software comercial [23][135][136][137]. La única comparación rigurosa de la usabilidad entre una aplicación OSS y su análoga comercial que hemos encontrado es el trabajo de Eklund et al. [61] que compara la satisfacción de un grupo de usuarios al usar dos hojas de cálculo: *StarOffice Calc* (aplicación OSS) y *Microsoft Excel*. Los participantes en el estudio (estudiantes de la Universidad de California), tuvieron que realizar 15 tareas. Casi todas las tareas fueron calificadas como ligeramente más fáciles de hacer en *Microsoft Excel* que en *StarOffice Calc*, pero la diferencia entre las dos aplicaciones sólo fue significativa en dos tareas (unir celdas e insertar gráficos), que fueron calificadas como más fáciles de hacer en *Microsoft Excel*. El estudio concluye que la satisfacción promedio de los participantes fue significativamente mayor al usar *Microsoft Excel* que *StarOffice Calc*. Existen muchas diferencias entre una aplicación OSS y una comercial que pueden influir y en ocasiones hacer que los resultados de tales comparaciones sean polémicos. Ejemplos de tales diferencias son el tiempo de desarrollo, recursos para el desarrollo, madurez del software, existencia previa de software similar, etc. Algunos de estos factores son diferentes entre el desarrollo open source y el comercial haciendo difícil determinar que sea una "comparación equitativa" [136][137].

5.1.2. Efectos Adversos del Proceso de Desarrollo OSS en la Usabilidad

El proceso de desarrollo es una de las principales diferencias entre las aplicaciones OSS y las comerciales que pueden afectar la usabilidad. Hemos organizado en cuatro grupos principales los numerosos argumentos que afirman que algunas características del desarrollo OSS van en contra de la producción de software usable. A continuación, enunciamos brevemente cada una de estas razones, para posteriormente discutir detalladamente cada una.

- Dado que los desarrolladores OSS normalmente han desarrollado software para sí mismos, el desarrollador y el usuario es a menudo la misma persona [108][109][136][161] y el concepto de usuario queda diluido.
- Los desarrolladores OSS no consideran la usabilidad un problema y prefieren trabajar en el desarrollo de funcionalidades [44]. Como los desarrolladores se centran en el desarrollo de funcionalidades nadie asume la responsabilidad de estudiar la usabilidad de la aplicación [44][108][136]. Los desarrolladores tienden a desarrollar funcionalidades sin tener en cuenta otras aplicaciones que podrían ser una fuente de ideas para mejorar la usabilidad [95][108].

- La práctica de parches incrementales usada a menudo para corregir errores funcionales no es buena para mejorar la usabilidad [28][136]. El reporte y posterior corrección de los errores de usabilidad es una labor compleja porque los usuarios no-desarrolladores no saben como reportarlos adecuadamente y porque algunos errores pueden ser potencialmente subjetivos [35][137][195][205][206]. Esto se complica aún más por el hecho de que las herramientas que se utilizan para reportar estos errores no fueron diseñadas con este propósito y son difíciles de usar por los usuarios no-desarrolladores [44][47][137][195][201].
- OSS es desarrollado en entornos distribuidos utilizando métodos de comunicación basados en texto como correos electrónicos, foros de discusión, etc. Esto no favorece el diseño visual de la interfaz de usuario [114]. Además, es difícil en entornos distribuidos obtener una muestra representativa de usuarios.

Ahora discutamos cómo estos factores influyen en la baja usabilidad. Los proyectos OSS son una asociación de desarrolladores que tradicionalmente han desarrollado para sí mismos el software que necesitan. Debido a que los desarrolladores son los usuarios, no hay usuarios en el sentido tradicional y por tanto sus aplicaciones tienden a tener interfaces que no son usables para los usuarios no-desarrolladores. Sin embargo, estas interfaces son perfectamente adecuadas para los usuarios a los cuales están dirigidas [136].

Para agravar la situación anterior, los desarrolladores tienen poco conocimiento del perfil, objetivos y contexto de uso de los usuarios no-desarrolladores [35][132][137] y tradicionalmente, han considerado la usabilidad como un tema de menor importancia [44]. Si los desarrolladores perciben que resolver problemas o realizar mejoras a la usabilidad no representan un reto técnico o simplemente lo consideran menos interesante que mejorar alguna funcionalidad, entonces la probabilidad de que trabajen en temas relacionados con la mejora de la usabilidad es menor. Esto significa que muchos problemas de usabilidad sigan sin resolverse [23][136][201].

Gracias a la arquitectura modular de las aplicaciones OSS, los desarrolladores OSS usualmente asumen la responsabilidad de la funcionalidad que necesita o quiere añadir [28][35][197]. Esto significa, que no hay nadie responsable de examinar la usabilidad de tales aplicaciones globalmente [44][108][136]. En la comunidad OSS no hay una entidad encargada de la estandarización de las interfaces [81]. Además, los desarrolladores realizan la práctica de agregar muchas opciones alternativas a sus aplicaciones, lo que es potencialmente bueno para los usuarios expertos, pero no para usuarios novatos o no-desarrolladores [136].

Los desarrolladores OSS tienden a no fijarse en soluciones estándares estables para el desarrollo de funcionalidades, que pueden ayudarles a mejorar la usabilidad. Esto es porque está mal visto, dentro de la comunidad OSS, fijarse en qué hacen las aplicaciones comerciales [95][108]. Sin embargo, el software comercial está siempre explorando las nuevas características de sus competidores que sean potencialmente útiles para adoptarlas en sus aplicaciones.

Para la corrección de los errores funcionales los desarrolladores siguen la práctica habitual de utilizar parches incrementales. Sin embargo, existen aspectos de usabilidad que impactan transversalmente toda la aplicación. Por lo tanto, puede ser difícil para los desarrolladores realizar mejoras incrementales a la usabilidad de sus aplicaciones siguiendo esta práctica [28][136].

Los errores de usabilidad en los proyectos OSS son reportados siguiendo la práctica habitual de reporte de errores funcionales y usando la misma herramienta para ambos tipos de errores. Sin embargo, reportar un error de usabilidad es más complejo que reportar un error funcional, por dos razones. En primer lugar, los usuarios no-desarrolladores no saben cómo reportar

adecuadamente un problema de usabilidad y, por lo general, están confundidos con lo que les ha pasado (no entienden porqué la interfaz de usuario ha tenido un comportamiento inesperado, o ni siquiera saben qué acción ocasionó tal comportamiento) [35][137][205][206]. En segundo lugar, algunos errores de usabilidad son subjetivos. Por ejemplo, una persona puede encontrar un elemento particular de la interfaz de usuario confuso o ambiguo pero otros no [137][195].

El proceso de reportar un error de usabilidad se complica aún más por el hecho de que las herramientas utilizadas no son óptimas. Según el estudio empírico realizado por Çetin y Göktürk [46], el 40 % de los expertos en usabilidad encuestados conocen personas que no han podido reportar errores de usabilidad debido a las herramientas utilizadas para tal propósito. Del mismo modo, cerca del 50 % de los desarrolladores encuestados ha recibido al menos una queja de un usuario que no pudo reportar un error debido también a los problemas de usabilidad presentes en las herramientas para reporte de errores.

Hay tres razones por las cuales las herramientas para reporte de errores no son óptimas para los problemas de usabilidad. Primero, no cuentan con un mecanismo que permita cargar, mostrar, mantener y comentar las imágenes o los vídeos que puedan necesitar los usuarios para reportar un error de usabilidad [47][137], ya que muchos de estos errores no pueden ser explicados textualmente [195]. Segundo, estas herramientas están especialmente diseñadas para que puedan ser reportados todos los tipos de problemas que pueda tener un desarrollador, pero no así para que los usuarios no-desarrolladores reporten errores de usabilidad [47]. Tercero, pocos proyectos OSS tienen un componente de propósito específico dedicado a la usabilidad dentro de la herramienta para el reporte de errores. Esto significa que los errores de usabilidad no pueden ser rastreados y hace difícil ejecutar búsquedas de errores de usabilidad en la base de datos de errores [44].

Los errores de usabilidad reportados son analizados, discutidos y posteriormente corregidos. Un error de usabilidad puede afectar la interacción del usuario con toda la aplicación e involucrar muchos elementos de la interfaz. Por tanto, la corrección de un error de usabilidad puede requerir no solo una revisión completa de toda la interfaz sino también del código asociado y si hay más de un desarrollador trabajando de manera autónoma y distribuida en el diseño de la misma, entonces es más difícil controlar que no existan inconsistencias [136][137][195]. En el desarrollo de software comercial, los desarrolladores se encuentran por lo general centralizados y gestionados de algún modo [136].

Los desarrolladores OSS están distribuidos por todo el mundo y gran parte de su comunicación se lleva a cabo a través de medios electrónicos, como correos electrónicos y foros de discusión. Esto significa que el flujo de información es limitado y asíncrono [47]. Estos canales de comunicación dan lugar a limitaciones en el diseño de software usable porque el diseño de la interfaz es un proceso interactivo y visual y los protocolos de comunicación son textuales [114].

En estos entornos distribuidos es difícil conseguir una muestra representativa de usuarios. Sin embargo, los desarrolladores de software comercial que trabajan en entornos igualmente distribuidos tienen un mayor interés por conocer y por obtener una muestra de sus usuarios porque se dan cuenta que el éxito de un producto depende en gran medida de la satisfacción del usuario. Además, en los proyectos OSS el público objetivo no está bien definido, porque no tienen que vender el software que desarrollan a un grupo específico de usuarios [43][108].

5.1.3. Obstáculos para la Adopción de Prácticas de Usabilidad en OSS

Principalmente los proyectos OSS pequeños no aplican las recomendaciones de usabilidad. Por lo general, no utilizan ningún método para ayudar a mejorar la usabilidad porque su uso es potencialmente contrario a la filosofía de la comunidad OSS [44][139][206]. Además, hay poca conciencia colectiva dentro de la comunidad OSS de las técnicas existentes para mejorar la usabilidad, a excepción de las *Human-Interface Guidelines* (HIGs) [188]. Las HIGs son el método más conocido para mejorar la usabilidad en OSS, aunque no todos los proyectos OSS utilizan estas guías para conducir el diseño visual de sus aplicaciones [44][137]. Hay proyectos OSS que han intentado, aunque sin éxito, adoptar técnicas de usabilidad (como Personas [38][139] y evaluación heurística [38][139] en el proyecto TYPO3 -un sistema de gestión de contenidos-).

Proyectos como TYPO 3 han intentado tanto investigar quienes eran los usuarios no-desarrolladores de la aplicación como introducir un conjunto de heurísticas de usabilidad. Sin embargo, ambos intentos fueron fallidos debido a la poca participación de la comunidad y a que ninguno de los desarrolladores del proyecto usó las heurísticas [38][139]. La poca participación de la comunidad se debe, quizás a que la cultura del hacer (desarrollar) en OSS es mayor a la necesidad de comprender los objetivos y las necesidades de los usuarios no-desarrolladores [114][137][139]. Según la IPO, la carencia del análisis de usuarios ocasiona que el diseño de las aplicaciones no se realice de acuerdo a las necesidades de los usuarios [114].

Según las recomendaciones de la IPO, para la construcción de software usable, es una buena práctica incluir expertos en usabilidad en el proceso de desarrollo. Esta sugerencia ha sido seguida por muchas empresas en sus desarrollos de software comercial. Sin embargo, en los proyectos OSS existe poca participación de expertos en usabilidad por dos razones. En primer lugar, algunos proyectos OSS trabajan con presupuestos pequeños (en especial aquellos que no cuentan con el apoyo de compañías) porque son realizados por voluntarios. Esto significa que los proyectos no pueden contratar a expertos externos (como diseñadores gráficos o expertos en usabilidad) [35][136][137][159][206]. En segundo lugar, hay pocos incentivos para que los expertos participen en proyectos OSS, además de ser pagados. Por ejemplo, el reconocimiento entre pares (uno de los principales incentivos para que los desarrolladores contribuyan en los proyectos OSS) es inexistente porque no hay una masa crítica de expertos en usabilidad que trabajen en proyectos OSS [114]. Además, parece que algunos expertos no se quieren involucrar porque no están interesados o motivados por el enfoque OSS de la misma forma en que lo están muchos de los desarrolladores [136][208]. Esta falta de motivación, quizás se deba a que su modo de trabajo no encaja con el de la comunidad OSS [89][114][136][208].

Los pocos expertos en usabilidad que participan en proyectos OSS se enfrentan a cuatro problemas. En primer lugar, problemas de comunicación entre los expertos y los desarrolladores. Estos problemas surgen porque los desarrolladores no siempre entienden los beneficios y resultados que se pueden obtener al trabajar con un experto [114], o porque los expertos no conocen el enfoque de trabajo de los proyectos OSS [44][45][114]. En segundo lugar, para los expertos en usabilidad es difícil demostrar su valía de la misma forma que se hace entre los desarrolladores, es decir, evaluando sus habilidades y competencias a través de la experiencia. Los desarrolladores no pueden evaluar el trabajo de los expertos en usabilidad porque no están familiarizados con las habilidades o conocimientos que necesitan para hacer su trabajo [23][28][114]. En tercer lugar, la participación de los expertos se limita sólo a ser asesores, porque la comunidad OSS tiene miedo de que la participación directa de los expertos en la toma de decisiones anule su democracia [23]. En cuarto lugar, las opiniones de los expertos en usabilidad son ignoradas porque son mucho menos numerosos que los desarrolladores y siempre están en minoría [35][136][195]. Sin un grupo representativo de expertos en usabilidad puede ser difícil para ellos establecer la legitimidad de sus argumentos en contra de las

pretensiones centradas en funcionalidades de los usuarios expertos o desarrolladores. Los expertos en usabilidad pueden sentirse en inferioridad numérica y desistir de defender sus argumentos [195][207]. Además, no habrá incentivos de reconocimiento entre pares a menos que haya un grupo representativo de expertos en usabilidad [136].

Resumiendo, los grandes problemas a los que se enfrenta el desarrollo OSS para producir software usable son la falta de conocimiento por parte de los desarrolladores del perfil de los usuarios novatos o no-desarrolladores, la falta de interés de los desarrolladores por los asuntos relacionados con la usabilidad, y los medios de comunicación y herramientas utilizadas durante el desarrollo. El problema de la baja usabilidad de las aplicaciones OSS se acrecienta porque algunos proyectos OSS no siguen las recomendaciones de la IPO, y aquellos que han intentado adoptar estas recomendaciones se han encontrado con ciertas dificultades. Algunas de estas dificultades tienen su origen en la filosofía e idiosincrasia de la comunidad OSS.

A pesar de todas estas dificultades, la usabilidad ha ido ganando terreno dentro de la comunidad OSS en los últimos años, como veremos en la siguiente sección.

5.2. Percepción de la Usabilidad en la Comunidad OSS

En esta sección discutimos algunos estudios empíricos recientes que muestran que la percepción de la usabilidad de la comunidad OSS está cambiando. Uno de los objetivos de estos estudios es comprender lo que los participantes en los proyectos OSS entienden por usabilidad y lo importante que es para ellos.

En 2006 Andreasen et al. [23] realizó uno de los primeros estudios para investigar lo que la comunidad OSS entiende por usabilidad. En este estudio los autores encontraron que muchos de los desarrolladores OSS tenían un conocimiento limitado de qué es la usabilidad. Aunque los desarrolladores manifestaron querer desarrollar software usable, las definiciones que dieron del término usabilidad fueron muy diferentes entre sí: algunas definiciones estaban alejadas de lo que es realmente la usabilidad y otras fueron muy simples y probablemente difíciles de aplicar en la práctica. Algunas de las definiciones hacían énfasis en que las interfaces de usuario debían ser intuitivas: un usuario debería ser capaz de usar lo básico de las aplicaciones sin ayuda de la documentación. En otras palabras, la interfaz debe tener en cuenta el modelo mental del usuario (por ejemplo, representar la acción *borrar archivos* con un ícono que represente un cubo de basura).

Otro estudio similar fue el realizado recientemente [188]. En 2010 diferentes miembros (desarrolladores, escritores de documentación, etc.) de proyectos OSS fueron entrevistados para conocer lo que ellos entendían por usabilidad. Los entrevistados tenían la libertad de definir el término usabilidad como quisieran, y algunos mencionaron más de un concepto en sus definiciones. La situación ha mejorado: las definiciones cubren todo un rango de significados que se encuentran en los libros de la IPO. Se puede decir que hoy en día la comunidad OSS en su conjunto tiene una buena noción de lo que es usabilidad.

En el estudio empírico realizado por Raza et al. [160], una de las preguntas realizadas a los encuestados (usuarios OSS) estaba relacionada con qué entendían por usabilidad. Casi todos los encuestados (94 %) estuvieron de acuerdo con la afirmación: *"la usabilidad trata la 'experiencia total del usuario', no sólo sobre lo atractiva de la interfaz de usuario"*. Parece que la comunidad OSS está dejando a un lado la tradicional creencia de que la usabilidad sólo está relacionada con las interfaces de usuario. Otra pregunta de este mismo estudio pone de relieve este punto. A los encuestados se les preguntó si estaban de acuerdo en que si el software era *"atractivo al usuario"* no implicaba necesariamente que fuera usable, y el 70 %

estuvo de acuerdo. Según el estudio empírico realizado por Çetin y Göktürk [46], cerca del 73 % de los desarrolladores encuestados ha leído al menos un capítulo de un libro de la IPO y el 88 % ha leído al menos un artículo sobre el mismo tema.

En cuanto a la importancia de la usabilidad en la comunidad OSS, la mayoría (83 %) de los participantes (desarrolladores) que completaron el cuestionario como parte del estudio empírico realizado por Andreasen et al. [23] en el 2006 consideró que la importancia de la usabilidad es *alta*, *muy alta* o *extremadamente alta*. Aunque el cuestionario reveló que entre los participantes la usabilidad tiene una prioridad alta, estas buenas intenciones no necesariamente son llevadas a la práctica, porque las entrevistas realizadas posteriormente, revelaron que para algunos desarrolladores es más interesante desarrollar nuevas funcionalidades que corregir problemas de usabilidad en la interfaz de usuario. Según el estudio empírico realizado por Çetin y Göktürk [46], la mayoría de los desarrolladores encuestados (84,2 %) tiene un alto sentido de respeto hacia los errores de usabilidad y los considera como errores reales, reconociendo su importancia. Sin embargo, los resultados de la encuesta realizada por Raza y Capretz [158] en 2012 a un grupo de desarrolladores revelan que el 70 % (de un total de 72 que respondieron la encuesta) no considera la usabilidad como una cuestión prioritaria. Raza et al. [162] también en 2012 han encontrado que sólo el 10,95 % (192 de 1.753) de los proyectos OSS estudiados destacan problemas de usabilidad en los foros de discusión.

Resumiendo, parece que la comunidad OSS ha pasado, en relativamente poco tiempo (cinco años), de tener definiciones equívocas de lo que es la usabilidad a comprender el concepto. Algunos proyectos consideran que la usabilidad es importante en el desarrollo de software, mientras que otros no consideran la usabilidad como una prioridad.

Adicionalmente, como discutiremos en la sección 5.4, algunos proyectos OSS han comenzado a adoptar técnicas de la IPO para mejorar la usabilidad de sus aplicaciones. Pero para analizar las técnicas de usabilidad que están siendo adoptadas por la comunidad OSS, necesitamos tener un universo estructurado de técnicas de la IPO. La siguiente sección presenta un catálogo de técnicas de la IPO para mejorar la usabilidad.

5.3. Estructurando el Universo de Técnicas de la IPO

Con el fin de conocer qué técnicas de la IPO se están utilizando en los desarrollos OSS nos es necesario identificar el universo de tales técnicas. Esto no es sencillo. En la IPO hay una gran diversidad de técnicas donde la misma técnica puede tener distintos nombres dependiendo del autor y pueden existir diversas variantes para una misma técnica. Afortunadamente, algunos autores de IS ya han hecho el trabajo de compilar un catálogo de técnicas de la IPO [68]. Ferré et al. [68] compilaron una lista de técnicas desde reconocidas fuentes de la IPO (ver Anexo F). Presentamos aquí un resumen que permita al lector seguir el resto de esta investigación donde analizamos cuáles técnicas de usabilidad son utilizadas en los desarrollos OSS y cómo y en qué medida han tenido que ser adaptadas a la filosofía OSS.

De acuerdo a Ferré et al. [68], las actividades más representativas del proceso de la IPO son: *especificación del contexto de uso*, *especificaciones de usabilidad*, *desarrollo del concepto del producto*, *prototipado*, *diseño de la interacción* y *evaluación de la usabilidad*. Ferré et al. [68] mapean estas actividades (y sus correspondientes técnicas asociadas) a las actividades de desarrollo de la IS. En algunos casos las actividades de la IPO se integran en actividades IS existentes. Por ejemplo, la actividad especificaciones de usabilidad puede ser integrada en la especificación de requisitos. En otros casos, sin embargo, es necesario añadir actividades adicionales, como el diseño de la interacción, que no son usualmente llevadas a cabo en un

proceso de desarrollo no centrado en el usuario. Estas actividades adicionales tendrán como nombre el mismo que reciben en la IPO.

Las actividades de la IPO han sido mapeadas teniendo en cuenta las actividades de desarrollo de la IS: **ingeniería de requisitos**, **diseño** y **evaluación**. Las técnicas de la IPO en el catálogo están clasificadas según lo que significan ingeniería de requisitos, diseño y evaluación para la IS. Las actividades de la IPO mapeadas en la actividad de la IS **ingeniería de requisitos** son: especificación del contexto de uso, especificaciones de usabilidad, desarrollo del concepto del producto y prototipado. La Tabla F.1 (Anexo F) recoge todas las técnicas de usabilidad agrupadas por las actividades de la IS relacionadas con la ingeniería de requisitos: educción, análisis, especificación y validación de requisitos. La actividad de la IPO que se mapea en la actividad de **diseño** de la IS es el *diseño de la interacción*. Según Ferré et al. [68], una nueva actividad (*diseño de la interacción*) debe ser adicionada a las actividades de diseño de la IS. El diseño de la interacción no trata únicamente el diseño de los elementos visibles de la interfaz de usuario, sino también es responsable de definir los entornos de interacción y su comportamiento. La Tabla F.2 (Anexo F) presenta las técnicas de usabilidad agrupadas según las actividades de la IS relacionadas con el diseño. Finalmente, en la actividad de **evaluación** se mapea la actividad de la IPO: *evaluación de la usabilidad*. En la IPO hay tres principales actividades de evaluación de la usabilidad: evaluación por expertos, test de usabilidad y estudios de seguimiento de sistemas instalados. Nótese que estas tres actividades necesitan ser añadidas a la actividad de evaluación de la IS con el fin de evaluar la usabilidad del software. La Tabla F.3 (Anexo F) detalla las técnicas de la IPO agrupadas dentro de la actividad de evaluación de la IS.

5.4. Uso de Técnicas de la IPO en OSS

La comunidad OSS está empezando a adoptar una serie de técnicas de usabilidad en su proceso de desarrollo, principalmente gracias al apoyo de algunas compañías como Canonical, Novell, Red Hat, y Sun Microsystems. Las primeras contribuciones en este sentido fueron realizadas por Sun para los proyectos NetBeans, GNOME y OpenOffice. Para el proyecto GNOME, por ejemplo, Sun elaboró guías de usabilidad (HIGs) [35] y realizó estudios de usabilidad [75].

Hemos clasificado las técnicas de la IPO que están empezando a ser adoptadas en los proyectos de desarrollo OSS en tres grupos. El primer grupo incluye las técnicas de la IPO que están siendo adoptadas sin ningún tipo de adaptación y serán discutidas en detalle en la subsección 5.5.1. Un segundo grupo incluye técnicas que han sido sometidas a ligeras modificaciones para adaptarlas a la filosofía de trabajo de OSS. Este grupo de técnicas será discutido en la subsección 5.5.2. Aunque la mayoría de las técnicas de usabilidad usadas en la comunidad OSS han sido adaptadas de la IPO, algunas técnicas han sido materializadas en la comunidad OSS. Estas técnicas son el resultado de la filosofía colaborativa y distribuida de esta comunidad. Estas técnicas parecen a primera vista ser *exclusivas* de OSS, es decir, técnicas no propuestas por la comunidad de la IPO para mejorar la usabilidad. Sin embargo, un análisis más profundo revela que tales técnicas son esencialmente equivalentes a las técnicas de la IPO existentes. Lo que ocurre con estas técnicas es que han sido reinterpretadas para OSS. Éstas constituyen el tercer grupo que será discutido en la subsección 5.5.3. Esta reinterpretación puede ser vista como un *disfraz* mediante el cual la técnica es adaptada a los principios de la comunidad OSS. Nótese que la comunidad OSS ha tratado de adoptar sólo un subconjunto reducido de las técnicas de la IPO. La Figura 5.1 muestra un mapa mental que ilustra el catálogo de técnicas de la IPO para mejorar la usabilidad descritas en la sección 5.3 agrupadas por actividad de desarrollo. Las ramas terminales representan las técnicas de la IPO y las otras ramas representan las actividades genéricas de la IS.

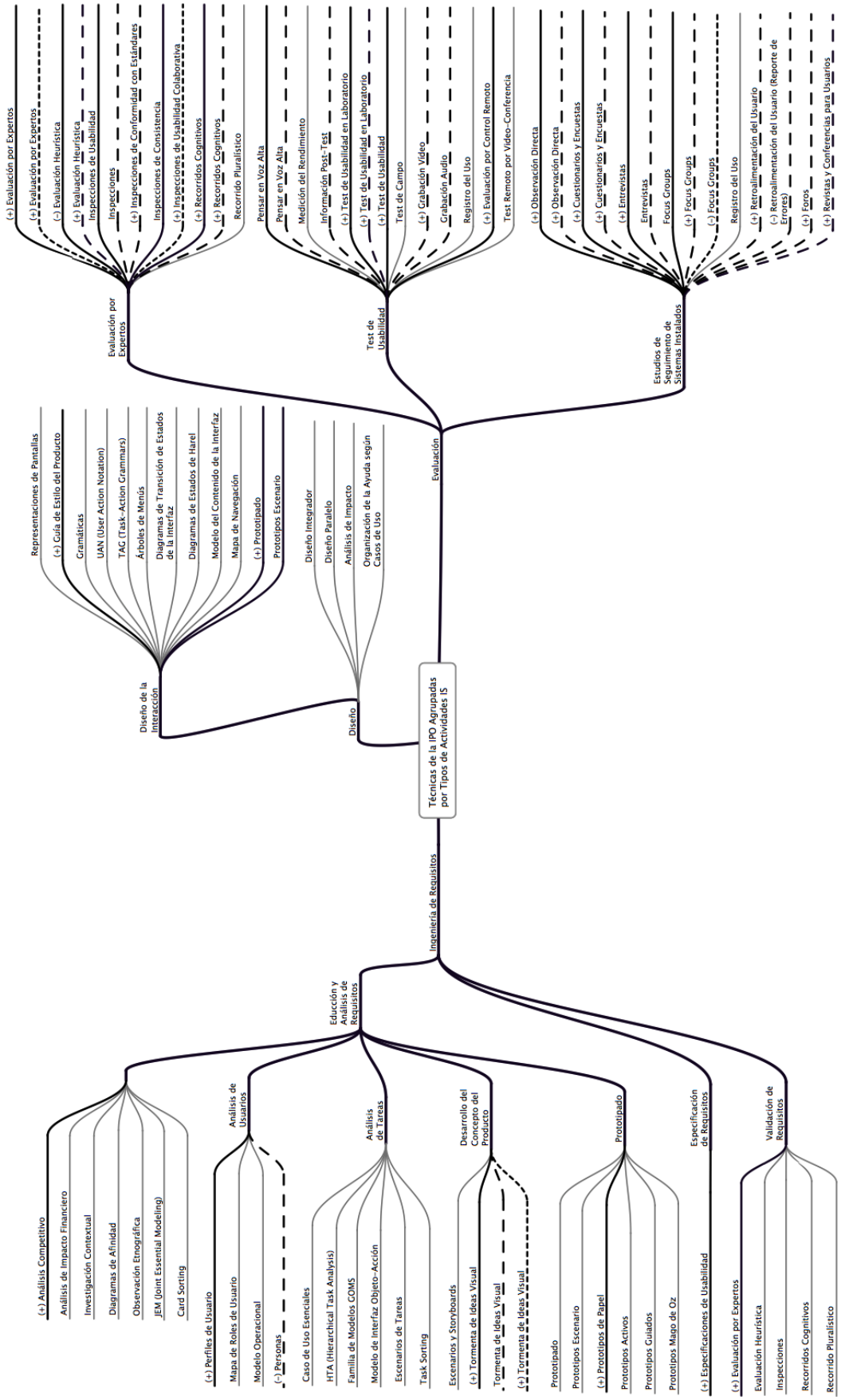


Figura 5.1: Técnicas de la IPO Agrupadas por Tipos de Actividad IS.

Las ramas terminales trazadas por una línea de color negro sólido representan las técnicas adoptadas puras por OSS. Las técnicas que ha adoptado OSS con ligeras modificaciones se representan con una línea discontinua de trazo largo y color negro. Las técnicas reinterpretadas se muestran con una línea discontinua de trazo corto y color negro. Las técnicas que OSS no ha adoptado tienen línea continua y color gris. Las técnicas que han sido adoptadas satisfactoriamente están marcadas con (+). Las técnicas cuya adopción fué insatisfactoria en al menos un caso están marcadas con (-). Si la técnica no está marcada ni por (+) ni por (-), es porque no se reportó información para decidir si la adaptación fue satisfactoria o no. Por ejemplo, la técnica análisis competitivo ha sido adoptada pura (línea de color negro sólido) y satisfactoriamente (está marcada con (+)).

Como se observa en la Figura 5.1, la comunidad OSS ha adoptado técnicas de usabilidad relacionadas con todas las actividades de desarrollo de la ingeniería de requisitos, diseño y evaluación. Sin embargo, los proyectos OSS han adoptado pocas técnicas relacionadas con las actividades de ingeniería de requisitos y diseño. La mayoría de proyectos OSS ha adoptado técnicas relacionadas con la evaluación.

La Tabla 5.1 resume, para cada una de las actividades de desarrollo, el número de técnicas de la IPO (de acuerdo al catálogo usado - ver Anexo F), y el número total y porcentaje de técnicas adoptadas por OSS en cualquier forma (es decir, puras, con ligeras modificaciones o reinterpretadas). Nótese que los tres tipos de adopción no son conjuntos disyuntos. La misma técnica puede haber sido adoptada en más de una forma. Por ejemplo, la técnica *perfiles de usuario* sólo ha sido adoptada pura, mientras que la técnica *evaluación heurística* ha sido adoptada pura y con ligeras modificaciones, y la técnica *focus groups* ha sido adoptada pura, con ligeras modificaciones y ha sido reinterpretada.

Nótese que para el cálculo del número total de técnicas de la IPO y las técnicas adoptadas por OSS hemos tenido en cuenta tanto la técnica genérica (por ejemplo, pensar en voz alta) y cualquiera de sus variantes (por ejemplo, interacción constructiva, test retrospectivo, toma de incidentes críticos y método de entrenamiento) porque OSS ha adoptado la técnica genérica, como también alguna de sus variantes. Por otro lado, si una técnica genérica tiene diferentes nombres asignados por diferentes autores de la IPO, sólo se cuenta la técnica una sola vez. Por ejemplo, la técnica pensar en voz alta recibe diferentes nombres en la literatura (por ejemplo, toma del protocolo verbal concurrente, test formales de usabilidad) como se reportó en [68] y se muestra en las subsecciones 5.5.1, 5.5.2 y en el Anexo F.

Tabla 5.1: Porcentaje de Técnicas de la IPO Adoptadas en OSS.

Tipo de Actividad IS		**Nro. de Técnicas de la IPO (siguiendo [68])**	**Nro. Total de Técnicas Adoptadas por OSS**	**% de Técnicas Adoptadas**		
				Puras	**Ligeras Modificaciones**	**Reinterpretadas**
Ingeniería de Requisitos	Educción y Análisis de Requisitos	25	5 (20 %)	16	8	4
	Especificación de Requisitos	1	1 (100 %)	100	0	0
	Validación de Requisitos	7	1 (14,29 %)	14,28	0	0
Diseño		16	3 (18,75 %)	18,75	0	0
Evaluación		39	18 (46,15 %)	33,33	41,02	7,7

Como se observa en la Tabla 5.1 muchas de las técnicas de la IPO que están siendo usadas en OSS han sido adoptadas puras, mientras que las técnicas reinterpretadas representan el porcentaje más pequeño de técnicas adoptadas.

Para cada actividad de desarrollo, la Tabla 5.2 muestra en términos porcentuales cómo de satisfactorios fueron los resultados observados en la adopción de las técnicas. Hemos deducido de los artículos revisados en nuestro SMS si la adopción de la técnica ha sido satisfactoria. Este porcentaje es calculado de forma separada para cada tipo de adopción de la técnica (es decir, puras, con ligeras modificaciones y reinterpretadas). Nótese que si una misma técnica ha sido adoptada tanto pura como con ligeras modificaciones, entonces se ha considerado dos veces para el cálculo del porcentaje correspondiente. Los tres valores posibles que se muestran en la Tabla 5.2 son: satisfactorio (S), insatisfactorio (I) y no reportado (N-R) -es decir, ninguna información fue suministrada a este respecto-. Por ejemplo, un valor cero debajo de la (I) en la Tabla 5.2 significa que ninguna de las técnicas ha sido adoptada insatisfactoriamente, como es el caso de las técnicas adoptadas puras en la actividad de diseño. Un guión debajo de la (S) indica que no es posible determinar cómo fue la adopción de la técnica, porque ninguna de ellas ha sido adoptada. Esto aplica, por ejemplo, a las técnicas relacionadas con diseño y con ligeras modificaciones.

Tabla 5.2: Porcentaje de Satisfacción en la Incorporación de las Técnicas de la IPO en OSS.

Tipo de Actividad IS	**% (S)atisfacción / (I)nsatisfacción**								
	Puras			**Ligeras Modificaciones**			**Reinterpretadas**		
	S	**I**	**N-R**	**S**	**I**	**N-R**	**S**	**I**	**N-R**
Ingeniería de Requisitos	87,5	0	12,5	0	16,67	83,33	66,67	0	33,33
Diseño	50,0	0	50,0	-	-	-	-	-	-
Evaluación	66,67	2,56	30,77	61,88	2,5	35,62	76,67	6,66	16,67

Como se observa en la Tabla 5.1, dentro del grupo de las técnicas adoptadas **puras**, encontramos que las técnicas más a menudo adoptadas en los desarrollos OSS son las relacionadas con la evaluación de la usabilidad (33,33 %). Los resultados de aplicar técnicas de la IPO para la evaluación de la usabilidad fueron satisfactorios en el 66,67 % de los casos, y solamente un 2,56 % han sido insatisfactorios (ver Tabla 5.2). Ninguna información fue suministrada sobre el uso de las otras técnicas de evaluación de la usabilidad (30,77 %). Nosotros podemos concluir que unas pocas técnicas de evaluación de la usabilidad (22,23 %) han sido usadas puras en los desarrollos OSS sin muchos problemas.

El porcentaje de uso de las técnicas adoptadas puras relacionadas con ingeniería de requisitos y diseño son muy similares entre sí (cerca del 20 %). Las técnicas relacionadas con la ingeniería de requisitos adoptadas puras fueron las más satisfactorias (87,5 %). Además, ninguna de las técnicas adoptadas puras fue insatisfactoria, mientras que los resultados de su uso no fueron reportados en el 12,5 % de los casos.

Los resultados de adoptar técnicas relacionadas con diseño fueron satisfactorios en el 50 % de los casos. El resultado del uso del otro 50 % no fue reportado. Ninguna de las técnicas relacionadas con diseño ha tenido resultados insatisfactorios. Podemos concluir que OSS encuentra la mayoría de técnicas de usabilidad de requisitos o diseño difíciles de adoptar puras (el 75 % no han sido adoptadas). Pero cuando se adoptaron, estuvo bien (los desarrolladores estaban en su mayoría satisfechos).

Dentro del grupo de técnicas adoptadas con **ligeras modificaciones**, las técnicas más frecuentemente adoptadas (41,02 %) fueron de nuevo las relacionadas con la evaluación de la usabilidad. El porcentaje de satisfacción e insatisfacción para estas técnicas es del 61,88 %

y 2,5 % respectivamente (ver Tabla 5.2). En el 35,62 % de los casos los resultados del uso de las técnicas no fueron reportados. Podemos concluir que la adaptación de técnicas para la evaluación de la usabilidad a la filosofía OSS es bastante generalizada (25,38 %), y el resultado es bastante satisfactorio.

Solo el 8 % de las técnicas adoptadas con ligeras modificaciones están relacionadas con la ingeniería de requisitos. Los resultados para estas fueron insatisfactorios en el 16,66 % de los casos, mientras que los resultados de su uso no fueron reportados en el 83,33 % de los casos. Ninguna de las técnicas relacionadas con la ingeniería de requisitos adoptadas con ligeras modificaciones fueron satisfactorias. No hay técnicas relacionadas con diseño adoptadas con ligeras modificaciones. Podemos concluir que no es fácil modificar técnicas de usabilidad para ingeniería de requisitos (6,06 %) y no hay datos sobre el nivel de satisfacción.

Con respecto a las técnicas **reinterpretadas**, el más alto porcentaje de técnicas adoptadas (7,7 %) está de nuevo relacionado con la evaluación de la usabilidad. De estas técnicas, los resultados fueron satisfactorios en la mayoría de los casos (76,67 %), y sólo el 6,66 % fue insatisfactorio. Los resultados de las técnicas usadas no fueron reportados para el otro 16,67 % de técnicas adoptadas.

Sólo el 4 % de las técnicas reinterpretadas estaban relacionadas con la ingeniería de requisitos. Los resultados fueron satisfactorios para la mayoría (66,67 %) de las técnicas reinterpretadas y no hay casos insatisfactorios. Los resultados de su uso no fueron reportados en todos los otros casos. De nuevo no hay técnicas reinterpretadas relacionadas con diseño. Podemos concluir que es difícil reinterpretar técnicas de usabilidad, aunque la satisfacción con las técnicas reinterpretadas está casi garantizada.

En resumen, la mayoría de las técnicas de la IPO (70 %) no se ha utilizado en proyectos OSS. Las técnicas de evaluación (20 %) representan la mayor parte del 30 % de las técnicas que se han adoptado. Los proyectos OSS han tenido acceso a los recursos necesarios para aplicar técnicas, como por ejemplo la colaboración de expertos en usabilidad. La participación de expertos fue posible gracias al apoyo suministrado por algunas empresas.

En cuanto a las técnicas adoptadas con ligeras modificaciones, la mayoría son técnicas relacionadas con la evaluación. Lo mismo aplica a las técnicas reinterpretadas, donde de nuevo la mayoría de las técnicas están relacionadas con la evaluación. Nuestra hipótesis es que esto se debe al hecho de que la adopción de técnicas de evaluación tiene un menor impacto en el proceso de desarrollo OSS. En otras palabras, las técnicas de evaluación de la usabilidad son menos intrusivas porque el desarrollo puede seguir adelante como es habitual ya que la evaluación se realiza al final por parte de terceros (por ejemplo, usuarios, expertos en usabilidad).

Las técnicas relacionadas con diseño son las técnicas menos frecuentemente adoptadas (sólo han sido adoptadas puras). Tenemos dos hipótesis para ello: (i) los desarrolladores no están familiarizados con las técnicas de la IPO aplicables al diseño y (ii) los canales de comunicación basados en texto (por ejemplo, correos electrónicos, foros de discusión, etc.) que se utilizan en los entornos distribuidos en los que se desarrolla OSS son un obstáculo para el diseño de la interfaz de usuario. En términos generales, la adopción de técnicas de la IPO en proyectos OSS ha sido satisfactoria.

Con respecto a las técnicas de la IPO que no han sido adoptadas en los desarrollos OSS, los proyectos OSS no han adoptado el 78,79 % de las técnicas de usabilidad relacionadas con la ingeniería de requisitos, mientras que el 81,25 % del grupo de técnicas relacionadas con el diseño no han sido adoptadas. Éste es el grupo con el mayor número de técnicas no utilizadas. OSS no ha adoptado el 53,85 % de las técnicas relacionadas con la evaluación (incluyendo evaluación por expertos, test de usabilidad y estudios de seguimiento de sistemas instalados). En la sección 5.6 reportamos el análisis completo de las técnicas de la IPO que no han sido adoptadas en los desarrollos OSS. A continuación, discutiremos las técnicas que se han utilizado en los proyectos OSS por tipo de adopción.

La Tabla 5.3 presenta un resumen de las técnicas de la IPO **adoptadas puras** por la comunidad OSS en las actividades de ingeniería de requisitos y diseño, mientras que la Tabla 5.4 resume las técnicas de la IPO adoptadas puras por la comunidad OSS para la evaluación de la usabilidad utilizando evaluación por expertos, test de usabilidad y estudios de seguimiento de sistemas instalados. Para cada técnica, especificamos su nombre genérico, el nombre usado por los autores, el nivel de adopción (puras o con ligeras modificaciones), las referencias, los proyectos OSS donde han sido adoptadas y si la adopción ha sido (S)atisfactoria, (I)nsatisfactoria o no reportada (N-R). En algunos casos se especifica el porcentaje de entrevistados o de proyectos que han usado cada técnica. Cuando OSS no ha adoptado ninguna técnica relacionada con una actividad específica, hemos dejado la fila en blanco (por ejemplo, análisis de tareas). En la subsección 5.5.1 discutimos los proyectos donde se aplicaron las técnicas y reportamos la experiencia.

Tabla 5.3: Técnicas de la IPO Adoptadas Puras Relacionadas con los Tipos de Actividades IS Relativas a la Ingeniería de Requisitos y Diseño.

Tipo de Actividad IS		**Técnica de la IPO**	**Nombre Dado por los Autores OSS**	**Ref.**	**Uso**		**¿Incorp. Satisfactoria?**
					Proyectos OSS	**% Entrev. / % Proyect.**	
Educción y Análisis de Requisitos		Análisis Competitivo	Competitive analysis	[SU20]	OpenOffice		S
	Análisis de Usuarios	Perfiles de Usuario	Defining a target user group	[SU37]	3DA, BG, DTP, VG		N-R
			Identify user profiles	[SU36]	GIMP		S
	Análisis de Tareas						
	Desarrollo del Concepto del Producto	Tormenta de Ideas Visual	Brainstorming	[SU25]	Carrot2		S
	Prototipado	*Prototipos de Papel*	Paper prototyping	[SU10]		44,4 % exper. 20 % desarr.	S
Especificación de Requisitos		Especificaciones de Usabilidad	Drawing up specifications	[SU37]	3DA, BG, OS, VG, WB		N-R
			Dialogue and interaction design	[SU20]	OpenOffice		S
			Writing specifications	[SU3]	OpenOffice		S
			User interface specification document	[SU3]	NetBeans		S
Validación de Requisitos		Evaluación por Expertos	Via dedicated UX people	[SU37]	BG, CMS, DWE, OS, WB		S

Tabla 5.3: Técnicas de la IPO Adoptadas Puras Relacionadas con los Tipos de Actividades IS Relativas a la Ingeniería de Requisitos y Diseño (continuación).

Tipo de Actividad IS	Técnica de la IPO	Nombre Dado por los Autores OSS	Ref.	Uso		¿Incorp. Satisfactoria?
				Proyectos OSS	% Entrev. / % Proyect.	
Diseño de la Interacción	Guía de Estilo del Producto	By developing and applying UI design patterns	[SU37]	DWE		N-R
		Use of human interface guidelines	[SU37]	BG, DWE, OS, VE		N-R
		User human interface guidelines	[SU23]	(no especif.)	(no especifica)	N-R
		User interface guidelines	[SU1]		79 % desarrollad.	S
		Interface guidelines	[SU10]		57,5 % proyectos	S
		Human interface guidelines	[SU3]	GNOME		S
	Prototipos Escenario	Creating scenarios of user to guide design	[SU37]	BG		N-R
	Prototipado	Prototyping of the new design	[SU25]	Carrot2		S
		Rapid prototype	[SU15]	TrueCrypt		S

3DA: paquete de animación 3D; BG: aplicación de gráficos de mapa de bits; CMS: sistema de gestión de contenidos para la web; DTP: aplicación de edición; DWE: entorno de ventanas de escritorio; OS: sistema operativo de escritorio; VE: aplicación para edición de video; VG: aplicación de gráficos basada en vectores; WB: navegador web.

Tabla 5.4: Técnicas de la IPO Relacionadas con las Actividades de Evaluación Adoptadas Puras.

Tipo de Actividad IS	Técnica de la IPO	Nombre Dado por los Autores OSS	Ref.	Uso		¿Incorp. Satisfactoria?
				Proyectos OSS	% Entrev. / % Proyect.	
Evaluación por Expertos	Evaluación por Expertos	Via dedicated UX people	[SU37]	BG, CMS, DWE, OS, WB		S
		Expert review	[SU10]		69,8 % proyectos	S
	Inspecciones de Consistencia	By finding inconsistencies in the "rules" of the interface	[SU37]	3DA, BG, DUT		N-R
	Evaluación Heurística	Heuristic evaluation	[SU10]	KDE	50 % expertos 27 % desarroll.	S
		Usability heuristic	[SU36]	KDE		S
		User interface heuristics	[SU4] [SU24]	TYPO3		I
	Inspecciones	Usability inspections	[SU10]	KDE		N-R
	Recorridos Cognitivos	Cognitive walkthrough	[SU15]	TrueCrypt		S

Tabla 5.4: Técnicas de la IPO Relacionadas con las Actividades de Evaluación Adoptadas Puras (continuación).

Tipo de Actividad IS	Técnica de la IPO	Nombre Dado por los Autores OSS	Ref.	Uso		¿Incorp. Satisfactoria?
				Proyectos OSS	% Entrev. / % Proyect.	
Test de Usabilidad	Test de Usabilidad	Usability test	[SU10]	KDE	52 % proyectos	S
		Usability testing of the new design	[SU25]	Carrot2		S
		Usability test	[SU20]	OpenOffice		S
	Evaluación por Control Remoto	Via expert reviews, performed remotely by UX members	[SU37]	DWE, VE		S
	Pensar en Voz Alta	By conducting think-aloud studies	[SU37]	DUT, OS, WB		N-R
		Think aloud	[SU15]	TrueCrypt		S
	Test de Usabilidad en Laboratorio	By performing usability studies in controlled settings	[SU37]	CMS, DWE, OS		N-R
		Usability evaluation in laboratory	[SU1]		8 % desarroll.	S
Estudios de Seguimiento de Sistemas Instalados	Cuestionarios y Encuestas	Through surveys of user base	[SU37]	WB, OS		S
		Directed program asking for users	[SU37]	WB, OS		S
		Surveys	[SU36]	GIMP, TV-Browser		S
	Entrevistas	Interviews of users	[SU37]	DWE		S
		Get feedback from early adopters	[SU10]		42,4 % proyectos	S
	Focus Groups	Focus groups	[SU10]		21,9 % proyectos	N-R
	Observación Directa	User and task observations	[SU36]	GIMP, TV-Browser		S
		Site visits	[SU20]	OpenOffice		S

DUT: utilidad de escritorio; FE: editor de fuentes.

La Tabla 5.5 presenta un resumen de las técnicas de la IPO **adoptadas con ligeras modificaciones** por la comunidad OSS en la actividad de ingeniería de requisitos. La Tabla 5.6 muestra las técnicas de la IPO adoptadas con ligeras modificaciones por la comunidad OSS para la evaluación de la usabilidad. La comunidad OSS no ha adoptado ninguna de las técnicas de la IPO relacionadas con la actividad de diseño. En la subsección 5.5.2 discutimos los proyectos en los cuales fueron aplicadas las técnicas con ligeras modificaciones.

OSS ha reinterpretado solo unas pocas, cinco para ser exactos, técnicas: *design-by-blog* [137][144][188], *open content projects* [188], *seasons of usability* [152][188], *discutir en comunidad* [35][137][188] y *design workshops* [35][188]. Sólo una de estas técnicas, *design-by-blog* [137] ha sido adoptada en la actividad de ingeniería de requisitos, mientras que las restantes han sido adoptadas en las actividades de evaluación.

Tabla 5.5: Técnicas de la IPO Relacionadas con las Actividades de Ingeniería de Requisitos Adoptadas con Ligeras Modificaciones.

Tipo de Actividad IS		**Técnica de la IPO**	**Nombre Dado por los Autores OSS**	**Ref.**	**Uso**		**¿Incorp. Satisfactoria?**
					Proyectos OSS	**% Entrev. / % Proyect.**	
Educción y Análisis de Requisitos	Análisis de Usuarios	Personas	Creating personas of users	[SU37]	OS, WB		N-R
			Personas	[SU10]	KDE		N-R
			Personas	[SU4] [SU24]	TYPO3		I
	Análisis de Tareas						
	Desarrollo del Concepto del Producto	Tormenta de Ideas Visual	Interface brainstorming wiki	[SU37]	BG		N-R
	Prototipado						
Especificación de Requisitos							
Validación de Requisitos							

Tabla 5.6: Técnicas de la IPO Relacionadas con las Actividades de Evaluación Adoptadas con Ligeras Modificaciones.

Tipo de Actividad IS	**Técnica de la IPO**	**Nombre Dado por los Autores OSS**	**Ref.**	**Uso**		**¿Incorp. Satisfactoria?**
				Proyectos OSS	**% Entrev. / % Proyect.**	
Evaluación por Expertos	*Inspección de Conform. con Estándares*	Bug hunting seasons	[SU10]	KDE		S
	Evaluación Heurística	Heuristic evaluations	[SU29]	Role-playing roguelike game		S
	Inspecciones	Inspection by usability expert	[SU1]		42 % desarrollad.	N-R
	Recorridos Cognitivos	By discovering what doesn't work for them as they develop the software	[SU37]	3DA, BG, CMS, DTP, DUT, DWE, FE, OS, VE, VG, WB		N-R
		Cognitive walkthrough	[SU29]	Role-playing roguelike game		S
Test de Usabilidad	Grabación Vídeo	Video recording	[SU29]	Role-playing roguelike game		S
	Grabación Audio	Audio recording	[SU29]	Role-playing roguelike game		N-R
	Información Post-Test	Interviewed the test persons after the usability testing sessions	[SU29]	Role-playing roguelike game		N-R
	Pensar en Voz Alta	Synchronous remote usability evaluation	[SU1]		21 % desarroll.	N-R
	Test de Usabilidad en Laboratorio	Laboratory usability testing	[SU29]	Role-playing roguelike game		S

Tabla 5.6: Técnicas de la IPO Relacionadas con las Actividades de Evaluación Adoptadas con Ligeras Modificaciones (continuación).

Tipo de Actividad IS	Técnica de la IPO	Nombre Dado por los Autores OSS	Ref.	Uso		¿Incorp. Satisfactoria?
				Proyectos OSS	% Entrev. / % Proyect.	
Estudios de Seguimiento de Sistemas Instalados	Cuestionarios y Encuestas	Via "reference users", "bleeding edge" users of nightly builds, and professional users	[SU37]	3DA, BG, DTP, DUT, FE, VG, WB		S
	Entrevistas	By giving tutorials on the software	[SU37]	BG, DTP		N-R
	Observación Directa	Through informal observations of friends and family	[SU37]	BG, DUT, VG, WB		S
	Focus Groups	Annual, monthly, weekly, or *ad hoc* meeting	[SU37]	3DA, BG, DWE, OS		S
	Foros	By paying attention to what is asked, discussed, or requested in IRC, mailing lists, and forums	[SU37]	3DA, BG, CMS, DTP, DUT, DWE, FE, OS, VE, VG, WB		S
	Retroalimentación del Usuario (Reporte de Errores)	Through bug reports	[SU37]	3DA, BG, CMS, DTP, DUT, DWE, FE, OS, VE, VG, WB		I
		Users assumed to report bugs	[SU16]	(no especifica)	(no especifica)	N-R
		Reporting a usability problem	[SU23]	(no especifica)	(no especifica)	I
		Descriptions of the usability problems in the wiki	[SU29]	Role-playing roguelike game		S
	Retroalimentación del Usuario	Seeking feedback on mock-ups, prototypes	[SU37]	3DA, BG, CMS, DTP, DUT, OS, VE, WB		S
	Revistas y Conferencias para Usuarios	Conference	[SU3]	OpenOffice		S
		Conference	[SU37]	BG, DUT, OS, VG, WB		S

La Tabla 5.7 resume todas las técnicas de la IPO adoptadas por la comunidad OSS (técnicas reinterpretadas). La subsección 5.5.3 discute los proyectos en los cuales estas técnicas fueron aplicadas y reportamos la experiencia. La sección 5.6 discute las técnicas que no han sido adoptadas en los proyectos OSS.

Tabla 5.7: Técnicas de Usabilidad Reinterpretadas en OSS.

Técnica de la IPO que Rememora	Principio de la Cultura OSS	Nombre Dado por los Autores OSS	Ref.	Uso		¿Incorp. Satisfactoria?
				Proyectos OSS	% Entrev. / % Proyect.	
Tormenta de Ideas Visual	Democracia. Todo se discute en comunidad	Bloggins about designs, getting feedback from user base	[SU37]	OS, WB		S
		Design-by-blog	[SU23]	(no especif.)	(no especif.)	S
		Blogs about user interface & usability	[SU26]	GNOME Python		N-R
Inspecciones de Usabilidad Colaborativas	Participación. Contribución	Open content projects	[SU37]	3DA		S
Evaluación por Expertos	Participación. Contribución. Todos en comunidad aportan algo (trabajo)	Participation in the season of usability	[SU37]	DWE, OS		N-R
		The student usability teams	[SU29]	Role-playing roguelike game		S
Focus Groups	Democracia. Todo se discute en comunidad	Discussion	[SU3]	NetBeans		I
		Discussions or IRC and mailing lists	[SU37]	3DA, BG, CMS, DTP, DUT, DWE, FE, OS, VE, VG, WB		S
		Discussion in the OSS community of the solutions to usability problems	[SU23]	(no especif.)	(no especif.)	S
Focus Groups	Democracia. Todo se discute en comunidad	Running design clinics at conferences	[SU37]	OS		S
		Design workshops	[SU3]	OpenOffice		S

En resumen, la comunidad OSS está aplicando su visión del desarrollo de software a la usabilidad. En el desarrollo de software comercial, hay un experto o grupo de expertos encargados de dar las pautas sobre cómo enfocar o mejorar la usabilidad. Sin embargo en la comunidad OSS, si bien algunos proyectos cuentan con expertos en usabilidad, todos sus miembros participan y discuten cuál es la mejor manera para resolver aspectos de usabilidad, por ejemplo, cómo resolver un problema de usabilidad o cuál es el diseño más adecuado para una funcionalidad específica, según las necesidades y expectativas de los usuarios no-desarrolladores.

En la comunidad OSS existe una simbiosis entre dos grupos de miembros diferentes: los desarrolladores y los usuarios no-desarrolladores. Ambos grupos interaccionan estrechamente para conseguir un objetivo común. Por un lado, los usuarios ayudan a los desarrolladores con ideas sobre cómo resolver un determinado problema de usabilidad y opinan sobre diferentes propuestas de diseños que han realizado los desarrolladores. Por otro lado, los desarrolladores materializan estas ideas o diseños en algo tangible.

La comunidad OSS es fuerte en el desarrollo de software, pero no en cuestiones de usabilidad. Sin embargo, es un vasto organismo, que tiene la capacidad de generar ideas y aprender. A medida que este organismo descubre lo que es la usabilidad y cuáles técnicas están disponibles

para su mejora, es capaz de aplicar sus principios, filosofía y metodología de trabajo a esta cuestión, reinterpretar y percibir la usabilidad desde su propia perspectiva (cultura).

En las siguientes secciones analizaremos las técnicas de usabilidad adoptadas por OSS. Este análisis es realizado en tres partes, una por cada tipo de adaptación (puras, con ligeras modificaciones y reinterpretadas). Además, analizaremos las consecuencias de las técnicas de usabilidad no usadas por OSS de ninguna manera.

5.5. Técnicas de la IPO Adoptadas por OSS

En esta sección analizaremos las técnicas de usabilidad que OSS ha adoptado puras, con ligeras modificaciones o reinterpretadas. Es importante mencionar que a lo largo del presente trabajo de investigación, cuando nos referimos a las técnicas de usabilidad que OSS ha adoptado en sus desarrollos nos estamos refiriendo a las técnicas que han sido adoptadas y a su vez han sido reportadas en la literatura. Es decir, es probable que haya técnicas adaptadas por algún proyecto OSS, pero de las cuales no tenemos constancia. A continuación, iniciaremos con el análisis de las técnicas de usabilidad adoptadas puras por OSS.

5.5.1. Técnicas de la IPO Adoptadas Puras

Esta sección describe las técnicas que se han adoptado puras en el proceso de desarrollo OSS. La descripción de cada técnica ha sido agrupada de acuerdo a la actividad IS con la que está relacionada (es decir, ingeniería de requisitos, diseño y evaluación).

5.5.1.1. Técnicas de Usabilidad Relacionadas con Ingeniería de Requisitos

Observando la Tabla F.1 (Anexo F), las técnicas de usabilidad que OSS ha adoptado en las actividades de ingeniería de requisitos son: *análisis competitivo* [132], *perfiles de usuario* [165][188], *tormenta de ideas visual* [142], *prototipos de papel* [46], *especificaciones de usabilidad* [35][132][188] y *evaluación por expertos* [188]. La técnica *análisis competitivo* ha sido adoptada en el proyecto OpenOffice con el objetivo de mejorar la usabilidad analizando productos similares existentes y extrayendo objetivos generales para el producto [132]. La técnica *perfiles de usuario* ha sido adoptada en diferentes proyectos OSS (por ejemplo, GIMP, un paquete de animación 3D, una aplicación de gráficos de mapas de bits) para definir los tipos representativos de usuarios en función de las tareas que van a realizar y las habilidades y conocimientos relacionados con las tareas [92]. La adopción de esta técnica permitió conocer a los usuarios no-desarrolladores [165][188].

El proyecto Carrot2 adoptó la técnica *tormenta de ideas visual* pura con el objetivo de generar ideas para el diseño de la nueva interfaz [142]. Según el estudio empírico realizado por Çetin y Göktürk [46], el 44,4 % de los expertos en usabilidad y el 22,2 % de los desarrolladores encuestados afirmaron que utilizaban los *prototipos de papel* como un método para evaluar la usabilidad.

Diversos proyectos OSS, entre los más conocidos OpenOffice y Netbeans, han adoptado las *especificaciones de usabilidad*. Estas especificaciones fueron realizadas por un grupo de expertos en usabilidad con el objetivo de describir la interacción detallada y el diseño visual de la mayoría de las nuevas funcionalidades [35][132][188].

Una técnica de *evaluación por expertos* ha sido adoptada en el desarrollo de un entorno de ventanas de escritorio. Terry et al. [188] no proporcionan ninguna información adicional para aclarar cuál de las técnicas de evaluación por expertos ha sido adoptada en este proyecto. Los expertos dieron su opinión sobre el diseño de las interfaces de usuario propuestas por

los desarrolladores en reuniones que se celebraban mensualmente. Estas reuniones también ayudan a recordarle a los desarrolladores la importancia de considerar la usabilidad en el desarrollo de las aplicaciones [188].

5.5.1.2. Técnicas de Usabilidad Relacionadas con Diseño

En las actividades de diseño, la comunidad OSS ha adoptado tanto principios básicos de usabilidad como técnicas. Los principios adoptados son dos: *diseñar pensando en el usuario final* [142] y *diseñar antes de codificar* [192]. El principio de usabilidad *diseñar pensando en el usuario final* fue seguido en el proyecto Carrot2. En este proyecto, el diseño original de la aplicación fue replanteado de acuerdo a los usuarios finales para quienes estaba dirigida (investigadores en el área de minería de datos). Gracias a este rediseño, todos los problemas de usabilidad de la aplicación fueron resueltos [142]. En el proyecto Netscape, una de las lecciones aprendidas fue *diseñar antes de codificar*. No seguir este principio ocasionó mucho retrabajo y reporte de errores [192].

Del grupo de técnicas listado en la Tabla F.2, OSS ha adoptado las siguientes: *guía de estilo del producto* [23][35][46][137][188], *prototipos escenarios* [188], y *prototipado* [84][142]. Las *guías de estilo del producto* adoptadas por OSS son conocidas como HIGs. Estas HIGs son un conjunto específico de reglas que siguen los desarrolladores con el fin de crear un *'look and feel'* para la interfaz de usuario y garantizar su consistencia a través de toda la aplicación. Las HIGs usadas con mayor frecuencia fueron las desarrolladas por el grupo de expertos del proyecto GNOME [23][35][46][137][188]. La adopción de la técnica *prototipos escenarios* ha sido útil para guiar el diseño y tratar los problemas de usabilidad de una aplicación de gráficos de mapa de bits [188]. La técnica de *prototipado* fue adoptada en el proyecto Carrot2 para permitir una mayor flexibilidad durante el diseño de la nueva interfaz de usuario. El desarrollo del prototipo se basó en ficheros HTML estáticos [142]. Esta técnica también ha sido adoptada en el proyecto TrueCrypt (herramienta para el cifrado de discos) para construir un prototipo de la nueva interfaz [84].

5.5.1.3. Técnicas de Usabilidad Relacionadas con la Evaluación

De acuerdo a la Tabla F.3 (Anexo F), existen tres grandes grupos de técnicas para evaluar la usabilidad como parte de las actividades de evaluación: *evaluación por expertos*, *test de usabilidad* y *estudios de seguimiento de sistemas instalados*. Las técnicas de usabilidad pertenecientes a la **evaluación por expertos** que ha adoptado OSS son: *evaluación por expertos* [46][188], *inspecciones de consistencia* [188], *evaluación heurística* [38][46][139][165], *inspecciones* [46], y *recorridos cognitivos* [84]. A continuación, describiremos las técnicas adoptadas puras.

Técnicas de *evaluación por expertos* [46][188] han sido adoptadas en diferentes proyectos (aunque no se ha especificado que técnicas han sido adoptadas), por ejemplo, en un gestor de contenidos Web, un sistema operativo de escritorio y en una aplicación de gráficos basados en vectores. Las evaluaciones realizadas por los expertos suministraron retroalimentación a los desarrolladores sobre la usabilidad de las aplicaciones [188]. La técnica *inspecciones de consistencia* ha sido adoptada en diversos proyectos, como, por ejemplo, en una aplicación de gráficos de mapa de bits [188]. El objetivo de esta técnica es verificar la consistencia en la terminología, color, diseño, etc., de las interfaces de usuario [180].

La técnica *evaluación heurística* analiza la conformidad de la interfaz de usuario con base en unos principios reconocidos de usabilidad (heurísticas) [138]. Las *evaluaciones heurísticas* fueron realizadas en el proyecto KDE por un grupo de expertos creado para mejorar la usabilidad [46][165].

En las *inspecciones* de usabilidad varios expertos evalúan el grado de usabilidad de una aplicación basándose en la inspección o examen de su interfaz. Esta técnica ha sido adoptada en el proyecto KDE [46]. Çetin y Göktürk [46] no especifican las variantes de la técnica inspecciones que fueron adoptadas.

La técnica *recorridos cognitivos* tiene por objetivo realizar una inspección de la usabilidad para evaluar la facilidad de aprendizaje de un diseño básicamente por exploración y está motivada por la observación de que muchos usuarios prefieren aprender a utilizar el software mediante la exploración de sus opciones [149]. Esta técnica ha sido adoptada en el proyecto TrueCrypt [84].

En cuanto a las técnicas para la evaluación de la usabilidad del grupo **test de usabilidad**, la comunidad OSS ha adoptado: *test de usabilidad* [46][132][142], *evaluación por control remoto* [188], *pensar en voz alta* [84][188] y *test de usabilidad en laboratorio* [23][188]. A continuación, describiremos las técnicas adoptadas puras.

Los *tests de usabilidad* han sido adoptados puros en las aplicaciones Amarok (reproductor de música), Kivio (diseñador visual), y Kget (gestor de descargas). Todas estas aplicaciones forman parte del proyecto KDE [46]. Esta técnica también ha sido adoptada en el proyecto Carrot2 [142]. En ningún caso se preciso qué técnicas del grupo de *test de usabilidad* fueron adoptadas. En el proyecto Carrot2, los *test de usabilidad* fueron realizados de manera informal a la interfaz de usuario. Los participantes en estos test recibieron una explicación breve de la aplicación y del objetivo de las pruebas, haciendo énfasis en que era la interfaz de usuario la que estaba siendo evaluada y no sus acciones. Los participantes realizaron una serie de tareas previamente establecidas, sin recibir ninguna clase de ayuda durante la ejecución de las mismas. Durante la realización de estas tareas, se observaron cuidadosamente a los participantes, tomando nota de sus interacciones con la aplicación.

La técnica *evaluación por control remoto* permite realizar una evaluación de usabilidad en el entorno habitual del usuario sin que el evaluador se encuentre físicamente presente. Esto ahorra costos y evita problemas de planificación [120]. Esta técnica ha sido adoptada en una aplicación para edición de video y en un entorno de ventanas de escritorio. En ambos casos, la adopción de esta técnica ha permitido descubrir problemas de usabilidad [188].

En la técnica *pensar en voz alta* los usuarios individualmente expresan en voz alta y libremente sus pensamientos, sentimientos y opiniones sobre cualquier aspecto (diseño, funcionalidad, etc.), mientras que interaccionan con la aplicación en presencia de un experto en usabilidad [120]. Esta técnica ha sido adoptada en el desarrollo de un sistema operativo de escritorio, un navegador Web, una utilidad de escritorio y en el proyecto TrueCrypt. La adopción de esta técnica en estos proyectos ha permitido descubrir problemas de usabilidad [84][188].

De las técnicas del grupo **test de usabilidad**, la técnica *test de usabilidad en laboratorio* es la técnica que ha sido menos frecuentemente adoptada en proyectos OSS debido al alto costo de crear un laboratorio para realizar las pruebas. De acuerdo al estudio empírico realizado por Andreasen et al. [23], sólo el 8 % de los encuestados había realizado test de usabilidad en laboratorio.

Finalmente, de las técnicas para la evaluación de la usabilidad del grupo **estudios de seguimiento de sistemas instalados** (ver Tabla F.3 en el Anexo F), la comunidad OSS ha adoptado: *cuestionarios y encuestas* [165][188], *entrevistas* [46][188], *focus groups* [46] y *observación directa* [132][165].

Los *cuestionarios y encuestas* fueron administrados por expertos en usabilidad a los usuarios no-desarrolladores y ayudaron a los desarrolladores a descubrir que la mayoría de los usuarios compartían las mismas expectativas y problemas de usabilidad [165][188]. Además, los *cuestionarios y encuestas* también fueron útiles para identificar los perfiles de usuarios [165]. La técnica *entrevistas* ha sido adoptada en el desarrollo de un entorno de ventanas de escritorio. Las entrevistas fueron realizadas por expertos en usabilidad a usuarios no-desarrolladores y permitieron descubrir problemas de usabilidad [188].

La técnica *focus groups* consiste en una serie de reuniones donde participan usuarios finales y expertos en usabilidad. En las reuniones se discuten los problemas y las preocupaciones de los usuarios, relacionados con la interfaz de usuario [138]. Según el estudio empírico realizado por Çetin y Göktürk [46], la técnica de evaluación de la usabilidad menos preferida fue *focus groups*. Sólo el 22 % de los proyectos adoptaba esta técnica.

El principal objetivo de la técnica *observación directa* es comprender cómo los usuarios de las aplicaciones realizan sus tareas y más específicamente conocer todas las acciones que realizan durante la ejecución de las tareas. Para ello, los observadores visitan los usuarios representativos en su lugar de trabajo donde realizan las actividades objeto de estudio y donde serán observados [138]. Aunque la observación directa consume mucho tiempo, desempeña un rol importante en la definición de la estrategia del producto [138]. Esta técnica ha sido adoptada en los proyectos OpenOffice [132], GIMP (aplicación para manipulación de imágenes) y TV-Browser (guía de TV) [165].

5.5.2. Técnicas de la IPO Adoptadas con Ligeras Modificaciones

En esta sección describimos las técnicas de usabilidad adoptadas con ligeras modificaciones en los desarrollos OSS. La descripción de cada técnica está agrupada de acuerdo a la actividad IS con la que está relacionada (es decir, ingeniería de requisitos y evaluación). La comunidad OSS no ha adoptado ninguna técnica de la IPO con ligeras modificaciones en la actividad de diseño.

5.5.2.1. Técnicas de Usabilidad Relacionadas con Ingeniería de Requisitos

Observando la Tabla F.1 (Anexo F), las técnicas de usabilidad que OSS ha adoptado en las actividades de ingeniería de requisitos son: *Personas* [38][46][139][188] y *tormenta de ideas visual* [188].

La técnica *Personas* consiste en representaciones de los usuarios finales que ayudan a guiar el diseño de las aplicaciones, teniendo en mente siempre al usuario y evitando que los desarrolladores desarrollen software para ellos mismos [52]. Las *Personas* han sido creadas con la participación tanto de expertos en usabilidad como de la comunidad OSS. Esta técnica no se ha llevado a cabo como prescribe la IPO porque la creación de *Personas* se realizó con base en las descripciones que la propia comunidad OSS suministró y no a través de una serie de entrevistas presenciales a un grupo pequeño de usuarios [38][46][139][188]. Las personas fueron creadas a partir de descripciones libres, es decir, descripciones que no seguían ningún tipo de estructura establecida con anterioridad.

La técnica *tormenta de ideas visual* ha sido adoptada con ligeras modificaciones en el desarrollo de una aplicación de gráficos de mapa de bits [188]. La adaptación realizada en esta técnica consistió en que las ideas fueron recolectadas a través de una wiki y no en reuniones presenciales como establece la IPO. Gracias a la wiki cualquier involucrado en el proyecto podía aportar sus ideas para el diseño de la interfaz.

5.5.2.2. Técnicas de Usabilidad Relacionadas con Evaluación

Las técnicas de evaluación de la usabilidad del grupo de **evaluación por expertos** adoptadas por OSS con ligeras modificaciones son: *inspección de conformidad con estándares* [46], *evaluación heurística* [152], *inspecciones* [23] y *recorridos cognitivos* [152][188].

La técnica *inspección de conformidad con estándares* recibe el nombre de *bug hunting seasons* en la comunidad OSS. Las *bug hunting seasons* consisten en la búsqueda, por parte de usuarios no-desarrolladores, de todas las infracciones evidentes a las HIGs. A los usuarios se les pide que encuentren cualquier aspecto que pueda afectar la usabilidad, como por ejemplo, falta de retroalimentación en determinadas acciones, barras de herramientas, menús o cuadros de diálogo de configuración sobrecargados. Los usuarios reportan, en un sistema de seguimiento de errores, los errores de usabilidad que ellos encuentran. Estos errores son etiquetados con HIG para que los desarrolladores puedan posteriormente buscarlos y corregirlos durante el ciclo de revisión que se realiza posteriormente [46]. La adaptación con respecto a la técnica *inspección de conformidad con estándares* es que en OSS es llevada a cabo por usuarios finales voluntarios en lugar de expertos en usabilidad como lo prescribe la IPO [50][120][149].

La técnica *evaluación heurística* ha sido adoptada en el proyecto Roguelike (juego de rol). Esta técnica fue adaptada, y un grupo de estudiantes de la IPO dirigidos por un experto en usabilidad desempeñó el rol del experto [152].

Según el estudio empírico realizado por Andreasen et al. [23], el 42 % de los desarrolladores encuestados afirmó que usaban *inspecciones*, pero que éstas rara vez se llevaban a cabo por profesionales de la usabilidad. Andreasen et al. [23] no especifican las variantes de la técnica *inspecciones* que fueron adoptadas.

La técnica *recorridos cognitivos* ha sido adoptada por diferentes proyectos OSS (por ejemplo, un sistema operativo y una utilidad de escritorio). La adaptación de esta técnica consiste en que es el desarrollador y no un experto en usabilidad quien realiza la inspección. El desarrollador inspecciona el sistema y la funcionalidad, mientras que el experto explora el software de acuerdo a una serie de datos y acciones previamente definidas. Los desarrolladores descubren los problemas de usabilidad mientras desarrollan o mejoran al software [188]. Con el objetivo de entender qué hace el software y cómo lo hace, los desarrolladores exploran y prueban el sistema existente para detectar problemas. En el proyecto Roguelike, esta técnica también fue adoptada con ligeras modificaciones. Pero en este caso, la modificación consistió en que el recorrido cognitivo se llevó a cabo por un grupo de estudiantes de la IPO dirigido por un experto en usabilidad.

En cuanto a las técnicas para la evaluación de la usabilidad del grupo *test de usabilidad*, la comunidad OSS ha adoptado: *grabación de video* [152], *grabación de audio* [152], *información post-test* [152], *pensar en voz alta* [23] y *test de usabilidad en laboratorio* [152]. A continuación, describiremos las técnicas adoptadas con ligeras modificaciones.

Grabación de video, *grabación de audio* e *información post-test* son técnicas para recopilar datos durante o al final de los test de usabilidad [50][92]. Todas estas técnicas han sido adoptadas con ligeras modificaciones en el proyecto Roguelike. La modificación consistió en

que la técnica fue aplicada por un grupo de estudiantes dirigido por un experto en usabilidad [152] en lugar de por un experto en usabilidad como lo prescribe la comunidad de la IPO.

De acuerdo al estudio empírico realizado por Andreasen et al. [23], el 21 % de los desarrolladores OSS encuestados ha usado la técnica *pensar en voz alta*. La adaptación realizada en esta técnica consiste en que es un desarrollador y no un experto en usabilidad quien está presente con el usuario mientras interacciona con la aplicación. Tanto el desarrollador como el usuario se comunican de forma remota [23].

La técnica *test de usabilidad en laboratorio* ha sido adoptada en el proyecto Roguelike. La modificación consistió en que la técnica fue aplicada por estudiantes de la IPO dirigidos por un experto en usabilidad [152] y no por expertos en usabilidad como prescribe la comunidad de la IPO.

Finalmente, de las técnicas de evaluación de la usabilidad del grupo de *estudios de seguimientos de sistemas instalados* (ver Tabla F.3 en el Anexo F), la comunidad OSS ha adoptado: *cuestionarios y encuestas* [188], *entrevistas* [188], *focus groups* [188], *observación directa* [188], *foros* [188], *retroalimentación del usuario* [89][137][152][188] y *revistas y conferencias para usuarios* [35][188].

La técnica *cuestionarios y encuestas* ha sido adoptada en varios proyectos OSS (por ejemplo, una aplicación de edición, un editor de fuentes, un navegador web). En la adaptación los usuarios no fueron encuestados porque, al ser más proactivos, tendían a proponer ideas sin necesidad de ser preguntados [188].

La técnica *entrevistas* ha sido adoptada en una aplicación de gráficos de mapa de bits y en una aplicación de edición. La técnica *entrevistas* ha sido adoptada de manera similar a la técnica *cuestionarios y encuestas*. Los usuarios no fueron estrictamente hablando entrevistados porque voluntariamente ofrecieron sus opiniones en los tutoriales o durante las capacitaciones de la aplicación [188].

La técnica *focus groups* ha sido adoptada en varios proyectos OSS (por ejemplo, en un entorno de ventanas de escritorio y en un sistema operativo de escritorio). En la técnica *focus groups*, un experto en usabilidad se reúne en persona o a través de Internet Relay Chat (IRC) con los desarrolladores. Estas reuniones se celebran con cierta periodicidad (semanal, mensual o anualmente) y tienen como objetivo que las aplicaciones o los diseños propuestos para una nueva funcionalidad sean valorados por un experto en usabilidad [188]. En este caso, son los desarrolladores OSS los que participan en los *focus groups* y no los usuarios finales.

Las *observaciones directas* han sido realizadas de manera informal cuando familiares y amigos de los desarrolladores usan la aplicación, o cuando usuarios avanzados realizan demostraciones en conferencias. En estas observaciones no hay un objeto de estudio previamente definido [188]. Esta técnica ha sido adoptada en varios proyectos OSS, incluyendo una aplicación de gráficos de mapa de bits y una herramienta de escritorio.

Las técnicas *foros* [188], *retroalimentación del usuario* [89][137][152][188] y *revistas y conferencias para usuarios* [35][188] han sido adoptadas con ligeras modificaciones. Los *foros* son usados por los usuarios para publicar mensajes abiertos y preguntas a los diseñadores de la interfaz [180]. En OSS, estos mensajes y preguntas no son solo publicados en foros, sino también son comunicados a través de IRC y listas de correo. Los usuarios no desarrolladores usan estos medios de comunicación para participar en un diálogo permanente y directo con los desarrolladores sobre sus necesidades y los problemas de usabilidad. Los *foros* constituyen,

dentro del desarrollo OSS, el principal medio para descubrir y tratar problemas de usabilidad [188].

La técnica *retroalimentación del usuario* ha sido adoptada en tres formas diferentes. En la primera, la retroalimentación es solicitada de forma explícita a los usuarios. Esta solicitud es realizada a los usuarios más cercanos al proyecto, es decir, aquellos que tienen un contacto frecuente con los desarrolladores. Estos usuarios prueban las versiones del software antes de que la versión final sea liberada. De esta forma, los principales problemas de usabilidad pueden ser detectados y corregidos antes de que el software llegue a otros usuarios cuando se libere la versión final. Esta técnica es uno de los métodos más importantes para descubrir y tratar los problemas de usabilidad en el desarrollo OSS [188]. En los desarrollos OSS esta técnica ha sido diseñada para que los desarrolladores y los usuarios finales interactúen directamente durante el desarrollo de software. En la segunda, la retroalimentación es realizada a través del reporte de errores. En el desarrollo OSS está muy extendido el uso del reporte de errores. Las herramientas para el reporte de errores son útiles no solo para reportar los problemas de usabilidad, sino también para discutirlos [89][137][188]. En la tercera, la retroalimentación se realiza a través de una wiki. Los desarrolladores usan la wiki para comentar los problemas de usabilidad detectados en los test de usabilidad ejecutados previamente. La wiki abrió un diálogo entre los expertos en usabilidad y los desarrolladores [152].

La técnica *revistas y conferencias para usuarios* es utilizada en proyectos con un número considerable de usuarios dispersos geográficamente. Las conferencias son un lugar donde los trabajadores pueden intercambiar experiencias con sus colegas y las reuniones presenciales aumentan el sentido de comunidad entre los usuarios [180]. Esta técnica ha sido adoptada con ligeras modificaciones en OSS. Los usuarios no son sólo participantes pasivos en las conferencias, ellos también enseñan a otros participantes del evento [188].

5.5.3. Técnicas de la IPO Reinterpretadas

Las técnicas reinterpretadas por la comunidad OSS son: *design-by-blog* [137][144][188], *open content projects* [188], *seasons of usability* [152][188], *discutir en comunidad* [35][137][188] y *design workshops* [35][188]. A continuación, describimos estas técnicas.

En las actividades de ingeniería de requisitos, ha sido adoptada la técnica *design-by-blog* [137][144][188]. Usando esta técnica, cualquier miembro de la comunidad OSS (incluidos los usuarios no-desarrolladores) puede analizar y discutir los diseños de las interfaces de usuario y de las nuevas funcionalidades a través de blogs. Un blog orientado al diseño proporciona una descripción de los problemas, objetivos y las restricciones o los criterios de diseño, las capturas de pantalla del diseño propuesto y los diferentes comentarios sobre el diseño [137][188]. El objetivo del *design-by-blog* es resolver un problema particular, descrito en un blog junto con una idea inicial de cómo resolverlo. Los comentarios de quienes participan en el blog permiten explorar otras ideas, soluciones alternativas o variaciones de la solución inicial, manteniendo cierta consistencia con el diseño.

El objetivo del *design-by-blog* y el método utilizado para cumplir tal objetivo rememora la técnica de la IPO *tormenta de ideas visual*. La *tormenta de ideas visual* permite explorar y validar diseños alternativos en pequeñas reuniones. Una vez creados los diseños iniciales, las mejores ideas pueden ser desarrolladas construyendo representaciones en cartulinas del diseño. Estos diseños pueden ser evaluados con los usuarios [149]. Ambas técnicas son esencialmente la misma, comparten el mismo espíritu: los usuarios expresan sus opiniones sobre las nuevas ideas de diseño generadas sobre la base de un diseño inicial. El medio utilizado en la *tormenta de ideas visual* para representar el diseño (cartulinas) es reemplazado en la técnica *design-by-blog* por blogs. El uso de estos blogs permite que cualquiera pueda dar su opinión, manifestando

el principio OSS: discutir en comunidad. Es precisamente la manera en que se aplica este principio el disfraz de la técnica.

Las técnicas adoptadas en las actividades de evaluación son: *open content projects* [188], *seasons of usability* [152][188], *discutir en comunidad* [35][137][188] y *design workshops* [35][188]. La técnica *open content projects* surge en el proyecto 3DA (un visualizador 3D), y consiste en la construcción de un producto creativo, como por ejemplo un cortometraje o un videojuego. Esta técnica es realizada con cierta regularidad y es financiada gracias a donaciones, subvenciones y pre-órdenes de compra de los contenidos que serán producidos. Los participantes en los *open content projects* tienen una doble motivación. Por un lado, tienen un objetivo claro y bien definido y, por otro lado, el producto final será mostrado y apreciado por el público en general. Los desarrolladores y los usuarios finales (artistas) participan en la construcción de un producto creativo, trabajando juntos para conseguir un objetivo común [188].

La técnica *open content projects* es aplicada con el objetivo de descubrir problemas y mejorar la usabilidad. Los problemas de usabilidad son descubiertos cuando la aplicación es usada de manera conjunta por los desarrolladores y los usuarios finales. El objetivo de esta técnica y cómo lo consigue rememora a la técnica de la IPO *inspecciones de usabilidad colaborativas*. Las *inspecciones de usabilidad colaborativas* consisten de un proceso de revisión de la usabilidad de un producto finalizado o prototipo desde el punto de vista de los usuarios finales. El proceso de revisión es realizado por un equipo de trabajo que incluye desarrolladores, usuarios finales y expertos en usabilidad [50].

El objetivo principal de las *inspecciones de usabilidad colaborativas* es mejorar la usabilidad e identificar los problemas de usabilidad de la aplicación que está siendo revisada. Las inspecciones son realizadas en dos fases. En la primera (inspección interactiva), la aplicación es usada siguiendo escenarios de uso previamente establecidos, preparados y documentados. En la segunda (inspección estática), se inspeccionan todos los componentes de la interfaz de usuario -pantallas, menús, o cajas de diálogo- [50].

Si bien, las *inspecciones de usabilidad colaborativas* se realizan de una manera estructurada, con tareas previamente definidas, tienen la misma esencia que los *open content projects*, donde un equipo de trabajo usa la aplicación pero de una manera más libre, siguiendo el principio de libertad de la comunidad OSS. Considerando solo la esencia de ambas técnicas, podemos decir que son la misma salvo por la manera en que se aplica este principio OSS, que a primera vista hace que parezcan técnicas totalmente diferentes.

Las *seasons of usability* son una serie de proyectos, por lo general, patrocinados para animar a los estudiantes de usabilidad y de diseño de la interacción que se involucren en proyectos OSS [188]. Sin embargo, en algunos casos estos proyectos no cuentan con el apoyo de la empresa, como en el caso del proyecto Roguelike [152]. Durante un periodo de tres a seis meses o incluso dos años, los estudiantes trabajan en colaboración con un mentor experto en usabilidad y con los desarrolladores principales del proyecto OSS con el objetivo de mejorar la usabilidad [188]. La esencia de esta técnica es precisamente mejorar la usabilidad gracias a la participación de personas con conocimientos en usabilidad, rememorando a las técnicas de la IPO del tipo *evaluación por expertos*. El disfraz lo constituyen quienes participan en el rol de expertos en usabilidad (estudiantes de la IPO y su mentor) y en la naturaleza voluntaria de esta participación. Esta técnica promulga el principio OSS de la participación voluntaria.

En el desarrollo OSS, los problemas de usabilidad se *discuten en comunidad* a través de herramientas para el reporte de errores (por ejemplo, Bugzilla) [137] y medios de comunicación

basados en internet, como IRC y listas de correo [188]. La esencia de esta técnica es discutir en comunidad las necesidades de los usuarios no-desarrolladores, así como también aspectos que afectan negativamente la usabilidad de las aplicaciones. Los *focus groups* tienen la misma esencia. En esta técnica de la IPO, se reúne presencialmente un grupo de entre seis y nueve usuarios para discutir nuevos conceptos e identificar temas relevantes para la usabilidad de la aplicación. Estas reuniones tienen una duración de unas dos horas y son llevadas por un moderador que es responsable de mantener el enfoque de grupo [138]. Esta técnica OSS está *disfrazada* por la manera en que se llevan a cabo las discusiones. Mientras que en los *focus groups* las reuniones son presenciales, en OSS son reuniones virtuales y en algunos casos las discusiones son asíncronas. Además, el número de participantes en estas discusiones no está limitado. El principio OSS detrás de este *disfraz*, es la democracia -todos participan en las discusiones-.

Algunos proyectos OSS, especialmente aquellos que tienen el apoyo de compañías como Sun Microsystems, celebran conferencias con cierta periodicidad (por ejemplo, anualmente). Dentro de estas conferencias se llevan a cabo reuniones en las que participan expertos en usabilidad y desarrolladores. Estas reuniones son conocidas como *design workshops*. En estas reuniones, los expertos en usabilidad asesoran a los desarrolladores sobre diferentes aspectos de usabilidad, por ejemplo, si es conveniente o no adicionar ciertas características a la aplicación [35][188].

Esta técnica nos recuerda de nuevo a los *focus groups*. Si bien en los *focus groups* se reúnen usuarios y en los *design workshops* participan desarrolladores, la esencia es la misma: discutir sobre aspectos relacionados con la usabilidad. Los *design workshops* están disfrazados por la forma en que se llevan a cabo. Estas reuniones se realizan como parte de un evento más grande (una conferencia) proporcionando la oportunidad de discutir en comunidad diferentes aspectos, entre ellos la usabilidad. Esta técnica es de nuevo disfrazada por el principio OSS de discutir en comunidad.

5.6. Técnicas de la IPO No Adoptadas en los Desarrollos OSS

En esta sección analizaremos las técnicas de usabilidad que no han sido adoptadas de ninguna manera (es decir, ni puras, ni con ligeras modificaciones, ni reinterpretadas). Este análisis es realizado en tres partes, una por cada uno de los tipos de actividades IS (ingeniería de requisitos, diseño y evaluación).

La Tabla 5.8 presenta el listado de técnicas de usabilidad no adoptadas en OSS relacionadas con la ingeniería de requisitos. Como se observa en la Tabla 5.8, los proyectos OSS no han adoptado de ninguna manera el 78,79 % de las técnicas de usabilidad relacionadas con la ingeniería de requisitos: *análisis de impacto financiero*, *investigación contextual*, *diagramas de afinidad*, *observación etnográfica*, *JEM (Joint Essential Modeling)*, *card sorting*, *mapa de roles de usuario*, *modelo operacional*, *casos de uso esenciales*, *HTA (Hierarchical Task Analysis)*, *familia de modelos GOMS*, *modelo de interfaz objeto-acción*, *escenarios de tareas*, *task sorting*, *escenarios y storyboards*, *prototipado*, *prototipos escenario*, *prototipos activos*, *prototipos guiados*, *prototipos mago de Oz*, *evaluación heurística*, *inspecciones de conformidad con estandares*, *revisión de guías*, *inspecciones de consistencia*, *inspecciones de usabilidad colaborativas*, *recorridos cognitivos* y *recorridos pluralístico*.

Las técnicas *análisis de impacto financiero*, *investigación contextual*, *diagramas de afinidad*, *observación etnográfica* y *JEM* tienen por objetivo especificar el contexto de uso, es decir, comprender y registrar las características del contexto previsto de uso del software en cuanto éstas puedan ser relevantes para la usabilidad del producto software final [68]. Del grupo

de técnicas para la especificación del contexto de uso, solo un proyecto OSS ha adoptado la técnica *análisis competitivo*, con lo cual es necesario que la comunidad OSS adopte más técnicas de este grupo. No adoptar la técnica *card sorting* no es crítico porque es una técnica complementaria en la ingeniería de requisitos.

Tabla 5.8: Técnicas de la IPO No Adoptadas en OSS Relacionadas con la Ingeniería de Requisitos.

Tipo de Actividad IS		**Técnicas No Adoptadas en OSS**
Educción y Análisis de Requisitos		Análisis de Impacto Financiero
		Investigación Contextual
		Diagramas de Afinidad
		Observación Etnográfica
		JEM (Joint Essential Modeling)
		Card Sorting
	Análisis de Usuarios	Mapa Roles de Usuario
		Modelo Operacional
	Análisis de Tareas	Casos de Uso Esenciales
		HTA (Hierarchical Task Analysis)
		Familia de Modelos GOMS (Goals, Operations, Methods and Selection Rules)
		Modelo de Interfaz Objeto-Acción
		Escenarios de Tareas
		Task Sorting
	Desarrollo del Concepto del Producto	Escenarios y Storyboards
	Prototipado	Prototipado
		Prototipos Escenario
		Prototipos Activos
		Prototipos Guiados
		Prototipos Mago de Oz
Validación de Requisitos	Evaluación Heurística	Evaluación Heurística
	Inspecciones	*Inspecciones de Conformidad con Estándares*
		Revisión de Guías
		Inspecciones de Consistencia
		Inspecciones de Usabilidad Colaborativas
	Recorridos Cognitivos	Recorridos Cognitivos
	Recorrido Pluralístico	Recorrido Pluralístico

Las técnicas *mapa de roles de usuario* y *modelo operacional* permiten realizar el análisis de usuarios. Aunque OSS ha adoptado otras técnicas para el análisis de usuarios, como *perfiles de usuario* y *Personas*, resultan insuficientes, sobre todo teniendo en cuenta que la adopción de *Personas* no fue exitosa en todos los casos, con lo cual hay un esfuerzo pendiente que realizar en este sentido.

Las técnicas *casos de uso esenciales*, *HTA*, *familia de modelos GOMS*, *modelo de interfaz objeto-acción*, *escenarios de tareas* y *task sorting*, son útiles para el análisis de tareas. La comunidad OSS no ha adoptado ninguna de estas técnicas para el análisis de tareas. Esto es crítico porque hay un alto riesgo de que las tareas que los usuarios necesitan realizar con el sistema no sean detectadas.

La técnica *escenarios y storyboards* es usada para desarrollar el concepto del producto, es decir, considerar las reglas generales que gobernarán el funcionamiento de las aplicaciones, sus espacios de interacción principales y cómo se trabajarán con las mismas [68]. Aunque la técnica *tormenta de ideas visual*, que es otra técnica usada para desarrollar el concepto del

producto, ha sido adoptada, su uso en OSS se ha limitado a recopilar ideas para el diseño de las interfaces de usuario. Por tanto, es crítico la no adopción de la técnica *escenarios y storyboards* porque es muy probable que el usuario no-desarrollador no sea capaz de comprender la lógica de la aplicación, debido a que no existen reglas generales que gobiernen su funcionamiento.

La técnica de *prototipado* y sus variantes (*prototipos escenario*, *prototipos activos*, *prototipos guiados* y *prototipos mago de Oz*) son usadas para educir información de los usuarios sobre las funcionalidades necesarias del sistema. OSS no ha adoptado ninguna de las técnicas anteriores, pero ha aplicado la técnica *prototipos de papel*, que cumple el mismo objetivo que las anteriores. Por este motivo, no resulta crítico que sólo adopte una de las técnicas de prototipado.

Las técnicas de usabilidad relacionadas con diseño no adoptadas en OSS aparecen listadas en la Tabla 5.9. Los proyectos OSS no adoptan de ninguna manera el 81,25 % del grupo de técnicas de usabilidad relacionadas con diseño: *representaciones de pantallas*, *gramáticas*, *UAN (User Action Notation)*, *TAG (Task-Action Grammars)*, *árboles de menús*, *diagramas de transición de estados de la interfaz*, *diagramas de estados de Harel*, *modelo del contenido de la interfaz*, *mapa de navegación*, *diseño integrador*, *diseño paralelo*, *análisis de impacto*, *organización de la ayuda según casos de uso.*

Tabla 5.9: Técnicas de la IPO No Adoptadas en OSS Relacionadas con el Diseño.

Tipo de Actividad IS		**Técnicas No Adoptadas en OSS**
Diseño	Diseño de la Interacción	Representaciones de Pantallas
		Gramáticas
		UAN (User Action Notation)
		TAG (Task-Action Grammars)
		Árboles de Menús
		Diagramas de Transición de Estados de la Interfaz
		Diagramas de Estados de Harel
		Modelo del Contenido de la Interfaz
		Mapa de Navegación
		Diseño Integrador
		Diseño Paralelo
		Análisis de Impacto
		Organización de la Ayuda según Casos de Uso

Las técnicas *representaciones de pantalla*, *gramáticas*, *UAN*, *TAG*, *árboles de menús*, *diagramas de transición de estados de la interfaz*, *diagramas de estados de Harel*, *modelo del contenido de la interfaz* y *mapa de navegación* son usadas para el diseño de la interfaz, y OSS no ha adoptado ninguna de éstas. Sin embargo, la no adopción de estas técnicas no es crítica porque OSS ha adoptado las *guías de estilo del producto* y las *especificaciones de usabilidad*. Si bien esta última técnica está relacionada con las actividades de la ingeniería de requisitos, OSS ha utilizado esta técnica para describir la interacción detallada de las nuevas funcionalidades.

En cuanto a las técnicas: *diseño integrador*, *diseño paralelo*, *análisis de impacto*, *organización de la ayuda según casos de uso*, ninguna de éstas ha sido adoptada por OSS. El hecho de que ninguna de las técnicas anteriores haya sido adoptada no resulta crítico porque son técnicas complementarias que se refieren a cómo las decisiones de diseño son tomadas.

Las técnicas de usabilidad relacionadas con la evaluación no adoptadas en los desarrollos OSS están listadas en la Tabla 5.10. Del grupo de técnicas de evaluación por expertos OSS no ha adoptado (ni puras, ni ligeramente modificadas, ni reinterpretadas) el 28,57 %: *revisión de guías* y *recorrido pluralístico*. Esto no es crítico porque OSS ha adoptado suficientes técnicas tanto puras como con ligeras modificaciones (ver Tablas 5.4 y 5.6).

Tabla 5.10: Técnicas de la IPO No Adoptadas en OSS Relacionadas con la Evaluación.

Tipo de Actividad IS	**Técnicas No Adoptadas en OSS**
Evaluación por Expertos	*Revisión de Guías*
	Recorrido Pluralístico
Test de Usabilidad	*Interacción Constructiva*
	Test Retrospectivo
	Toma de Incidentes Críticos
	Método de Entrenamiento
	Medición de Rendimiento
	Test de Campo
	Registro del Uso
	Registro de Pulsaciones en el Tiempo
	Registro de la Interacción
	Test Remoto por Video-Conferencia
Estudios de Seguimiento de Sistemas Instalados	*Observación Aleatoria*
	Entrevistas Estructuras
	Entrevistas Flexibles
	Registro de Uso
	Registro de Pulsaciones en el Tiempo
	Registro de la Interacción
	Monitores Software
	Servicios de Atención al Usuario en Línea
	Evaluación Remota Semi-Instrumentada

Los proyectos OSS no han adoptado de ninguna manera el 62,5 % de las técnicas para la evaluación de la usabilidad del tipo test de usabilidad: *interacción constructiva*, *test retrospectivo*, *toma de incidentes críticos*, *método de entrenamiento*, *medición del rendimiento*, *test de campo*, *registro del uso*, *registro de pulsaciones en el tiempo*, *registro de la interacción* y *test remoto por videoconferencia*. La no incorporación de estas técnicas no es crítico porque son complementarias y porque OSS ha adoptado técnicas para el mismo propósito.

En cuanto a las técnicas para la evaluación de la usabilidad del tipo estudios de seguimiento de sistemas instalados, OSS no ha adoptado de ninguna manera el 56,25 %: *observación aleatoria*, *entrevistas estructuradas*, *entrevistas flexibles*, *registro de uso*, *registro de pulsaciones en el tiempo*, *registro de la interacción*, *monitores software*, *servicios de atención al usuario en línea* y *evaluación remota semi-instrumentada*. La no adopción de estas técnicas no resulta crítico porque OSS ha adoptado otras para el mismo propósito (ver Tablas 5.4 y 5.6).

5.7. Discusión

Existen varios obstáculos para que la comunidad OSS se ocupe de la usabilidad, que era hasta hace unos pocos años un tema descuidado. Sin embargo, gracias a que la comunidad se hace gradualmente más consciente de su importancia y al apoyo de algunas compañías, están empezando a adoptar principios básicos y técnicas de la IPO en su proceso de desarrollo. La adopción de algunas de estas técnicas ha ayudado a la comunidad OSS a resolver algunos de los problemas de usabilidad que enfrenta. Por ejemplo, (i) algunos proyectos han adoptado las

técnicas *perfiles de usuario* y *Personas* para ayudar a los desarrolladores a determinar el perfil de los usuarios no-desarrolladores de sus aplicaciones, y (ii) la comunidad ha sacado partido de sus usuarios no-desarrolladores para que realicen labores de inspección de las interfaces de usuario porque hay pocos expertos en usabilidad como voluntarios en los proyectos OSS.

La mayoría de las técnicas de la IPO que la comunidad OSS está adoptando son técnicas para la evaluación de la usabilidad. Creemos que hay dos razones para esto. En primer lugar, una versión de la aplicación sobre la cual evaluar la usabilidad está disponible durante todo el proceso de desarrollo OSS. En segundo lugar, las técnicas de evaluación de la usabilidad son menos intrusivas que las técnicas relacionadas con ingeniería de requisitos o diseño, es decir, son técnicas que pueden ser aplicadas por terceros (por ejemplo, expertos en usabilidad, usuarios) y no tienen un gran impacto en la rutina de trabajo de los desarrolladores OSS.

En conjunto, la comunidad OSS ha adoptado un poco más del 50 % de las técnicas de la IPO relacionadas con la evaluación. Sin embargo, solo cerca del 20 % de las técnicas de usabilidad relacionadas con las actividades de la ingeniería de requisitos y diseño han sido adoptadas. Los problemas de la comunidad relacionados con la comprensión de los usuarios y sus expectativas son el resultado directo de esta escasez de técnicas. Por consiguiente, se requiere de una mayor investigación para apoyar la adopción de técnicas relacionadas con la ingeniería de requisitos (ver Tabla F.1 del Anexo F) en los desarrollos OSS.

Hay tres tipos diferentes de técnicas para la evaluación de la usabilidad: *evaluación por expertos*, *test de usabilidad* y *estudios de seguimiento de sistemas instalados*. La comunidad OSS ha reinterpretado el rol del experto en algunas de las técnicas de evaluación por expertos (por ejemplo, inspección de conformidad con estándares). En la versión de la comunidad OSS, los usuarios no-desarrolladores desempeñan el rol de experto en usabilidad, ayudados por un documento que describe las reglas que debe seguir la interfaz de usuario (HIGs).

Las principales adaptaciones realizadas por OSS a las técnicas de la IPO son cuatro. En primer lugar, el rol de experto en usabilidad es realizado por otra persona (por ejemplo, un desarrollador). En segundo lugar, los participantes en la técnica al no poder estar físicamente reunidos durante la ejecución de la técnica han mantenido contacto a través de artefactos web (por ejemplo, IRC, blogs) o exponen sus ideas a través de una wiki y no en reuniones presenciales. En tercer lugar, la retroalimentación del usuario no se realiza solamente a través de herramientas online para el reporte de errores, sino se utilizan también chats, foros de discusión y blogs. En cuarto lugar, se ha permitido la participación de cualquier miembro de la comunidad durante la ejecución de la técnica.

Los proyectos OSS han adoptado algunas técnicas de la IPO (por ejemplo, pensar en voz alta, observación directa) tanto puras, como con ligeras modificaciones. Además, otras técnicas, como por ejemplo focus groups, han sido adoptadas en todas las tres formas discutidas (puras, con ligeras modificaciones y reinterpretadas). La comunidad OSS ha conseguido incluso reinterpretar la misma técnica de diferentes maneras, como es el caso de nuevo de la técnica focus group. Lo que demuestra la versatilidad de la comunidad OSS en cuanto a adaptar las técnicas a su modo de desarrollo.

La mayoría de las técnicas adoptadas de manera satisfactoria por la comunidad OSS son técnicas adoptadas puras -87,5 %, 50 % y 66,67 % de resultados satisfactorios en técnicas relacionadas con ingeniería de requisitos, diseño y evaluación respectivamente- o reinterpretadas -66,67 % y 76,67 % de casos satisfactorios en técnicas relacionadas con ingeniería de requisitos y evaluación respectivamente-. En términos generales, las técnicas de la IPO han sido satisfactoriamente adoptadas en proyectos OSS. Las técnicas de la IPO

reinterpretadas fueron satisfactoriamente adoptadas por los proyectos OSS y funcionaban bien. Por tanto, la comunidad OSS está reinterpretando con éxito técnicas de la IPO.

La adopción de las técnicas de la IPO son insatisfactorias cuando la participación de las comunidad OSS es pobre y los desarrolladores OSS no adoptan las heurísticas definidas para el diseño de las interfaces de usuario. También las herramientas para el reporte de errores no son muy adecuadas para reportar errores de usabilidad, y las discusiones de usabilidad están fragmentadas a través de las listas de correo y reportes de error.

La comunidad OSS no ha adoptado alrededor del 70 % de todas las técnicas de la IPO (de acuerdo al catálogo del Anexo F). Creemos que la falta de adopción de estas técnicas puede deberse, por una parte, al hecho de que los desarrolladores no están familiarizados con las técnicas y, por el otro lado, que la adopción significaría cambiar el método de desarrollo OSS. La situación es aún más complicada en el caso particular de las técnicas relacionadas con diseño, porque las herramientas usadas en el desarrollo de proyectos OSS están en su mayoría basadas en texto (por ejemplo, IRC, foros de discusión) que es un obstáculo para el diseño de la interacción del usuario.

Las técnicas de la IPO adoptadas en los proyectos de desarrollo OSS han sido adoptadas puras en su mayoría. Por tanto, las técnicas de la IPO ideadas para desarrollos tradicionales están siendo adoptadas por la comunidad OSS porque muchos de los proyectos OSS identificados en la literatura por el SMS tienen el apoyo de las empresas que proporcionan los recursos necesarios (por ejemplo, expertos en usabilidad, laboratorios de usabilidad) para aplicar las técnicas según lo prescrito por la comunidad de la IPO. Sin embargo, esto generalmente no ocurre, es decir, pocos proyectos OSS están soportados por empresas. Por tanto, nuestro estudio posiblemente ofrece una visión más optimista que un proyecto OSS promedio, aunque es factible gracias a las técnicas adaptadas.

Casi todos los proyectos OSS identificados en la literatura por el SMS han adoptado al menos una técnica de evaluación de la usabilidad en su proceso de desarrollo. Algunos proyectos han adoptado más de nueve técnicas. Sin embargo, son pocos los proyectos que han adoptado técnicas relacionadas con los tres tipos de actividades de la IS: ingeniería de requisitos, diseño y evaluación.

La comunidad OSS desarrolla software siguiendo su propia filosofía alejándose de la manera tradicional de desarrollar software que establece la IS y está haciendo lo mismo con la usabilidad. En otras palabras, la comunidad OSS está tomando conciencia de que la usabilidad es importante y está comenzando a adoptar técnicas de usabilidad, que, sin embargo, adaptan a su cultura (por ejemplo, discutir en comunidad los diferentes diseños alternativos de la interfaz de usuario para una nueva funcionalidad) dando a las técnicas un nuevo giro. Esto es interesante porque las ideas generadas por la comunidad OSS pueden complementar el rol de los expertos en usabilidad, una gran ayuda de la que los desarrollos comerciales pueden beneficiarse. Considerando que lo que hace un experto en el desarrollo de software tradicional es dar su opinión, en lugar de ser una sola (aunque sea de calidad experta) en la versión OSS hay muchas personas opinando, trabajando colaborativamente y con mucho interés y motivación, porque ellos serán directamente los más beneficiados.

5.8. Conclusiones

Este capítulo presenta nuestra investigación sobre el estado de la consideración de la usabilidad en los desarrollos OSS. Examinamos las razones por las que los usuarios novatos o no-desarrolladores no encuentran las aplicaciones OSS muy usables. Muchas de estas razones

reportadas en la literatura están relacionadas con las características del método de desarrollo OSS (por ejemplo, los desarrolladores OSS han desarrollado software para su propio uso y no consideran la usabilidad un problema, es difícil obtener una muestra representativa de usuarios en entornos distribuidos) y algunas otras son debidas al fracaso de la comunidad OSS al aplicar técnicas específicas para el diseño de aplicaciones usables (es decir, técnicas de la IPO).

A medida que los usuarios de la comunidad OSS han crecido más allá de los desarrolladores de software, se ha incrementado el interés por la usabilidad. Ahora la comunidad está empezando a considerar la usabilidad un aspecto importante y gracias al apoyo de algunas compañías está comenzando a adoptar técnicas de usabilidad en sus desarrollos. Algunas técnicas de la IPO se están adoptando puras (sin ninguna adaptación) mientras que otras se han modificado ligeramente y unas pocas tiene adaptaciones importantes o han sido reinterpretadas para adaptarse a la filosofía de trabajo OSS. Algunas técnicas incluso han sido adoptadas puras, con ligeras modificaciones y reinterpretadas (por ejemplo, focus groups). Las técnicas reinterpretadas son técnicas que la comunidad OSS ha remodelado según los principios que rigen su método de desarrollo. A pesar de este esfuerzo, el 70 % de las técnicas de la IPO aún no se utilizan en los desarrollos OSS.

De los tres tipos de adopción de las técnicas de la IPO practicadas por la comunidad OSS, la mayoría de las técnicas han sido adoptadas puras. Por lo tanto, la mayoría de las técnicas de la IPO ideadas para desarrollos tradicionales pueden ser aplicadas en los proyectos OSS: la usabilidad no es incompatible con OSS.

La mayoría de las técnicas de la IPO adoptadas por la comunidad OSS son técnicas relacionadas con la evaluación. Pocos proyectos han adoptado técnicas de la IPO relacionadas con los tres tipos de actividades IS: ingeniería de requisitos, diseño y evaluación. De estos tres tipos de actividades, las técnicas relacionadas con ingeniería de requisitos y diseño son las que menos han sido adoptadas en el proceso de desarrollo OSS. La comunidad OSS no ha adoptado técnicas con ligeras modificaciones ni reinterpretadas relacionadas con la actividad de diseño. La comunidad OSS ha adoptado solo unas pocas técnicas para el análisis de usuarios (una de las cuales resultó ser insatisfactoria) y ninguna para el análisis de tareas. Además, son pocos los proyectos OSS que han adoptado técnicas para las actividades de ingeniería de requisitos. Pensamos que hay dos razones para esto: (i) la comunidad OSS no es consciente de que existen estas técnicas, y (ii) estas técnicas son más difíciles de adoptar porque implican cambios en el método de desarrollo OSS, como, por ejemplo, adicionar el análisis de usuarios o el análisis de tareas al proceso de desarrollo.

Los resultados de nuestra investigación sugieren que las aplicaciones OSS no están condenadas a tener una baja usabilidad porque una vez la comunidad tome conciencia colectiva de la importancia de la usabilidad y actúe en consecuencia, su propia cultura y filosofía de trabajo puede llevar a soluciones originales para mejorar la usabilidad.

CAPÍTULO 6

DETERMINACIÓN DEL USO DE TÉCNICAS DE USABILIDAD EN LOS DESARROLLOS OPEN SOURCE

Este capítulo está dedicado a detallar nuestro marco de integración de técnicas de usabilidad en los desarrollos OSS. Para comprender dicho marco es necesario conocer cuáles son las características de los proyectos OSS, por tal motivo, en primer lugar presentamos tales características. Luego, estudiamos cuáles son las condiciones desfavorables que impiden el uso de técnicas de usabilidad en los desarrollos OSS y analizamos qué tipos y cuáles transformaciones son necesarias para poder facilitar su uso en esta clase de proyectos. Una vez discutidos los impedimentos para incorporar técnicas de la IPO en los desarrollos OSS, estudiamos las técnicas de usabilidad creadas en los proyectos OSS y analizamos porqué estas técnicas en realidad no son nuevas. Posteriormente, analizamos cuáles técnicas de la IPO pueden ser incorporadas en OSS gracias a las transformaciones que estudiaremos en el presente capítulo. Finalmente, discutimos los resultados obtenidos y presentamos las conclusiones de nuestra propuesta de marco de integración de técnicas.

6.1. Caracterización de los Proyectos OSS

Las aplicaciones OSS típicamente son construidas por un grupo de desarrolladores independientes y voluntarios que se encuentran distribuidos por todo el mundo. La comunidad OSS emplea diferentes artefactos Web (por ejemplo, listas de correo, Internet Relay Chat (IRC), repositorios de código fuente y sistemas de reporte de errores) para la comunicación y sincronización de sus prácticas de trabajo. El texto es el principal medio de comunicación en la comunidad, así como el principal objeto de interés (más específicamente, el código fuente). En esta comunidad, los desarrolladores principalmente contribuyen con el desarrollo de nuevas funcionalidades y con la corrección de los errores reportados por los usuarios.

Al comienzo del movimiento OSS, los usuarios de sus aplicaciones eran los mismos desarrolladores que las construían, pero con el paso de los años y la creciente popularidad de este tipo de software el perfil del usuario ha cambiado. En la actualidad, dentro de los usuarios de las aplicaciones OSS se encuentran, básicamente, dos grupos. Por un lado, están los usuarios que tienen algún nivel de conocimiento en informática, tienen experiencia usando software o están muy interesados con todo lo relacionado con las tecnologías. Este grupo

de usuarios está más frecuentemente en contacto con los desarrolladores principales y son quienes suministran la mejor retroalimentación. Por otro lado, están aquellos usuarios cuyo interés principal es usar las aplicaciones, bien porque les ayudan con sus labores diarias o bien porque son sus herramientas de trabajo.

La Figura 6.1 ilustra los dos grupos de usuarios instanciados en el proyecto Linux [188]. Los usuarios identificados como *Core Users* son usuarios del primer grupo, mientras que los usuarios identificados como *Stable Release Users* y *Linux Distribution Users* son usuarios del segundo grupo.

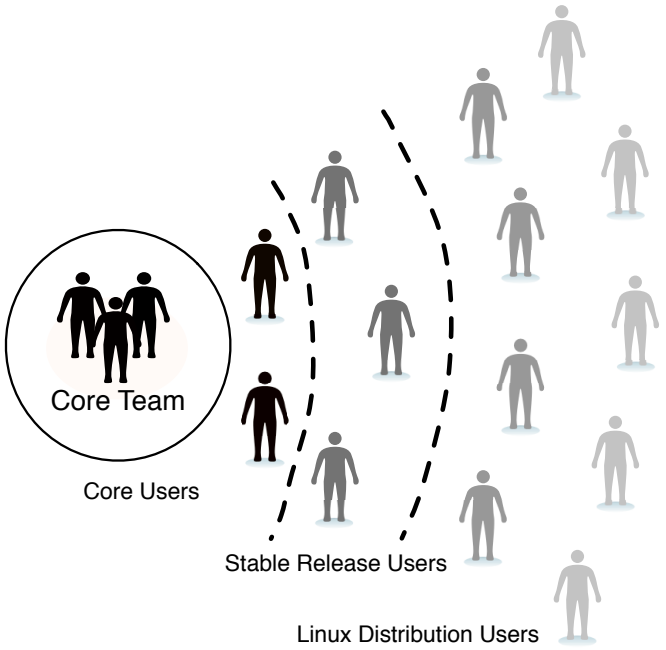


Figura 6.1: Grupos de Usuarios en el Proyecto OSS Linux (adaptada de [188]).

A pesar de que en la comunidad OSS los desarrolladores y los usuarios se encuentran distribuidos geográficamente, existen puntos de encuentro como conferencias o workshops, por lo general patrocinados por compañías. Estos eventos se realizan con el objetivo de lanzar oficialmente nuevas versiones de las aplicaciones y ofrecer tutoriales y talleres donde desarrolladores y usuarios intercambian opiniones. Por ejemplo, en junio del 2013 en Madrid (España) se llevó a cabo una reunión para la presentación de la versión 6.0 de BonitaSoft (una aplicación de gestión de procesos de negocio) donde participaron desarrolladores y usuarios.

Algunos proyectos OSS tienen el patrocinio de compañías, lo que les permite contar con los recursos necesarios (por ejemplo, expertos en usabilidad, laboratorios de usabilidad) para aplicar técnicas de usabilidad según lo prescrito por la IPO. Por ejemplo, las técnicas *especificaciones de usabilidad* [35][132][188] y *recorridos cognitivos* [84] han sido incorporadas en algunos proyectos OSS gracias a que contaban con la participación de expertos en usabilidad. Sin embargo, estos son casos excepcionales porque a menudo los proyectos OSS trabajan con presupuestos pequeños debido a su carácter voluntario, ocasionando que no sea posible tener expertos externos (como diseñadores gráficos o expertos en usabilidad) [35][136][137][159][206].

En la siguiente sección, estudiaremos las condiciones desfavorables (por ejemplo, la falta de participación de expertos en usabilidad), que impiden que muchas de las técnicas de usabilidad sean incorporadas en los desarrollos OSS.

6.2. Impedimentos para Aplicar Técnicas de Usabilidad en OSS

Existen ciertas técnicas de la IPO que pueden ser incorporadas tanto en el desarrollo comercial como en OSS. Determinados proyectos OSS han incorporado, tal cual lo prescribe la IPO, técnicas como: *cuestionarios y encuestas* [165][188] y *prototipos de papel* [46]. Sin embargo, hay técnicas de usabilidad cuyas exigencias no pueden ser satisfechas en la mayoría de los proyectos OSS. Por ejemplo, aquellas técnicas que exigen que los usuarios estén presentes (por ejemplo, *observación directa* [149] y *pensar en voz alta* [50]), puesto que los usuarios en OSS se encuentran distribuidos por todo el mundo. A continuación, identificamos y describimos las condiciones desfavorables que dificultan que muchas técnicas de usabilidad sean incorporadas en proyectos OSS si se sigue estrictamente lo prescrito por la IPO.

Existe un grupo de técnicas de la IPO que **exigen contar con expertos en usabilidad**. En muchos proyectos OSS la participación de expertos en usabilidad es poco común por dos razones. En primer lugar, los proyectos OSS (especialmente aquellos no patrocinados por compañías) trabajan con presupuestos pequeños y el trabajo es realizado por voluntarios. Esto significa que no pueden pagar expertos externos (como diseñadores gráficos o expertos en usabilidad) [35][136][137][159][208]. En segundo lugar, no hay muchos incentivos para que los expertos en usabilidad participen en proyectos OSS. Por ejemplo, el reconocimiento entre pares (uno de los principales incentivos para que los desarrolladores contribuyan en proyectos OSS) es inexistente porque no hay una masa crítica de expertos en usabilidad que trabaje en proyectos OSS [114]. Además, parece que algunos expertos no se quieren involucrar porque no están interesados o motivados por el enfoque OSS de la misma forma en que lo están muchos desarrolladores [136][139]. Esta falta de motivación podría deberse a que el modo de trabajo de los expertos en usabilidad no encaja con el de la comunidad OSS [89][136][114][208]. Ejemplos de algunas técnicas donde es necesario un experto en usabilidad son: *recorridos cognitivos* [180] y *pensar en voz alta* [50]. En los *recorridos cognitivos*, un grupo de expertos en usabilidad mantiene una reunión tipo juicio, con un moderador o juez, para presentar la interfaz de usuario y discutir sus fortalezas y debilidades [180]. La técnica *pensar en voz alta* implica tener un sujeto usando el sistema mientras, de forma continua, describe en voz alta lo que está pensando y mientras expertos en usabilidad observan y monitorizan lo que hace el usuario [50].

Otras técnicas de la IPO **requieren de una preparación previa**, por ejemplo definir una tarea que el usuario debe realizar. Esto supone una dificultad para aplicar la técnica, porque el trabajo que realiza la comunidad OSS es completamente voluntario y realizado en el tiempo libre de sus miembros [37][181][186][210]. Por tal razón, técnicas que requieren preparativos previos (es decir, que no se puedan aplicar sobre la marcha o cuando se presenta la oportunidad) no se ajustan a la filosofía de trabajo de la comunidad OSS. Ejemplos de estas técnicas son la *observación directa* [149] e *inspecciones de usabilidad colaborativas* [50]. En la *observación directa*, usuarios individuales son observados realizando tareas específicamente preparadas para la ocasión, mientras el observador anota su comportamiento [149]. El objetivo principal de las *inspecciones de usabilidad colaborativas* es mejorar la usabilidad e identificar los problemas de usabilidad de la aplicación bajo inspección. Para realizar tal inspección, la aplicación es usada siguiendo escenarios de uso previamente establecidos, preparados y documentados [50].

Similar a la condición desfavorable anterior, está el caso de las técnicas pesadas, es decir, aquellas técnicas que **requieren de múltiples pasos para su ejecución** demandando una considerable inversión de tiempo. De nuevo, debido a que el trabajo en la comunidad OSS es realizado por voluntarios en su tiempo libre esta condición dificulta la aplicación de ciertas técnicas. Un ejemplo de técnicas pesadas lo encontramos en *Personas* [52]. Esta técnica

consiste en la creación de representaciones de los usuarios finales (es decir, personas) que ayudan a guiar el diseño de las aplicaciones, teniendo en mente siempre al usuario y evitando que los desarrolladores desarrollen software para sí mismos. La técnica *Personas* consta de siete pasos, entre los que se encuentran por ejemplo, la identificación de las variables conductuales y el mapeo de los sujetos entrevistados a estas variables conductuales. En la identificación de las variables conductuales se listan los distintos aspectos de comportamiento observados en las entrevistas a los usuarios. Ejemplos de variables conductuales para una aplicación de e-commerce son frecuencia de compra, motivación de la compra y edad. Una vez identificadas las variables conductuales, el siguiente paso es mapear cada uno de los sujetos entrevistados en las variables conductuales identificadas previamente [52].

La participación de los usuarios es fundamental en algunas técnicas de usabilidad pero en los proyectos OSS a menudo los usuarios están distribuidos geográficamente. Encontramos dos grupos de técnicas donde son necesarios los usuarios. En el primero se encuentran aquellas técnicas donde es **necesaria la participación del usuario**, pero donde no se requiere que varios usuarios participen simultáneamente, como es el caso por ejemplo de las técnicas *grabación de video* [92] e *información post-test* [50]. La *grabación de video* es una técnica de recolección de datos que permite capturar todo lo que ocurre mientras el usuario realiza una tarea con la aplicación que se pretende evaluar [92]. La técnica *información post-test* consiste de una entrevista que se realiza a cada usuario después de finalizar la prueba de usabilidad. La entrevista tiene por objetivo solicitar al usuario retroalimentación y sugerencias tanto de la prueba realizada como de la aplicación que estaba siendo evaluada [50].

En el segundo grupo encontramos las técnicas que **exigen que varios usuarios se encuentren físicamente reunidos** en el mismo lugar. Las técnicas *focus groups* [138] y *tormenta de ideas visual* [149] son ejemplos de técnicas de este tipo. En un *focus groups*, se reúne a un grupo de entre seis y nueve usuarios para discutir nuevos conceptos e identificar temas relevantes en un período de unas dos horas. Las reuniones permiten evaluar las necesidades de los usuarios y sus sensaciones, tanto antes de que sea diseñada la interfaz de usuario como después de un tiempo de uso. La *tormenta de ideas visual* permite explorar diseños alternativos y validarlos en reuniones donde participan los usuarios. Una vez creados los diseños iniciales, las ideas pueden ser desarrolladas en mayor profundidad con la construcción de cartulinas que representan el diseño. Estos diseños son evaluados con los usuarios [149].

Para la aplicación de ciertas técnicas de usabilidad según lo prescrito por la IPO se **requiere tener previamente cierta información**, como por ejemplo en la técnica *diagramas de afinidad*. El objetivo de la técnica consiste en que cada observador anota en un post-it cada una de las observaciones que va recogiendo de la observación de los usuarios en su entorno habitual de trabajo. Las notas que parecen estar relacionadas se van agrupando. A partir de estas agrupaciones se obtiene el diagrama de afinidad que será de gran utilidad para especificar los requisitos de usabilidad [120]. El hecho de que la técnica requiera de notas obtenidas a través de observaciones realizadas a los usuarios en su entorno de trabajo, supone un impedimento para su aplicación porque los usuarios en la comunidad OSS se encuentran distribuidos por todo el mundo.

En resumen, hemos identificado las siguientes seis características de técnicas de usabilidad que resultan especialmente desfavorables para ser aplicadas en un desarrollo OSS:

- Requerir un experto de usabilidad para ser aplicadas.
- Necesitar de una preparación previa, por ejemplo definir una tarea que el usuario debe realizar.

- Ser pesadas, es decir, requieren de múltiples pasos para ser ejecutadas.
- Necesitar la participación de los usuarios.
- Requerir que varios usuarios estén físicamente reunidos.
- Necesitar de cierta información, obtenida previamente, para poder ser aplicadas.

A pesar de que ciertas técnicas de usabilidad demandan condiciones que impiden ser aplicadas en el desarrollo OSS, es posible realizar transformaciones a las técnicas para acercarlas a la idiosincrasia de los proyectos OSS. En la siguiente sección, describiremos las transformaciones que se han realizado y proponemos nuevas transformaciones.

6.3. Transformaciones que Facilitan el Uso de Técnicas de Usabilidad en OSS

Cuando una técnica de usabilidad **requiere un experto en usabilidad** para su aplicación, algunos proyectos OSS lo han sustituido por un desarrollador, un estudiante o grupo de estudiantes de la IPO (bajo la supervisión de un mentor) o por usuarios que aplican la técnica con la ayuda de un documento (por ejemplo, una guía de estilo). Por ejemplo, en la técnicas *evaluación heurística* [152], *grabación de vídeo* [152] e *información post-test* [152] incorporadas en un proyecto[1] OSS para el desarrollo de un videojuego, el rol del experto fue desempeñado por un estudiante de la IPO. En la técnica *inspección de conformidad con estándares* [46], fueron los usuarios con la ayuda de una guía de estilo (conocida como HIGs) quienes desempeñaron el rol del experto en usabilidad en el proyecto OSS KDE [46]. Los usuarios buscaron todas las infracciones evidentes a las HIGs y cualquier aspecto que pueda afectar a la usabilidad, como por ejemplo, falta de retroalimentación en determinadas acciones, barras de herramientas, menús o cuadros de diálogo de configuración sobrecargados [46]. Existen otras técnicas que demandan un experto y que los proyectos OSS no han utilizado hasta ahora (o no han reportado sobre su uso). Algunos ejemplos de técnicas no incorporadas por OSS y que proponemos podrían serlo gracias a la sustitución de un experto en usabilidad por otro recurso más asequible son: *representaciones de pantallas* y *recorrido pluralístico*. La técnica *representaciones de pantallas* consiste de una especificación de los contenidos de la parte visible de la interfaz [92]. En la técnica *recorrido pluralístico* se lleva a cabo una evaluación heurística colaborativa, donde todos sus participantes representan el rol del usuario [50]. Proponemos que en ambas técnicas el rol del experto en usabilidad sea realizado por un desarrollador o un miembro de la comunidad OSS con ciertos conocimientos en usabilidad.

Para las técnicas de la IPO que **exigen de una preparación previa**, en algunos proyectos OSS se ha suavizado esta condición de manera que los preparativos necesarios puedan realizarse sobre la marcha. Este es el caso, por ejemplo, de las técnicas *observación directa* [188] y *recorridos cognitivos* [152][188]. En la técnica *observación directa* [188], las observaciones a los usuarios fueron realizadas de manera informal cuando familiares y amigos de los desarrolladores usaron la aplicación o cuando usuarios avanzados realizaban demostraciones en conferencias. La tarea que el usuario debía desarrollar, mientras era observado no fue definida con anterioridad sino que fue oportunista. Esta técnica ha sido incorporada en varios proyectos entre los cuales se encuentra una aplicación de gráficos de mapa de bits y una aplicación de gráficos basada en vectores [188]. En la técnica *recorridos cognitivos* [188], la exploración del software no es realizada de acuerdo a una serie de datos y acciones previamente definidas. En su lugar, el desarrollador inspecciona el software mientras realiza algún desarrollo. La técnica *recorridos cognitivos* ha sido incorporada en diferentes proyectos OSS, como por ejemplo en el desarrollo de un paquete de animación 3D [188] y un navegador web [188]. La técnica

[1] Algunos autores se refieren de manera general a los proyectos OSS sin especificar su nombre.

interacción constructiva [50] también exige de una preparación previa. Esta técnica aún no ha sido usada en proyectos OSS y proponemos que podría ser incorporada de forma análoga a la técnica *observación directa*. La técnica *interacción constructiva* implica tener a dos usuarios utilizando el sistema juntos y se basa en el hecho de que las personas acostumbran a verbalizar cuando están intentando realizar una tarea de forma conjunta (como en la técnica *pensar en voz alta*, pero con dos usuarios). En la técnica *interacción constructiva* es necesario definir previamente la tarea específica que deben desempeñar los usuarios con la aplicación [50]. Proponemos que esta tarea no sea definida con anterioridad, sino que sea aprovechada una acción que los usuarios desempeñen habitualmente con la aplicación.

Las técnicas pesadas, que **exigen una serie de pasos para su ejecución**, pueden simplificarse, como por ejemplo en el caso de la técnica *Personas* [38][46][139][188] incorporadas en los proyectos OSS: KDE [46], TYPO3 [38][139], un sistema operativo de escritorio y en un navegador web [188]. La creación de personas ha sido realizada con base en las descripciones que la propia comunidad OSS suministró y no a través de una serie de entrevistas (con una estructura definida) presenciales realizadas a un grupo limitado de usuarios. Existen otras técnicas que también exigen una serie de pasos para su ejecución y que no han sido usadas en proyectos OSS, como por ejemplo la técnica *mapa de roles de usuario*. La técnica tiene por objetivo capturar la visión general de los usuarios del sistema a través de un mapa de roles de usuario. En el mapa se representan los roles de los usuarios y sus relaciones. La construcción del mapa de roles de usuario se realiza en cinco pasos: (i) generar ideas de cuáles serán los roles de los usuarios, (ii) ordenar y organizar por grupos o categorías los roles, (iii) agregar detalles específicos como la descripción de los roles, (iv) estudiar los roles identificados para mejorarlos y completarlos, y finalmente (v) construir el mapa interrelacionando los roles por afinidad, clasificación o composición [50]. Proponemos que esta técnica podría se incorporada en los desarrollos OSS simplificando los tres primeros pasos para convertirlos en uno solo. Como resultado de estos tres primeros pasos se obtienen los roles de los usuarios. Los roles de los usuarios serían obtenidos colaborativamente a través de la participación de los miembros del proyecto OSS. Serían los desarrolladores y los usuarios OSS quienes suministrarían ideas de cuáles son los roles.

La **participación de los usuarios es fundamental** en muchas técnicas de usabilidad. Sin embargo, en los proyectos OSS a menudo sus usuarios se encuentran distribuidos por todo el mundo. Para sortear esta condición desfavorable la comunidad OSS ha encontrado hasta seis soluciones para conseguir usuarios. En primer lugar, los usuarios participan cuando se imparten tutoriales o capacitaciones sobre el software o cuando se enseña el mismo en clases. Esta transformación se ha realizado para la técnica *cuestionarios y encuestas* [188] incorporada por ejemplo en los proyectos OSS: una aplicación de edición, un editor de fuentes y un navegador Web [188]. En segundo lugar, los usuarios son reemplazados por desarrolladores. En la técnica *focus groups* un experto en usabilidad se reúne con los desarrolladores con el objetivo de valorar aplicaciones o los diseños propuestos para una nueva funcionalidad [188]. Esta transformación de la técnica ha sido incorporada en los proyectos OSS: un entorno de ventanas de escritorio, un sistema operativo de escritorio, un paquete de animación 3D y una aplicación de gráficos de mapa de bits [188]. En tercer lugar, los usuarios son familiares y amigos de los desarrolladores o son conseguidos durante las demostraciones que se realizan durante las conferencias. En la técnica *observación directa* [188] encontramos esta transformación. Los desarrolladores realizaron observaciones oportunísticas a familiares y amigos mientras ellos usaban la aplicación de manera casual o cuando, en las conferencias realizadas sobre la aplicación, usuarios avanzados presentaban demostraciones de la aplicación [188]. Esta transformación de la técnica ha sido incorporada en el desarrollo de algunos proyectos OSS, por ejemplo: una aplicación de gráficos de mapa de bits y una aplicación de gráficos basada en vectores [188]. En cuarto lugar, cuando los usuarios asisten a las conferencias no lo hacen solo

como participantes pasivos sino también activamente impartiendo talleres a otros participantes [188], como ha ocurrido en el desarrollo de los proyectos OSS: una aplicación de gráficos de mapa de bits, una utilidad de escritorio, una aplicación de gráficos basada en vectores y un navegador Web [188]. En quinto lugar, los usuarios participan remotamente: dando su opinión en foros o a través de chats o correos electrónicos a los desarrolladores [188], realizando pruebas al software -desde su casa- antes de su liberación (por ejemplo, *retroalimentación del usuario* [188]), opinando a través de las herramientas utilizadas para el reporte de errores [89][137][188], como por ejemplo en la técnica *retroalimentación del usuario*, realizando comentarios en una wiki sobre, por ejemplo, problemas de usabilidad [152] o permitiendo al desarrollador acceder remotamente a su ordenador para corregir errores "en vivo" [23]. La técnica *retroalimentación del usuario* ha sido incorporada en varios proyectos OSS con esta transformación entre los que se encuentran: un editor de fuentes, una aplicación para edición de vídeo y un sistema de gestión de contenidos para la Web [188]. En sexto lugar, se encuentra el caso particular de las técnicas *grabación de audio* [152], *grabación de vídeo* [152], *información post-test* [152] y *test de usabilidad en laboratorio* [152] incorporadas en un proyecto OSS de desarrollo de un videojuego. En la incorporación de estas técnicas los usuarios fueron conseguidos preguntando a usuarios de videojuegos sus hábitos de juego y experiencia previa con videojuegos.

Proponemos que otras técnicas de usabilidad aún no usadas en proyectos OSS donde sea necesaria la participación de los usuarios pueden beneficiarse de algunas de las transformaciones anteriores. Por ejemplo, las técnicas *escenarios de tareas* y *card sorting*. Los *escenarios de tareas* son instancias de casos de uso que representan las tareas del trabajo en la vida real. Se elaboran estos escenarios para las tareas más representativas de cada tipo de usuario. La construcción de los escenarios de tareas es realizada por un equipo de trabajo, donde están incluidos algunos usuarios [120]. Proponemos que esta participación se realice de manera remota a través de chats o de una wiki donde los usuarios realicen comentarios de los escenarios de tareas que están siendo creados por el equipo de trabajo. La técnica *card sorting* permite comprender la representación de información que manejan los usuarios y consiste en pedir a los usuarios que agrupen una serie de conceptos del dominio. Cada concepto se escribe en una tarjeta y se pide al usuario que organice las tarjetas en pilas [12]. Proponemos que el equipo de trabajo se reúna remotamente a través de herramientas que permitan la comunicación de vídeo sobre internet. Las tarjetas, previamente escaneadas, serán enviadas a los usuarios para que las agrupen. La agrupación podría ser realizada etiquetando con una palabra clave todas las tarjetas de una pila.

Otras técnicas de la IPO requieren no solo la participación de los usuarios, sino también que **varios usuarios se encuentren físicamente reunidos**. En algunos proyectos OSS las reuniones han sido realizadas de manera virtual a través de chats (por ejemplo, en la técnica *focus groups* [188] incorporada en los proyectos OSS: un entorno de ventanas de escritorio, sistema operativo de escritorio, un paquete de animación 3D y una aplicación de gráficos de mapa de bits) o han sido reemplazadas por una wiki como en la técnica *tormenta de ideas visual* [188] incorporada en una aplicación de gráficos de mapa de bits. Otro ejemplo de técnica que requiere la participación de varios usuarios físicamente reunidos es *JEM (Joint Essential Modelling)*. Proponemos que esta técnica podría ser incorporada en proyectos OSS con una transformación de este tipo. La técnica *JEM (Joint Essential Modelling)* consiste en un proceso estructurado donde participan varios usuarios con el objetivo de desarrollar especificaciones de requisitos centrados en el uso [50]. Proponemos que las reuniones sean llevadas a cabo online, utilizando sistemas software que permitan la comunicación de texto, voz y vídeo sobre internet.

Para la aplicación de algunas técnicas de usabilidad es **necesario tener previamente cierta información** (por ejemplo, las notas obtenidas de una serie de entrevistas contextuales) como por ejemplo, en las técnicas *diagramas de afinidad* [120] y *escenarios de tareas* [120]. Técnicas como las anteriores no han sido incorporadas aún en proyectos OSS. Sin embargo, consideramos que podrían ser incorporadas con una nueva transformación análoga a las transformaciones realizadas para otras técnicas: obtener la información necesaria para aplicar la técnica por medios diferentes a lo prescritos por la IPO, por ejemplo, a través de entrevistas telefónicas o encuestas en lugar de realizar investigaciones contextuales que requieren entrevistar al usuario en su ambiente de trabajo.

Finalmente, encontramos el caso especial de dos transformaciones que no surgen para dar solución a condiciones desfavorables. Estas dos transformaciones surgen por la idiosincrasia y forma de trabajo de la comunidad OSS, particularmente por los artefactos Web que utilizan. La primera, consiste en que los usuarios no son encuestados o entrevistados directamente, porque se trata de usuarios proactivos y tienden a contribuir sin que se les pregunte. Esta transformación ha sido incorporada en diferentes proyectos OSS (por ejemplo, una aplicación de edición, un editor de fuentes y un navegador Web [188]). En la segunda, los mensajes y las preguntas que realizan los usuarios a los desarrolladores no son solamente publicados en foros, también son comunicados a través de IRC y listas de correo [188], como por ejemplo en los proyectos OSS: una aplicación de edición, un editor de fuentes y un navegador Web. Creemos que estas dos transformaciones suponen fuentes de información sobre los usuarios que podrían ser explotadas para mejorar la usabilidad de las aplicaciones OSS, aunque sean recursos no propuestos por la IPO para obtener información de los usuarios que no suelen darse en los proyectos comerciales.

La Tabla 6.1 resume las transformaciones que proponemos para permitir una mejor concordancia entre las técnicas de usabilidad y los proyectos OSS. Cada transformación ha sido identificada con un código alfanumérico para facilitar su posterior referencia. En las columnas 4 y 5 se listan tanto las técnicas de usabilidad que OSS ha incorporado en algún proyecto con esta transformación, así como las técnicas que OSS no ha incorporado y que proponemos podrían seguir tal transformación. Nótese que algunas celdas de las columnas 4 y 5 tienen fondo gris indicando que las técnicas vecinas no tienen ninguna relación entre sí. Salvo en el caso de las transformaciones A2-a y A3-a, donde el fondo gris en las celdas de las columnas 2, 4 y 5 sirve para indicar la transformación que debe ser aplicada a las técnicas de las columnas 4 y 5.

Las Tablas 6.2, 6.3 y 6.4 presentan otra vista de la información mostrada en la Tabla 6.1, pero donde la atención se centra en las técnicas de usabilidad y sus transformaciones asociadas. Por ejemplo, para poder aplicar en los desarrollos OSS la técnica *evaluación heurística* es necesario realizar la transformación identificada con el ID A1 que corresponde, como muestra la Tabla 6.1, a sustituir el experto en usabilidad por un desarrollador, un estudiante o grupo de estudiantes de la IPO o usuarios con la ayuda de un documento (HIGs). Las Tablas 6.2, 6.3 y 6.4 presentan las técnicas de la IPO organizadas por las actividades del desarrollo según prescribe la IS: ingeniería de requisitos, diseño y evaluación, respectivamente. Para cada una de las técnicas se especifica el nombre genérico, el nombre usado en la literatura de la IPO y el ID de la transformación que debe ser realizada para poder ser incorporada en los desarrollos OSS. Este ID es el mismo utilizado en la Tabla 6.1.

En la siguiente sección discutiremos las técnicas de usabilidad que se han propuesto como nuevas en la comunidad OSS pero que, analizadas desde la perspectiva de las transformaciones, descubrimos que en realidad son técnicas clásicas de la IPO que han sufrido una conjunción de transformaciones.

Tabla 6.1: Transformaciones a las Técnicas de Usabilidad para Adaptarlas a la Idiosincrasia del Desarrollo OSS.

ID	Transformación	Condición Desfavorable	Técnicas Incorporadas por OSS	Técnicas No Incorporadas por OSS
A1	El experto en usabilidad es sustituido por: ▪ Un desarrollador ▪ Un estudiante o grupo de estudiantes de la IPO (bajo la supervisión de un mentor) ▪ Usuarios con la ayuda de un documento (HIGs)	Es indispensable contar con un experto en usabilidad para aplicar la técnica	Personas [38][46][139][188]	Escenarios de Tareas
			Evaluación Heurística [152]	Especificaciones de Usabilidad
			Inspecciones de Conformidad con Estándares [46]	Representaciones de Pantallas
			Recorridos Cognitivos [152][188]	Revisión de Guías
			Pensar en Voz Alta [23]	Inspecciones de Consistencia
			Informacion Post-Test [152]	Inspecciones de Usabilidad Colaborativas
			Test de Usabilidad en Laboratorio [152]	Recorrido Pluralístico
			Grabación Vídeo [152]	Interacción Constructiva
			Grabación Audio [152]	Test Retrospectivo
			Entrevistas [188]	Toma de Incidentes Críticos
				Método de Entrenamiento
				Registro de Uso
				Registro de Pulsaciones en el Tiempo
				Registro de la Interacción
				Evaluación por Control Remoto
				Test Remoto por Video-Conferencia
				Entrevistas Estructuradas
				Monitores Software
				Evaluación Remota Semi-Instrumentada

Tabla 6.1: Transformaciones a las Técnicas de Usabilidad para Adaptarlas a la Idiosincrasia del Desarrollo OSS (continuación).

ID	Transformación	Condición Desfavorable	Técnicas Incorporadas por OSS	Técnicas No Incorporadas por OSS
A2-a	La tarea que deben realizar los usuarios no es definida previa e intencionadamente. En su lugar, se aprovechan las tareas que habitualmente realizan los usuarios	Es necesaria una preparación previa (por ejemplo, concebir la tarea que el usuario debe realizar) para aplicar la técnica	Observación Directa [188]	Observación Aleatoria Interacción Constructiva
A2-b	La exploración del software no es realizada de acuerdo a una serie de datos y acciones previamente definidas. En su lugar, el desarrollador inspecciona el software mientras realiza algún desarrollo		Recorridos Cognitivos [188]	
A3-a	La creación de personas se realizó con base en descripciones que la comunidad OSS suministró y no a través de entrevistas (con una estructura definida) presenciales realizadas a un grupo limitado de usuarios	Es una técnica pesada, es decir, requiere de varios pasos para su ejecución	Personas [38][46][139][188]	
A3-b	Los roles de los usuarios serían obtenidos en un solo paso gracias a la participación de la comunidad OSS y no a través de varios pasos			Mapa de Roles de Usuario

Tabla 6.1: Transformaciones a las Técnicas de Usabilidad para Adaptarlas a la Idiosincrasia del Desarrollo OSS (continuación).

ID	Transformación	Condición Desfavorable	Técnicas Incorporadas por OSS	Técnicas No Incorporadas por OSS
A4	Los usuarios: ▪ Participan cuando se imparten tutoriales o capacitaciones sobre el software o cuando se enseña el mismo en clases ▪ Son reemplazados por desarrolladores ▪ Son familiares y amigos de los desarrolladores o son conseguidos durante las demostraciones que se realizan en las conferencias ▪ Participan remotamente, a través de: (I) foros o correos electrónicos, (II) realizando pruebas al software antes de su liberación, (III) opinando a través de las herramientas para el reporte de errores, (IV) realizando comentarios en una wiki o (V) permitiendo al desarrollador acceder remotamente a su ordenador	Es necesaria la participación de los usuarios para aplicar la técnica	Personas [38][46][139][188] Tormenta de Ideas Visual [188] Inspecciones de Conformidad con Estándares [46] Pensar en Voz Alta [23] Información Post-Test [152] Test de Usabilidad en Laboratorio [152] Grabación Vídeo [152] Grabación Audio [152] Observación Directa [188] Cuestionarios y Encuestas [188] Entrevistas [188] Focus Groups [188] Retroalimentación del Usuario [89][137][152][188] Revistas y Conferencias para Usuarios [35][188]	Diagramas de Afinidad JEM (Joint Essential Modeling) Card Sorting Perfiles de Usuario Escenarios de Tareas Task Sorting Escenarios y Storyboards Prototipado Prototipos Escenario Prototipos de Papel Prototipos Activos Prototipos Guiados Gramáticas Árboles de Menús Diagramas de Transición de Estados de la Interfaz Modelo del Contenido de la Interfaz Test Retrospectivo Toma de Incidentes Críticos Método de Entrenamiento Entrevistas Estructuradas Entrevistas Flexibles

Tabla 6.1: Transformaciones a las Técnicas de Usabilidad para Adaptarlas a la Idiosincrasia del Desarrollo OSS (continuación).

ID	Transformación	Condición Desfavorable	Técnicas Incorporadas por OSS	Técnicas No Incorporadas por OSS
A5	Las reuniones: ▪ Son virtuales (por ejemplo, a través de chat) ▪ Son reemplazadas por una wiki, donde cualquier involucrado en el proyecto puede aportar sus ideas	Es necesario que varios usuarios estén físicamente reunidos	Personas [38][46][139][188] Tormenta de Ideas Visual [188] Focus Groups [188]	Diagramas de Afinidad JEM (Joint Essential Modeling) Perfiles de Usuario Casos de Uso Esenciales Escenarios de Tareas Modelo de Contenido de la Interfaz Diseño Integrador Diseño Paralelo Inspecciones de Consistencia Inspecciones de Usabilidad Colaborativ. Recorrido Pluralístico Interacción Constructiva
		Es necesario que varios desarrolladores estén físicamente reunid.		Mapa de Roles de Usuario Organización de la Ayuda según Casos de Uso
A6	La información necesaria para aplicar la técnica es conseguida por otros medios diferentes a los establecidos por la IPO (por ejemplo, encuestas en lugar de investigaciones context.)	Es necesario obtener cierta información para aplicar la técnica de la manera prescrita por la IPO		Diagramas de Afinidad Perfiles de Usuario HTA (Hierarchical Task Analysis) Escenarios de Tareas Guía de Estilo del Producto
A7	Los usuarios no son encuestados o entrevistados porque ellos son más proactivos y tienden a contribuir sin necesidad de que nadie les pregunte nada	(Esta transformación no surge para solucionar ninguna condición desfavorable, surge por la idiosincrasia de OSS)	Cuestionarios y Encuestas [188] Entrevistas [188]	

Tabla 6.1: Transformaciones a las Técnicas de Usabilidad para Adaptarlas a la Idiosincrasia del Desarrollo OSS (continuación).

ID	Transformación	Condición Desfavorable	Técnicas Incorporadas por OSS	Técnicas No Incorporadas por OSS
A8	Los mensajes y preguntas no son solo publicados en foros, también son comunicados a través de IRC y listas de correo	(Esta transformación no responde a dar solución a ninguna condición desfavorable, surge por la forma de trabajo de la comunidad OSS)	Foros [188]	

6.4. Nuevas Técnicas de Usabilidad en OSS que en Realidad No Son Tales (Técnicas Existentes con Nuevos Trajes)

Hemos identificado cuatro técnicas de usabilidad que diversos autores afirman haber sido inventadas en la comunidad OSS: *design-by-blog* [188], *design-workshops* [188], *open content projects* [137] y *season of usability* [188]. A continuación, explicaremos para cada una de estas supuestas nuevas técnicas con cuál técnica de la IPO se corresponden y estudiaremos las condiciones desfavorables que motivan las transformaciones que han sufrido que las hacen prácticamente irreconocibles.

6.4.1. Design-by-Blog

Muchos proyectos OSS tienen blogs para comunicar noticias sobre su progreso [144]. Los bloggers (quienes escriben los blogs) tienen diferentes motivaciones para escribirlos, por ejemplo, proporcionar comentarios y opiniones sobre un tema particular [134]. Sin embargo, desde la perspectiva de los proyectos OSS los blogs representan una forma sencilla de mantener la comunicación a través de una comunidad distribuida de miembros. Los blogs, poco a poco, están siendo utilizados no solo para comunicar noticias, sino también para analizar y discutir los diseños de las interfaces de usuario con cualquier miembro de la comunidad OSS, incluidos los usuarios. Un diseño orientado hacia el blog consiste de una descripción del problema, los objetivos y las restricciones o los criterios de diseño, las capturas de pantalla del diseño propuesto y los diferentes comentarios sobre el diseño [137][188]. El *design-by-blog* tiene como objetivo resolver un determinado problema descrito en un blog junto con una idea inicial de cómo resolverlo. Los comentarios de quienes participan en el blog permiten la exploración de otras ideas, soluciones alternativas o variaciones de la solución inicial, manteniendo cierta consistencia en el diseño. Esta técnica de la IPO ha sido incorporada en los proyectos OSS: Phyton [144], GNOME [144], un navegador Web [188] y un sistema operativo de escritorio [188]. El objetivo de los *design-by-blog* y el método utilizado para cumplir tal objetivo, rememora la técnica de la IPO *tormenta de ideas visual*. Esta técnica permite explorar diseños alternativos y validarlos en reuniones donde el número de participantes es limitado. Una vez creados los diseños iniciales, las ideas son desarrolladas en mayor profundidad con la construcción de cartulinas representando los diferentes diseños que serán evaluados con los usuarios [149].

Tabla 6.2: Técnicas de la IPO Relacionadas con la Actividad de Ingeniería de Requisitos y las Transformaciones que Permiten su Incorporación.

Nombre Genérico de la Técnica de la IPO	**Nombre Técnica Usada en la Literatura**	**ID Transformación**
Diagramas de Afinidad	Diagramas de Afinidad	A4 A5 A6
JEM (Joint Essential Modeling)	JEM (Joint Essential Modeling)	A4 A5
Card Sorting	Card Sorting	A4
Perfiles de Usuario	Perfiles de Usuario	A4 A6
	Característica de Usuario Individuales	
	Perfiles de Uso	
	Modelo Estructurado de Roles	
	Cuestionarios Perfiles de Usuario	
Mapa de Roles de Usuario	Mapa de Roles de Usuario	A3-b A5
Personas	Personas	A1 A3-a A4 A5
Casos de Uso Esenciales	Casos de Uso Esenciales	A5
HTA (Hierarchical Task Analysis)	HTA (Hierarchical Task Analysis)	A6
Escenarios de Tareas	Escenarios de Tareas	A1 A4 A5 A6
Task Sorting	Task Sorting	A4
Escenarios y Storyboards	Escenarios, Storyboards e Instantáneas	A4
	Escenarios	
	Escenarios y Storyboards	
Tormenta de Ideas Visual	Tormenta de Ideas Visual	A4 A5
Prototipado	Prototipado	A4
	Prototipos Escenario	A4
	Prototipos de Papel	A4
	Prototipos Activos	A4
	Prototipos Guiados	A4
Especificaciones de Usabilidad	Especificaciones de Usabilidad	A1
	Objetivos de Usabilidad	
Evaluación Heurística	Evaluación Heurística	A1
Inspecciones	*Inspecciones de Conformidad con Estándares*	A1 A4
	Revisión de Guías	A1
	Inspecciones de Consistencia	A1 A5
	Inspecciones de Usabilidad Colaborativas	A1 A5
Recorridos Cognitivos	Recorridos Cognitivos	A1 A2-b
Recorrido Pluralístico	Recorrido Pluralístico	A1 A5

Tabla 6.3: Técnicas de la IPO Relacionadas con la Actividad de Diseño y las Transformaciones que Permiten su Incorporación.

Nombre Genérico de la Técnica de la IPO	**Nombre Técnica Usada en la Literatura**	**ID Transformación**
Representaciones de Pantallas	Escenarios y Representaciones de Pantallas	A1
Guía de Estilo del Producto	Guía de Estilo del Producto	A6
Gramáticas	Gramáticas	A4
Árboles de Menús	Árboles de Menús	A4
Diagramas de Transición de Estados de la Interfaz	Diagramas de Transición de Estados de la Interfaz	A4
Modelo del Contenido de la Interfaz	Modelo del Contenido de la Interfaz	A4 A5
Prototipado	Prototipado	A4
	Prototipos Escenario	A4
Diseño Integrador	Diseño Integrador (Both-And Design)	A5
Diseño Paralelo	Diseño Paralelo	A5
Organización de la Ayuda según Casos de Uso	Organización de la Ayuda según Casos de Uso	A5

Tabla 6.4: Técnicas de la IPO Relacionadas con la Actividad de Evaluación y las Transformaciones que Permiten su Incorporación.

Nombre Genérico de la Técnica de la IPO	**Nombre Técnica Usada en la Literatura**	**ID Transformación**
Evaluación Heurística	Evaluación Heurística	A1
Inspecciones	*Inspecciones de Conformidad con Estándares*	A1 A4
	Revisión de Guías	A1
	Inspecciones de Consistencia	A1 A5
	Inspecciones de Usabilidad Colaborativas	A1 A5
Recorridos Cognitivos	Recorridos Cognitivos	A1 A2-b
Recorrido Pluralístico	Recorrido Pluralístico	A1 A5
Pensar en Voz Alta	Toma del Protocolo Verbal Concurrente	A1 A4
	Pensar en Voz Alta	
	Test Formales de Usabilidad (en las etapas iniciales)	
	Interacción Constructiva	A1 A2-a A5
	Test Retrospectivo	A1 A4
	Toma de Incidentes Críticos	A1 A4
	Método de Entrenamiento	A1 A4

Tabla 6.4: Técnicas de la IPO Relacionadas con la Actividad de Evaluación y las Transformaciones que Permiten su Incorporación (continuación).

Nombre Genérico de la Técnica de la IPO	**Nombre Técnica Usada en la Literatura**	**ID Transformación**
Test de Usabilidad en Laboratorio	Test en Laboratorio	A1 A4
	Laboratorios de Usabilidad	
	Test de Usabilidad y Laboratorios	
Grabación Vídeo	Grabación Vídeo	A1 A4
Grabación Audio	Grabación Audio	A1 A4
	Protocolo Verbal	
Registro del Uso	Instrumentación Interna de la Interfaz	A1
	Registro del Uso	
	Registro Software	
	Registro Continuo del Rendimiento del Usuario	
	Registro de Pulsaciones en el Tiempo	A1
	Registro de la Interacción	A1
Evaluación por Control Remoto	Evaluación por Control Remoto	A1
Test Remoto por Video-Conferencia	Test Remoto por Video-Conferencia	A1
Observación Directa	Observación Directa	A2-a A4
	Observación Aleatoria	A2-a
Cuestionarios y Encuestas	Cuestionarios	A4 A7
	Cuestionarios y Encuestas	
	Encuestas	
Entrevistas	Entrevistas	A1 A4 A7
	Entrevistas Estructuradas	A1 A4
	Entrevistas Flexibles	A4
Focus Groups	Focus Groups	A4 A5
Registro del Uso	Instrumentación Interna de la Interfaz	A1
	Registro del Uso Real	
	Registro Software	
	Registro Continuo del Rendimiento del Usuario	
	Evaluación Remota Instrumentada	
	Registro de Pulsaciones en el Tiempo	A1
	Registro de la Interacción	A1
	Monitores Software	A1
Retroalimentación del Usuario	Retroalimentación del Usuario	A4
	Buzón de Sugerencias o Reporte de Errores	
	Servicios de Atención al Usuario en Línea	A8
	Foros	A4
	Revistas y Conferencias para Usuarios	A1

Ambas técnicas (*design-by-blog* y *tormenta de ideas visual*) comparten el mismo espíritu: partiendo de un diseño inicial se generan nuevas ideas de diseño mediante la contribución de diversas personas; las ideas de diseño son evaluadas por los usuarios proporcionando sus opiniones; la iteración idea/evaluación se repite varias veces hasta llegar a un diseño satisfactorio. En la *tormenta de ideas visual* el medio utilizado para representar los diseños (las cartulinas) es reemplazado en el *design-by-blog* por los blogs. El uso de estos blogs permite que cualquiera pueda dar su opinión, manifestando el principio OSS: discusión abierta a toda la comunidad. Por tanto, *design-by-blog* no es una técnica nueva sino una versión transformada de la técnica de la IPO *tormenta de ideas visual*.

La utilización de los blogs por parte de la comunidad OSS viene a dar solución a dos requisitos exigidos por la técnica *tormenta de ideas visual* que dificulta o impide que sea aplicada en desarrollos OSS: la necesidad de la participación de los usuarios y la exigencia de que sea necesario que varios usuarios estén físicamente reunidos. La participación de los usuarios se logra a través del blog en forma de comentarios y es el mismo blog quien actúa como el espacio donde se reúnen los usuarios, con la ventaja añadida de que el número de participantes en este espacio no está limitado como sí lo está en la técnica *tormenta de ideas visual* tal cual la prescribe la IPO [149]. El hecho de que el número de participantes no esté limitado y debido a lo público que es el blog, permite que la técnica se adapte al principio de la comunidad OSS: discutir en comunidad. Además, el blog es el instrumento que reemplaza a las cartulinas para representar los diferentes diseños.

Existen otras técnicas que podrían ser incorporadas en OSS gracias a la transformación de las reuniones como espacios virtuales de participación representados por blogs y que aún no han sido usadas en desarrollos OSS. Proponemos que las técnicas *casos de uso esenciales* y *escenarios de tareas* pueden adoptar esta misma transformación para adaptarse a la idiosincrasia de los desarrollos OSS. En la técnica *casos de uso esenciales* se definen, en un nivel alto de abstracción, casos de uso. Un caso de uso esencial es una estructura narrativa, expresada en el lenguaje del dominio de la aplicación y de los usuarios, escrito con un enfoque libre de tecnología e independiente de la implementación. Los casos de uso plasman el propósito o intenciones subyacentes en la interacción del usuario con la aplicación [50]. Proponemos transformar la técnica de modo que los casos de uso sean sometidos a debate en un blog, con el objetivo de que desarrolladores y/o usuarios realicen comentarios o suministren retroalimentación sobre los mismos y sean posteriormente mejorados o adaptados. En nuestra propuesta se adiciona el componente participativo a la técnica, es decir, los usuarios participan cuando la prescripción de la técnica de la IPO no lo exige.

Los *escenarios de tareas* son instancias de casos de uso que representan las tareas del trabajo en la vida real. El equipo de trabajo que desarrolla los escenarios de tareas incluye usuarios clave. Los escenarios se elaboran para las tareas más representativas de cada tipo de usuario [120]. Proponemos, que los escenarios de tareas sean sometidos, en un blog, a consideración por parte de la comunidad OSS. Los comentarios y mejoras recogidos en el blog servirán para mejorar los escenarios.

6.4.2. Design Workshops

Algunos proyectos OSS, celebran conferencias con cierta periodicidad (por ejemplo, anualmente). Dentro de estas conferencias se llevan a cabo reuniones en las que participan expertos en usabilidad y desarrolladores. Estas reuniones son conocidas dentro de la comunidad OSS como *design workshops*. En estas reuniones, expertos en usabilidad asesoran a desarrolladores sobre diferentes aspectos de usabilidad, por ejemplo, si es conveniente o no adicionar ciertas características a la aplicación [35][188]. Es importante mencionar que solo unos pocos proyectos OSS (OpenOffice [35] y un sistema operativo de escritorio [188]),

especialmente aquellos que tienen el apoyo de compañías, celebran estas conferencias donde se enmarcan los *design workshops*.

La nueva técnica *design workshops* es en realidad una transformación de la técnica de la IPO *focus groups*. En los *focus groups* se les pregunta a un grupo de usuarios sus opiniones sobre un producto software determinado y sobre aspectos que afectan negativamente su usabilidad. Las preguntas son formuladas en un entorno de grupo interactivo donde los participantes son libres de hablar con otros miembros del grupo. Con los *focus groups* se pretende entender mejor las motivaciones de los usuarios y su percepción sobre el producto software bajo estudio [138]. Si bien en los *focus groups* se reúnen usuarios y en los *design workshops* participan desarrolladores, la esencia es similar: discutir sobre aspectos relacionados con la usabilidad. Las reuniones en los *design workshops* se enmarcan dentro de un evento más grande (una conferencia) que ofrece la oportunidad de discutir en comunidad diferentes aspectos, entre ellos la usabilidad.

Son dos las condiciones desfavorables que exigen transformaciones para que la técnica *focus groups* encaje con la filosofía de desarrollos OSS: la necesidad de la participación de los usuarios y la necesidad de que varios de ellos estén físicamente reunidos para aplicar la técnica. Los usuarios son reemplazados por desarrolladores y las reuniones se enmarcan dentro de un evento más grande (una conferencia) que permite discutir en comunidad aspectos sobre la usabilidad. En estas reuniones el número de participantes no está limitado. Para realizar estas transformaciones es necesario que se celebren conferencias sobre el proyecto OSS. Estas conferencias se realizan gracias al patrocinio de compañías, y por lo general, muchos proyectos OSS no lo tienen. Por tal razón, consideramos que esta transformación tiene poca aplicabilidad en otras técnicas donde sea necesario contar con la participación de varios usuarios.

6.4.3. Open Content Projects

Los *open content projects* consisten en la construcción de un producto creativo, como por ejemplo un videojuego, gracias al trabajo colaborativo entre desarrolladores y usuarios finales. Los *open content projects* proporcionan una forma de abordar y mejorar la usabilidad del software debido a que los desarrolladores y usuarios finales dependen mutuamente entre sí para lograr un objetivo común (es decir, la construcción de un producto creativo). El objetivo de los *open content projects* es descubrir problemas y mejorar la usabilidad. Los problemas de usabilidad son descubiertos cuando la aplicación es usada de manera conjunta por los desarrolladores y los usuarios finales. Esta técnica ha sido incorporada por OSS en el desarrollo de un paquete de animación 3D [188].

La nueva técnica *open content projects* es una transformación de la técnica de la IPO *inspecciones de usabilidad colaborativas*. Las *inspecciones de usabilidad colaborativas* de la IPO son un proceso de revisión de la usabilidad de un prototipo o producto finalizado desde el punto de vista de los usuarios finales. El equipo de trabajo que participa en el proceso de revisión está compuesto por desarrolladores, usuarios finales, expertos de la aplicación o del dominio y expertos en usabilidad. Durante la inspección, los desarrolladores necesitan mantener la mentalidad de un usuario intolerante e impaciente. La presencia de usuarios reales en las inspecciones ayuda a este fin [50].

El objetivo final de las *inspecciones de usabilidad colaborativas* es mejorar la calidad del producto en términos de usabilidad. Con este fin, su principal propósito es identificar defectos de usabilidad e inconsistencias en la interfaz de usuario de la aplicación que está siendo inspeccionada. Las inspecciones son realizadas en dos fases. En la primera (inspección interactiva), la aplicación es usada siguiendo escenarios de uso previamente establecidos, preparados y documentados. En la segunda (inspección estática), se inspeccionan todos los

componentes de la interfaz de usuario (es decir, pantallas, menús o cajas de diálogo) [50]. Aunque, las *inspecciones de usabilidad colaborativas* se realizan de una manera estructurada, con tareas previamente definidas, comparte la esencia de los *open content projects*, donde también es usada la aplicación por un equipo de trabajo pero de una manera más abierta. Considerando solo la esencia de ambas técnicas, podemos decir que son la misma transformada para que encaje con el principio de libertad de la comunidad OSS. Estas transformaciones hacen parecer a primera vista que ambas técnicas son totalmente diferentes cuando en realidad no lo son.

Identificamos tres condiciones desfavorables en el análisis de la técnica *inspecciones de usabilidad colaborativas*: la exigencia de un experto en usabilidad, la necesidad de la participación de los usuarios y que varios estén físicamente reunidos, y la necesidad de una exigente preparación previa antes de aplicar la técnica. En la transformación de las *inspecciones de usabilidad colaborativas* se ha prescindido del experto en usabilidad. El proyecto OSS (un paquete de animación 3D) donde se ha incorporado esta técnica ha conseguido la participación de los usuarios motivándolos con el hecho de que el producto creativo resultante de los *open content projects* será exhibido y apreciado por el público en general. Los participantes en los *open content projects* se han reunido en una ubicación compartida.

La inspección de la usabilidad en los *open content projects* es realizada de una manera más libre sin escenarios de uso previamente definidos como prescribe la IPO para las *inspecciones de usabilidad colaborativas*. En los *open content projects*, la inspección de la usabilidad se consigue como un efecto colateral al usar la aplicación intensamente. Es decir, sus participantes no tienen como objetivo usar la aplicación siguiendo unos escenarios de uso para encontrar problemas de usabilidad, simplemente la usan y es durante su uso donde encuentran inconsistencias en la interfaz de usuario y problemas de usabilidad. Con esta transformación se consigue que la inspección sea más libre y placentera para el usuario.

Existe otra técnica que podría ser transformada de manera similar a la técnica *inspecciones de usabilidad colaborativas* y que aún no ha sido usada en los desarrollos OSS: *inspecciones de consistencia*. La técnica de la IPO *inspecciones de consistencia* es una variante de la técnica *inspecciones de usabilidad colaborativas* [50]. En una inspección de consistencia expertos en usabilidad verifican la consistencia a lo largo de una familia de interfaces, comprobando la consistencia de terminología, color, disposición, formatos de entrada/salida, etc. [180]. Proponemos transformar esta técnica de modo que las inspecciones de consistencia sean llevadas a cabo durante la construcción de un producto creativo. Sin embargo, los problemas de usabilidad que serían detectados son todos aquellos relacionados con inconsistencias entre los mismos tipos de elementos de la interfaz de usuario, como por ejemplo, barras de menú y cajas de diálogo.

6.4.4. Seasons of Usability

Las *seasons of usability* son una serie de proyectos, por lo general, patrocinados para animar a estudiantes de usabilidad, de diseño de interfaces de usuario y de diseño de la interacción para que participen en proyectos OSS [16]. Sin embargo, la mayoría de proyectos OSS no cuentan con el apoyo de compañías. El proyecto Roguelike (un videojuego de roles) [152] es un ejemplo de proyecto OSS que no tiene el apoyo de compañías. En las *seasons of usability* patrocinadas, los proyectos OSS que desean participar deben inscribirse en las convocatorias realizadas para tal fin, por lo general, organizadas anualmente. Una vez ha finalizado el plazo de inscripción, los mentores expertos en usabilidad se ponen en contacto con los proyectos OSS e inician el proceso de selección. La selección de los proyectos OSS está a cargo de cada mentor y se basa en la necesidad que tiene el proyecto en mejorar su usabilidad y en la disponibilidad de un

mentor técnico por parte del proyecto [16]. Las *seasons of usability* pueden tener una duración de entre tres y seis meses o incluso de dos años, en algunos casos [152]. Durante este tiempo, los estudiantes trabajan colaborativamente con un mentor experto en usabilidad y con los principales desarrolladores para mejorar la usabilidad de una aplicación OSS.

La esencia de las *seasons of usability* es precisamente mejorar la usabilidad gracias a la participación de personas con conocimientos en usabilidad, rememorando a las técnicas de la IPO del tipo *evaluación por expertos*. Es decir, la nueva técnica *seasons of usability* es en realidad una transformación de la técnica *evaluación por expertos* donde se sustituye el rol de experto en usabilidad por estudiantes de la IPO guiados por un mentor. Las *seasons of usability* han sido aplicadas en el desarrollo de los proyectos: un entorno de ventanas de escritorio, un sistema operativo de escritorio [188] y un videojuego de roles [152].

La *evaluación por expertos* es una técnica de la IPO que evalúa de manera subjetiva la usabilidad de un producto, basada en la experiencia y juicio de un experto o expertos en usabilidad. El experto puede estar cualificado por una formación en ingeniería de usabilidad, diseño de la interfaz de usuario, diseño gráfico, ergonomía, interacción persona-ordenador, psicología industrial o cognitiva. Para la evaluación de la usabilidad el experto puede utilizar cualquier método ya sea formal o informal, pero los hallazgos en último término dependen del juicio del experto más que de los métodos empleados. La mayoría de expertos realizan algunos ensayos con el producto que se está evaluando. Los evaluadores pueden realizar una exploración abierta de la interfaz de usuario o realizar tareas representativas con la aplicación. El resultado de una evaluación por expertos es normalmente una lista de los problemas de usabilidad y un conjunto de recomendaciones para mejorarla [50].

Identificamos en la técnica *evaluación por expertos* como condición que dificulta su uso en proyectos OSS la necesidad de contar con uno o varios expertos en usabilidad para su aplicación. Para superar este impedimento, el rol de experto en usabilidad es realizado por estudiantes de usabilidad, de diseño de interfaces de usuario y de diseño de la interacción junto con la tutoría de un mentor. Es importante mencionar que en las *seasons of usability* patrocinadas debe ser el proyecto OSS el primer interesado en mejorar la usabilidad de su aplicación, porque es el proyecto quien se inscribe en las convocatorias. Mientras que en las *seasons of usability* no patrocinadas, son los estudiantes junto con su mentor quienes deciden en cuál proyecto desean participar, como por ejemplo en el estudio empírico realizado por Rajanen et al. [152].

Existen otras técnicas que podrían ser incorporadas de manera similar a la técnica *evaluación por expertos*, como por ejemplo la técnica de la IPO *inspecciones de conformidad con estándares*. En las inspecciones de conformidad, el objetivo es identificar desviaciones de los estándares de la interfaz de usuario [149]. En nuestra propuesta, las inspecciones de conformidad son realizadas por estudiantes de usabilidad, de diseño de interfaces de usuario y de diseño de la interacción como parte de las *seasons of usability*.

Resumiendo, estas técnicas que han sido consideradas nuevas en OSS, en realidad han aplicado transformaciones a técnicas de usabilidad para sortear la dificultad de la participación de varios usuarios físicamente reunidos. Para sortear este impedimento, hemos identificado básicamente dos transformaciones. Por un lado, el rol de los usuarios puede reemplazarse por desarrolladores o conseguir la participación física de los usuarios gracias a una atractiva motivación (por ejemplo, el producto resultante de la participación en la técnica sería exhibido y apreciado por el público). Por otro lado, los usuarios participan remotamente en forma de comentarios en las herramientas utilizadas para el reporte de errores o a través de artefactos Web como por ejemplo blogs o listas de correo. Los artefactos Web sirven también como punto

de reunión virtual de los usuarios. Estas transformaciones intentan aplicar a las técnicas de usabilidad dos principios de la cultura OSS: discutir en comunidad y participación voluntaria de sus miembros.

En las siguientes secciones analizaremos cuáles técnicas podrían ser incorporadas en OSS, generalizando las transformaciones estudiadas en las dos secciones anteriores y transformaremos la técnica *Personas* para su utilización en proyectos OSS.

6.5. Técnicas de Usabilidad Aplicables a OSS

Las transformaciones que hemos propuesto anteriormente pueden ser aplicadas a otras técnicas de la IPO que compartan las mismas condiciones desfavorables que motivaron tales transformaciones. Para poder saber cuántas y cuáles técnicas podrían incorporarse en OSS gracias a las transformaciones, necesitamos analizar cada una de las técnicas de usabilidad para determinar qué impedimentos presentan para poder ser incorporadas en OSS. Sin embargo, identificar todas las técnicas de la IPO no es una tarea sencilla. En la IPO hay una tremenda diversidad de técnicas y una falta de estandarización donde la misma técnica puede tener distintos nombres dependiendo del autor y pueden existir diversas variantes para una misma técnica. Afortunadamente, algunos investigadores del área de la IS ya se han realizado el trabajo de compilar un catálogo de técnicas de la IPO [68].

Como se ha mencionado, Ferré et al. [68] compiló una lista de técnicas a partir de las fuentes de la IPO más reconocidas (ver Anexo F). Las técnicas de la IPO en el catálogo están clasificadas en las actividades de ingeniería de requisitos, diseño y evaluación según se entienden en IS (no en la IPO). Del catálogo, los autores [69][70] seleccionaron un subconjunto de técnicas más cercanas a IS (ver última columna de las tablas del Anexo F) de acuerdo con los siguientes criterios: participación de los usuarios, necesidad de formación, aplicabilidad general, proximidad con la IS, relación esfuerzo/mejora de la usabilidad y representatividad. Los valores de estos seis criterios fueron asignados a cada técnica de la IPO desde una perspectiva de la IS. El objetivo de dicho catálogo es apoyar a los ingenieros de software para elegir las técnicas con las que hacer frente a la usabilidad. En nuestra investigación usamos el catálogo recopilado por Ferré et al. [68] como universo de técnicas de la IPO sobre las que trabajaremos para analizar cuántas y cuáles técnicas podrían ser incorporadas en OSS.

Para analizar cuáles técnicas podrían ser incorporadas en los desarrollos OSS, tenemos en cuenta cuatro criterios: la estrategia prescrita por la IPO para cada técnica; la idiosincrasia particular de los desarrollos OSS; las condiciones desfavorables que creemos impiden incorporar técnicas de usabilidad en proyectos OSS; y las transformaciones que permiten sortear tales condiciones desfavorables.

La Tabla 6.5 recoge las técnicas de usabilidad relacionadas con las actividades de ingeniería de requisitos en IS. Para cada una se especifica un nombre genérico para referirse a la técnica (siguiendo a [68]), el nombre dado por los diferentes autores en la literatura (en *itálica* las variantes de la técnica), si la técnica es recomendada o no para su integración desde la perspectiva del desarrollo de software comercial según [69][70] y si la técnica podría ser incorporada en OSS si se le aplica alguna transformación.

Tabla 6.5: Técnicas de la IPO Relacionadas con Actividades de Ingeniería de Requisitos Aplicables a OSS con Transformaciones.

Nombre Genérico de la Técnica de la IPO	**Nombre Técnica Usada en la Literatura**	**Recom. Ferré**	**Una Transf.**	**Más de Una Transf.**
Análisis Competitivo	Análisis Competitivo	Sí	No	No
Análisis de Impacto Financiero	Análisis de Impacto Financiero	No	No	No
Investigación Contextual	Investigación Contextual	Sí	No	No
	Entrevistas Contextuales			
Diagr. de Afinidad	Diagramas de Afinidad	Sí	Sí	No
Observación Etnográfica	Etnografía	Sí	No	No
	Observación Etnográfica			
JEM	JEM (Joint Essential Modeling)	Sí	Sí	No
Card Sorting	Card Sorting	Sí	Sí	No
Perfiles de Usuario	Perfiles de Usuario	Sí	Sí	Sí
	Característ. de Usuario Individuales			
	Perfiles de Uso			
	Modelo Estructurado de Roles			
	Cuestionarios Perfiles de Usuario			
Mapa Roles de Usuario	Mapa de Roles de Usuario	Sí	Sí	Sí
Modelo Operacional	Modelo Operacional	No	No	No
	Capacidades y Restricciones de Plataforma			
Personas	Personas	Sí	Sí	Sí
Casos de Uso Esenciales	Casos de Uso Esenciales	Sí	Sí	Sí
HTA	HTA (Hierarchical Task Analysis)	Sí	Sí	Sí
Familia de Modelos GOMS	GOMS (Goals, Operations, Methods and Selection Rules)	No	No	No
Modelo de Interfaz Objeto-Acción	Modelo de Interfaz Objeto-Acción	No	No	No
Escenarios de Tareas	Escenarios de Tareas	Sí	Sí	Sí
Task Sorting	Task Sorting	Sí	Sí	No
Escenarios y Storyboards	Escenarios, Storyboards e Instantáneas	Sí	Sí	Sí
	Escenarios			
	Escenarios y Storyboards			
Tormenta de Ideas Visual	Tormenta de Ideas Visual	Sí	Sí	Sí
Prototipado	Prototipado	Sí	Sí	No
	Prototipos Escenario	Sí	Sí	Sí
	Prototipos de Papel	Sí	Sí	No
	Prototipos Activos	Sí	Sí	No
	Prototipos Guiados	Sí	Sí	No
	Prototipos Mago de Oz	No	No	No

Tabla 6.5: Técnicas de la IPO Relacionadas con Actividades de Ingeniería de Requisitos Aplicables a OSS con Transformaciones (continuación).

Nombre Genérico de la Técnica de la IPO	**Nombre Técnica Usada en la Literatura**	**Recom. Ferré**	**Una Transf.**	**Más de Una Transf.**
Especificaciones de Usabilidad	Especificaciones de Usabilidad	Sí	Sí	No
	Objetivos de Usabilidad			
Evaluación Heurística	Evaluación Heurística	Sí	Sí	Sí
Inspecciones	*Inspecciones de Conformidad con Estándares*	Sí	Sí	Sí
	Revisión de Guías	Sí	Sí	Sí
	Inspecciones de Consistencia	Sí	Sí	Sí
	Inspecc. de Usabilidad Colaborativas	Sí	Sí	Sí
Recorridos Cognitivos	Recorridos Cognitivos	Sí	Sí	No
Recorrido Pluralístico	Recorrido Pluralístico	Sí	Sí	Sí

Hemos identificado tres casos diferentes en la Tabla 6.5, cada uno de ellos identificado con un color de fondo distinto. El color de fondo gris más claro corresponde a las técnicas recomendadas para el desarrollo comercial según Ferré et al. [69][70] pero que según nuestro análisis no es posible incorporar en OSS aunque se realicen transformaciones. Un color de fondo gris oscuro corresponde a las técnicas que no son recomendadas para el desarrollo comercial (según Ferré et al. [69][70]) ni para los desarrollos OSS (según nuestro análisis). El color de fondo blanco pertenece a las técnicas que son recomendadas para el desarrollo comercial (según Ferré et al. [69][70]) y que pueden ser incorporadas en los desarrollos OSS si se les aplica una o varias transformaciones. A lo largo de esta sección ilustraremos con ejemplos cada uno de estos casos.

Como se observa en la Tabla 6.5, existen tres técnicas (con fondo gris claro) que son recomendadas para el desarrollo comercial pero que no es posible incorporar en OSS aunque se realicen una o varias transformaciones: *análisis competitivo*, *investigación contextual* y *observación etnográfica*. El *análisis competitivo* consiste en analizar productos existentes de forma heurística, de acuerdo a guías de usabilidad establecidas, y llevar a cabo test empíricos de estos productos con usuarios [138]. Esto implica realizar comparaciones con aplicaciones similares o, lo que es lo mismo, con aplicaciones de la competencia. En la comunidad OSS está mal visto fijarse en otras aplicaciones y más si son aplicaciones comerciales [95][108]. Por esta razón, no creemos adecuado incorporar la técnica análisis competitivo puesto que contraviene una característica intrínseca al modo de hacer las cosas en la comunidad OSS.

En la *investigación contextual* es necesario entrevistar al usuario mientras realiza sus tareas habituales en su entorno habitual de trabajo [149]. Sin embargo, en OSS no es posible realizar este tipo de entrevistas porque los usuarios están distribuidos por todo el mundo. Cualquier transformación va en contra del espíritu que la IPO otorga a esta técnica.

En la técnica *observación etnográfica* los investigadores procuran sumergirse en la situación sobre la cual quieren aprender algo [149]. Al igual que ocurre en la *investigación contextual*, no es posible que el experto en usabilidad se incorpore al entorno del usuario dado que el trabajo distribuido es una característica intrínseca de los desarrollos OSS. Por tanto, no es posible incorporar la técnica *observación etnográfica* en OSS. Cualquier transformación que se realizase sobre esta técnica va en contra de su esencia.

Otras técnicas (en fondo gris oscuro), relacionadas con ingeniería de requisitos, no son recomendadas para el desarrollo comercial y menos aún ser incorporadas en OSS aunque se realicen una o varias transformaciones. Ejemplos de estas técnicas son: *modelo operacional*, *familia de modelos GOMS (Goals, Operations, Methods and Selection Rules)* y *prototipos mago de Oz*.

El *modelo operacional* es una colección de varias influencias operacionales (por ejemplo, características de los usuarios) y contextuales (por ejemplo, aspectos del entorno físico de trabajo) que pueden jugar un rol en la usabilidad. Un modelo operacional es una forma de describir los aspectos relevantes de las plataformas hardware/software sobre las que va a funcionar el sistema [50]. Esta técnica no encaja en OSS porque las aplicaciones OSS son desarrolladas según los conocimientos de los desarrolladores y las plataformas donde más se sienten a gusto trabajando. Por tanto, no hay necesidad de realizar un modelo operacional. El objetivo de la técnica *familia de modelos GOMS* es la representación de los métodos o planes que elabora un usuario para alcanzar metas específicas. Los métodos se componen de una serie de pasos, formados por operaciones que el usuario realiza [149]. Es una técnica que requiere de una formación extensa y es compleja de aplicar. Por tales razones, no recomendamos su incorporación en OSS. En los *prototipos mago de Oz* un usuario interacciona con una pantalla, pero en vez de un software respondiendo a las acciones del usuario, un desarrollador está situado en otra pantalla respondiendo a las peticiones del usuario [149]. Esta técnica no encaja con la cultura de desarrollo OSS donde lo importante es desarrollar software, no simularlo [114][137][139].

La Tabla 6.6 recoge las técnicas de usabilidad relacionadas con la actividad de diseño en IS. Se sigue el mismo código de color de fondo que en la Tabla 6.5. En la actividad de diseño existe un nuevo caso (representado con un color gris más oscuro). Este nuevo caso corresponde a las técnicas que no son recomendadas para el desarrollo comercial (según Ferré et al. [69][70]) pero que según nuestro análisis si es posible incorporar en los desarrollos OSS gracias a alguna transformación.

Como se observa en la Tabla 6.6 existe sólo una técnica (con fondo gris claro) recomendada para el desarrollo comercial, pero no así para OSS: *análisis de impacto*. La técnica *análisis de impacto* implica considerar la importancia relativa de los problemas de usabilidad (identificados en la evaluación de la usabilidad) junto con el coste de desarrollo de las posibles soluciones [138]. En la comunidad OSS no es importante el coste de desarrollar o resolver problemas debido a la naturaleza voluntaria del trabajo de sus miembros [37][104][186][210]. Si bien la técnica *análisis de impacto* puede usarse para priorizar la solución a problemas de usabilidad, en OSS la priorización se realiza de manera democrática contraveniendo lo prescrito por la IPO para esta técnica [203].

Algunas técnicas de usabilidad (fondo gris medio), relacionadas con diseño, no son recomendadas para el desarrollo comercial y consideramos tampoco pueden ser incorporadas en OSS, como por ejemplo: *gramáticas* y *mapa de navegación*. La técnica *gramáticas* es aplicable únicamente a interfaces de usuario basadas en línea de comando, las cuales conforman un porcentaje muy bajo de las IUs que se diseñan en la actualidad. Además, son difíciles de seguir cuando crecen y resultan confusas para muchos usuarios [180]. Debido a su baja aplicabilidad y a lo complejas que pueden llegar a ser, consideramos que esta técnica no es apropiada para ser incorporada en OSS.

Tabla 6.6: Técnicas de la IPO Relacionadas con Actividades de Diseño Aplicables a OSS con Transformaciones.

Nombre Genérico de la Técnica de la IPO	**Nombre Técnica Usada en la Literatura**	**Recom. Ferré**	**Una Transf.**	**Más de Una Transf.**
Representaciones de Pantallas	Escenarios y Representaciones de Pantallas	No	Sí	Sí
Guía de Estilo del Producto	Guía de Estilo del Producto	Sí	Sí	Sí
Gramáticas	Gramáticas	No	No	No
UAN	UAN (User Action Notation)	No	No	No
TAG	TAG (Task-Action Grammars)	No	No	No
Árboles de Menús	Árboles de Menús	Sí	Sí	Sí
Diagramas de Transición de Estados de la Interfaz	Diagramas de Transición de Estados de la Interfaz	Sí	Sí	Sí
Diagramas de Estados de Harel	Diagramas de Estados de Harel	No	No	No
Modelo del Contenido de la Interfaz	Modelo del Contenido de la Interfaz	Sí	Sí	No
Mapa de Navegación	Mapa de Navegación entre Contextos	Sí	No	No
Prototipado	Prototipado	Sí	Sí	No
	Prototipos Escenario	Sí	Sí	Sí
Diseño Integrador	Diseño Integrador (Both-And Design)	No	Sí	No
Diseño Paralelo	Diseño Paralelo	No	Sí	Sí
Análisis de Impacto	Análisis de Impacto	Sí	No	No
Organización de la Ayuda según Casos de Uso	Organización de la Ayuda según Casos de Uso	Sí	Sí	Sí

La técnica *mapa de navegación* permite representar la arquitectura general de la interfaz de usuario modelando las relaciones entre contextos de interacción. Cada contexto se representa con un rectángulo y las flechas que los conectan representan posibles transiciones entre un espacio de interacción y otro [50]. La técnica *mapa de navegación* es compleja de aplicar y al ser una representación abstracta de la interfaz de usuario no encaja con la cultura OSS de desarrollar, de lo tangible (por ejemplo, el código fuente) [114][137][139].

Existen técnicas no recomendadas para el desarrollo comercial pero que es posible incorporar en los desarrollos OSS a través de alguna transformación (fondo gris oscuro), como por ejemplo: *representaciones de pantallas* y *diseño paralelo*. Las *representaciones de pantallas* especifican los contenidos de la parte visible de la interfaz de usuario [92]. Es una técnica muy útil cuando se necesita especificar con mucho detalle el trabajo a realizar, porque no es posible contar con un equipo multidisciplinar, centralizado e integrado, por ejemplo por lejanía geográfica. Esta técnica es perfecta para OSS porque los desarrolladores se encuentran distribuidos. En el *diseño paralelo* varios diseñadores trabajan en diseños similares de manera independiente [138]. Esta técnica encaja con la forma de trabajo de la comunidad OSS porque los desarrolladores trabajan independientemente, gracias a la arquitectura modular de sus aplicaciones [28][35][197].

La Tabla 6.7 presenta las técnicas de usabilidad relacionadas con las actividades de evaluación. Como se observa en la Tabla 6.7, existen tres técnicas que son recomendadas para el desarrollo comercial y no es posible incorporar en OSS (fondo gris más claro), aunque se realicen transformaciones: *medición del rendimiento*, *test de usabilidad en laboratorio* y *servicios de atención al usuario en línea*. En la técnica *medición del rendimiento* se cuantifican importantes aspectos del uso real de la aplicación, bien en un entorno controlado de laboratorio, bien en el

entorno habitual de trabajo [50]. La técnica *test de usabilidad en laboratorio* implica llevar a cabo tests en un lugar específico configurado especialmente para la realización de los tests de usabilidad [180]. La comunidad OSS no cuenta con los recursos necesarios para un laboratorio de usabilidad [23] y sus usuarios están distribuidos por todo el mundo, con lo cual se hace difícil acceder a su entorno. Por tanto, consideramos que estas dos técnicas no son posibles de incorporar en OSS. Los *servicios de atención al usuario en línea* no existen en la comunidad OSS porque la mayoría de proyectos OSS no tienen los recursos necesarios para contar con operadores telefónicos.

Tabla 6.7: Técnicas de la IPO Relacionadas con Actividades de Evaluación Aplicables a OSS con Transformaciones.

Nombre Genérico de la Técnica de la IPO	**Nombre Técnica Usada en la Literatura**	**Recom. Ferré**	**Una Transf.**	**Más de Una Transf.**
Evaluación Heurística	Evaluación Heurística	Sí	Sí	Sí
Inspecciones	*Inspecciones de Conf. con Estándares*	Sí	Sí	Sí
	Revisión de Guías	Sí	Sí	Sí
	Inspecciones de Consistencia	Sí	Sí	Sí
	Inspecciones de Usab. Colaborativas	Sí	Sí	Sí
Recorridos Cognitivos	Recorridos Cognitivos	Sí	Sí	No
Recorrido Pluralístico	Recorrido Pluralístico	Sí	Sí	Sí
Pensar en Voz Alta	Toma del Protocolo Verbal Concurrente	Sí	Sí	No
	Pensar en Voz Alta			
	Test Formales de Usabilidad (en las etapas iniciales)			
	Interacción Constructiva	Sí	Sí	No
	Test Retrospectivo	Sí	Sí	No
	Toma de Incidentes Críticos	Sí	Sí	Sí
	Método de Entrenamiento	Sí	Sí	No
Medición del Rendimiento	Tareas de Referencia	Sí	No	No
	Métricas de Rendimiento			
	Test Formales de Usabilidad (en etapas avanzadas)			
Información Post-Test	Información Post-Test	Sí	Sí	Sí
Test de Usabilidad en Laboratorio	Test en Laboratorio	Sí	No	No
	Laboratorios de Usabilidad			
	Test de Usabilidad y Laboratorios			
Test de Campo	Test de Campo	No	No	No
Grabación Vídeo	Grabación Vídeo	No	No	No
Grabación Audio	Grabación Audio	No	No	No
	Protocolo Verbal			
Registro del Uso	Instrumentación Interna de la Interfaz	Sí	Sí	No
	Registro del Uso			
	Registro Software			
	Registro Continuo del Rendimiento del Usuario			
	Registro de Pulsaciones en el Tiempo	Sí	Sí	No
	Registro de la Interacción	Sí	Sí	No
Evaluación por Control Remoto	Evaluación por Control Remoto	Sí	Sí	No

Tabla 6.7: Técnicas de la IPO Relacionadas con Actividades de Evaluación Aplicables a OSS con Transformaciones (continuación).

Nombre Genérico de la Técnica de la IPO	**Nombre Técnica Usada en la Literatura**	**Recom. Ferré**	**Una Transf.**	**Más de Una Transf.**
Test Remoto por Video-Conferencia	Test Remoto por Video-Conferencia	Sí	Sí	No
Observación Directa	Observación Directa	No	Sí	Sí
	Observación Aleatoria	No	Sí	Sí
Cuestionarios y Encuestas	Cuestionarios	Sí	Sí	No
	Cuestionarios y Encuestas			
	Encuestas			
Entrevistas	Entrevistas	Sí	Sí	No
	Entrevistas Estructuradas	Sí	Sí	No
	Entrevistas Flexibles	Sí	Sí	No
Focus Groups	Focus Groups	No	Sí	Sí
Registro del Uso	Instrumentación Interna de la Interfaz	Sí	Sí	No
	Registro del Uso Real			
	Registro Software			
	Registro Continuo del Rendimiento del Usuario			
	Evaluación Remota Instrumentada			
	Registro de Pulsaciones en el Tiempo	Sí	Sí	No
	Registro de la Interacción	Sí	Sí	No
	Monitores Software	Sí	Sí	No
Retroalimentación del Usuario	Retroalimentación del Usuario	Sí	Sí	Sí
	Buzón de Sugerencias o Reporte de Errores en Línea			
	Servicios de Atención al Usuario en Línea	Sí	No	No
	Foros	Sí	Sí	Sí
	Revistas y Conferencias para Usuarios	Sí	Sí	No
	Evaluación Remota Semi-Instrumentada	Sí	Sí	No

Algunas técnicas no son recomendadas para el desarrollo comercial y consideramos no podrían ser incorporadas en OSS de ninguna manera (fondo gris medio), por ejemplo: *grabación de vídeo* y *test de campo*. La técnica *grabación de vídeo* requiere el análisis de los vídeos grabados mientras los usuarios realizan los test de usabilidad realizados generalmente en laboratorios de usabilidad [92]. Esta técnica ha sido incorporada en un proyecto OSS [152]. Sin embargo, consideramos que esta técnica no es idónea para ser aplicada en OSS porque el análisis de los vídeos consume mucho tiempo y el trabajo en los proyectos OSS es realizado por voluntarios en su tiempo libre. En la técnica *test de campo* se requiere el acceso a los usuarios finales en el entorno previsto donde se usará la aplicación [50]. Sin embargo, acceder a los usuarios en su entorno de trabajo no es posible porque los usuarios están geográficamente distribuidos. Cualquier transformación va en contra de la esencia de la técnica.

La Tabla 6.8 presenta un resumen, para cada una de las actividades de IS (ingeniería de requisitos, diseño y evaluación), del número de técnicas de la IPO disponibles según el catálogo recopilado por Ferré et al. [68], el número de técnicas de la IPO que pueden recomendarse para su uso en desarrollos tradicionales según [69][70], el número de técnicas que hemos encontrado

en la literatura han sido incorporadas por OSS, el número de técnicas que proponemos podrían ser incorporadas en OSS en cualquier nivel de transformación (es decir, una transformación o una conjunción de varias transformaciones) y el número de técnicas que creemos no podrían ser incorporadas aún con las transformaciones. Nótese que todos los porcentajes en la Tabla 6.8 son calculados con base en el número de técnicas de la IPO disponibles según el catálogo recopilado por Ferré et al. [68]. Por ejemplo, el número de técnicas de la IPO relacionadas con la actividad de requisitos que han sido incorporadas por OSS en cualquier nivel de transformación es solo 2 con respecto del total disponible para esta actividad (es decir, 33).

Tabla 6.8: Resumen del Número de Técnicas de la IPO Incorporadas por OSS y que Podrían Serlo con las Transformaciones Propuestas.

Tipo de Actividad IS		**Nro. Técnicas de la IPO Disponibles**	**Técnicas Recomendadas en un Desarrollo Tradicional**		**Técnicas Incorporadas por OSS en Cualquier Nivel de Transfor.**		**Técnicas que Proponemos podrían ser Incorporadas en Cualquier Nivel de Transfor.**		**Técnicas que Consideramos No podrían ser Incorporadas en Ninguno de los dos Niveles de Transfor.**	
			Nro.	**%**	**Nro.**	**%**	**Nro.**	**%**	**Nro.**	**%**
Ingeniería de Requisitos		33	28	84,85	2	6,06	25	75,76	8	24,24
Diseño		16	9	56,25	0	0	10	62,50	6	37,50
Evaluación	Evaluación por Expertos	7	7	100	7	100	7	100	0	0
	Test de Usabilidad	16	13	81,25	11	68,75	11	68,75	5	31,25
	Estudios de Seguimiento de Sistemas Instalados	16	13	81,25	15	93,75	15	93,75	1	6,25
Totales		88	70	**79,55**	35	**39,77**	68	**77,27**	20	**22,73**

Como se observa en la Tabla 6.8, según el análisis realizado por Ferré et al. [69][70] no todas las técnicas de usabilidad son aplicables en un desarrollo no centrado en el usuario[2]. El porcentaje de técnicas de la IPO que pueden usarse en desarrollos tradicionales de IS es del 79,55 %. A continuación, analizaremos cuántas técnicas de usabilidad del universo de técnicas de la IPO (ver columna 2 de la Tabla 6.8) podrían incorporarse en los desarrollos OSS gracias a las transformaciones propuestas. Hemos realizado el análisis de las técnicas sobre el total de las técnicas (88), es decir, sobre todo el catálogo, no solamente sobre las técnicas utilizables en los desarrollos comerciales.

El porcentaje de técnicas de la IPO relacionadas con las actividades IS de ingeniería de requisitos y diseño que OSS ha incorporado con algún nivel de transformación (es decir, una transformación o una conjunción de varias transformaciones) es muy bajo, 6,06 % y 0 % respectivamente. El porcentaje de técnicas relacionadas con la evaluación incorporadas por OSS y reportadas en la literatura sube hasta casi el 85 %. Creemos que hay dos razones para esto. Por un lado, las técnicas de evaluación de la usabilidad son menos intrusivas que las técnicas relacionadas con requisitos y diseño. Es decir, incorporar técnicas de evaluación de la usabilidad no tiene un gran impacto en el trabajo de los desarrolladores. Basta con añadirlas al final del desarrollo. Por otro lado, las dos principales condiciones desfavorables

[2]Recuérdese que la IPO postula un tipo de desarrollo centrado en el usuario que no concuerda al 100 % con el desarrollo tradicional postulado por IS. Las diferencias entre ambos tipos de desarrollo desaconsejan el uso de algunas técnicas de la IPO en desarrollos de IS.

(es decir, necesidad de expertos en usabilidad y participación de los usuarios) que impiden que las técnicas de evaluación de la usabilidad sean incorporadas han logrado ser resueltas a través de transformaciones (discutidas en la sección 7.1). Si analizamos de forma global (ingeniería de requisitos, diseño y evaluación) el porcentaje de técnicas incorporadas por OSS con transformaciones, éste llega cerca del 40 % del total de las técnicas de usabilidad.

Sin embargo, según nuestro análisis consideramos que el 77,27 % del total de técnicas de la IPO disponibles en el catálogo [68], podrían ser incorporadas en OSS. Es decir, la aplicación sistemática de determinadas transformaciones a las técnicas de usabilidad permitirían incorporar la mayoría de las técnicas de usabilidad en los desarrollos OSS. En otras palabras, muchas de las técnicas de usabilidad presentan impedimentos para ser incorporadas en OSS, pero tales impedimentos pueden ser sorteados para el 77 % de las técnicas. Además, proponemos que el porcentaje de técnicas que pueden ser incorporadas gracias a una conjunción de varias transformaciones es poco más del 40 %. Este porcentaje es menor que el de técnicas que pueden sufrir una simple transformación por dos razones. En primer lugar, no todas las técnicas de usabilidad necesitan sufrir transformaciones múltiples, porque con la transformación simple es suficiente para su incorporación en OSS. En segundo lugar, las técnicas idóneas para sufrir transformaciones múltiples son aquellas que requieren de la participación de varios usuarios físicamente reunidos, y no todas las técnicas de usabilidad sufren esta condición desfavorable.

El porcentaje de técnicas de usabilidad que no podrían ser incorporadas aunque se realicen una o varias transformaciones, alcanza casi al 23 % del total de técnicas disponibles del catálogo [68]. Son cuatro las razones por las que el 23 % de las técnicas de usabilidad no pueden ser incorporadas en los desarrollos OSS: algunas técnicas tienen una baja aplicabilidad (por ejemplo, *gramáticas*); existen técnicas a las cuales no es posible realizar transformación alguna, porque cualquier transformación sobre la técnica va en contra de su esencia (por ejemplo, *observación etnográfica*); algunas técnicas son complejas de aplicar (por ejemplo, *familia de modelos GOMS*); ciertas técnicas no encajan en OSS, porque van en contra de su filosofía (por ejemplo, *análisis competitivo*) o no se ajustan a sus intereses (por ejemplo, *prototipos mago de Oz*).

Resumiendo, según nuestro análisis, consideramos que en los desarrollos OSS podrían ser incorporadas cerca del 80 % del universo de técnicas de usabilidad (según el catálogo recopilado por Ferré et al. [68]).

6.6. Transformación de la Técnica Personas para OSS

En esta sección se transforma la Técnica Personas de Cooper et al. [52] para comprobar que a partir de las condiciones desfavorables de una técnica de usabilidad determinadas previamente es posible aplicar el conjunto pertinente de transformaciones propuestas en este trabajo para poder ser usada en los desarrollos OSS.

La Técnica Personas, perteneciente a la actividad de análisis de usuarios de la IPO, permite recolectar, analizar y sintetizar la información relacionada con los usuarios que interactuarán con el sistema software y, por tanto, ayuda a centrar el análisis y diseño del software en las características y objetivos del usuario final del producto. Esta técnica recoge datos sobre los usuarios, comprende sus características, define personas ficticias sobre la base de esta comprensión y favorece que los desarrolladores se centren en tales personas durante todo el proceso de desarrollo de software. La principal condición desfavorable de esta técnica detectada es la necesidad de la participación de los usuarios para realizar las entrevistas etnográficas que involucra la técnica original y sintetizar las variables conductuales de estos usuarios.

Ante esta condición desfavorable, se ha propuesto que los desarrolladores hipoteticen sobre la conducta de los usuarios (que en muchos casos pueden ser ellos mismos) mediante un proceso de introspección y asimismo modelen conductas a partir de lo que ellos piensan que harían los usuarios involucrados o bien solicitar directamente a los usuarios descripciones de sus conductas a través de correos electrónicos. Todo esto para determinar las hipótesis de las personas a las que irá dirigido el sistema software y definir las variables conductuales de las mismas.

La Tabla 6.9 corresponde a la adaptación que se hizo de la Técnica Personas de Cooper. Nuestra propuesta consiste en un grupo de actividades que en conjunto llevan a la creación de personas.

Tabla 6.9: Adaptación de la Técnica Personas de Cooper.

Pasos de la Técnica	Descripción del Paso de la Técnica	Descripción Ligera Modificación
1. Identificación de Variables Conductuales	En este paso se listan los distintos aspectos de comportamiento observados. Este listado se conoce como el conjunto de variables conductuales. Estas variables conductuales son útiles en el desarrollo de arquetipos de usuarios.	- Solicitar a los desarrolladores descripciones de sus usuarios. - Estas descripciones siguen una plantilla que se suministra (por ejemplo, estructura del documento fundación de personas) a los desarrolladores. También se les pide que realicen storyboards. - Las descripciones también pueden solicitarse por correo electrónico a los propios usuarios (los más cercanos) para que se describan.
2. Mapeo de Sujetos Entrevistados a Variables Conductuales	Una vez que se ha identificado la lista de variables conductuales obtenidas a través de las entrevistas a los sujetos, el paso a seguir consiste en el mapeo de cada sujeto entrevistado con cada variable conductual. Estas variables conductuales representan o un rango continuo de conductas o bien múltiples selecciones discretas.	
3. Identificación de Patrones de Comportamiento Significativos	En este paso se deben observar grupos de sujetos que se encuentran ubicados en múltiples rangos o variables. Un grupo de sujetos que agrupa de seis a ocho variables diferentes podría representar un patrón de comportamiento significativo, que formará la base de una persona.	
4. Síntesis de las Características y los Objetivos relevantes	En este paso se deben sintetizar los detalles de cada uno de los patrones de comportamiento significativos identificados en el paso anterior. Esta síntesis debe describir el ambiente de uso potencial, un día de trabajo típico y relaciones relevantes con los demás. Para realizar esta síntesis es suficiente con breves puntos que describan las características de los comportamientos.	- Con base en las descripciones suministradas (tanto por los desarrolladores como por los usuarios) un experto en usabilidad (o alguien que juegue ese rol pero con conocimientos en la IPO) elabora el Documento Fundación de Personas.
5. Chequeo de Completitud y Redundancia	En este paso se deben comprobar los mapeos, las características de las personas y sus objetivos para determinar si hay algún hueco que necesite ser cubierto. Es importante asegurarse que el conjunto de personas es completo y que cada persona es significativamente distinta de las demás.	- Un grupo de desarrolladores se reúne remotamente para validar si el Documento Fundación de Personas no omite información importante que describa o caracterice las personas.
6. Expansión de la Descripción de Atributos y Comportamientos	La narrativa en tercera persona es muy útil para transmitir las actitudes, necesidades, y problemas de las personas a los otros miembros del equipo. Una típica descripción de una persona debe ser una síntesis de los más importantes detalles observados durante la investigación, relevantes a esta persona.	- Un grupo de desarrolladores reunido remotamente realiza la Narrativa.
7. Designación de Tipos de Personas	El desarrollo requiere un objetivo. Típicamente, el objetivo más especifico es el mejor. El objetivo es encontrar una sola persona cuyas necesidades y objetivos puedan ser completamente satisfechas por una sola interfaz sin defraudar a ninguna de las otras personas. Esto se lleva a cabo a través de un proceso de designar tipos de personas. Existen seis tipos de personas: primaria, secundaria, suplementaria, cliente, servida y negativa.	- La designación de tipos de personas (primaria, secundaria) puede discutirse en comunidad, a través de un foro. - Esta designación es realizada en la conferencias cuando se reúnen los usuarios y los desarrolladores.

La Tabla 6.10 presenta todas las actividades con sus objetivos y productos asociados. Las nuevas actividades o productos (no existentes en ninguna versión anterior de Personas) se encuentran sombreadas en gris. Nótese que incluso para las actividades existentes ha sido necesario realizar una labor de identificación y definición tanto de sus objetivos como de sus productos utilizando como fuentes todas las versiones existentes de Personas [24][51][52][53][150][151]. Esto se debe a que se ha encontrado que además la Técnica Personas adolece de dos carencias típicas de las técnicas IPO: no presenta una definición detallada tanto de las actividades como de los productos que la componen. Estos problemas provocan que la introducción de Personas en desarrollos para enriquecer las actividades de la ingeniería de requisitos resulte excesivamente compleja y ambigua para los profesionales del desarrollo de software.

Tabla 6.10: Conjunto de Actividades Propuestas para Personas.

Actividades		**Rfcias. que Mencionan Tareas Similares**	**Objetivos**	**Productos**
Actividad 1: Elaborar Hipótesis	Subactividad 1.1: Identificar Posibles Personas	[150]	Formular hipótesis iniciales acerca de las posibles *personas* que serán creadas.	• Lista de Hipótesis de Personas
	Subactividad 1.2: Modelar Posibles Personas		Basados en estas hipótesis, realizar esbozos de motivaciones, habilidades y sentimientos de las posibles personas para determinar datos de comportamientos de las mismas.	• Bocetos, Storyboards
Actividad 2: Identificar Variables Conductuales		[52]	Obtener la lista completa de variables conductuales.	• Síntesis de las Entrevistas
Actividad 3: Mapear los Sujetos Modelados a las Variables Conductuales	Subactividad 3.1: Identificar los Rangos de Valores de las Variables Conductuales		Para cada una de las variables conductuales encontradas identificar su rango de valores posibles.	• Rangos de Valores de las Variables Conductuales
	Subactividad 3.2: Mapear los Sujetos Modelados	[52]	Representar exactamente la forma en que múltiples sujetos se agrupan con respecto a cada una de las variables conductuales significativas.	• Mapeo de Sujetos Entrevistados
Actividad 4: Identificar Patrones de Conductas Significativos		[52, 150]	Identificar agrupamientos de sujetos particulares, que ocurren en múltiples rangos o variables.	• Patrones de Conductas Significativos
Actividad 5: Sintetizar Características y Objetivos Relevantes		[52, 150, 151]	Sintetizar las características y objetivos relevantes. Describir las características de las *personas*.	• Documento Fundación de Personas
Actividad 6: Comprobar la Redundancia y la Completitud		[52]	Comprobar los mapeos, características de las personas y sus objetivos.	• Documento de Validación

Tabla 6.10: Conjunto de Actividades Propuestas para Personas (continuación).

Actividades		Rfcias. que Mencionan Tareas Similares	Objetivos	Productos
Actividad 7: Expandir la Descripción de Atributos y Conductas		[52, 150]	Transmitir las actitudes de las *personas*, su personalidad, necesidades, y problemas a otros miembros del equipo.	• Narrativa
Actividad 8: Designar Tipos de Personas	Subactividad 8.1: Seleccionar Representantes de Personas para Educir Requisitos	[52]	Priorizar las *personas* creadas para determinar quién debe ser el objetivo de diseño primario, es decir encontrar una sola *persona* primaria del conjunto cuyas necesidades y objetivos pueden ser completa y felizmente satisfechos por una sola interfaz.	• Asociación del Tipo de Persona.
	Subactividad 8.2: Enriquecer el Sistema con Personas Secundarias		Determinar que necesidades de la *persona* secundaria pueden enriquecer el sistema.	(Especificación de Requisitos de Software enriquecida)
Actividad 9: Elaborar los Casos de Uso		[151]	Realizar el Diagrama de Caso de Uso con Notas. Realizar la Especificación de los Casos de Uso.	• Diagrama de Caso de Uso con Notas • Especificación de Caso de Uso
Actividad 10: Implementar y Evaluar los Prototipos	Subactividad 10.1: Realizar las Maquetas		Elaborar las Maquetas.	• Maquetas
	Actividad 10.2: Evaluar las Maquetas		Validar las Maquetas.	• Documento de Evaluación de las Maquetas

En la nueva actividad inicial Elaborar Hipótesis se prepone la Lista de Hipótesis de Personas que se crearán, así como desarrollar y realizar storyboards de los futuros posibles usuarios del sistema para así recolectar la información necesaria para llevar a cabo las demás actividades. Con las hipótesis de personas se pretenden identificar variables que puedan distinguir a los usuarios basados en sus necesidades y comportamientos o conductas. En la Figura 6.2 mostramos un ejemplo de una Lista de Hipótesis de Personas. La identificación de hipótesis de personas es la primera actividad que debe realizarse hacia la identificación y síntesis de personas.

En la tarea ya existente de Identificar Variables Conductuales, Cooper et al. [52] establecen que debe generarse una lista de variables conductuales a partir de los distintos aspectos de las conductas, en esta transformación de la técnica, modeladas, intuidas y razonadas por los desarrolladores OSS, a fin de determinar conductas comunes y sus rangos, presentados en los documentos Lista de Variables Conductuales y Rango de Valores de las Variables Conductuales propuestos en este trabajo.

Hipótesis	Personas	Justificación
H0	Tercera Edad, Jubilados	Las personas de la tercera edad y los jubilados disponen de tiempo suficiente para realizar distintos viajes durante todo el año.
H1	Profesionales, Empleados	Los profesionales y empleados en general cuentan con unos ingresos que en pequeña o mediana escala les permiten realizar viajes.
H2	Agentes de Viajes, Hombres de Negocios	Debido a su ocupación estas personas requieren viajar muy seguido.

Figura 6.2: Lista de Hipótesis de Personas para el Sistema Web de Reserva de Billetes Aéreos.

Según Personas [21][52], las variables conductuales son tipos de comportamiento que varían a través de un espectro. Cooper [52], menciona las variables conductuales de tipo: Actividades, Actitudes, Aptitudes, Motivaciones y Capacidades. Las variables conductuales relevantes deben ser obtenidas por el desarrollador para cada sistema software pues dependen del dominio y de la tarea. A menudo las variables conductuales se encuentran de manera implícita en cada una de las Hipótesis de Personas creadas en la actividad inicial donde se realizó la investigación y recolección de información. Por ejemplo, si una de las hipótesis corresponde a un estudiante universitario, implícitamente está que su edad se encuentra comprendida entre los 18 y 25 años, y que cuenta con ciertos conocimientos de informática, por lo que las variables implícitas en este caso son: edad y experiencia con el uso de ordenadores. La Figura 6.3 ilustra un ejemplo de un Listado de Variables Conductuales.

Variables Conductuales Observadas
Actitud hacia la tecnología de la información
Experiencia con el uso de ordenadores/internet
Frecuencia de compra
Lugar donde accede a internet
Dispositivo desde donde accede a internet
Motivación de la compra
Nivel educativo
Deseo para viajar
Edad
Profesión u oficio
Finalidad del viaje

Figura 6.3: Ejemplo de Lista de Variables Conductuales para el Sistema Web de Reserva de Billetes Aéreos.

En la tarea ya existente de Mapear los Sujetos Modelados a las Variables Conductuales, proponemos una nueva actividad para identificar los posibles rangos de valores que cada una de las variables conductuales puede tomar. Por ejemplo, la variable conductual *Actitud hacia la Tecnología* puede tomar valores desde cauto y defensivo hasta experimental y curioso. La identificación de estos rangos de valores de las variables proponemos sea realizada con base en la síntesis de storyboards realizados en la primera actividad. La Figura 6.4 ilustra un ejemplo del documento Rangos de Valores de las Variables Conductuales para un sistema Web de reserva de billetes aéreos.

Variable	Escala
Actitud hacia la tecnología de la información	Cauto, defensivo ←→ Experimental, Curioso
Experiencia con el uso de Ordenadores/Internet	Infrecuente ←→ Frecuente
Frecuencia de Compra	Infrecuente ←→ Frecuente
Lugar donde accede a Internet	En Casa ←→ Oficina
Dispositivo desde donde accede a Internet	Ordenador de Sobremesa ←→ Móvil, PDA
Motivación de la Compra	Cazando Ofertas ←→ Buscando justo el ítem adecuado
Nivel Educativo	Graduado de la ESO ←→ Universitario
Deseo para Viajar	Ama viajar ←→ Detesta viajar
Edad	Entre 20 y 30 ←→ Mayor de 61
Profesión u Oficio	Ama de Casa ←→ Profesional
Finalidad del Viaje	Placer ←→ Negocio

Figura 6.4: Ejemplo de Rangos de Valores de las Variables Conductuales para el Sistema Web de Reserva de Billetes Aéreos.

Teniendo los rangos de valores definidos, se puede realizar más fácilmente el mapeo de los sujetos modelados. Cooper [52] describe este mapeo como la ubicación para cada variable conductual, de cada uno de los sujetos entrevistados en uno de los valores. La precisión de este mapeo no es crítica pero sí lo es identificar la ubicación de los sujetos entrevistados en relación con cada uno de los demás sujetos. No hay a menudo ninguna forma de medir la precisión de este mapeo; se debe confiar en la percepción de la persona que lo realiza, quien se basa en las observaciones realizadas a los sujetos entrevistados. Personas [99][151] establece que el mapeo pretende mostrar cómo se posiciona cada sujeto entrevistado en relación con las variables conductuales seleccionadas. En esta transformación de la Técnica Personas no existen sujetos entrevistados, pero sí los desarroladores han pensado sobre lo que ellos intuyen serán los comportamientos de los posibles usuarios del sistema, y ahora tienen que pensar en la ubicación exacta de los sujetos modelados en relación con cada uno de los demás sujetos. La Figura 6.5 muestra un fragmento de un mapeo realizado para el sistema Web de reserva de billetes aéreos.

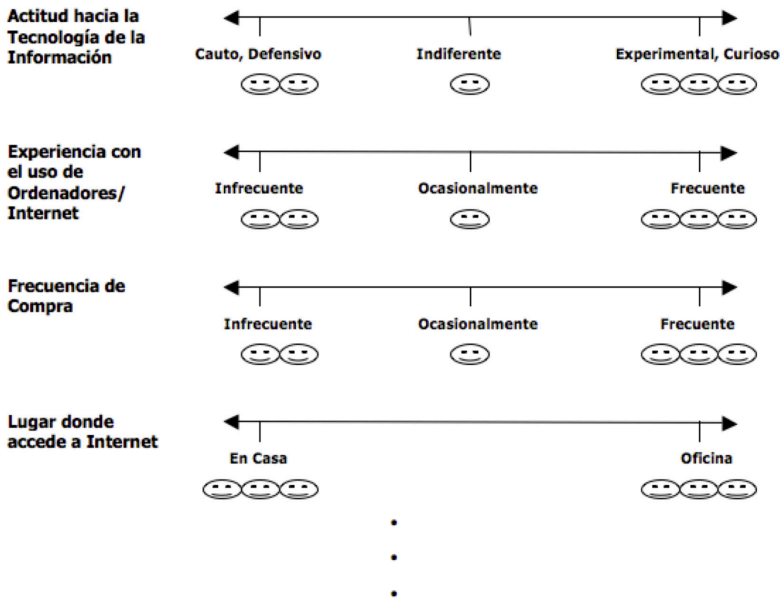


Figura 6.5: Fragmento del Mapeo de Sujetos Entrevistados para el Sistema Web de Reserva de Billetes Aéreos.

La tarea ya existente de Identificar Patrones de Conductas Significativos, tiene como objetivo [52] observar agrupamientos de sujetos particulares que ocurren sobre múltiples rangos o variables. Un conjunto de sujetos que se agrupan entre seis y ocho variables diferentes representará un patrón de conducta significativo que formará la base de una persona [82]. Para que un patrón sea válido, debe haber una relación lógica o causal entre las conductas agrupadas [82], y no una correlación falsa o espontánea. Por ejemplo, hay una clara relación lógica si los datos muestran que la gente que regularmente compra CDs, también le gusta bajar archivos MP3; pero no hay probablemente ninguna relación lógica si los datos muestran que a los entrevistados que frecuentemente compran CDs, también les gusta coleccionar estampillas. Proponemos obtener de esta actividad el producto Patrones de Conductas Significativos. La Figura 6.6 ilustra un ejemplo del documento Patrones de Conductas Significativos para el sistema Web de reserva de billetes aéreos.

Durante la tarea ya existente de Sintetizar Características y Objetivos Relevantes para cada patrón de conducta significativo identificado, proponemos que se deben sintetizar los detalles de los datos obtenidos a través de storyboards diseñados y realizados en la Actividad Elaborar Hipótesis, y de la síntesis de las variables conductuales determinadas en la Actividad Identificar Variables Conductuales. Es decir, describir el potencial ambiente de uso, un día de trabajo típico (u otro periodo de tiempo importante), frustraciones, y relaciones relevantes con otros [82]. Es suficiente, con la especificación de puntos breves que describan las características de las conductas. Se deben incluir las conductas observadas como sea posible. Sin embargo, demasiada biografía de ficción, idiosincrasia, etc., son una distracción y hacen las personas menos creíbles [82]. Nótese que se está creando una herramienta de diseño, no un esbozo de un personaje para una novela. Solamente datos concretos pueden apoyar el desarrollo y las decisiones de negocio que finalmente el equipo implementará [52][150]. Como resultado de esta actividad proponemos crear, para cada una de las personas identificadas, el Documento Fundación de Personas. Cuando se está decidiendo qué características incluir en los documentos fundación, se debe pensar en el tipo de información que será útil al equipo de desarrollo. En la Figura 6.7, se presenta un fragmento de la estructura e información que proponemos para el Documento Fundación de Personas.

La tarea ya existente para Comprobar la Redundancia y la Completitud, se realiza [52] para buscar lagunas de información y conocimiento que sea necesario cubrir, para lo cual es posible que se requiera de una investigación adicional, que puede llevar a encontrar conductas que no se encuentran en los ejes conductuales, y esto impactaría en todas las demás actividades. Se puede también necesitar chequear las notas para ver si hay alguna persona que se necesite adicionar para satisfacer las suposiciones o peticiones de los implicados. Para llevar a cabo esta validación, se deben responder preguntas como [52]: ¿Todas las personas creadas son significativamente distintas? o ¿Todas las personas creadas representan suficientemente la diversidad de conductas y necesidades del mundo real? Si se encuentra que dos personas parecen variar sólo en datos sociodemográficos, puede decidirse eliminar una de las personas redundantes o profundizar en las características de las personas para diferenciarlas aún más. Cada persona debe diferenciarse de todas las demás en al menos una conducta significativa. Si se ha hecho un buen trabajo de mapeo, esto no debe ser un problema. Se debe asegurar que el conjunto de personas sea distinto y completo y se debe mantener un conjunto manejable de personas. Como producto de esta actividad, proponemos crear el Documento de Validación. La Figura 6.8, muestra una posible estructura de este documento.

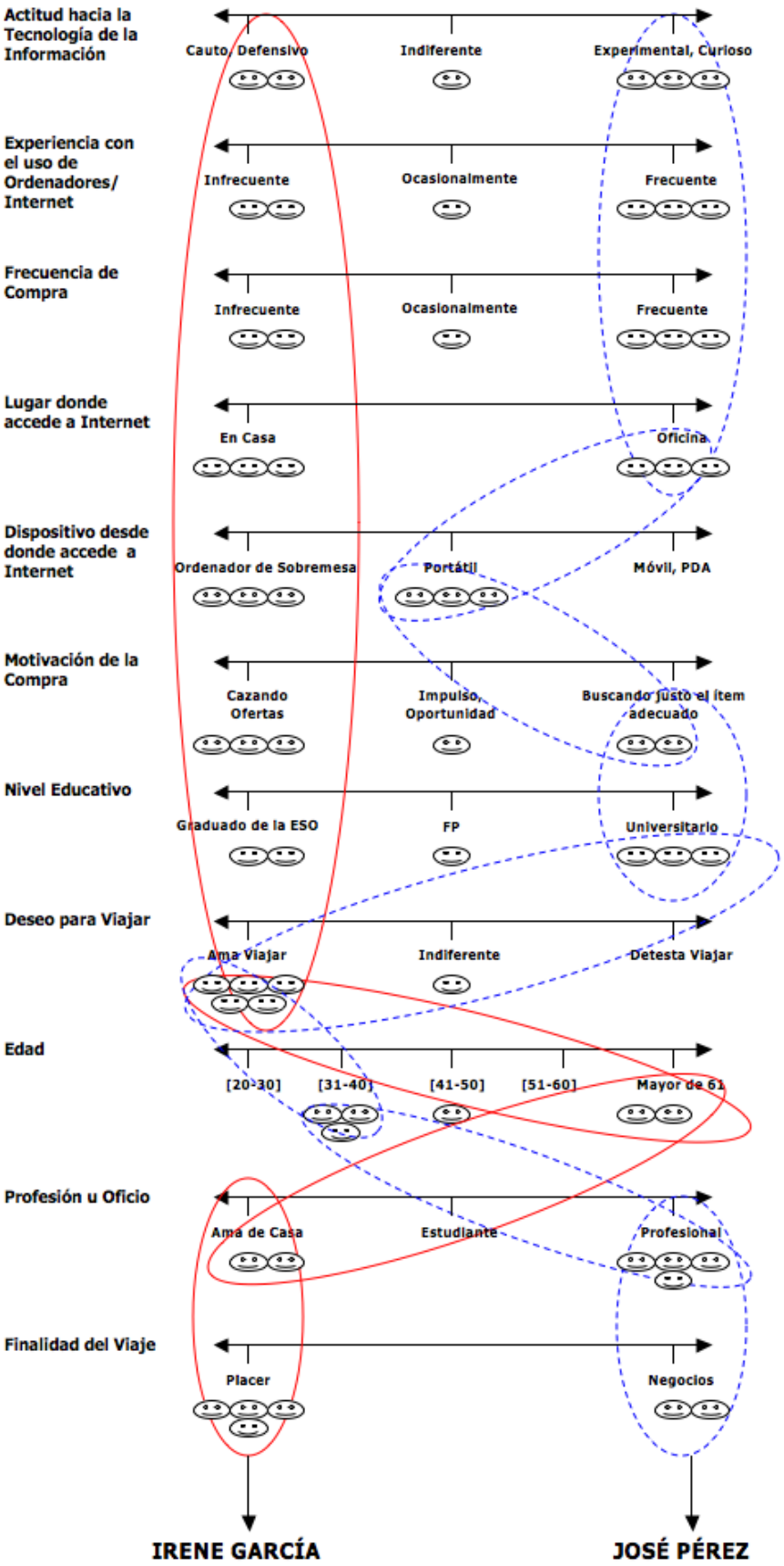
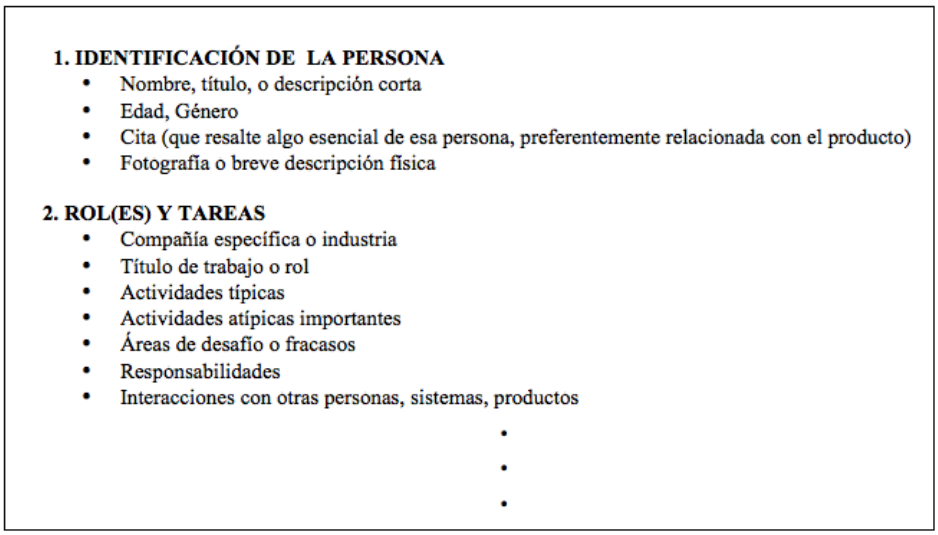


Figura 6.6: Ejemplo de Patrones de Conductas Significativos para el Sistema Web de Reserva de Billetes Aéreos.



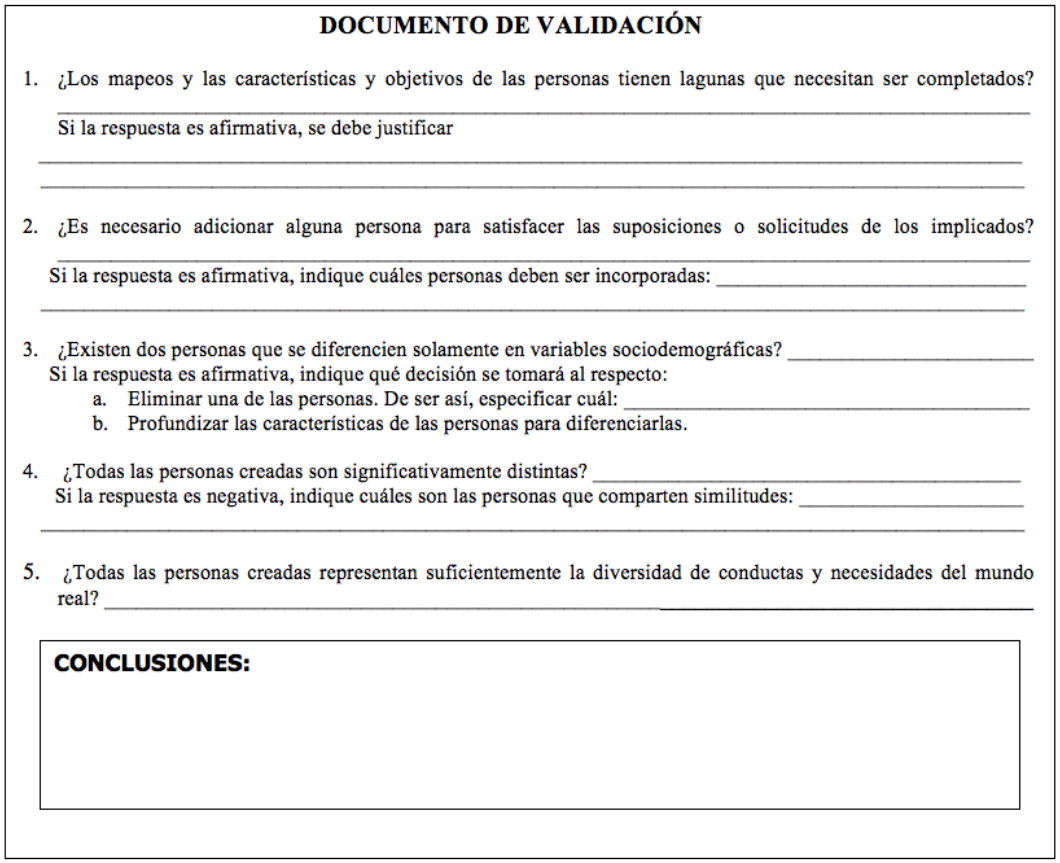
1. IDENTIFICACIÓN DE LA PERSONA

- Nombre, título, o descripción corta
- Edad, Género
- Cita (que resalte algo esencial de esa persona, preferentemente relacionada con el producto)
- Fotografía o breve descripción física

2. ROL(ES) Y TAREAS

- Compañía específica o industria
- Título de trabajo o rol
- Actividades típicas
- Actividades atípicas importantes
- Áreas de desafío o fracasos
- Responsabilidades
- Interacciones con otras personas, sistemas, productos

Figura 6.7: Fragmento del Documento Fundación de Personas.



DOCUMENTO DE VALIDACIÓN

1. ¿Los mapeos y las características y objetivos de las personas tienen lagunas que necesitan ser completados? _____
   Si la respuesta es afirmativa, se debe justificar
   _____
   _____
2. ¿Es necesario adicionar alguna persona para satisfacer las suposiciones o solicitudes de los implicados? _____
   Si la respuesta es afirmativa, indique cuáles personas deben ser incorporadas: _____
   _____
3. ¿Existen dos personas que se diferencien solamente en variables sociodemográficas? _____
   Si la respuesta es afirmativa, indique qué decisión se tomará al respecto:
   a. Eliminar una de las personas. De ser así, especificar cuál: _____
   b. Profundizar las características de las personas para diferenciarlas.
4. ¿Todas las personas creadas son significativamente distintas? _____
   Si la respuesta es negativa, indique cuáles son las personas que comparten similitudes: _____
   _____
5. ¿Todas las personas creadas representan suficientemente la diversidad de conductas y necesidades del mundo real? _____

CONCLUSIONES:

Figura 6.8: Estructura Propuesta para el Documento de Validación.

El Documento Fundación de Personas que proponemos, apunta a la esencia de conductas complejas, pero deja muchos aspectos en forma implícita. Por esta razón, existe una tarea para Expandir la Descripción de Atributos y Conductas donde para cada una de las personas se crea una Narrativa [52][150], que consiste en un documento de una página de longitud donde se describe a la persona y un día típico de trabajo en su vida. La mejor narrativa es aquella que introduce rápidamente a la persona en términos de su trabajo o estilo de vida y brevemente esboza un día en su vida, incluyendo preocupaciones e intereses que tienen implicaciones directas sobre el producto [150]. La narrativa debe expresar lo que la persona

está buscando en el producto a través de una conclusión. En la Figura 6.9 se muestra un ejemplo de una narrativa.

PERSONA PRIMARIA: IRENE LA COMPRADORA POR COMPARACIÓN

Irene (62) trabajaba como secretaria en la Universidad Complutense de Madrid. Pedro, su marido de 30 años, se retiró de su trabajo que llevaba en una pequeña tienda doméstica. Ellos siempre habían querido viajar luego de jubilarse, y ahorraron cuidadosamente para que pudieran permitirse el lujo de hacerlo. El primer año después de la jubilación, ellos se unieron a un Tour al campo italiano. Esto les dejó deseando viajar más. Este año, ellos desean ir de nuevo a Italia (pero esta vez solos). El viaje del último año costó demasiado y ellos quieren controlar los costos.

Su hijo ha estado diciéndoles sobre las baratas tarifas que pueden conseguir en Internet, si se busca duro. Irene ha estado pasándose mucho tiempo mirando sitios de viaje, intentando encontrar billetes para ir a Roma y hoteles para quedarse una vez lleguen allí. Ellos quieren visitar algunas viñas e ir al Vaticano por un día.

Irene es una pintora aficionada y está interesada en visitar sitios asociados con el arte. Ella también es aficionada a la comida italiana, pero no todo el tiempo. Le gusta quedarse en áreas turísticas para que pueda conseguir comida española fácilmente cuando ella quiera.

Cuando ella busca los sitios de viaje, busca tarifas baratas. Ella es flexible sobre las fechas. Mientras busca billetes aéreos, también busca hoteles. Planeando su itinerario, ella busca información de viaje sobre varias ciudades que está pensando visitar. No le importa si termina tomando un poco de tiempo en encontrar lo que quiere, pero se siente muy frustrada cuando el sistema no le da toda la información que está buscando. Por ejemplo, le gusta saber cuánto tiempo exactamente tomará el vuelo, de cuánto será el impuesto, y lo que pagará por cambiar o cancelar un billete. No le gusta hacer clic en numerosos enlaces para obtener cada pieza de información.

Figura 6.9: Ejemplo de Documento Narrativa para el Sistema Web de Reserva de Billetes Aéreos.

Conocer la audiencia que usará el software, ayuda a los desarrolladores a tener un objetivo de desarrollo concreto y explícito. Las personas creadas representan candidatos para este objetivo. La interfaz principal del sistema software se debe diseñar para el uso del sistema por una persona. Es necesario priorizar las personas creadas para determinar quién debe ser el objetivo principal de desarrollo. Se debe encontrar una sola persona del conjunto cuyas necesidades y objetivos pueden ser completamente y felizmente satisfechas por una sola interfaz sin desencantar a ninguna de las otras personas. Para llevar a cabo esto, se realiza [52] un proceso de designación de tipos de personas, teniendo en cuenta toda la información recolectada en las actividades anteriores. Este proceso se lleva a cabo en la Actividad Designar Tipos de Personas. Los dos principales tipos de personas que considera Cooper [52] son las primarias y las secundarias. Las primarias son aquellas que representan el objetivo primario de desarrollo del software, y las secundarias son las personas que tienen algunas necesidades específicas adicionales que no pueden ser cubiertas por la interfaz de la persona primaria, pero que no perturban las funcionalidades que brinda la interfaz a la persona primaria. Para estas personas se genera interfaces alternativas. Por ejemplo, perfiles de uso para usuarios novatos o expertos.

Hemos definido una nueva actividad que liga el trabajo realizado con Personas con el resto del desarrollo: Elaborar los Casos de Uso donde se componen los casos de uso a partir de los documentos Fundación de Personas y Narrativa, y el conocimiento del usuario adquirido a través de todas las actividades anteriormente realizadas. Como resultado de esta actividad proponemos realizar el Diagrama de Caso de Uso con Notas. Este diagrama tiene como base el diagrama de caso de uso tradicional, al cual se le adiciona una breve descripción de cada una

de las personas involucradas en el caso de uso. Proponemos que la descripción de la persona contenga una breve reseña para aspectos como, el Nombre y Tipo de Persona, así como una nota informativa sobre el caso de uso.

Finalmente, proponemos una nueva actividad para Implementar y Evaluar los Prototipos donde construir las maquetas y evaluarlas. Para la elaboración de las maquetas, proponemos tener en cuenta los casos de uso construidos en la actividad anterior, y el conocimiento del usuario adquirido a través de las actividades de Personas. La evaluación proponemos sea realizada o bien en el propio entorno donde los posibles usuarios realizan sus tareas habitualmente o bien en salas de reuniones adecuadas donde se puedan reunir los posibles usuarios y el evaluador.

6.7. Conclusiones

Este capítulo presenta nuestra investigación sobre cómo permitir el uso de técnicas de usabilidad en los desarrollos OSS. Proponemos un marco que permite la integración de ciertas técnicas de usabilidad en los desarrollos OSS. Dicho marco está conformado por todas las posibles transformaciones que deben sufrir las técnicas de usabilidad para poder ser aplicadas en OSS. Tras identificar y analizar cuáles técnicas de usabilidad están siendo incorporadas en los desarrollos OSS, nos hemos dado cuenta que es posible generalizar muchas de las transformaciones de modo que otras técnicas (otros proyectos OSS) puedan beneficiarse de ellas. No todas las técnicas de usabilidad pueden sufrir todas las adaptaciones. La transformación depende de la idiosincrasia de la técnica. Por tal razón, el marco también propone para cada técnica y sus características intrínsecas y condiciones desfavorables asociadas qué adaptaciones pueden ser realizadas a la misma.

Con el marco que proponemos, cualquier persona que quiera aplicar una técnica de usabilidad en OSS puede elegir la técnica que necesita dependiendo de su desarrollo y de su proyecto y aplicar alguna de las transformaciones sugeridas. De esta manera, logrará que una técnica que antes no se podía aplicar directamente en OSS, porque no se adapta a las circunstancias de los desarrollos OSS, pueda hacerlo gracias a nuestro marco. Como prueba de concepto, hemos realizado la transformación de la Técnica Personas para su aplicación adecuada en los proyectos OSS.

En este trabajo de investigación analizamos las condiciones desfavorables que impiden que las técnicas de usabilidad sean incorporadas por OSS. Estas condiciones pueden ser clasificadas en tres grupos. En primer lugar, aquellas técnicas donde es necesario contar con un experto en usabilidad y en la mayoría de proyectos OSS la participación de expertos es poco común. En segundo lugar, las técnicas donde es necesario la participación presencial de los usuarios y en la comunidad OSS todos sus miembros se encuentran geográficamente distribuidos. En tercer lugar, las técnicas que requieren de una preparación previa o de muchos pasos para su ejecución. En la comunidad OSS el trabajo que se realiza es completamente voluntario y desarrollado en el tiempo libre de sus miembros.

Para sortear las condiciones desfavorables, proponemos realizar ciertas transformaciones a las técnicas de usabilidad. Estas transformaciones se dan en dos niveles. En el primero, se encuentran transformaciones unitarias (por ejemplo, los usuarios son familiares y amigos de los desarrolladores y no usuarios representativos de la aplicación). En el segundo, están las transformaciones que son una conjunción de varias transformaciones. Estas transformaciones se basan en dos de los principios de la cultura OSS: discutir en comunidad y el trabajo voluntario de sus miembros.

Algunas de las condiciones desfavorables cuentan con una familia de transformaciones, es decir, existen varias soluciones alternativas. La participación de los usuarios es la condición desfavorable con la mayor familia de transformaciones. Consideramos que esto se debe a dos razones. En primer lugar, por la importancia que tienen los usuarios. Esta importancia radica en que son ellos finalmente quienes determinan el éxito o fracaso de una aplicación y si esta es o no usable. En segundo lugar, es importante para la comunidad OSS que todos sus miembros (incluidos los usuarios) puedan participar de alguna manera en el desarrollo o mejora de sus aplicaciones.

Consideramos que las transformaciones propuestas pueden ser aplicadas a otras técnicas de usabilidad que presenten las mismas condiciones desfavorables. Con el objetivo de determinar cuáles técnicas podrían ser incorporadas en OSS, gracias a las transformaciones, realizamos un análisis de todas las técnicas de usabilidad (según el catálogo tomado como base). En nuestro análisis, hemos considerado lo que prescribe la IPO para cada una de las técnicas, el escenario típico en el que desarrolla la comunidad OSS y las condiciones desfavorables de cada técnica. Según el análisis, consideramos que cerca del 80 % de las técnicas de usabilidad podrían ser incorporadas en OSS con transformaciones unitarias. Sin embargo, sólo cerca del 40 % de las técnicas podrían ser adaptadas con una conjunción de transformaciones. Las técnicas de usabilidad candidatas para ser incorporadas en OSS gracias a una conjunción de transformaciones son todas aquellas que requieran de la participación de varios usuarios físicamente reunidos.

Los resultados de nuestra investigación sugieren que a pesar de los impedimentos que presentan las técnicas de usabilidad es posible incorporar un buen porcentaje de estas técnicas en los desarrollos OSS. Esta incorporación se logra a través de transformaciones a las técnicas. Transformaciones que no deben ir en contra de la esencia de la técnica, ni en contra de la filosofía de trabajo de la comunidad OSS. La transformación que se ha realizado de la técnica *Personas* así lo constata. Además, al participar como voluntarios en dos proyectos reales OSS pudimos validar que es posible incorporar técnicas de la IPO gracias a ciertas transformaciones, como veremos en el siguiente capítulo. Estas transformaciones fueron realizadas según nuestro marco de integración.

CAPÍTULO 7

MODIFICANDO TÉCNICAS DE USABILIDAD PARA PODER SER APLICADAS EN OSS: DOS CASOS DE ESTUDIO

El presente capítulo detalla el estudio realizado sobre la viabilidad de aplicación del marco de integración de técnicas de usabilidad en los desarrollos OSS, propuesto en el presente trabajo de investigación. Para la validación de la viabilidad de dicho marco hemos utilizado como método de investigación el estudio de casos. En primer lugar, detallamos las transformaciones realizadas a las técnicas de usabilidad incorporadas en los proyectos OSS *FreeMind* y *OpenOffice Writer*, siendo estos los dos casos estudiados. Posteriormente, presentamos el diseño del caso de estudio, que incluye la definición de la pregunta de investigación y los procedimientos de recolección, análisis y validación de datos. Una vez definido el diseño del caso de estudio, detallamos la aplicación de las técnicas de usabilidad con transformaciones en los proyectos OSS FreeMind y OpenOffice Writer. Finalmente, presentamos las conclusiones de incorporar técnicas de usabilidad con transformaciones en los desarrollos OSS.

7.1. Transformaciones Realizadas a las Técnicas de Usabilidad Incorporadas

El objetivo de la presente sección consiste en describir las técnicas de usabilidad que han sido incorporadas con transformaciones en los proyectos OSS FreeMind y OpenOffice Writer. Como hemos discutido a lo largo del presente trabajo de investigación, la mayoría de técnicas de usabilidad de la IPO requieren de ciertas condiciones (por ejemplo, el usuario debe estar físicamente presente) para su aplicación. Condiciones que no se cumplen en los desarrollos OSS debido a las características intrínsecas de su proceso de desarrollo (por ejemplo, desarrolladores y usuarios distribuidos geográficamente por todo el mundo). Por tal razón, se hace necesario la transformación de la mayoría de técnicas de usabilidad de la IPO. A continuación, describiremos las transformaciones realizadas a las técnicas de usabilidad incorporadas.

7.1.1. Técnica Perfiles de Usuario y sus Transformaciones

La técnica perfiles de usuario tiene como objetivo definir clases representativas de usuarios en términos de las tareas que realizarán y las habilidades y conocimientos que dichos usuarios aportan a las tareas [92]. Para recolectar la información necesaria para la creación de los

perfiles de usuario, Nielsen [138] recomienda observar y hablar con los usuarios reales en su propio entorno de trabajo, porque son ellos la principal fuente de información. Una vez recolectada dicha información, es necesario que los implicados en la creación de los perfiles de usuario se reúnan para la definición de los mismos [92].

Debido a que los usuarios de las aplicaciones OSS se encuentran distribuidos por todo el mundo, no es posible observarlos y hablar con ellos en su entorno de trabajo. Por tal razón, es necesario realizar una transformación a la técnica en este sentido para poder conseguir la información necesaria para la creación de los perfiles de usuario. La transformación adoptada en este caso consiste en conseguir la información por otros medios. En nuestro caso particular hemos diseñado un cuestionario para tal fin. Este cuestionario es enviado a los usuarios OSS por correo electrónico para que sea completado de manera asíncrona y remota. De nuevo, en vista de que los miembros de la comunidad OSS se encuentran distribuidos por todo el mundo, no fue posible realizar reuniones presenciales para discutir la estructura y contenido de dicho cuestionario. En su lugar, el diseño del cuestionario fue realizado intercambiando varios correos electrónicos con uno de los administradores del proyecto OpenOffice Writer.

7.1.2. Técnica Observación Directa y sus Transformaciones

La técnica observación directa tiene como principal objetivo comprender cómo los usuarios de las aplicaciones realizan sus tareas y más específicamente conocer todas las acciones que realizan durante la ejecución de determinadas tareas. Para ello, los observadores visitan los usuarios representativos en su lugar de trabajo donde realizan las actividades objeto de estudio y donde serán observados [138].

Como comentamos anteriormente, los usuarios de las aplicaciones OSS se encuentran geográficamente distribuidos por lo que no es posible realizar observaciones en el lugar donde utilizan las aplicaciones. Por tal razón, es necesario realizar transformaciones para poder aplicar esta técnica. Particularmente, hemos realizado dos transformaciones. En primer lugar, hemos realizado un muestreo sesgado que incluye a familiares y amigos del observador como usuarios. En segundo lugar, hemos realizado observaciones remotas a usuarios OSS distribuidos geográficamente. Para realizar tales observaciones hemos utilizado diferentes herramientas que permiten la transferencia de texto, voz y vídeo sobre internet.

7.1.3. Técnica Información Post-Test y sus Transformaciones

La técnica información post-test consiste de una entrevista que se realiza a cada usuario después de finalizar la prueba de usabilidad. La entrevista tiene por objetivo solicitar al usuario retroalimentación y sugerencias tanto de la prueba realizada como de la aplicación que esta siendo evaluada [50]. Esta técnica requiere de la participación de un experto en usabilidad, quien es el responsable de diseñar y realizar la entrevista al usuario [50].

En vista de que los usuarios de las aplicaciones OSS se encuentran distribuidos geográficamente por todo el mundo y a que por regla general los proyectos OSS no cuentan con expertos en usabilidad, es necesario realizar tres transformaciones para poder aplicar la técnica información post-test. En primer lugar, hemos realizado un muestreo sesgado que incluye a familiares y amigos del observador como usuarios. En segundo lugar, hemos realizado entrevistas de manera remota a los usuarios OSS geográficamente distribuidos. Para ello, hemos utilizado las mismas herramientas que en la aplicación de la técnica observación directa. En tercer lugar, reemplazamos el rol del experto en usabilidad por uno o varios estudiantes de la IPO bajo la supervisión de un mentor.

7.2. Diseño del Caso de Estudio

El *caso de estudio* se encuentra entre las formas de estudio empírico cualitativo más populares. En este tipo de estudio, el fenómeno de interés se estudia en su contexto real. Concretamente, para la presente investigación, el fenómeno de interés es la incorporación de técnicas de usabilidad con transformaciones, mientras que el contexto real lo constituyen los proyectos OSS. En la presente sección se describe de manera general el procedimiento seguido en la realización de dos *casos de estudio*, correspondientes a sendos proyectos OSS reales: uno grande, OpenOffice Writer, y otro pequeño, FreeMind.

Ambos *casos de estudio* forman parte, a su vez, de un Trabajo de Fin de Grado (TFG) realizado en el seno de la Universidad Autónoma de Madrid (UAM) [117] y llevado a cabo bajo la supervisión del autor de esta tesis doctoral. La presente tesis doctoral, además de haber dado lugar a publicaciones en revistas y congresos, ha motivado una parte fundamental de dicho TFG, brindando el marco investigador y todos los materiales y lineamientos necesarios para la aplicación de las técnicas de usabilidad. Es importante mencionar que el autor de la presente investigación estuvo presente durante todas las sesiones de aplicación de las técnicas, con el objetivo de realizar las observaciones necesarias de cara a validar si es posible la incorporación de técnicas de usabilidad en los desarrollos OSS. Desde la perspectiva de la contribución a esta tesis, la autora del TFG, Cristina Martín Montero, ha colaborado en labores de observación y aportando comentarios y sugerencias. El presente texto, en lo sucesivo, se referirá a la autora del TFG y colaboradora en los *casos de estudio* por su nombre, Cristina.

7.2.1. Pregunta de Investigación

Ambos *casos de estudio* parten de la siguiente pregunta de investigación:

RQ: Comprender si ciertas transformaciones de las técnicas de usabilidad permiten su uso en proyectos OSS.

Las técnicas a las que hace referencia la pregunta de investigación han sido introducidas en la Sección 7.1. Los *casos de estudio* se completan con sendas encuestas sobre usabilidad SUS (*System Usability Scale*) [40][110][194]. La encuesta SUS se encuentra disponible gratuitamente para su uso en investigación e industria, con la única condición de citar la fuente utilizada. Consta de un conjunto de 10 preguntas en la escala Likert y de un conjunto de reglas para asignar una puntuación $0 \leq p \leq 100$ a cada combinación de respuestas.

7.2.2. Procedimiento de Recolección de Datos

Antes de contactar con los usuarios, se establecieron conversaciones con los administradores de ambos proyectos OSS para hacerles llegar nuestro interés en aplicar técnicas de usabilidad y tratar de obtener su apoyo. Aunque se había considerado la posibilidad de que los responsables de los proyectos mantuvieran listas de correos electrónicos de usuarios representativos, sólo uno de ellos contaba con una lista de usuarios y en ningún caso se había identificado a los representativos. La falta de conocimiento por parte de los desarrolladores OSS del perfil de sus usuarios es un problema que ya ha sido identificado en otros trabajos de investigación [35][132][137]. Nuestro trabajo contribuye aportando evidencia empírica de ello, proponiendo transformaciones, con base en nuestro marco de integración, de la técnica *Perfiles de Usuario*.

Se establecieron dos perfiles de usuario: usuario junior y usuario senior. Cuando los usuarios son familiares y amigos, el nivel académico marca la forma de clasificar. La clase junior está compuesta por estudiantes de grado (por ejemplo, *Grado en Ingeniería Informática* de la UAM), mientras que la clase senior la conforman estudiantes de posgrado (por ejemplo, *Máster en Ingeniería Informática* y *Máster en Investigación e Innovación en TIC* de la UAM). Por

el contrario, los usuarios reales son clasificados de acuerdo a su nivel de experiencia con la aplicación en cuestión. Cabe mencionar que, en ambos casos, se partía de la base de que el usuario había tenido contacto previo con la aplicación. Para asegurar esto, se pidió a familiares y amigos, con suficiente antelación, que se familiarizaran con las opciones básicas de sus respectivas aplicaciones objetivo (siempre disponibles en la web), sin llegar a precisar por nuestra parte ninguna habilidad o conocimiento que debieran adquirir obligatoriamente.

El protocolo seguido en las sesiones de observación directa y remota es prácticamente el mismo. En ambos *casos de estudio* se contó tanto con usuarios junior como senior, a los que se aplicó un procedimiento casi idéntico. Al comienzo de cada sesión, concertada previamente con el usuario, se le explicaba que iba a realizar una serie de tareas con la aplicación en cuestión mientras un observador (Cristina) tomaba notas. Se hizo saber al usuario que en ningún caso se le estaba evaluando a él, sino a la aplicación. La información registrada comprendía los problemas y dificultades encontrados en el transcurso de la actividad, así como aquellos gestos que denotaran sentimientos hacia la aplicación. Las Figuras 7.1 y 7.2 muestran los documentos para registrar los sentimientos del usuario en observación directa y remota, respectivamente, mientras que la Figura 7.3 es un ejemplo de registro de incidencias de un usuario junior en el caso de *FreeMind*.

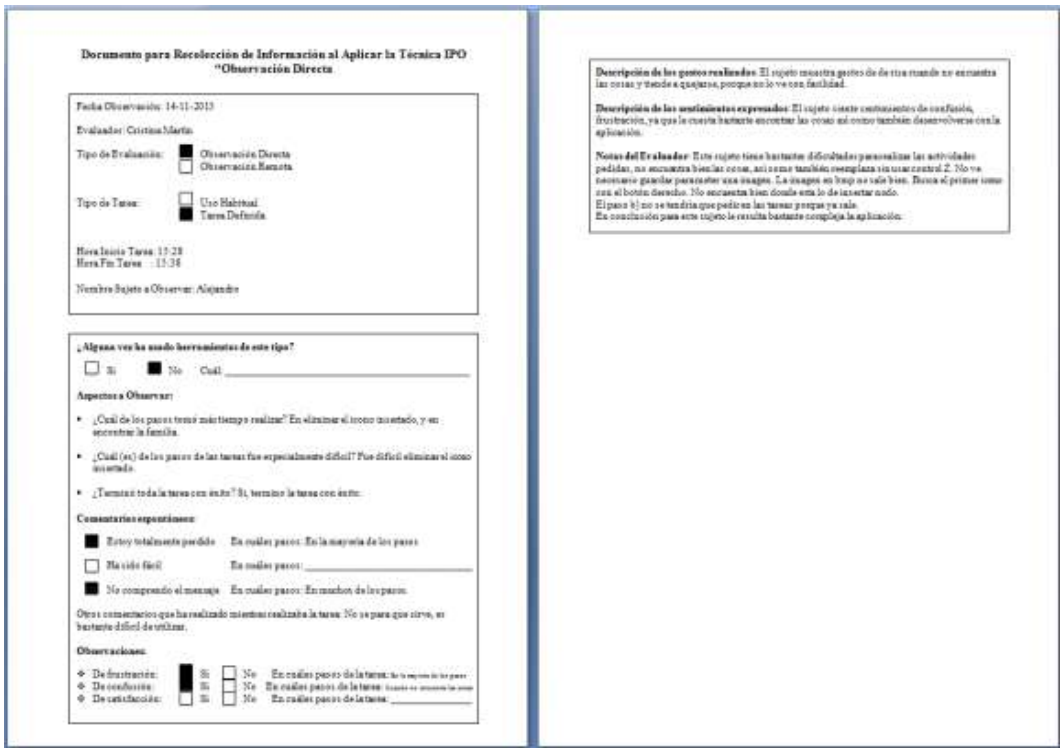
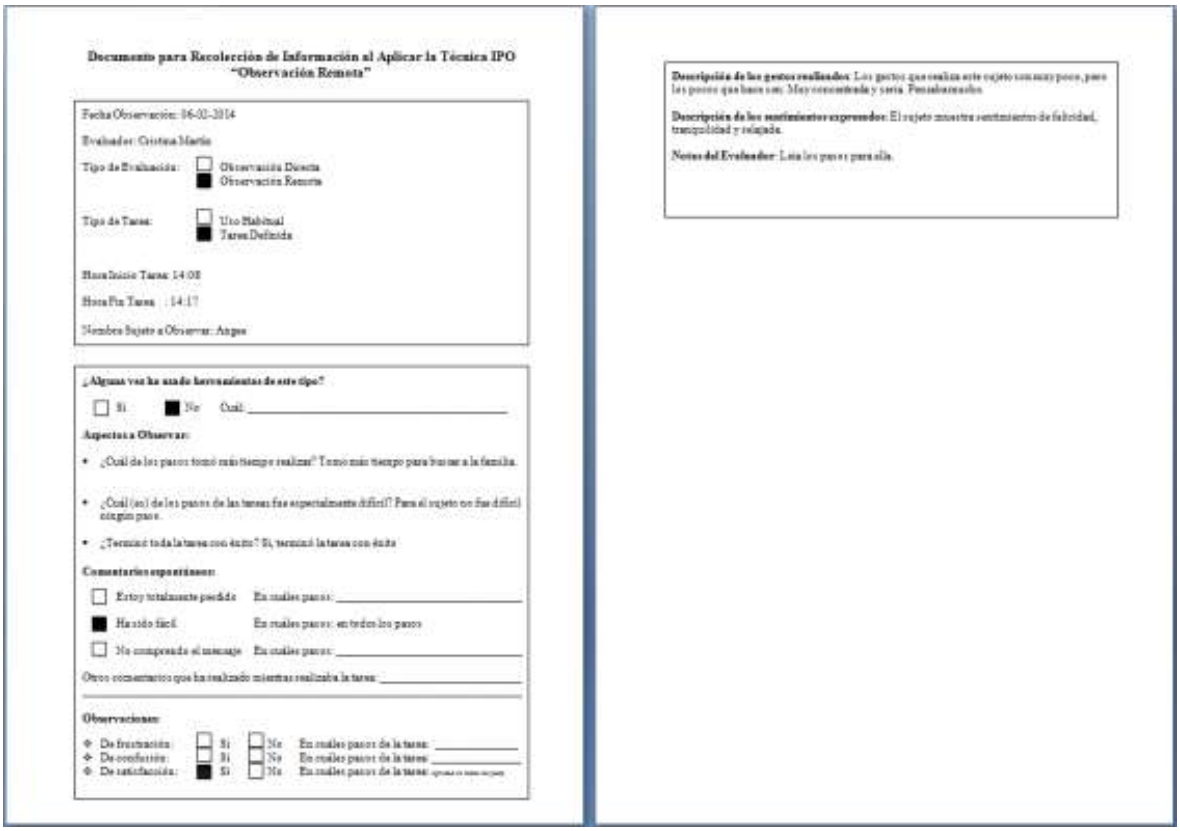


Figura 7.1: Documento para la Recolección de Información Durante la Aplicación de la Técnica Observación Directa.

Finalmente, tras completar las tareas, se invitaba al usuario a realizar una entrevista para obtener información post-test, a cuyo término se le agradecía su participación. La entrevistadora, al contrario que en la aplicación pura de la técnica, no se trataba de una experta en usabilidad, sino de una estudiante de la IPO (Cristina). El usuario se limitaba a contestar a las preguntas de la entrevistadora, siendo ésta la encargada de anotar las respuestas. Supervisando la entrevista se hallaba siempre un mentor (el autor de esta tesis) que, con sus comentarios, pretendía incidir en los puntos más importantes y ayudar al entrevistado a aportar la máxima información posible. La Figura 7.4 muestra el documento empleado para conducir la entrevista. Además, en el transcurso de la sesión, ya fuera antes o después de la observación, se pedía al usuario que completase la encuesta SUS.



Documento para Recolección de Información al Aplicar la Técnica IPO "Observación Remota"

Fecha Observación: 06-02-2014

Evaluador: Cristina Martín

Tipo de Evaluación: ☐ Observación Directa ■ Observación Remota

Tipo de Tarea: ☐ Uso Habitual ■ Tarea Definida

Hora Inicio Tarea: 14:08

Hora Fin Tarea : 14:17

Nombre Sujeto a Observar: Angee

¿Alguna vez ha usado herramientas de este tipo?

☐ Si ■ No Cuál: _____

Aspectos a Observar:

- ¿Cuál de los pasos tomó más tiempo realizar? Tomo más tiempo para buscar a la familia.
- ¿Cuál (es) de los pasos de las tareas fue especialmente difícil? Para el sujeto no fue difícil ningún paso.
- ¿Terminó toda la tarea con éxito? Si, terminó la tarea con éxito

Comentarios espontáneos:

☐ Estoy totalmente perdido En cuales pasos: _____

■ Ha sido fácil En cuales pasos: en todos los pasos

☐ No comprendo el mensaje En cuales pasos: _____

Otros comentarios que ha realizado mientras realizaba la tarea: _____

Observaciones:

- De frustración: ☐ Si ☐ No En cuales pasos de la tarea: _____
- De confusión: ☐ Si ☐ No En cuales pasos de la tarea: _____
- De satisfacción: ■ Si ☐ No En cuales pasos de la tarea:

Descripción de los gestos realizados: Los gestos que realiza este sujeto son muy poco, pero los pocos que hace son: Muy concentrada y seria. Pensativa mucho.

Descripción de los sentimientos expresados: El sujeto muestra sentimientos de felicidad, tranquilidad y relajada.

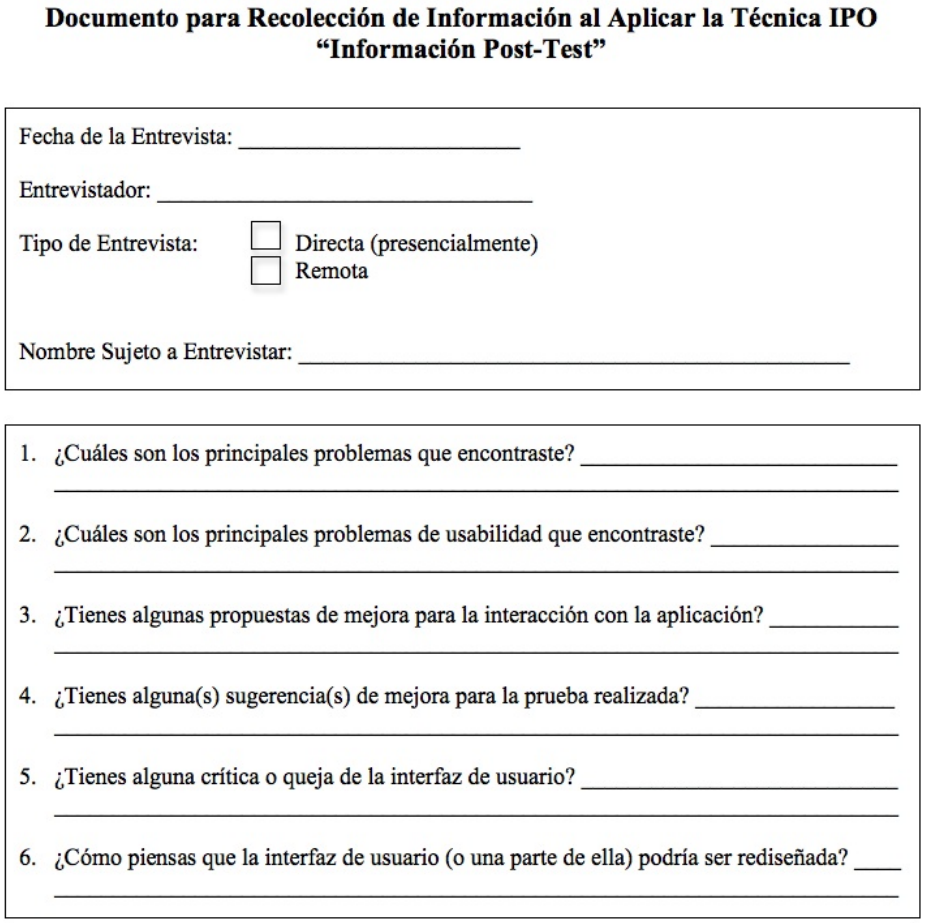
Notas del Evaluador: Lista los pasos para ella.

Figura 7.2: Documento para la Recolección de Información Durante la Aplicación de la Técnica Observación Remota.

Tareas a realizar en la Herramienta "FreeMind"
(Anotaciones del Observador)

		¿El usuario sabe como hacerlo? (Sí / No)	¿Se ha encontrado el usuario con algún problema? ¿Cuál?	Otros comentarios
a.	Abrir la aplicación *FreeMind*	Si	No	
b.	Crear un nuevo mapa	Si	No	Ya está hecho al abrir la aplicación
c.	Colocar como nombre al nodo principal: *Las Leyes de los Mapas Mentales*	Si	No	
d.	Insertar nuevo nodo hermano con nombre: *Imágenes*	Si	Si, porque no lo encuentra con facilidad	
e.	Insertar en el nodo principal un nuevo nodo hermano con nombre: *Palabras*	Si	No	
f.	Insertar en el nodo *Palabras* un nodo hijo con nombre: *Símbolos*	Si	No	
g.	Eliminar el nodo con nombre: *Imágenes*	Si	No	
h.	Una vez hecho esto, y visto que si necesita el nodo *Imágenes*, lo crea de nuevo	Si	No	No usa control Z, sino lo vuelve a crear
i.	Insertar en el nodo principal un nuevo nodo hijo con nombre: *Estructura*	Si	No	
j.	Insertar una imagen cualquiera en el nodo con nombre: *Palabras*	Si	No	
k.	Insertar en el nodo principal un nuevonodo hijo con nombre: *Estilo*	Si	No	
l.	Insertar icono que represente el símbolo admiración (!) en el nodo con nombre: *Estilo*	Si	No	Lo hace con botón derecho
m.	Insertar otro icono, esta vez que represente una familia, en el nodo con nombre: *Estructura*	Si	Si, porque es muy parecido al otro paso	Pretende buscarlo con el botón derecho y no sale
n.	Eliminar el primer icono insertado, es decir, el símbolo admiración (!)	Si	Si, hay que indicarle donde está la opción de eliminar	Lo elimina con el icono de la parte izquierda de la interfaz
o.	Guarde el mapa mental con el nombre *MapaMental* en la carpeta documentos	Si	No	

Figura 7.3: Documento para la Recolección de Incidencias Durante la aplicación de la Técnica Observación Directa/Remota (FreeMind).



Documento para Recolección de Información al Aplicar la Técnica IPO
"Información Post-Test"

Fecha de la Entrevista: _____

Entrevistador: _____

Tipo de Entrevista: ☐ Directa (presencialmente) ☐ Remota

Nombre Sujeto a Entrevistar: _____

1. ¿Cuáles son los principales problemas que encontraste? _____
2. ¿Cuáles son los principales problemas de usabilidad que encontraste? _____
3. ¿Tienes algunas propuestas de mejora para la interacción con la aplicación? _____
4. ¿Tienes alguna(s) sugerencia(s) de mejora para la prueba realizada? _____
5. ¿Tienes alguna crítica o queja de la interfaz de usuario? _____
6. ¿Cómo piensas que la interfaz de usuario (o una parte de ella) podría ser rediseñada? ____

Figura 7.4: Documento para la Recolección de Información Durante la Aplicación de la Técnica Información Post-Test.

Resulta importante recalcar ciertas divergencias en la metodología dependientes del tipo de usuario protagonista de la sesión. Por supuesto, cuando el usuario era remoto, es decir, participaba en una observación remota, toda la comunicación debía realizarse por vía telemática, incluyendo las explicaciones, el suministro de materiales, la entrevista post-test y la toma de datos. Concretamente, se utilizaron dos herramientas: Skype, para poder hablar con el usuario y ver sus reacciones, y TeamViewer, para visualizar su interacción con la aplicación accediendo remotamente a la vista de su pantalla. Una característica de dichas herramientas que se contempló como ventajosa fue la posibilidad que ofrecían de grabar audio y vídeo, lo cual permitiría analizar cuantas veces fuese necesario las sesiones. Finalmente, resultó un recurso útil porque fué necesario revisar las grabaciones en varias oportunidades con el objetivo de resolver dudas (por ejemplo, determinar si un usuario realizó o no una de las tareas según lo indicado en el documento entregado) que surgían al analizar los datos recolectados de manera escrita.

Otra diferencia entre las sesiones presencial y remota fue el momento en el que se abordó la encuesta SUS, cuyo formulario, común a todas las aplicaciones, puede consultarse en la Figura 7.5. En el primer caso, se realizó al finalizar la observación directa, mientras que en el segundo se hizo antes. Este punto no resulta especialmente relevante, ya que el conocimiento del que debían disponer los usuarios para rellenar la encuesta no dependía de que hubieran llevado a cabo la actividad. Por otra parte, los usuarios junior, en ambos *casos de estudio*, realizaban sólo un subconjunto de las tareas realizadas por los senior. El motivo de esta decisión estriba en la mayor destreza que se presuponía a los usuarios senior, por la cual se esperaba poder obtener más conocimiento de estos.

Encuesta SUS

Por favor, marque la casilla que refleje su respuesta inmediata a cada afirmación. No piense demasiado sobre cada afirmación. Asegúrese de que responde todas las afirmaciones. Si no sabe que responder, simplemente marque la casilla "3".

	Totalmente en desacuerdo	En desacuerdo	Neutral	De acuerdo	Totalmente de acuerdo
1. Pienso que me gustaría utilizar este producto con frecuencia	1	2	3	4	5
2. Encontré el producto innecesariamente complejo	1	2	3	4	5
3. Me pareció que el producto era fácil de usar	1	2	3	4	5
4. Creo que voy a necesitar la ayuda de una persona técnica para poder utilizar este producto	1	2	3	4	5
5. Me pareció que las diversas funciones de este producto están bien integradas	1	2	3	4	5
6. Me pareció que había demasiada inconsistencia en este producto	1	2	3	4	5
7. Me imagino que la mayoría de personas aprenderían a usar este producto muy rápidamente	1	2	3	4	5
8. Me pareció que el producto es muy complicado de usar	1	2	3	4	5
9. Me sentí con mucha confianza al usar el producto	1	2	3	4	5
10. Tenía que aprender muchas cosas antes de que pudiera comenzar a utilizar este producto	1	2	3	4	5

Figura 7.5: Encuesta SUS.

7.2.3. Procedimiento de Análisis de Datos

La parte más importante del proceso de análisis de datos ha consistido en la creación de tablas que sintetizan la información obtenida mediante observación directa, remota y entrevista post-test y en la propuesta de mejoras a los problemas de usabilidad encontrados con dichas técnicas. Debido a que la prescripción de las técnicas de usabilidad por parte de la IPO no contempla ningún tipo de documento para la recolección de información durante la aplicación de las mismas, proponemos varias plantillas de tablas que sirven tanto para presentar los resultados de FreeMind como los de OpenOffice Writer.

Las Tablas 7.1 y 7.2, que se presentan en secciones posteriores, son dos ejemplos de tablas que responden al mismo patrón de columnas. Destaca la columna *nivel de relevancia*, cuyos valores han de interpretarse, en ambos *casos de estudio*, de la siguiente manera:

- **Alto**: Problemas de usabilidad que afectan en gran medida a la funcionalidad básica de la aplicación.

- **Medio**: Problemas de usabilidad que afectan levemente a la funcionalidad básica de la aplicación.
- **Bajo**: Problemas de usabilidad que no afectan a la funcionalidad básica de la aplicación.

La funcionalidad básica en el caso de FreeMind recogería todas aquellas tareas de creación, edición, eliminación o manipulación de mapas mentales. Por su parte, la funcionalidad básica para OpenOffice Writer podría entenderse exactamente de la misma forma, pero considerando al documento de texto o HTML como objeto central, en lugar del mapa mental. La columna *prioridad* de las Tablas 7.6 y 7.13, entre otras, ha de interpretarse similarmente a la columna *nivel de relevancia*, pero sustituyendo la figura del *problema de usabilidad* por la de la *mejora de usabilidad*.

La segunda parte del análisis de datos comprende el cálculo de la puntuación p de las encuestas SUS para todos los usuarios y el cálculo de la puntuación media en cada grupo de usuarios (junior y senior). El amplio uso de la encuesta SUS, en el contexto de la evaluación de tipos de aplicaciones muy diversos [31][194], ha dado lugar en la literatura a la siguiente interpretación de la puntuación final p:

$$Usabilidad(p) = \begin{cases} \text{no aceptable}, & p < 50 \\ \text{marginal}, & 50 \leq p \leq 70 \\ \text{aceptable}, & p > 70 \end{cases} .$$

7.2.4. Procedimiento de Validación

Los inconvenientes encontrados en la incorporación de las técnicas de usabilidad varían según la técnica aplicada (observación directa/remota, información post-test y encuesta), según el tipo de sesión (presencial o remota) y, en menor medida, según el proyecto en cuestión (FreeMind y OpenOffice Writer). A continuación, se describirán para cada una de las técnicas, las principales dificultades encontradas en su aplicación.

Para la técnica *observación directa*, el principal escollo en todos los casos ha sido la disponibilidad del usuario, ya que gran parte de ellos se regían por un horario difícilmente compatible con la realización de las pruebas. Cabe mencionar que durante la aplicación de la técnica no surgió ningún problema referente al funcionamiento de ninguna de las aplicaciones que entorpeciese la prueba. En el caso de la técnica observación directa pero aplicada de manera remota, se han detectado cuatro principales obstáculos. El primero, la coordinación con los usuarios, dado que la mayoría de ellos procedía de fuera de España; concretamente, de países del norte de Europa, Australia y América, siendo la diferencia horaria en estos dos últimos casos especialmente pronunciada. En segundo lugar, la conexión a internet. Muchos de los usuarios tuvieron problemas con la conexión, que resultaba demasiado lenta. Como consecuencia, no era posible verlos a través de la *webcam*, impidiendo observar sus expresiones; no se escuchaban correctamente sus comentarios y no era posible establecer la conexión mediante la herramienta TeamViewer. En tercer lugar, el idioma de los usuarios, porque la mayoría de ellos tenía como lengua materna el inglés, lo que llevó a requerir la ayuda de un traductor que tomase nota de sus comentarios. En cuarto y último lugar, una minoría de los usuarios no disponía, en el momento acordado para la realización de la evaluación, de los programas necesarios instalados en sus ordenadores. Esto suponía un aumento en el tiempo de aplicación de la técnica y una pérdida de tiempo que, en ocasiones, retrasaba el inicio de las sesiones programadas con otros usuarios.

En el caso de la técnica *información post-test*, los problemas encontrados son los mismos que los que se derivan de las observaciones directa o remota, según tuviera lugar la entrevista en uno u otro tipo de sesión. Una notable diferencia, no obstante, es que, para la entrevista, no

ver al usuario a través de la *webcam* no supone un grave problema, ya que lo fundamental son los comentarios de los usuarios y no sus expresiones faciales.

Para la técnica *encuesta* han sido dos los problemas encontrados. En primer lugar, un gran número de usuarios no tenía instalado Microsoft Excel, por lo que no podían acceder a la encuesta. Para solucionar este problema hubo que cambiar el formato de la encuesta a uno común, compatible con cualquier hoja de cálculo. Por último, en el caso de usuarios remotos, dado que las encuestas eran enviadas por correo electrónico y podían realizarse asíncronamente y con bastante flexibilidad, unos pocos usuarios finalmente olvidaron rellenarlas. No ocurrió así con los usuarios presenciales, porque la encuesta estaba impresa y la completaban al término de la sesión.

Aparte de los problemas derivados de la aplicación de las técnicas, también surgieron dificultades a la hora de seleccionar a los participantes. Ninguno de los administradores de proyecto, Christian Foltin (FreeMind) y Rob Weir (OpenOffice Writer), contaba con una relación de usuarios representativos de sus respectivas aplicaciones. De hecho, sólo el segundo contaba con una lista de correos electrónicos para contactar con usuarios, de modo que se optó por buscar los usuarios de FreeMind en foros, siguiendo una recomendación del administrador de este proyecto.

En resumen, de las tres técnicas de usabilidad aplicadas, la técnica observación remota es la que presenta la mayor cantidad de inconvenientes, puesto que es la más vulnerable a limitaciones de naturaleza material o tecnológica. Por ejemplo, resulta fundamental contar con una buena conexión a internet y que los usuarios tengan instaladas una serie de herramientas originalmente desconocidas para ellos.

7.3. Caso FreeMind

En esta sección se presenta el caso de estudio de la aplicación FreeMind, comenzando con una breve descripción del contexto del proyecto en el que se enmarca, de sus características y funcionalidades básicas y de las circunstancias concretas experimentadas en el reclutamiento de sus usuarios. Seguidamente, se presentarán documentos utilizados en la realización de este *caso de estudio* y, finalmente, se presentarán los resultados del caso de estudio.

7.3.1. Selección del Caso y los Sujetos

FreeMind es una herramienta OSS, programada en Java, que permite la elaboración de mapas mentales. Esta herramienta constituye la alternativa libre a la aplicación MindManager, desarrollada por la empresa MindJet, distribuyéndose bajo las licencias GNU (*GNU's Not Unix*) y GPL (*General Public License*). Fué nominada en la categoría de *best project* de los *SourceForge.net's Community Choice Awards* de 2008 en representación de los proyectos OSS.

FreeMind resulta de utilidad en el análisis y la recopilación de ideas generadas en grupo. Permite al usuario construir una jerarquía de ideas alrededor de un concepto central, por ejemplo, empleando la conocida técnica de *brainstorming* (tormenta de ideas). Además de crear mapas mentales, permite publicarlos en la web como páginas HTML o Java e insertarlos en *wikis* (por ejemplo, DokuWiki) mediante la configuración de un *plug-in*.

Como aplicación Java, FreeMind puede ejecutarse en múltiples plataformas (por ejemplo, Microsoft Windows, Linux y Mac OS X) conservando la misma interfaz de usuario, salvo pequeñas variaciones dependientes del sistema operativo.

El número de usuarios de FreeMind que participaron en la aplicación de las técnicas observación directa e información post-test fue de 22. La primera mitad se compone de amigos y familiares con los que se podía contactar personalmente. La segunda mitad incluye a amigos, familiares y también usuarios reales de la aplicación, procedentes de diversas partes del mundo.

El proceso de reclutamiento de usuarios reales de FreeMind para observación remota resultó doblemente desafiante, ya que, tal y como se señaló entre las dificultades del estudio, el director de proyecto (Christian Foltin) no sólo no conocía a los usuarios más representativos de la aplicación, sino que tampoco podía proporcionar una lista de correos electrónicos de la que partir. Ante esta situación, se optó por buscar correos electrónicos de usuarios en foros de ayuda de FreeMind. Resultó que no todos los miembros de los foros eran apropiados para la evaluación, al responder algunos a un perfil demasiado técnico, a tenor del tipo de mensajes que publicaban (por ejemplo, errores al compilar una determinada clase Java).

Se contactó vía correo electrónico con un total de 100 usuarios. En el *correo electrónico* enviado a estos usuarios se les preguntaba si estaban interesados en participar en la aplicación de técnicas de usabilidad. Este *correo electrónico* fue respondido por un total de 18 usuarios. A este grupo de usuarios interesados se les escribió un segundo *correo electrónico* donde se describían las técnicas de usabilidad, las actividades que se iban a realizar, el tiempo requerido, las herramientas necesarias, se les informaba del horario disponible y se les preguntaba por el horario en el cual podían realizar tales actividades. Este segundo *correo electrónico* fue respondido por 13 usuarios. Un tercer *correo electrónico* fue enviado a estos usuarios con el propósito de solicitar su usuario de Skype, confirmar la fecha y hora de reunión e indicarles los sitios Web desde donde podían descargar las herramientas necesarias. Se obtuvo respuesta de 6 usuarios. Finalmente, se consiguió una reunión virtual con 4 de estos 6, mientras que los otros 2 no se conectaron en la fecha y hora previamente acordada y confirmada. La Figura 7.6 resume en cifras y clases el reclutamiento de usuarios para el caso de estudio.

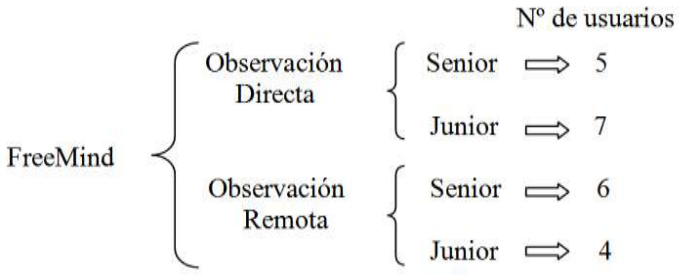


Figura 7.6: Número de Usuarios a los que se Aplicó cada Técnica.

7.3.2. Ejecución

El material suministrado a los usuarios para la realización de las tareas viene dado por las Figuras 7.7 y 7.8, dependiendo de si el sujeto es junior o senior, respectivamente.

7.3.3. Resultados

Comenzamos la presente subsección mostrando los resultados individuales derivados de la encuesta SUS realizada por usuarios junior (Figura 7.9) y senior (Figura 7.10) de la aplicación FreeMind. Por su parte, la Figura 7.11 sintetiza los anteriores resultados e incluye la interpretación de las puntuaciones en términos de niveles de usabilidad.

Tareas a realizar en la Herramienta "FreeMind"
(Usuario junior)

Utilice la aplicación "*FreeMind*" para realizar la siguiente tarea.

Las acciones a realizar son:

a) Abrir la aplicación *FreeMind*
b) Crear un nuevo mapa
c) Colocar como nombre al nodo principal: *Las Leyes de los Mapas Mentales*
d) Insertar nuevo nodo hermano con nombre: *Imágenes*
e) Insertar en el nodo principal un nuevo nodo hermano con nombre: *Palabras*
f) Insertar en el nodo *Palabras* un nodo hijo con nombre: *Símbolos*
g) Eliminar el nodo con nombre: *Imágenes*
h) Una vez hecho esto, y visto que si necesita el nodo *Imágenes*, lo crea de nuevo
i) Insertar en el nodo principal un nuevo nodo hijo con nombre: *Estructura*
j) Insertar una imagen cualquiera en el nodo con nombre: *Palabras*
k) Insertar en el nodo principal un nuevo nodo hijo con nombre: *Estilo*
l) Insertar icono que represente el símbolo admiración (!) en el nodo con nombre: *Estilo*
m) Insertar otro icono, esta vez que represente una familia, en el nodo con nombre: *Estructura*
n) Eliminar el primer icono insertado, es decir el símbolo admiración (!)
o) Guarde el mapa mental con el nombre *MapaMental* en la carpeta documentos.

Figura 7.7: Tareas a Realizar por los Usuarios Junior con FreeMind.

Tareas a realizar en la Herramienta "FreeMind"
(Usuario senior)

Utilice la aplicación "*FreeMind*" para realizar la siguiente tarea.

Las acciones a realizar son:

a) Abrir la aplicación *FreeMind*
b) Crear un nuevo mapa
c) Colocar como nombre al nodo principal: *Las Leyes de los Mapas Mentales*
d) Insertar nuevo nodo hermano con nombre: *Imágenes*
e) Insertar en el nodo principal un nuevo nodo hermano con nombre: *Palabras*
f) Insertar en el nodo *Palabras* un nodo hijo con nombre: *Símbolos*
g) Eliminar el nodo con nombre: *Imágenes*
h) Una vez hecho esto, y visto que si necesita el nodo *Imágenes*, lo crea de nuevo
i) Insertar en el nodo principal un nuevo nodo hijo con nombre: *Estructura*
j) Insertar una imagen cualquiera en el nodo con nombre: *Palabras*
k) Insertar en el nodo principal un nuevo nodo hijo con nombre: *Estilo*
l) Insertar icono que represente el símbolo admiración (!) en el nodo con nombre: *Estilo*
m) Insertar otro icono, esta vez que represente una familia, en el nodo con nombre: *Estructura*
n) Eliminar el primer icono insertado, es decir el símbolo admiración (!)
o) Guarde el mapa mental con el nombre *MapaMental* en la carpeta documentos
p) Insertar en el nodo *Estilo* un nodo hijo con nombre: *Personal*
q) Insertar en el nodo *Estilo* un segundo nodo hijo con nombre: *Armonía*
r) Insertar en el nodo *Estilo* un tercer nodo hijo con nombre: *Claridad*
s) Insertar una nube que incluya al nodo *Estilo* y a todos sus nodos hijos
t) Añadir un enlace gráfico entre el nodo *Personal* y el nodo *Armonía*
u) Cambiar el color de la nube por amarillo
v) Insertar un enlace a un fichero pdf
w) Subir el nodo *Claridad* un nivel
x) Cambiar el formato del nodo *Imágenes* a nodo parpadeante
y) Unir los nodos *Armonía* y *Claridad*
z) Colocar color al fondo del nodo principal. Por ejemplo rosa
aa) Exportar Mapa Mental como pdf a la carpeta documentos

Figura 7.8: Tareas a Realizar por los Usuarios Senior con FreeMind.

SUS Calculation

Participant	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	SUS Score
Alejandro	1	5	2	3	1	4	2	4	2	2	25,0
Angee	4	1	5	1	3	1	5	1	5	1	92,5
Carlos	2	3	1	2	4	2	1	1	3	2	52,5
Endika	2	1	4	2	4	3	5	2	4	2	72,5
Linda	3	2	4	2	3	3	2	3	3	2	57,5
Miguel	2	4	3	2	3	4	2	3	2	1	45,0
Miguel Ángel	3	3	3	2	4	2	4	2	3	3	62,5
Olympia	4	3	2	1	2	3	5	2	4	2	65,0
Rubén	2	4	3	2	3	5	2	3	2	2	40,0
Javier	3	2	4	2	4	2	4	1	3	2	72,5
Samuel	3	2	4	2	3	3	3	2	4	2	65,0
									Promedio Total		59,1

Figura 7.9: Puntuaciones SUS Individuales de los Usuarios Junior.

SUS Calculation

Participant	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	SUS Score
John	5	2	4	1	4	1	4	1	4	1	87,5
Nathan	5	2	5	2	4	2	3	2	4	3	75,0
Andrés	4	1	5	1	4	1	5	1	5	1	95,0
Cristina	3	1	5	1	4	2	5	1	5	1	90,0
Dan	5	2	4	2	4	2	5	2	4	2	80,0
François	4	2	4	1	4	2	4	1	4	1	82,5
María	3	1	5	1	4	3	5	1	4	1	85,0
Ricardo	5	3	4	1	3	4	2	2	4	1	67,5
Rodrigo	1	1	5	1	4	2	5	1	5	1	85,0
Javier F.	2	5	4	1	3	3	2	2	3	2	52,5
Yanedt	3	4	4	2	2	4	2	4	3	2	45,0
									Promedio Total		76,8

Figura 7.10: Puntuaciones SUS Individuales de los Usuarios Senior.

Nombre del Proyecto: FreeMind	Versión del Proyecto: 0.9.0
Nombre de la Aplicación: FreeMind	Fecha: 19/12/2011

Preparado por: Cristina Martín Montero	Departamento: Ingeniería Informática, Escuela Politécnica Superior
Fecha del Informe: 14/03/2014	Email: cristina.martinm01@estudiante.uam.es

1. Estadísticas de los Participantes

Número Total de Participantes:
22

Tipo de Usuario	Número	Porcentaje
Junior	11	50%
Senior	11	50%

2. Resultados

Número de Preguntas	Tasa de Respuesta
10	100%

Tipo de Usuario	Promedio	Interpretación
Junior	59,1	Usabilidad Marginal
Senior	76,8	Usabilidad Aceptable

Figura 7.11: Informe de la Encuesta SUS.

Como resultado de la incorporación de la técnica observación directa, hemos encontrado diferentes problemas en la aplicación FreeMind. Las Tablas 7.1 y 7.2 presentan el listado de los problemas encontrados por los usuarios junior y senior, respectivamente. Las recomendaciones dadas por los usuarios para mejorar la usabilidad de FreeMind se presentan en la Tabla 7.3.

Tabla 7.1: Problemas Encontrados por los Usuarios Junior Durante la Observación Directa.

Problema	**Nro. de Personas**	**Nivel de Relevancia**	**Mejoras**
No se ve fácilmente la opción de insertar un nodo hijo	1	Alto	Mejorar la visibilidad de las opciones relacionadas con la inserción y eliminación de nodos.
El símbolo *familia* no aparece al hacer *click* derecho	1	Alto	Permitir el acceso a todos los símbolos a través del botón derecho del ratón.
El tamaño de la imagen es excesivo	1	Alto	Redimensionar la imagen.
La imagen elimina el texto del nodo	1	Alto	Permitir que imagen y texto coexistan en un nodo.
No se ve fácilmente la opción de insertar un nuevo nodo	2	Alto	Mejorar la visibilidad de las opciones relacionadas con la inserción y eliminación de nodos.
Dificultad para encontrar los símbolos	2	Alto	Clasificar los iconos por categorías.
Dificultad para encontrar los botones para eliminar y mover nodos	2	Alto	Mejorar la visibilidad de las opciones relacionadas con la inserción, eliminación y desplazamiento de nodos.
Se solicita al usuario guardar al introducir una imagen	3	Alto	No solicitar al usuario que guarde cuando introduzca una imagen.
La baja usabilidad de la aplicación	3	Alto	Mejorar la usabilidad en general, especialmente pensando en los usuarios principiantes.
Los símbolos son muy pequeños	4	Alto	Situar los símbolos en la barra de herramientas y aumentar su tamaño.
Imposibilidad de mover los nodos	1	Medio	Permitir el desplazamiento de cualquier nodo en el mapa mental.
No se encuentra la opción de eliminar un nodo	1	Medio	Mejorar la visibilidad de las opciones relacionadas con la inserción y eliminación de nodos o incluir un símbolo para ello.

Tabla 7.1: Problemas Encontrados por los Usuarios Junior Durante la Observación Directa (continuación).

Problema	**Nro. de Personas**	**Nivel de Relevancia**	**Mejoras**
No se encuentra la opción de eliminar un icono	2	Medio	Mejorar la visibilidad de la opción de eliminar un icono o modificarla para que sea más fácilmente reconocible.
Imposibilidad de eliminar un icono concreto entre varios	2	Medio	Permitir eliminar un icono que no sea necesariamente el último insertado.
El gran parecido entre los símbolos *familia* y *grupo*.	4	Medio	Modificar el símbolo *familia*.
Dificultad para ver el símbolo *"!"*	1	Bajo	Organizar los iconos en la barra de herramientas por categorías.
Seleccionar un nodo modifica su texto	1	Bajo	Ofrecer vías claramente diferenciadas para, por un lado, seleccionar un nodo y, por otro, editar su texto asociado.
Nodo hijo y nodo padre tienen la misma forma	1	Bajo	Asegurar que la forma del nodo hijo es distinta de la del nodo padre.
Imposibilidad de desplazar el mapa conceptual en conjunto	1	Bajo	Permitir mover de un sitio a otro el diagrama como un *todo*.

Tabla 7.2: Problemas Encontrados por los Usuarios Senior Durante la Observación Directa.

Problema	**Nro. de Personas**	**Nivel de Relevancia**	**Mejoras**
Dificultad para encontrar el símbolo *familia* con el botón derecho del ratón	1	Alto	Permitir el acceso a todos los símbolos a partir del botón derecho.
La aplicación es poco flexible y poco intuitiva	1	Alto	Modificar la interfaz, especialmente pensando en los usuarios principiantes.
El efecto de seleccionar un nodo no es perceptible	1	Alto	Sombrear un nodo cuando es seleccionado.
Los símbolos son muy pequeños	1	Alto	Situar los símbolos en la barra de herramientas y aumentar su tamaño.
Modo de operación deficiente	1	Alto	Homogeneizar todas las opciones que se pueden realizar.

Tabla 7.2: Problemas Encontrados por los Usuarios Senior Durante la Observación Directa (continuación).

Problema	**Nro. de Personas**	**Nivel de Relevancia**	**Mejoras**
No se produce ningún efecto al interactuar con el botón *guardar*	1	Alto	Programar el botón *guardar* para que responda visualmente, tal vez con un movimiento, ante la pulsación del usuario.
Se solicita al usuario guardar al introducir una imagen	1	Alto	No solicitar al usuario que guarde cuando introduzca una imagen.
El color de fondo del nodo no se actualiza al efectuar un cambio	1	Alto	Actualizar de inmediato el color de fondo del nodo tras un cambio.
No se ve fácilmente la opción de insertar un nodo hermano	1	Alto	Mejorar la visibilidad de las opciones relacionadas con la inserción y eliminación de nodos.
La opción *color nodo* puede referirse tanto al color de fondo como al del texto asociado al mismo, creando ambigüedad	1	Alto	Precisar a qué color se refiere la opción *color nodo*: color de fondo o del texto.
No todas las opciones de la barra de herramientas están escritas en el mismo idioma	1	Alto	Emplear un único idioma para escribir las opciones de la barra de herramientas.
El tamaño de la imagen es excesivo	2	Alto	Redimensionar la imagen.
Imposibilidad de eliminar un icono concreto entre varios	3	Alto	Permitir eliminar un icono que no sea necesariamente el último insertado.
Los símbolos no parecen guardar ningún orden	4	Alto	Organizar los iconos en la barra de herramientas por categorías.
Se solicita confirmación al usuario para eliminar un nodo que no tiene descendientes	1	Medio	Evitar pedir confirmación al usuario cuando el nodo a eliminar no tiene descendientes.
Jerarquía de menús poco equilibrada	1	Medio	Organizar la jerarquía de menús de tal forma que no existan submenús con profundidades muy dispares.
Imposibilidad de seleccionar varios nodos simultáneamente	1	Medio	Permitir seleccionar varios nodos simultáneamente.

Tabla 7.2: Problemas Encontrados por los Usuarios Senior Durante la Observación Directa (continuación).

Problema	Nro. de Personas	Nivel de Relevancia	Mejoras
Símbolo *familia* muy parecido al símbolo *grupo*	1	Medio	Modificar el símbolo *familia* de tal manera que se diferencie claramente del símbolo *grupo*.
Imposibilidad de desplazar un nodo	1	Medio	Permitir desplazar un nodo.
No existe diferencia entre insertar un nodo hijo o un nodo hermano en el nodo raíz	1	Medio	Cambiar la forma del nodo hijo y del nodo hermano con respecto al raíz.
Existen símbolos en la barra de herramientas que, aún desempeñando distintas funciones, tienen la misma apariencia	2	Medio	Cambiar los símbolos por otros.
Apariencia general poco agradable	1	Bajo	Mejorar el diseño general de la interfaz.
Al cambiar el color de fondo de una nube que abarca varios nodos, el nuevo color no se aprecia porque el color gris que indica la selección de la nube no deja ver el nuevo color	1	Bajo	Cambiar el color de la nube desde cualquier nodo dentro de la nube, y no solo desde el nodo padre.
La aplicación no cuenta con una funcionalidad que avise al usuario de la existencia de nuevas versiones	1	Bajo	Avisar al usuario de nuevas versiones.
Los símbolos no son muy variados y no se pueden añadir nuevos	1	Bajo	Aumentar la variedad de iconos y permitir añadir nuevos.
La apariencia de ramas y nodos es demasiado seria	3	Bajo	Cambiar la forma de los nodos y colorear las diferentes ramas.

Tabla 7.3: Recomendaciones de Usabilidad Dadas por los Usuarios Durante la Observación Directa.

Mejora	Clasificación	Prioridad
Incorporación de imágenes sin necesidad de guardar el documento	Mejora de la interacción (nueva funcionalidad)	Alta
Agrupación de los iconos según categoría de funcionalidad	Diseño de la interfaz	Alta
Redimensionamiento de la imagen	Mejora de la interacción	Alta
Agrupación de los iconos según categoría de funcionalidad	Diseño de la interfaz	Alta
Menor semejanza entre los símbolos *familia* y *grupo*	Diseño de la interfaz	Alta

Tabla 7.3: Recomendaciones de Usabilidad Dadas por los Usuarios Durante la Observación Directa (continuación).

Mejora	Clasificación	Prioridad
Mayor visibilidad de las opciones de insertar nodo, nodo hermano y nodo hijo	Diseño de la interfaz	Alta
Utilización de un mismo lenguaje para todas las opciones de la barra de herramientas	Diseño de la interfaz	Alta
Mayor visiblidad de los iconos	Diseño de la interfaz	Alta
Selección de un nodo sin que se modifique su texto asociado	Mejora de la interacción	Alta
Posibilidad de mover los nodos	Mejora de la interacción	Alta
Posibilidad de que un nodo posea tanto texto como imagen	Mejora de la interacción	Alta
Apariencia más agradable, con colores y formas diferentes	Mejora de la interacción	Media
Iconos más grandes	Diseño de la interfaz	Media
Adición de un botón *papelera* y otro *mover* a la barra de herramientas	Diseño de la interfaz	Media
Retoque en la interfaz para facilitar el manejo del sistema	Diseño de la interfaz y mejora de la interacción	Media
Posibilidad de mover el diagrama haciendo *click* en un espacio en blanco del área de trabajo	Mejora de la interacción	Media
Cambio de "Eliminar último icono" por "Eliminar último icono introducido"	Diseño de la interfaz	Media
Ampliación de la documentación	Mejora de la documentación	Baja
Actualización de la herramienta con versiones	Mejora de la funcionalidad	Baja
Mayor variedad de iconos	Diseño de la interfaz	Baja
Posibilidad de añadir más símbolos	Mejora de la interacción	Baja
Cambio en la forma de los nodos hijo	Diseño de la interfaz	Baja

De manera análoga a las Tablas 7.1 y 7.2, las Tablas 7.4 y 7.5 presentan los diferentes problemas encontrados por los usuarios durante la aplicación de la técnica observación directa pero aplicada de manera remota. Estos problemas están relacionados directamente con el uso de la aplicación. Las recomendaciones dadas por los usuarios para mejorar la usabilidad de FreeMind se presentan en la Tabla 7.6.

Tabla 7.4: Problemas Encontrados por los Usuarios Junior Durante la Observación Remota.

Problema	Nro. de Personas	Nivel de Relevancia	Mejoras
Las imágenes insertadas aparecen con su tamaño original	1	Alto	Redimensionar la imagen.
El icono *equis* "X" (en la paleta de iconos del lado izquierdo de la interfaz) se interpreta erróneamente como un botón para la acción borrar	1	Alto	Eliminar este icono de la paleta de iconos o rediseñar el icono.

Tabla 7.4: Problemas Encontrados por los Usuarios Junior Durante la Observación Remota (continuación).

Problema	Nro. de Personas	Nivel de Relevancia	Mejoras
Hacer *click* sobre un nodo sólo permite editar su texto asociado	2	Alto	Añadir una mayor variedad de acciones a realizar sobre un nodo.
Confusión, a la hora de eliminar un icono, acerca de cuál ha sido el último insertado	2	Alto	Permitir eliminar cualquier icono.
Tamaño de imagen excesivo	2	Alto	Redimensionar la imagen.
Iconos muy ocultos	1	Medio	Facilitar el acceso a los íconos del menú secundario.
Dificultad para diferenciar entre nodo *hermano* e *hijo*	1	Medio	Cambiar la forma del nodo hijo.
Imposibilidad de ordenar los nodos	1	Medio	Permitir ordenar los nodos por todo el mapa.
Iconos muy desordenados	1	Medio	Ordenar los iconos por tipo.
El botón *deshacer* y la secuencia equivalente (CTRL+Z) no funcionan siempre	1	Medio	Aumentar la fiabilidad del botón y la secuencia de teclas.
Un nodo seleccionado sólo se desmarca al seleccionar otro	1	Medio	Permitir que un nodo se desmarque al hacer *click* en cualquier punto del espacio de trabajo.

Tabla 7.5: Problemas Encontrados por los Usuarios Senior Durante la Observación Remota.

Problema	Nro. de Personas	Nivel de Relevancia	Mejoras
Sólo parte de los iconos son accesibles desde el botón derecho del ratón	1	Alto	Permitir el acceso a todos los iconos a través del botón derecho del ratón.
Se solicita al usuario guardar al introducir una imagen	1	Alto	No solicitar al usuario que guarde cuando introduzca una imagen.
Las imágenes insertadas aparecen con su tamaño original	2	Alto	Redimensionar la imagen.
La imagen elimina el texto del nodo	2	Alto	Permitir que imagen y texto coexistan en un nodo.
El color de fondo del nodo no se actualiza al efectuar un cambio	2	Alto	Actualizar de inmediato el color de fondo del nodo tras un cambio.
No se solicita confirmación al eliminar un nodo o icono	1	Medio	Solicitar confirmación al eliminar un nodo o icono.
Excesiva complejidad de la acción de juntar dos nodos	1	Medio	Reducir el número de pasos necesarios para completar la acción de juntar dos nodos.

Tabla 7.5: Problemas Encontrados por los Usuarios Senior Durante la Observación Remota (continuación).

Problema	Nro. de Personas	Nivel de Relevancia	Mejoras
Dificultad para diferenciar entre nodo *hermano* e *hijo*	2	Medio	Cambiar la forma del nodo hijo.
Imposibilidad de mover los nodos	2	Medio	Permitir organizar los nodos en el mapa.
No es posible crear un nodo de un tamaño determinado. Actualmente el tamaño de un nodo viene dado por la longitud de la cadena de texto que contiene	1	Bajo	Permitir la creación de nodos de tamaño fijo.
Excesiva complejidad en el uso de filtros	1	Bajo	Proporcionar ayuda para el uso de filtros.

Tabla 7.6: Recomendaciones de Usabilidad Dadas por los Usuarios Durante la Observación Remota.

Mejora	Clasificación	Prioridad
Redimensionamiento de los iconos	Diseño de la interfaz	Alta
Redimensionamiento de la imagen	Mejora de la interacción	Alta
Agrupación de los iconos	Mejora de la interacción	Alta
Posibilidad de mover los nodos	Mejora de la interacción	Alta
Cambio instantáneo del color de fondo del nodo	Mejora de la interacción	Alta
Al hacer *click* derecho no desaparecen nodo *hermano* e *hijo*	Mejora de la interacción	Alta
Iconos menos ocultos con el botón derecho	Mejora de la interacción	Media
Diseño de la interfaz más moderna	Diseño de la interfaz	Media
Click derecho tiene las mismas opciones que la barra de herramientas	Mejora de la interacción	Media
Posibilidad de situar el nodo principal en cualquier lugar	Mejora de la interacción	Media
Adición de barras laterales para poder mover el mapa	Mejora de la interacción	Media
Reubicación de los botones *borrar* y *papelera* en la barra de herramientas	Diseño de la interfaz	Media
Selección de iconos más importantes	Diseño de la interfaz	Media
Mejor acceso al menú de filtros	Mejora de la interacción	Media
Potenciación del uso del teclado, añadiendo atajos para determinadas acciones	Mejora de la interacción	Media
Adición de una opción de vista preliminar de todo el mapa	Mejora de la interacción	Media
Al seleccionar un nodo las opciones disponibles para editar el mismo no son las mismas si se despliegan las opciones con click derecho o con el menú principal	Mejora de la interacción	Media
Los botones de la barra de herramientas ofrecen *feedback* visual al usuario cuando éste pasa por encima de ellos con el ratón	Mejora de la interacción	Media

Tabla 7.6: Recomendaciones de Usabilidad Dadas por los Usuarios Durante la Observación Remota (continuación).

Mejora	Clasificación	Prioridad
Refuerzo de la ayuda a usuarios principiantes	Mejora de la ayuda	Baja
La interfaz es compleja para usuarios principiantes. En especial la parte inferior de la misma que permite realizar diferentes opciones de edición y de formato para el mapa mental	Diseño de la interfaz	Baja
Lista de iconos más pequeña	Diseño de la interfaz	Baja
Ramas más bonitas	Diseño de la interfaz	Baja
Fondo detrás del mapa	Diseño de la interfaz	Baja
Posibilidad de añadir iconos a la paleta de iconos	Mejora de la interacción	Baja
Realizar modificaciones para que FreeMind tenga muchas de las funcionalidades que tiene FreePlane	Mejora de la funcionalidad	Baja

Finalmente, a manera de resumen la Tabla 7.7 presenta todos los diferentes problemas de usabilidad detectados por los usuarios durante la aplicación de las técnicas observación directa/remota e información post-test. Para cada uno de estos problemas se propone una mejora.

Tabla 7.7: Listado de los Diferentes Problemas y Mejoras Encontrados Durante la Aplicación de las Técnicas Directa e Información Post-Test.

Problema	Técnicas	Frecuencia	Mejoras	Autor Mejora
No se encuentra el símbolo de exclamación *"!"*	Directa y Post-test	3 de 22	Situar el símbolo en un lugar más visible.	Cristina
No se encuentra la opción de insertar un nuevo nodo	Directa y Post-test	6 de 22	Aumentar la visibilidad de la opción.	Cristina
Confusión entre los símbolos *familia* y *grupo*	Directa y Post-test	8 de 22	Ordenar los símbolos.	Usuario
Se solicita guardar para insertar una imagen	Directa y Post-test	8 de 22	Permitir insertar la imagen sin necesidad de guardar.	Usuario
No encuentra la opción de cambio de color de la fuente del nodo	Directa y Post-test	1 de 22	Cambiar "Color nodo" por "Color fuente de nodo".	Usuario
La herramienta no es muy flexible	Directa y Post-test	1 de 22	Permitir mover el mapa conceptual y cada una de sus partes con mayor libertad.	Usuario
No se solicita confirmación para eliminar un nodo no hoja	Directa y Post-test	1 de 22	Preguntar al usuario si desea eliminar el nodo y, con él, sus descendientes.	Usuario
No se aprecia si se ha seleccionado un nodo	Directa y Post-test	1 de 22	Ofrecer *feedback* visual al usuario cuando seleccione un nodo.	Cristina
Menús poco homogéneos	Directa y Post-test	1 de 22	Homogeneizar los menús tanto de la barra de herramientas como los contextuales.	Cristina

Tabla 7.7: Listado de los Diferentes Problemas y Mejoras Encontrados Durante la Aplicación de las Técnicas Directa e Información Post-Test (continuación).

Problema	Técnicas	Frecuencia	Mejoras	Autor Mejora
La opción de cambiar el color de la nube no siempre está disponible	Directa y Post-test	1 de 22	Averiguar por qué sólo está disponible en ciertas ocasiones.	Cristina
Color de fondo de un nodo no aparece hasta que no pinchas fuera	Directa y Post-test	1 de 22	Color aparezca de modo inmediato cuando se ha seleccionado un color.	Usuario
Es difícil encontrar *nodo parpadeante*	Directa y Post-test	1 de 22	Elegir una ubicación alternativa para los símbolos, ordenarlos y redimensionarlos.	Usuario
Es difícil encontrar la opción de enlazar dos nodos	Directa y Post-test	1 de 22	Situar la opción en una zona de mayor visibilidad.	Cristina
Es difícil encontrar los filtros	Directa y Post-test	1 de 22	Situar los filtros en el menú de herramientas.	Cristina
No es posible arrastrar los elementos	Directa y Post-test	1 de 22	Permitir arrastrar y mover los elementos.	Usuario
Existe una gran dependencia del ratón en el uso de la aplicación	Directa y Post-test	1 de 22	Permitir el uso alternativo de comandos rápidos con el teclado.	Usuario
Click en un nodo sólo permite modificar el texto	Directa y Post-test	1 de 22	Permitir un mayor número de acciones a realizar sobre un nodo seleccionado.	Cristina
Los iconos a los que se accede con el botón derecho del ratón no son muy visibles	Directa y Post-test	1 de 22	Facilitar el acceso a estos iconos a través del botón derecho del ratón aumentando su visibilidad.	Usuario
Nodo hermano y nodo hijo son iguales	Directa y Post-test	1 de 22	Distinguir entre nodo hermano y nodo hijo.	Usuario
No hay ayuda para los usuarios principiantes	Directa y Post-test	1 de 22	Añadir documentación de ayuda orientada a usuarios principiantes.	Usuario
El botón *suprimir* (CTRL+ Z) no funcionan siempre	Directa y Post-test	1 de 22	Asegurar que esta acción siempre está disponible.	Usuario
Icono "X" crea confusión porque es una acción	Directa y Post-test	1 de 22	Eliminar esta acción de la lista de iconos.	Cristina
Se entra en modo edición al hacer *click* sobre un nodo	Directa y Post-test	1 de 22	Proporcionar un mayor número de acciones a realizar sobre un nodo.	Usuario
Un nodo sólo deja de estar seleccionado cuando se selecciona otro	Directa y Post-test	1 de 22	Permitir que un nodo deje de estar seleccionado cuando el usuario haga *click* sobre cualquier punto del área de trabajo.	Usuario
Usabilidad general	Directa y Post-test	2 de 22	Mejorar la interfaz de la aplicación.	Usuario

Tabla 7.7: Listado de los Diferentes Problemas y Mejoras Encontrados Durante la Aplicación de las Técnicas Directa e Información Post-Test (continuación).

Problema	**Técnicas**	**Frecuencia**	**Mejoras**	**Autor Mejora**
Interfaz demasiado cargada, plagada de elementos que dificultan las tareas del usuario	Directa y Post-test	2 de 22	Rediseñar la interfaz para que el usuario alcance sus objetivos con mayor facilidad.	Usuario
En la barra de herramientas hay 4 símbolos iguales que sólo se diferencian por el color	Directa y Post-test	2 de 22	Distinguir los símbolos.	Usuario
Los nodos carecen de color	Directa y Post-test	2 de 22	Añadir diferentes colores a los nodos, según su categoría.	Cristina
Es necesario guardar primero el mapa mental para poder insertar una imagen	Directa y Post-test	2 de 22	Dejar insertar la imagen sin tener que guardar.	Usuario
No es posible seleccionar el icono que simboliza una familia con el clic derecho porque no aparecen todos los iconos	Directa y Post-test	2 de 22	Permitir que todo el listado de iconos sean desplegados con clic derecho.	Cristina
Iconos poseen tamaño fijo	Directa y Post-test	2 de 22	Redimensionar los iconos.	Usuario
No existe la posibilidad de mover un nodo	Directa y Post-test	3 de 22	Añadir una opción para mover un nodo utilizando el botón secundario del ratón.	Usuario
Cuando se inserta una imagen en un nodo, el texto asociado a este queda oculto por la imagen	Directa y Post-test	3 de 22	Permitir que imagen y texto puedan coexistir en un nodo.	Usuario
Iconos muy desordenados	Directa y Post-test	3 de 22	Ordenar los iconos por categorías.	Usuario
Símbolos muy pequeños	Directa y Post-test	4 de 22	Situar los símbolos en un lugar alternativo, ordenarlos y aumentar su tamaño.	Usuario
Imagen demasiado grande	Directa y Post-test	4 de 22	Adaptar la imagen a un tamaño deseable.	Usuario
Sólo es posible eliminar el último icono insertado en un nodo (como en una pila)	Directa y Post-test	4 de 22	Permitir eliminar un icono arbitrario entre los insertados en un nodo.	Usuario
Dificultades para encontrar con el botón derecho del ratón, el símbolo que representa una *familia*	Directa y Post-test	5 de 22	No caben todos los símbolos si se hace con el botón secundario en la pantalla, proporcionarlo a todas las pantallas.	Cristina
Se echa en falta una *papelera* o ícono *eliminar*	Directa y Post-test	6 de 22	Cambiar "Eliminar el último icono" por "Eliminar icono".	Usuario

7.4. Caso OpenOffice Writer

En esta sección se presenta el caso de estudio de la aplicación OpenOffice Writer, comenzando con una breve descripción del contexto del proyecto en el que se enmarca, de sus características y funcionalidades básicas y de las circunstancias concretas experimentadas en el reclutamiento de sus usuarios. Seguidamente, se presentarán documentos utilizados en la realización de este *caso de estudio* y, finalmente, se presentarán los resultados del caso de estudio.

7.4.1. Selección del Caso y los Sujetos

OpenOffice es uno de los proyectos OSS más populares en la actualidad. A la hora de hablar de proyectos OSS exitosos, resulta ineludible referirse a OpenOffice. Se trata de un proyecto OSS de gran envergadura, bien organizado y estructurado, y que además cuenta con una gran comunidad de usuarios [20]. Apache OpenOffice Writer (OpenOffice.org Writer hasta diciembre de 2011) es un procesador de texto multiplataforma que forma parte del conjunto de aplicaciones de la *suite* ofimática Apache OpenOffice. Además de otros formatos de documento estándares y ampliamente utilizados, es compatible, casi completamente, con el formato propietario *.doc* de Microsoft Word. El formato nativo para exportar documentos es XML, si bien también puede exportar ficheros PDF nativamente sin usar programas intermedios.

La versión actual de la aplicación es la 4.0.1. Si bien la anterior versión estable, la 1.1.5, presentaba una apariencia poco atractiva, las versiones 2.x (todavía disponibles en su página Web) no sólo han mejorado en este aspecto, sino también en cuanto a compatibilidad con otros formatos de archivo y a sencillez de uso. Actualmente la aplicación permite proteger documentos con contraseña, mantener versiones del mismo documento e insertar imágenes, objetos OLE, firmas digitales, símbolos, fórmulas, tablas de cálculo, gráficos, hiperenlaces, marcadores y formularios, entre otros. OpenOffice Writer es también un potente editor HTML, con la misma facilidad de uso que proporciona como procesador de texto e incorporando una extensa galería de imágenes, texturas y botones. Permite crear fácilmente etiquetas y tarjetas de presentación, sin necesidad de modificar el formato del documento de texto, y es totalmente configurable, permitiendo modificar cualquier opción de página, botones, barra de herramientas, idioma, autocorrección, ortografía, etc. Además, ante cualquier duda, el usuario dispone de un servicio de ayuda adecuado.

Para OpenOffice Writer el número de usuarios que participaron en la aplicación de las técnicas observación directa e información post-test fue de 16: una mitad, familiares y amigos; la otra, usuarios reales. Al contrario que en el caso de FreeMind, OpenOffice Writer sí contaba con una lista de correos de usuarios, de modo que no fue necesario buscar en foros. La lista de correos de OpenOffice Writer contaba con 9.000 usuarios. Debido a la confidencialidad de los *correos electrónicos*, hubo de ser uno de los administradores del proyecto quien enviase un primer *correo electrónico* a todos los usuarios. En él se pedía a los usuarios que completaran una encuesta y se les solicitaba autorización para compartir los datos de la encuesta con un grupo de investigadores. Entre los datos que se solicitaban en la encuesta se encontraba el *correo electrónico* de contacto del usuario. Un total de 1.121 usuarios respondieron la encuesta, de los cuales 956 autorizaron compartir la información, pero sólo 644 usuarios suministraron su *correo electrónico*. Enviamos un segundo correo a dichos 644 usuarios con el objetivo de describir las técnicas de usabilidad que serían aplicadas, las actividades que se iban a realizar y las herramientas necesarias, así como para informar del tiempo que requeriría la actividad. De los 644 correos electrónicos, 9 fueron rechazados por el servidor de correo (direcciones inexistentes), de suerte que finalmente se pudo contactar con 635 usuarios, de los cuales respondieron 132.

De estos 132 usuarios, 60 no aceptaron participar en la aplicación de las técnicas, mientras que 72 sí mostraron su interés. Tras leer los 72 correos, se distinguieron dos grupos diferentes de usuarios. Por un lado, los usuarios que estaban interesados en participar pero que, debido a ciertas circunstancias (por ejemplo, no tienen cámara Web, conexión de internet lenta, no pueden hablar bien debido a una enfermedad) no podían hacerlo; por otro lado, los usuarios que estaban interesados en participar y podían hacerlo. Cada uno de estos dos grupos estaba constituido por 36 usuarios. Se envió un correo a 15 de los usuarios interesados y en disposición de participar en el que se informaba del horario disponible y se preguntaba por el horario en el cual podían reunirse virtualmente para aplicar las técnicas. De los 15 correos enviados, 10 tuvieron respuesta. Finalmente, se consiguió una reunión con un total de 8 usuarios. No fue posible aplicar las técnicas con los otros dos usuarios porque uno de ellos declinó finalmente participar, a pesar del interés mostrado en los correos iniciales, mientras que el otro canceló la cita en el último momento a causa de un imprevisto en el trabajo. La Figura 7.12 resume en cifras y clases el reclutamiento de usuarios para el caso de estudio.

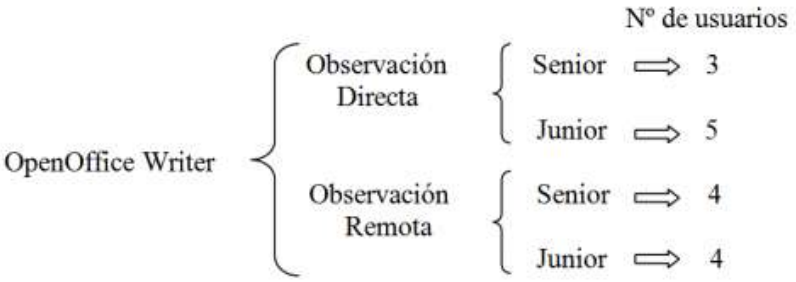


Figura 7.12: Número de Usuarios a los que se Aplicó cada Técnica.

7.4.2. Ejecución

Para la aplicación de la técnica observación directa/remota, el material suministrado a los usuarios para la realización de las tareas viene dado por las Figuras 7.13 y 7.14, dependiendo de si el sujeto es junior o senior, respectivamente.

Con respecto a la técnica *perfiles de usuario*, todos los usuarios que participaron lo hicieron remotamente. La participación de los usuarios consistía en completar un cuestionario que era enviada por correo electrónico. Dicho cuestionario fue diseñada en conjunción con uno de los administradores del proyecto OpenOffice Writer. A partir de una versión inicial e intercambiando correos electrónicos con propuestas de mejora para el cuestionario se logró llegar -luego de varias iteraciones- a una versión final que satisfacía a ambas partes (administrador-investigadores). Posteriormente, se envió este cuestionario por correo electrónico a los usuarios obteniendo la información necesaria para la creación de los perfiles de usuario. La Figura 7.15 ilustra el cuestionario diseñado. El cuestionario fue enviado a un total de 9.000 usuarios, de los cuales respondieron 1.121. Sin embargo, solo 956 autorizaron compartir los datos con los autores del presente trabajo de investigación.

Tareas a realizar en la Herramienta "OpenOffice Writer"

Utilice la aplicación *"OpenOffice Writer"* para realizar la siguiente tarea.

Las acciones a realizar son:

1. Abrir la aplicación *OpenOffice*
2. Abrir un *Documento de texto*
3. Crear un nuevo documento
4. Digitar el siguiente texto:

   Apache OpenOffice es una suite ofimática libre (código abierto y distribución gratuita) que incluye herramientas como procesador de textos, hoja de cálculo, presentaciones, herramientas para el dibujo vectorial y base de datos. Está disponible para varias plataformas, tales como Microsoft Windows, GNU/Linux, BSD, Solaris y Mac OS X. Soporta numerosos formatos de archivo, incluyendo como predeterminado el formato estándar ISO/ IECOpenDocument (ODF), entre otros formatos comunes, así como también soporta más de 110 idiomas, desde febrero del año 2010.

5. Justificar el párrafo anterior.
6. Use clic derecho para cambiar el tipo de letra a Times New Roman y el tamaño de letra a 11, al párrafo anterior.
7. Colocar sobre la palabra Solaris la siguiente nota al pie de página: Sistema operativo de tipo Unix desarrollado inicialmente por Sun Microsystems.
8. Debajo del párrafo anterior, insertar una imagen de la galería de OpenOffice Writer (que represente a un turista).
9. Colocar borde a todos los lados de la imagen insertada anteriormente.
10. Cambiar la margen izquierda de la página a 3 cm.
11. Colocar paginación en la esquina inferior izquierda a todas las páginas del documento.
12. Debajo de la imagen insertar una tabla con 3 columnas y 7 filas. Utilizar el autoformato de tabla Gris. La tabla resultado debe ser igual a la siguiente:

	Nombre de la Aplicación	Descripción
	Writer	**Procesador de textos similar a *Microsoft Word*.**
	Calc	**Hoja de cálculo similar a *Microsoft Excel* o *Lotus 1-2-3*.**
	Impress	**Programa de presentación similar a *Microsoft PowerPoint* o *Keynote de Apple*.**
	Base	**Programa de base de datos similar a *Microsoft Access*.**
	Draw	**Editor de gráficos vectoriales y herramientas de diagramación similar a *Microsoft Visio*.**
	Math	**Aplicación diseñada para la creación y edición de fórmulas matemáticas.**

13. En la tabla anterior, centrar verticalmente el contenido de las celdas de la columna "Nombre de la Aplicación".
14. Guardar el archivo con el nombre OpenOffice-tareas en formato texto de OpenOffice (es decir, con extensión .odt) en el escritorio.
15. Exportar el archivo a formato pdf y guardarlo con el mismo nombre en el escritorio.

Figura 7.13: Tareas a Realizar por los Usuarios Junior con la Aplicación OpenOffice Writer.

7.4.3. Resultados

Comenzamos la presente subsección mostrando los resultados individuales derivados de la encuesta SUS realizada por usuarios junior (Figura 7.16) y senior (Figura 7.17) de la aplicación OpenOffice Writer. Por su parte, la Figura 7.18 sintetiza los anteriores resultados e incluye la interpretación de las puntuaciones en términos de niveles de usabilidad.

Tareas a realizar en la Herramienta "OpenOffice Writer"

Utilice la aplicación *"OpenOffice Writer"* para realizar la siguiente tarea.

Las acciones a realizar son:

1. Abrir la aplicación *OpenOffice*
2. Abrir un *Documento de texto*
3. Crear un nuevo documento
4. Digitar el siguiente texto:

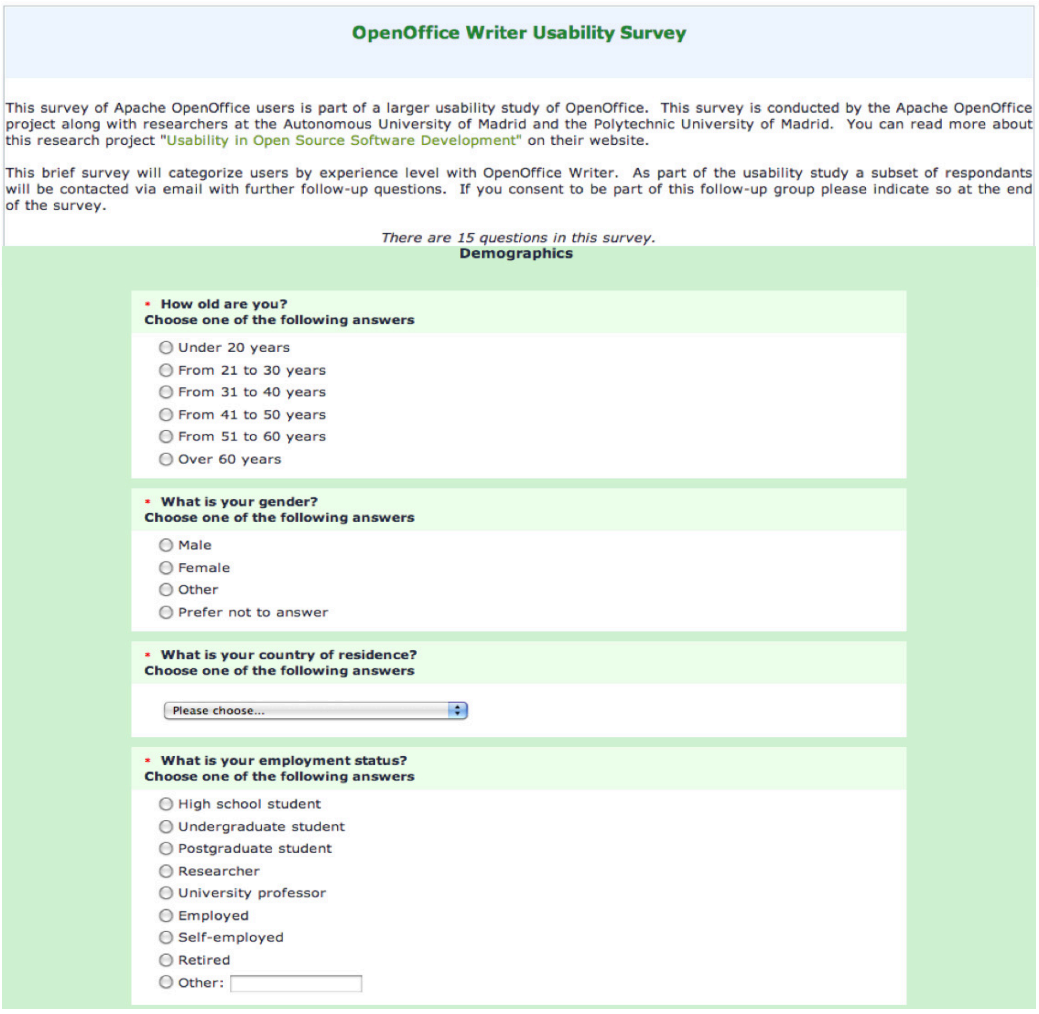
   Apache OpenOffice es una suite ofimática libre (código abierto y distribución gratuita) que incluye herramientas como procesador de textos, hoja de cálculo, presentaciones, herramientas para el dibujo vectorial y base de datos. Está disponible para varias plataformas, tales como Microsoft Windows, GNU/Linux, BSD, Solaris y Mac OS X. Soporta numerosos formatos de archivo, incluyendo como predeterminado el formato estándar ISO/ IECOpenDocument (ODF), entre otros formatos comunes, así como también soporta más de 110 idiomas, desde febrero del año 2010.

5. Justificar el párrafo anterior.
6. Use clic derecho para cambiar el tipo de letra a Times New Roman y el tamaño de letra a 11, al párrafo anterior.
7. Colocar sobre la palabra Solaris la siguiente nota al pie de página: Sistema operativo de tipo Unix desarrollado inicialmente por Sun Microsystems.
8. Debajo del párrafo anterior, insertar una imagen de la galería de OpenOffice Writer (que represente a un turista).
9. Colocar borde a todos los lados de la imagen insertada anteriormente.
10. Cambiar la margen izquierda de la página a 3 cm.
11. Colocar paginación en la esquina inferior izquierda a todas las páginas del documento.
12. Debajo de la imagen insertar una tabla con 3 columnas y 7 filas. Utilizar el autoformato de tabla Gris. La tabla resultado debe ser igual a la siguiente:

	Nombre de la Aplicación	Descripción
	Writer	**Procesador de textos similar a *Microsoft Word*.**
	Calc	**Hoja de cálculo similar a *Microsoft Excel* o *Lotus 1-2-3*.**
	Impress	**Programa de presentación similar a *Microsoft PowerPoint* o *Keynote de Apple*.**
	Base	**Programa de base de datos similar a *Microsoft Access*.**
	Draw	**Editor de gráficos vectoriales y herramientas de diagramación similar a *Microsoft Visio*.**
	Math	**Aplicación diseñada para la creación y edición de fórmulas matemáticas.**

13. En la tabla anterior, centrar verticalmente el contenido de las celdas de la columna "Nombre de la Aplicación".
14. Guardar el archivo con el nombre OpenOffice-tareas en formato texto de OpenOffice (es decir, con extensión .odt) en el escritorio.
15. Exportar el archivo a formato pdf y guardarlo con el mismo nombre en el escritorio.
16. Insertar el siguiente encabezamiento al documento: OpenOffice Writer.
17. Insertar el siguiente pie de pagina al documento: Evaluación de la Usabilidad.
18. Realizar los siguientes cambios a la tabla insertada en el punto 12:
    - Cambiar el color de la fuente por negro a toda la tabla.
    - Centrar los títulos de las columnas "Nombre de la Aplicación" y "Descripción".
    - Colocar como título a la primera columna "Aplicaciones OpenOffice".
    - Unir todas las celdas (incluido el título) de la primera columna.
    - Girar 90° el texto de la primera columna.
19. Ordenar toda la tabla alfabéticamente por la columna "Nombre de la Aplicación".
20. Seleccionar todo el texto del párrafo (ingresado en el punto 4) solo a través de clics con el botón izquierdo y modificar la separación silábica a 4 caracteres al final de la línea.
21. Justo después del final del párrafo ingresado en el punto 4 insertar un hiperenlace a la siguiente web: http://es.wikipedia.org/wiki/Apache_OpenOffice.
22. Insertar después de la tabla una nueva imagen a partir de un archivo y vincular esta al documento.
23. Ajustar el tamaño de la imagen insertada en el punto anterior.
24. Modificar la imagen anterior para reflejar verticalmente la misma.
25. Exportar el archivo a formato pdf con calidad de compresión JPEG del 60%y guardarlo con el mismo nombre en el escritorio.

Figura 7.14: Tareas a Realizar por los Usuarios Senior con la Aplicación OpenOffice Writer.



OpenOffice Writer Usability Survey

This survey of Apache OpenOffice users is part of a larger usability study of OpenOffice. This survey is conducted by the Apache OpenOffice project along with researchers at the Autonomous University of Madrid and the Polytechnic University of Madrid. You can read more about this research project "Usability in Open Source Software Development" on their website.

This brief survey will categorize users by experience level with OpenOffice Writer. As part of the usability study a subset of respondants will be contacted via email with further follow-up questions. If you consent to be part of this follow-up group please indicate so at the end of the survey.

There are 15 questions in this survey.

Demographics

* **How old are you?**
Choose one of the following answers
- Under 20 years
- From 21 to 30 years
- From 31 to 40 years
- From 41 to 50 years
- From 51 to 60 years
- Over 60 years

* **What is your gender?**
Choose one of the following answers
- Male
- Female
- Other
- Prefer not to answer

* **What is your country of residence?**
Choose one of the following answers

Please choose...

* **What is your employment status?**
Choose one of the following answers
- High school student
- Undergraduate student
- Postgraduate student
- Researcher
- University professor
- Employed
- Self-employed
- Retired
- Other:

Figura 7.15: Fragmento del Cuestionario para Perfiles de Usuario.

SUS Calculation

Participant	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	SUS Score
Alejandro	4	2	3	1	3	3	3	2	3	2	**65,0**
Carlos	1	3	1	1	1	5	1	3	1	4	**22,5**
Cristina	1	2	3	1	4	1	5	1	3	3	**70,0**
Miguel Ángel	3	2	3	2	3	3	4	1	3	1	**67,5**
Olympia	3	1	4	1	4	1	5	2	5	1	**87,5**
Osvaldo	3	4	3	1	3	3	2	3	4	4	**50,0**
John	5	2	4	1	4	1	4	1	4	1	**87,5**
John O	5	2	4	2	3	2	5	2	4	2	**77,5**
Keith	4	2	4	1	4	2	4	1	4	1	**82,5**
									Promedio Total		**67,8**

Figura 7.16: Puntuaciones SUS Individuales de los Usuarios Junior.

SUS Calculation

Participant	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	SUS Score
Bernie	5	2	4	1	3	1	4	2	5	1	**85,0**
Donn	4	2	4	1	3	2	5	1	4	2	**80,0**
Fernando	5	1	4	2	4	1	3	2	5	2	**82,5**
José Miguel	5	1	4	2	3	2	2	1	4	2	**75,0**
JohnC	4	2	4	1	3	2	4	1	4	2	**77,5**
Yanedt	4	2	4	1	3	3	5	1	4	1	**80,0**
Sharon	4	3	2	1	3	3	2	1	4	4	**57,5**
									Promedio Total		**76,8**

Figura 7.17: Puntuaciones SUS Individuales de los Usuarios Senior.

Nombre del Proyecto: OpenOffice Writer	Versión del Proyecto: 4.0.1
Nombre de la Aplicación: OpenOffice Writer	Fecha: 01/10/2013

Preparado por: Cristina Martín Montero	Departamento: Ingeniería Informática, Escuela Politécnica Superior
Fecha del Informe: 14/05/2014	Email: cristina.martinm01@estudiante.uam.es

1. Estadísticas de los Participantes

Número Total de Participantes:
16

Tipo de Usuario	Número	Porcentaje
Junior	9	56,25%
Senior	7	43,75%

2. Resultados

Número de Preguntas	Tasa de Respuesta
10	100%

Tipo de Usuario	Promedio	Interpretación
Junior	67,8	Usabilidad Marginal
Senior	76,8	Usabilidad Aceptable

Figura 7.18: Informe de la Encuesta SUS.

Como resultado de la incorporación de la técnica observación directa, hemos encontrado diferentes problemas en la aplicación OpenOffice Writer. Las Tablas 7.8 y 7.9 presentan el listado de los problemas encontrados por los usuarios junior y senior, respectivamente. Las recomendaciones dadas por los usuarios para mejorar la usabilidad de OpenOffice Writer se presentan en la Tabla 7.10.

Tabla 7.8: Problemas Encontrados por los Usuarios Junior Durante la Observación Directa.

Problema	Nro. de Personas	Nivel de Relevancia	Mejoras
Los números de página no aparecen a pie de página	2	Alto	Insertar números de página de manera automática (a pie de página).
Opciones de menú apenas visibles	3	Alto	Aumentar la visibilidad de las opciones básicas.
El menú *insertar* no incluye una opción *galería*	3	Alto	Incluir la opción *galería* en el menú *insertar*.
No aparecen todos los tipos de letra cuando se buscan con clic derecho del ratón	5	Alto	Permitir la selección de cualquier tipo de letra con clic derecho del ratón.
Cambiar el margen izquierdo es poco intuitivo	1	Medio	Opción más visible a la hora de diseñar la página.

Tabla 7.8: Problemas Encontrados por los Usuarios Junior Durante la Observación Directa (continuación).

Problema	Nro. de Personas	Nivel de Relevancia	Mejoras
La herramienta de autoformato de tabla no se ve con facilidad y sólo está disponible al crear la tabla	1	Medio	Permitir cambiar el autoformato una vez creada la tabla.
Es difícil encontrar la opción de insertar números de página	1	Medio	Aumentar la visibilidad de las opciones básicas.
Es difícil encontrar la opción insertar notas a pie de página	1	Bajo	Aumentar la visibilidad de las opciones básicas.
Arrastrar la imagen de la galería	1	Bajo	Pinchar la imagen para que aparezca.
La aplicación no prima las opciones y botones que el usuario utiliza con más frecuencia	1	Bajo	Resaltar los botones y opciones más utilizadas por el usuario.
Quitar el panel izquierdo	1	Bajo	Situar el panel en la barra de herramientas.
Interfaz complicada, pésima	1	Bajo	Rediseñar la interfaz para que sea más intuitiva y fácil de usar.

Tabla 7.9: Problemas Encontrados por los Usuarios Senior Durante la Observación Directa.

Problema	Nro. de Personas	Nivel de Relevancia	Mejoras
El menú *insertar* no incluye una opción *galería*	1	Alto	Incluir la opción *galería* en el menú *insertar*.
Tipo de letra con el botón derecho	2	Alto	Todos los tipos de letras a través del botón derecho.
Los números de página no aparecen a pie de página	2	Alto	Insertar números de página de manera automática (a pie de página).
Resulta difícil vincular una imagen a un documento	1	Medio	Facilitar la opción de vincular una imagen a un documento.
Menús poco intuitivos (por ejemplo, menú *campos* y *carácter*)	2	Medio	Utilizar nombres más claros y significativos para los menús.
Ordenar la tabla no se puede coger el encabezamiento	1	Bajo	Poder ordenar cogiendo toda la tabla y decir porqué fila/columna ordenar.
La galería si se abre con Herramientas, se queda abierta, si no se quita el tick, y el panel derecho también se queda abierto	1	Bajo	Cerrar la galería una vez seleccionada la imagen.

Tabla 7.10: Recomendaciones de Usabilidad Dadas por los Usuarios Durante la Observación Directa.

Mejora	Clasificación	Prioridad
Resolver el tipo de letra con el botón derecho	Mejora de la interacción	Alta
Facilitar el acceso a la opción de insertar números de página	Mejora de la interacción	Alta
Facilitar el acceso a *galería*	Mejora de la interacción	Alta
Poner en cada pestaña las opciones de cada herramienta relacionada con ello	Mejora de la interacción	Alta
Preguntar al usuario dónde insertar los números de página: en el encabezado o en el pie de página	Mejora de la interacción	Alta
Cambiar los nombres de algunos menús por otros más claros y significativos	Mejora de la interacción	Alta
La opción Herramientas/Galería tendría que ser Insertar/Imagen/Galería	Mejora de la interacción y diseño de la interfaz	Alta
Eliminar el panel derecho de la interfaz	Diseño de la interfaz	Media
Retocar la interfaz para que sea más intuitiva	Mejora de la interacción	Media
Diseño para los encabezados	Diseño de la interfaz	Media
Extraer el desplegable del menú *campos* y situarlo dentro de las opciones de *insertar*, separado de las otras opciones	Diseño de la interfaz	Media
Cerrar la galería una vez seleccionada la imagen, ya sea en *herramientas* o en el panel derecho	Mejora de la interacción	Media
Desmarcar de forma automática la opción de vincular la imagen, cuando se sale de la ventana	Mejora de la interacción	Media
Valor por defecto de forma automática la opción de calidad de compresión, cuando se sale de la ventana	Mejora de la interacción	Media
Proporcionar un ayudante virtual	Mejora de la interacción	Baja
Resaltar los botones y opciones que más se utilicen y quitar las que no se utilicen a menudo	Diseño de la interfaz	Baja
Ordenar la tabla sin eliminar su encabezado	Mejora de la interacción	Baja

De manera análoga a las Tablas 7.8 y 7.9, las Tablas 7.11 y 7.12 presentan los diferentes problemas encontrados por los usuarios durante la aplicación de la técnica observación directa pero aplicada de manera remota. Estos problemas están relacionados directamente con el uso de la aplicación. Las recomendaciones dadas por los usuarios para mejorar la usabilidad de OpenOffice Writer se presentan en la Tabla 7.13.

Tabla 7.11: Problemas Encontrados por los Usuarios Junior Durante la Observación Remota.

Problema	Nro. de Personas	Nivel de Relevancia	Mejoras
El menú *insertar* no incluye una opción *galería*	2	Alto	Incluir la opción *galería* en el menú *insertar*.
Cambiar el margen	1	Medio	Ubicación más sencilla de encontrar el margen.
Resulta complicado buscar las opciones pedidas en el menú *herramientas*	2	Medio	Rediseñar el menú *herramientas* para que sea más intuitivo.
El formato del documento cambia al abrirlo con Microsoft Word	1	Bajo	Explicar como abrir documentos en otra aplicación.

Tabla 7.12: Problemas Encontrados por los Usuarios Senior Durante la Observación Remota.

Problema	Nro. de Personas	Nivel de Relevancia	Mejoras
El menú *insertar* no incluye una opción *galería*	1	Alto	Incluir la opción *galería* en el menú *insertar*.
La herramienta de autoformato de tabla no se ve con facilidad y sólo está disponible al crear la tabla	1	Alto	Permitir cambiar el autoformato una vez creada la tabla.
La imagen salía entre el texto, difícil de mover	1	Medio	Al arrastrar la imagen facilidad de posicionamientos.
Dificultad al encontrar la separación silábica	1	Medio	Situar en una ubicación más accesible la separación silábica.
Tamaño del icono *galería*	1	Medio	Situar la opción *galería* en el menú *insertar* o aumentar el tamaño del icono.
Los iconos que salen al insertar la imagen no son representativos	1	Medio	Utilizar iconos más representativos para la imagen.
Iconos poco representativos	1	Bajo	Utilizar iconos más representativos en la barra de herramientas.
Es difícil ordenar una tabla, porque hay que seleccionar todas sus celdas.	1	Bajo	Ordenar con sólo seleccionar una fila/columna.
Las ventanas emergentes no son conocidas por los usuarios	1	Bajo	Poner título a las ventanas emergentes.
Los iconos no están ordenados por su función	1	Bajo	Ordenar los iconos por función.
El cursor cambia al situarse sobre un fondo gris	1	Bajo	Mantener el mismo tipo de cursor para todos los colores de fondo.
Se alteran las fuentes al copiar desde un documento PDF y pegar sin formato	1	Bajo	Conservar el formato original del documento PDF.

Tabla 7.13: Recomendaciones de Usabilidad Dadas por los Usuarios Durante la Observación Remota.

Mejora	Clasificación	Prioridad
Barra de herramientas más grande y visible	Diseño de la interfaz	Alta
Posicionar la imagen porque no hay un wizard para ponerla con opciones	Mejora de la interacción	Alta
Mejoras a la hora de hacer los índices	Mejora de la interacción	Alta
Hay problemas con el formato, los estilos no funcionan bien	Mejora de la interacción	Alta
Inserción de números de página más intuitiva	Mejora de la interacción	Alta
La aplicación debe ser más intuitiva como Kingsoff Office, ya que es más fácil de manipular	Mejora de la interacción	Media

Tabla 7.13: Recomendaciones de Usabilidad Dadas por los Usuarios Durante la Observación Remota (continuación).

Mejora	Clasificación	Prioridad
Iconos más estéticos	Diseño de la interfaz	Media
La interfaz debe mantenerse simple	Diseño de la interfaz	Media
Facilitar la búsqueda en la aplicación, pues posee mucha funcionalidad	Mejora de la interacción y diseño de la interfaz	Media
Botones situados en la parte superior son confusos	Diseño de la interfaz	Media
Numeración funciona de manera diferente	Mejora de la interacción	Media
Solucionar el problema de hacer el diccionario de las palabras más buscadas	Mejora de la interacción	Media
Buscar y reemplazar expresiones regulares	Mejora de la funcionalidad	Media
Crear *macros* en OpenOffice Writer es difícil	Mejora de la interacción	Media
Facilidad de acceso a la ayuda directamente desde la acción que se está realizando	Mejora de la interacción	Media
A la interfaz le falta algo más de color y de diseño minimalista	Diseño de la interfaz	Baja
Problemas al abrir el fichero, porque tiene demasiado zoom	Mejora de la interacción	Baja
El corrector ortográfico se ve afectado por el teclado al trabajar con algunos idiomas	Mejora de la interacción	Baja

Finalmente, a manera de resumen la Tabla 7.14 presenta todos los diferentes problemas de usabilidad detectados por los usuarios durante la aplicación de las técnicas observación directa/remota e información post-test. Para cada uno de estos problemas se propone una mejora.

Tabla 7.14: Listado de los Diferentes Problemas y Mejoras Encontrados Durante la Aplicación de las Técnicas Directa e Información Post-Test.

Problema	Técnicas	Frecuencia	Mejoras	Autor Mejora
Es difícil encontrar la opción de insertar notas a pie de página	Directa y Post-test	1 de 16	Aumentar la visibilidad de las opciones básicas.	Cristina
Arrastrar la imagen de la galería	Directa y Post-test	1 de 16	Pinchar la imagen para que aparezca.	Cristina
Cambiar el margen izquierdo es poco intuitivo	Directa y Post-test	1 de 16	Aumentar la visibilidad de la opción.	Cristina
La aplicación no prima las opciones y botones que el usuario usa con más frecuencia	Directa y Post-test	1 de 16	Resaltar los botones y opciones más utilizadas por el usuario.	Usuario
Quitar el panel izquierdo	Directa y Post-test	1 de 16	Poner el panel en la barra de herramientas.	Usuario
Interfaz complicada, pésima	Directa y Post-test	1 de 16	Interfaz más intuitiva y fácil.	Usuario
Es dificil encontrar la opción de insertar números de página	Directa y Post-test	1 de 16	Aumentar la visibilidad de las opciones básicas.	Cristina

Tabla 7.14: Listado de los Diferentes Problemas y Mejoras Encontrados Durante la Aplicación de las Técnicas Directa e Información Post-Test (continuación).

Problema	**Técnicas**	**Frecuencia**	**Mejoras**	**Autor Mejora**
Resulta difícil vincular una imagen a un documento	Directa y Post-test	1 de 16	Facilitar la opción de vincular una imagen a un documento.	Cristina
Ordenar la tabla no se puede coger el encabezamiento	Directa y Post-test	1 de 16	Poder ordenar cogiendo toda la tabla y decir porqué fila/columna ordenar.	Usuario
La galería si se abre con Herramientas, se queda abierta, si no se quita el tick, y el panel derecho también se queda abierto	Directa y Post-test	1 de 16	Cerrar la galería una vez seleccionada la imagen.	Usuario
Vincular la imagen se queda marcada cuando se sale de la ventana y se vuelve a entrar	Directa y Post-test	1 de 16	Desmarcar de forma automática la opción cuando se sale de la ventana.	Usuario
Calidad de compresión 60 %, se queda guardado	Directa y Post-test	1 de 16	Valor por defecto de forma automática la opción cuando se sale de la ventana.	Usuario
El formato del documento cambia al abrirlo con Microsoft Word	Directa y Post-test	1 de 16	Explicar cuál es el formato para abrir documentos en otra aplicación.	Cristina
Cambiar el margen	Directa y Post-test	1 de 16	Ubicación más sencilla de encontrar el cambio de margen.	Cristina
La imagen salía entre el texto, difícil de mover	Directa y Post-test	1 de 16	Al arrastrar la imagen facilidad de posicionamientos.	Cristina
Dificultad al encontrar la separación silábica	Directa y Post-test	1 de 16	Situar en una ubicación más accesible la separación silábica.	Cristina
Tamaño del icono *galería*	Directa y Post-test	1 de 16	Poner la opción de Galería en el menú de Insertar o aumentar el tamaño del icono.	Cristina
Iconos poco representativos	Directa y Post-test	1 de 16	Utilizar iconos más representativos en la barra de herramientas.	Usuario
Los iconos que salen al insertar la imagen no son representativos	Directa y Post-test	1 de 16	Utilizar iconos más representativos para la imagen.	Usuario
Es difícil ordenar una tabla, porque hay que seleccionar todas sus celdas.	Directa y Post-test	1 de 16	Ordenar con sólo seleccionar una fila/columna.	Usuario
El usuario no reconoce las ventanas emergentes	Directa y Post-test	1 de 16	Poner título a las ventanas emergentes.	Cristina

Tabla 7.14: Listado de los Diferentes Problemas y Mejoras Encontrados Durante la Aplicación de las Técnicas Directa e Información Post-Test (continuación).

Problema	Técnicas	Frecuencia	Mejoras	Autor Mejora
No ubica bien los iconos según su funcionalidad	Directa y Post-test	1 de 16	Ordenar los iconos por función.	Cristina
El cursor cambia al situarse sobre un fondo gris	Directa y Post-test	1 de 16	Mantener el mismo tipo de cursor para todos los colores de fondo.	Cristina
Se alteran las fuentes al copiar desde un documento PDF y pegar sin formato	Directa y Post-test	1 de 16	Conservar el formato original del documento PDF.	Cristina
La herramienta de autoformato de tabla no se ve con facilidad y sólo está disponible al crear la tabla	Directa y Post-test	2 de 16	Permitir cambiar el autoformato una vez creada la tabla.	Usuario
Menús poco intuitivos (por ejemplo, menú *campos* y *carácter*)	Directa y Post-test	2 de 16	Utilizar nombres más claros y significativos para los menús.	Usuario
Resulta complicado buscar las opciones pedidas en el menú *herramientas*	Directa y Post-test	2 de 16	Rediseñar el menú *herramientas* para que sea más intuitivo.	Cristina
Opciones de menú poco visibles	Directa y Post-test	3 de 16	Aumentar la visibilidad de las opciones básicas.	Cristina
Número de página no aparece en la parte inferior y tiene que ir con pie de página	Directa y Post-test	4 de 16	Insertar números de página automáticamente en la parte inferior.	Usuario
Tipo de letra con el botón derecho	Directa y Post-test	7 de 16	Todos los tipos de letras a través del botón derecho.	Usuario
El menú *insertar* no incluye una opción *galería*	Directa y Post-test	7 de 16	Incluir la opción *galería* en el menú *insertar*.	Cristina

7.5. Conclusiones

Los resultados obtenidos al incorporar técnicas de la IPO con transformaciones son satisfactorios. Al participar como voluntarios en dos proyectos reales OSS hemos podido validar que es posible incorporar técnicas de la IPO gracias a ciertas transformaciones. Desde un principio, los administradores de ambos proyectos se han mostrado entusiasmados con la idea de aplicar técnicas de usabilidad y siempre han estado dispuestos a colaborar para todo aquello en lo que han sido requeridos. Asímismo, algunos de los usuarios se han mostrado muy interesados, desde el primer instante, por las técnicas y por conocer más sobre ellas, llegando a agradecer este tipo de iniciativas. Por tanto, la incorporación de técnicas de usabilidad en OSS contará con un aspecto muy positivo: el entusiasmo tanto de los jefes de proyecto como de los participantes.

Sin embargo, durante la aplicación de las técnicas hemos encontrado tres problemas. En primer lugar, lo difícil de coordinar la cita con el usuario debido a la diferencia horaria que existe entre los diferentes países donde se encuentran los usuarios. En segundo lugar, la conexión a Internet. Algunos usuarios tenían problemas con la conexión lo que dificultaba verlos a través de la webcam o escucharlos claramente. En tercer lugar, una minoría de los usuarios

no disponía de los programas necesarios instalados en sus ordenadores (a pesar de que fueron informados con anterioridad y en repetidas ocasiones), por lo que había que ayudarles a instalarlos ocasionando que el tiempo para aplicar las técnicas aumentara.

Tres técnicas de evaluación de la usabilidad han sido aplicadas a los proyectos FreeMind y OpenOffice Writer: observación directa, información post-test y encuesta SUS. Esta última es la única que ha sido incorporada según prescribe la IPO. Para todas las demás ha sido necesario realizar transformaciones para poder ser incorporadas en los desarrollos OSS. Con estas técnicas y con la participación de diferentes tipos de usuarios, tanto junior como senior, se han extraído los diferentes problemas de usabilidad de cada aplicación y se han llevado a cabo propuestas de mejora. Problemas y mejoras, en ambas aplicaciones, están relacionados con el diseño de la interacción y el diseño de la interfaz de usuario y de la organización de ésta. Para usuarios senior, los problemas y sus mejoras asociadas están relacionados mayoritariamente con la interacción, mientras que para los usuarios junior la mayoría de los problemas encontrados y mejoras propuestas están relacionados con el diseño de la interfaz.

Como resultado de la aplicación de estas técnicas, se ha descubierto que el número de usuarios con interés en participar es mayor cuando su participación no requiere una gran inversión de tiempo. Si su participación exige, por ejemplo, la instalación de software adicional o permitir el acceso remoto a su ordenador, el número de usuarios interesados en participar disminuye considerablemente. No obstante, es importante destacar que siempre se encontraron usuarios comprometidos para los que el tiempo no supuso impedimento alguno.

CAPÍTULO 8

CONCLUSIONES

Finalizada la exposición de la solución propuesta y de los resultados obtenidos en la aplicación de la misma, en este capítulo se valoran las conclusiones obtenidas de la realización del presente trabajo, así como las oportunidades que deja abiertas para continuar la investigación en nuevas direcciones.

La creciente importancia y popularidad de las aplicaciones OSS en los últimos años ha llevado a la comunidad científica y a la industria a estudiar diferentes aspectos de los desarrollos OSS. Este interés ha generado una gran variedad de publicaciones. En la literatura existen pocos trabajos de investigación que realicen una recopilación y clasificación de estas publicaciones. Estas recopilaciones de literatura son generales. Los trabajos de Jalali y Wohlin [96] y Mulazzani et al. [130] realizan una recopilación de la literatura a través de un SMS. Mientras que los trabajos de Aksulu y Wade [22] y Crowston et al. [54] realizan recopilaciones ad hoc. Sin embargo, no existe un trabajo de investigación que realice una recopilación de los modelos de proceso de desarrollo OSS publicados en la literatura.

El presente trabajo de investigación realiza un estudio de los procesos de desarrollo OSS publicados en la literatura. La recopilación de la literatura se ha realizado a través de un SMS. Al realizar el estudio de los diferentes modelos de proceso de desarrollo OSS, se han encontrado dos problemas. Primero, los nombres de las actividades dados por los autores eran confusos. Segundo, la descripción de una actividad realmente se corresponde con dos o más actividades diferentes. No hemos encontrado en la literatura un trabajo de investigación que analice si los nombres de las actividades de los procesos de desarrollo OSS son acordes con las descripciones dadas por los autores.

En la literatura existe una variedad de trabajos de investigación que estudian diferentes aspectos relacionados con la usabilidad en la comunidad OSS. A través de un SMS, hemos clasificado esta literatura en tres grupos. En el primero, clasificamos los trabajos que plantean la problemática de la usabilidad en los desarrollos OSS (por ejemplo, [28][44][47][56]). En el segundo, catalogamos los trabajos que analizan la percepción que tiene la comunidad OSS de la usabilidad (por ejemplo, [39][160][188]). En el tercero, registramos los trabajos que estudian las técnicas de usabilidad incorporadas en los desarrollos OSS (por ejemplo, [23][35][38][188]). Sin embargo, no existe en la literatura un trabajo de investigación que analice de manera conjunta estos tres aspectos (problemática, percepción y técnicas de usabilidad) y exponga el estado actual de la usabilidad en la comunidad OSS, como sí lo hace el presente trabajo.

Los diferentes problemas relacionados con la usabilidad a los que se enfrenta la comunidad OSS están dispersos en la literatura. Existen trabajos de investigación dedicados a estudiar

algunos de estos problemas (por ejemplo, [47][136][137]). Mientras que otros se limitan a comentarlos (por ejemplo, [35][44][89]), solo como parte de la introducción de su trabajo. Sin embargo, no existe un trabajo que agrupe ordenadamente estos problemas y analice de qué manera afectan la usabilidad de las aplicaciones OSS. La presente investigación realiza tal análisis, clasificando los problemas en dos grandes grupos. En el primer grupo, clasificamos los problemas relacionados con el modo de desarrollo OSS. En el segundo grupo, agrupamos los problemas relacionados con el hecho de que los proyectos OSS, por lo general, no siguen las recomendaciones del área de la IPO para mejorar la usabilidad o los problemas derivados al intentar adoptar tales recomendaciones.

Al realizar el análisis de los trabajos de investigación que estudian las técnicas de usabilidad incorporadas en OSS, hemos identificado tres problemas. Primero, algunos autores describen la técnica de usabilidad sin asociarla al nombre generalmente conocido para esa técnica [46][132][152][188]. Por tanto, los autores no han asociado estas técnicas con las técnicas de usabilidad existentes en la literatura de la IPO. Esta práctica contribuye a dificultar la comprensión de qué técnicas de la IPO han sido usadas en los desarrollos OSS.

Segundo, ciertos investigadores afirman que la comunidad OSS ha creado nuevas técnicas para mejorar la usabilidad de sus aplicaciones [137][188]. Un análisis profundo de estas técnicas y su comparación con las técnicas de usabilidad existentes, nos ha permitido descubrir que estas nuevas supuestas técnicas no son una creación de la comunidad OSS. El área de la IPO tiene un amplio recorrido investigando, entre otras cosas, cómo mejorar la usabilidad de los sistemas software a través de la aplicación de técnicas de usabilidad, con lo cual la comunidad OSS no ha inventado nada. Realmente, lo que ha ocurrido es que la técnica ha sufrido grandes transformaciones que, a primera vista, la hacen parecer una completamente nueva.

Tercero, los autores describen para ciertas técnicas de usabilidad cómo estas han sido aplicadas en algunos proyectos OSS. Sin embargo, no comparan si la forma en que fueron aplicadas las técnicas se corresponde con lo prescrito por la IPO. Los autores no se cuestionan las razones por las cuales la técnica ha sido incorporada con modificaciones, ni tampoco se cuestionan porqué hay diferencias con respecto a la teoría (si las hubiese). Estos tres problemas no han sido previamente identificados por otros investigadores. Este trabajo identifica y contribuye a resolver dichos problemas.

Unos pocos trabajos de investigación [47][44][89][188] han propuesto que las técnicas de usabilidad de la IPO deben ser adaptadas para que concuerden con la cultura de la comunidad OSS y su modo de desarrollo. Sin embargo, estos trabajos de investigación no explican cuáles son estas adaptaciones, qué motiva estas adaptaciones, qué problemática resuelven las adaptaciones, ni en qué consisten tales adaptaciones, no exponen si la adaptación afecta a todas las técnicas o únicamente a unas pocas, ni cuáles técnicas de usabilidad pueden ser o no adaptadas y porqué no pueden serlo otras. El trabajo de investigación de Nichols y Twidale [136] presenta algunas ideas generales para mejorar la usabilidad en OSS (como por ejemplo, involucrar estudiantes de IPO en los proyectos OSS). Sin embargo, no explica por ejemplo las consideraciones que deben ser tenidas en cuenta al incorporar tales técnicas en los desarrollos OSS, ni tampoco detalla cuáles técnicas de usabilidad pueden ser aplicadas por los estudiantes de la IPO, ni cómo deben participar los estudiantes en los proyectos OSS. Proponemos un marco de integración que permite incorporar la mayoría de las técnicas de usabilidad en los desarrollos OSS. Para ello se deben aplicar diversas transformaciones a las técnicas. Estas transformaciones dependen de las exigencias de la técnica que no pueden ser satisfechas por el modo de trabajo de la comunidad OSS.

Esta tesis también realiza una contribución con respecto a la praxis de aplicar técnicas de usabilidad en los proyectos OSS. Concretamente hemos participado en los proyectos OpenOffice Writer y FreeMind incorporando tres técnicas de usabilidad (perfiles de usuario, observación directa e información post-test) con diferentes transformaciones. Por ejemplo, en las técnicas observación directa e información post-test la participación de los usuarios ha sido en algunos casos presencial (los usuarios han sido amigos o familiares) y en otras remota (los usuarios están distribuidos por el mundo). La incorporación de éstas técnicas ha sido posible gracias a las transformaciones que hemos realizado a las técnicas. Las transformaciones realizadas a las técnicas han sido tomadas de nuestro marco de integración.

Benson et al. [35] en su trabajo de investigación cuestiona si en las listas de correo de los proyectos OSS se encuentran los usuarios de sus aplicaciones. Hemos encontrado con la participación como voluntarios en dos proyectos OSS reales que la crítica de Benson et al. [35] solamente es válida en algunos casos. En las listas de correo de los proyectos OSS grandes (por ejemplo, OpenOffice Writer) es posible encontrar toda una variedad de usuarios, desde usuarios con poca experiencia usando las aplicaciones hasta usuarios expertos que han usado la aplicación desde sus primeras versiones. Es posible caracterizar y conocer qué tipos de usuarios usan las aplicaciones OSS a través, por ejemplo, de técnicas de usabilidad como perfiles de usuario, tal como ha evidenciado nuestra investigación.

Existen pocos trabajos en la literatura que reporten la experiencia al incorporar técnicas de usabilidad en los desarrollos OSS. En el estudio empírico realizado por Rajanen et al. [152] un grupo de estudiantes de la IPO participan como voluntarios en un proyecto OSS. Los participantes aplican algunas técnicas de usabilidad como lo prescribe la IPO -gracias a que contaban con usuarios presenciales y con toda la infraestructura necesaria (por ejemplo, un laboratorio de usabilidad)- salvo por el hecho que el rol del experto en usabilidad es reemplazado por estudiantes de la IPO. Sin embargo, no existe un trabajo que reporte la experiencia de incorporar en proyectos OSS técnicas de usabilidad con adaptaciones, justifique porqué tales adaptaciones deben ser realizadas y proponga un marco general que permita incorporar ciertas técnicas en los desarrollos OSS, como se realiza en la presente investigación.

La solución propuesta al problema, tratado en el presente trabajo, de la mano con los resultados prometedores alcanzados, ha abierto nuevas líneas de investigación. En concreto, las siguientes líneas parecen prometedoras:

- Realizar un estudio empírico para determinar las técnicas de usabilidad que están siendo incorporadas en la comunidad OSS. Existe un gran número y variedad de proyectos OSS. Sin embargo, hemos encontrado en los resultados de nuestra investigación que la literatura existente solo ha estudiado las técnicas de usabilidad incorporadas en unos pocos proyectos OSS. En nuestro estudio, seleccionaremos una muestra representativa de comunidades OSS que abarque un rango de proyectos que varíen en términos del tipo de software producido y madurez del proyecto. Para la recolección de la información se diseñara un cuestionario que abarque las principales técnicas del catálogo que hemos utilizado como universo de técnicas de usabilidad. El cuestionario será enviado por email a los administradores de los proyectos OSS y a sus principales desarrolladores.

- Incorporar otras técnicas de usabilidad en los proyectos OpenOffice Writer y FreeMind aprovechando que ya somos parte de su comunidad de voluntarios. Además, seleccionaremos nuevos proyectos OSS (de la muestra definida previamente) donde incorporar técnicas de usabilidad. Para poder incorporar estas técnicas realizaremos las transformaciones correspondientes de acuerdo a nuestra propuesta de marco integración. Analizaremos los resultados obtenidos para determinar una guía de mejores prácticas en la incorporación de técnicas de usabilidad.

- Realizar un estudio de las herramientas existentes que faciliten la aplicación de técnicas de usabilidad en los proyectos OSS, debido a que los miembros de estos proyectos se encuentran distribuidos geográficamente por todo el mundo. Conocemos la existencia de algunas herramientas como por ejemplo *OptimalSort* y *XSort* que ayudan a la aplicación de la técnica de usabilidad card sorting. Es necesario investigar qué otras técnicas cuentan con herramientas similares.

Bibliografía

[1] Debian.org. *Debian Constitution.* Available at https://www.debian.org/devel/constitution, December 1998. [Accesed March 2014].

[2] Opensource.org. *Halloween Document I.* Available at http://www.catb.org/esr/halloween/halloween1.html, August 1998. [Accesed March 2014].

[3] Mozilla.org. *XPToolkit Architecture.* Available at http://www-archive.mozilla.org/xpfe/aom/AOM.html, March 1999. [Accesed March 2014].

[4] Mozilla.org. *Documentation Graveyard.* Available at http://www-archive.mozilla.org/classic/, August 2001. [Accesed March 2014].

[5] Mozilla.org. *Mozilla Development Documentation.* Available at https://developer.mozilla.org/en-US/docs/Developer_Guide, December 2001. [Accesed March 2014].

[6] Mozilla.org. *Quality Assurance at Mozilla.* Available at https://developer.mozilla.org/en-US/docs/Mozilla/QA, March 2001. [Accesed March 2014].

[7] Opensource.org. *Open Source Initiative.* Available at http://opensource.org/, 2002. [Accesed March 2014].

[8] Opensource.org. *Open Source Licenses.* Available at http://opensource.org/licenses, 2002. [Accesed March 2014].

[9] Opensource.org. *The Open Source Definition.* Available at http://opensource.org/docs/definition.html, 2002. [Accesed March 2014].

[10] Freestandards.org. *Filesystem Hierarchy Standard.* Available at http://www.pathname.com/fhs/, October 2003. [Accesed March 2014].

[11] *Linux Usability Report version 1.01.* Available at http://www.relevantive.com/Linux-Usabilitystudy_e.html, 2003. [Accesed February 2014].

[12] *Return on Investment for Usability.* Available at http://www.useit.com, January 2003. [Accesed April 2014].

[13] Freem. *A Portrait of J. Random Hacker.* Available at http://freem.vmth.ucdavis.edu/ saintly/bio/portrait.html, January 2004. [Accesed June 2014].

[14] Mozilla.org. *Layout Engine Technical Documentation.* Available at http://www-archive.mozilla.org/newlayout/doc/, April 2008. [Accesed March 2014].

[15] *GNOME Usability Project.* Available at https://wiki.gnome.org/UsabilityProject/, August 2010. [Accesed February 2014].

[16] *Season of Usability.* Available at http://season.openusability.org/index.php/home, 2010. [Accesed April 2014].

[17] Openusability.org. *Open Usability*. Available at http://season.openusability.org/index.php/about, 2011. [Accesed February 2014].

[18] *Better Desktop Usability*. Available at https://www.suse.com/products/desktop/features/usability.html, 2011. [Accesed February 2014].

[19] *KDE Usability Project*. Available at http://techbase.kde.org/Projects/Usability, December 2013. [Accesed February 2014].

[20] Openoffice.org. *OpenOffice User Experience Project*. Available at https://www.openoffice.org/ux/, February 2014. [Accesed February 2014].

[21] T. Adlin and J. Pruitt. *The Essential Persona Lifecycle: Your Guide to Building and Using Personas*. Morgan Kaufmann, Burlington, MA, USA, 1st edition, 2010.

[22] A. Aksulu and M. Wade. A Comprehensive Review and Synthesis of Open Source Research. *Journal of the Association for Information Systems*, 11(11):576–656, November 2010.

[23] M. S. Andreasen, H. V. Nielsen, S. O. Schrøder, and J. Stage. Usability in Open Source Software Development: Opinions and Practice. *Information Technology and Control*, 35A(3):303–312, 2006.

[24] M. Aoyama. Persona-Scenario-Goal Methodology for User-Centered Requirements Engineering. In *Proceedings of the 15th International Requirements Engineering Conference*, (RE'07), pages 185–194, New Delhi, India, October 2007. IEEE Computer Society.

[25] V. Arief, C. Gacek, and T. Lawrie. Software Architectures and Open Source Software–Where can Research Leverage the Most? In J. Feller, B. Fitzgerald, and A. Van der Hoek, editors, *Proceedings of the 1st International Workshop on Open Source Software Engineering at ICSE 2001*, pages 1–3, Toronto, Ontario, Canadá, May 2001. IEEE Computer Society.

[26] R. M. Arlein and V. K. Gurbani. An Extensible Framework for Constructing SIP User Agents. *Bell Labs Technical Journal*, 9(3):87–100, November 2004.

[27] U. Asklund and L. Bendix. Configuration Management for Open Source Software. In J. Feller, B. Fitzgerald, and A. Van der Hoek, editors, *Proceedings of the 1st Workshop on Open Source Software Engineering at ICSE 2001*, Toronto, Ontario, Canadá, May 2001. IEEE Computer Society.

[28] P. M. Bach and J. M. Carroll. FLOSS UX Design: An Analysis of User Experience Design in Firefox and OpenOffice.org. In C. Boldyreff, K. Crowston, B. Lundell, and A. I. Wasserman, editors, *Open Source Ecosystems: Diverse Communities Interacting (OSS'09)*, volume 299 of *IFIP Advances in Information and Communication Technology*, pages 237–250. Springer-Verlag Berlin Heidelberg, Skövde, Sweden, June 2009.

[29] P. M. Bach, R. DeLine, and J. M. Carroll. Designers Wanted: Participation and the User Experience in Open Source Software Development. In *Proceedings of the 27th Conference on Human Factors in Computing Systems*, CHI'09, pages 985–994, Boston, MA, USA, April 2009. ACM.

[30] B. R. Baliga and J. G. Hunt. *Emerging Leadership Vistas*, chapter 8. An Organizational Life Cycle Approach to Leadership, pages 129–149. Lexington Books, Lexington, MA, USA, 1988.

[31] A. Bangor, P. T. Kortum, and J. T. Miller. An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction*, 24(6):574–594, September 2008.

[32] G. Bauer. *User Experience Specifications.* Available at http://www-archive.mozilla.org/projects/ui/communicator/, December 2001. [Accesed March 2014].

[33] B. Behlendorf. *Open Sources: Voices from the Open Source Revolution*, chapter 11. Open Source as a Business Strategy, pages 70–78. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, January 1999.

[34] C. Benson, A. Elman, S. Nickell, and C. Z. Robertson. *GNOME Humane Interface Guidelines 1.0.* The GOME Usability Project, 2002.

[35] C. Benson, M. Müller-Prove, and J. Mzourek. Professional Usability in Open Source Projects: GNOME, OpenOffice.Org, NetBeans. In *Proceedings of the CHI'04 Extended Abstracts on Human Factors in Computing Systems*, CHI EA'04, pages 1083–1084, Vienna, Austria, April 2004. ACM.

[36] N. Bezroukov. A Second Look at the Cathedral and the Bazaar. *First Monday*, 4(12), December 1999.

[37] J. Bitzer, W. Schrettl, and P. J. H. Schröder. Intrinsic Motivation in Open Source Software Development. *Journal of Comparative Economics*, 35(1):160–169, March 2007.

[38] M. Bødker, L. Nielsen, and R. N. Orngreen. Enabling User Centered Design Processes in Open Source Communities. In N. Aykin, editor, *Usability and Internationalization, part. I, HCI and Culture, UI-HCII'07*, volume 4559 of *Lecture Notes in Computer Science*, pages 10–18. Springer-Verlag Berlin Heidelberg, Beijing, China, July 2007.

[39] A. M. Braccini, C. Silvestri, and S. Za. Interactions with Open Source Software: A Pilot Study on End Users' Perception. In A. D'Atri and D. Saccà, editors, *Information Systems: People, Organizations, Institutions, and Technologies*, pages 549–556. Springer Physica-Verlag Berlin Heidelberg, 2010.

[40] J. Brooke. SUS: A Quick and Dirty Usability Scale. In P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, editors, *Usability Evaluation in Industry*, pages 189–194. Taylor & Francis, London, 1996.

[41] F. P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering.* Addison Wesley Longman, Inc., annniversary edition, August 1995.

[42] A. Capiluppi and M. Michlmayr. From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects. In J. eller, B. Fitzgerald, W. Scacchi, and A. Sillitti, editors, *Open Source Development, Adoption and Innovation*, volume 234 of *IFIP The International Federation for Information Processing*, pages 31–44. Springer US, Limerick, Ireland, June 2007.

[43] G. Çetin and M. Göktürk. Usability in Open Source: Community. *Interactions*, 14(6):38–40, November/December 2007.

[44] G. Çetin and M. Göktürk. A Measurement Based Framework for Assessment of Usability-Centricness of Open Source Software Projects. In *Proceedings of the 4th International Conference on Signal Image Technology and Internet Based Systems*, SITIS'08, pages 585–592, Bali, Indonesia, November/December 2008. IEEE Computer Society.

[45] G. Çetin and M. Göktürk. Collaboration in Open Source Domains: A Perspective on Usability. *International Journal of Open Source Software & Processes*, 1(4):167–178, 2009.

[46] G. Çetin and M. Göktürk. Assessing Usability Readiness of Collaborative Projects. *International Journal of Computer Systems Science & Engineering*, 26(4):259–274, July 2011.

[47] G. Çetin, D. Verzulli, and S. Frings. An Analysis of Involvement of HCI Experts in Distributed Software Development: Practical Issues. In D. Schuler, editor, *Online Communities and Social Computing (OCSC'07)*, volume 4564 of *Lecture Notes in Computer Science*, pages 32–40. Springer-Verlag Berlin Heidelberg, Beijing, China, 2007.

[48] C. Cleveland. The Past, Present, and Future of PC Mod Development. *Game Developer Magazine*, 8(2):46–49, February 2001.

[49] R. O. Colbert, D. S. Compton, R. L. Hackbarth, J. D. Herbsleb, L. A. Hoadley, and G. J. Wills. Advanced Services: Changing How We Communicate. *Bell Labs Technical Journal*, 6(1):211–228, September 2001.

[50] L. L. Constantine and L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-centered Design*. ACM Press/Addison-Wesley Publishing Co., New York, USA, 1999.

[51] A. Cooper. *The Inmates are Running the Asylum*. Sams, Indianapolis, Indiana, USA, 1999.

[52] A. Cooper, R. Reimann, and D. Cronin. *About Face 3: The Essentials of Interaction Design*. Wiley Publishing, Inc., Indianapolis, Indiana, USA, 3rd edition, May 2007.

[53] A. Cooper, R. Reimann, and H. Dubberly. *About Face 2.0: The Essentials of Interaction Design*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2003.

[54] K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/Libre Open-Source Software Development: What We Know and What We Do Not Know. *ACM Computing Surveys*, 44(2):1–35 (article 7), February 2012.

[55] B. Curtis, H. Krasner, and N. Iscoe. A Field Study of the Software Design Process for Large Systems. *Communications of the ACM*, 31(11):1268–1287, November 1988.

[56] M. I. de Porto Alegre Muniz and A. de Moraes. Usability Issues in Learning Management Systems (LMS). *Work: A Journal of Prevention, Assessment and Rehabilitation*, 41(1):832–837, February 2012.

[57] T. T. Dinh-Trong and J. M. Bieman. The FreeBSD Project: A Replication Case Study of Open Source Development. *IEEE Transactions on Software Engineering*, 31(6):481–494, June 2005.

[58] S. Egan. *Open Source Messaging Application Development: Building and Extending Gaim*, chapter 2. The Open Source Development Process. Apress, 1st edition, July 2005.

[59] B. Eich. *Mozilla Development Roadmap*. Available at http://www-archive.mozilla.org/roadmap/roadmap-25-Sep-2000.html, September 2000. [Accesed February 2014].

[60] B. Eich and M. Baker. *Mozilla "Super-Review"*. Available at http://www-archive.mozilla.org/hacking/reviewers.html, April 2008. [Accesed March 2014].

[61] S. Eklund, M. Feldman, and M. Trombley. StarOffice Calc v. MS Excel: Improving the Usability of an Open Source Spreadsheet Application. Technical Report InfoSys 271, Quantitative Reseach Methods for Information Management, School of Information Management and Systems, University of California, Berkeley, Berkeley, CA, USA, December 2001.

[62] H. Erdogmus. A Process That Is Not. *IEEE Software*, 26(6):4–7, November/December 2009.

[63] A. Ezeala, H. Kim, and L. A. Moore. Open Source Software Development: Expectations and Experience from a Small Development Project. In *Proceedings of the 46th ACM Southeast Regional Conference*, ACM-SE'08, pages 243–246, Auburn University, Auburn, AL, USA, March 2008. ACM.

[64] M. E. Fagan. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, 38(2-3):258–287, June 1999.

[65] J. Feller. Meeting Challenges and Surviving Success: The 2Nd Workshop on Open Source Software . In *Proceedings of the 24th International Conference on Software Engineering*, ICSE'02, pages 669–670, Orlando, Florida, USA, 2002. ACM.

[66] J. Feller and B. Fitzgerald. *Understanding Open Source Software Development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[67] J. Feller, B. Fitzgerald, and A. van der Hoek. Making Sense of the Bazaar: 1st Workshop on Open Source Software Engineering. *SIGSOFT Software Engineering Notes*, 26(6):51–52, November 2001.

[68] X. Ferré, N. Juristo, and A. M. Moreno. *Deliverable D.5.1. Selection of the Software Process and the Usability Techniques for Consideration*. STATUS Project (code IST-2001-32298) financed by the European Commission from December of 2001 to December of 2004. Available at http://is.ls.fi.upm.es/status/results/STATUS_D5.1_v1.0.pdf, April 2002.

[69] X. Ferré, N. Juristo, and A. M. Moreno. *Deliverable D.5.2. Specification of the Software Process with Integrated Usability Techniques*. STATUS Project (code IST-2001-32298) financed by the European Commission from December of 2001 to December of 2004. Available at http://is.ls.fi.upm.es/status/results/STATUS_D5.2_v1.0.pdf, November 2002.

[70] X. Ferré, N. Juristo, and A. M. Moreno. Framework for Integrating Usability Practices into the Software Process. In F. Bomarius and S. Komi-Sirviö, editors, *Product Focused Software Process Improvement (PROFES'05)*, volume 3547 of *Lecture Notes in Computer Science*, pages 202–215. Springer-Verlag Berlin Heidelberg, Oulu, Finland, June 2005.

[71] X. Ferré, N. Juristo, H. Windl, and L. Constantine. Usability Basics for Software Developers. *IEEE Software*, 18(1):22–29, January 2001.

[72] R. T. Fielding. Shared Leadership in the Apache Project. *Communications of the ACM*, 42(4):42–43, April 1999.

[73] B. Fitzgerald. The Transformation of Open Source Software. *MIS Quarterly*, 30(3):587–598, September 2006.

[74] K. Fogel. *Open Source Development with CVS*. Coriolis Press, May 1999.

[75] N. Frishberg, A. M. Dirks, C. Benson, S. Nickell, and S. Smith. Getting to Know You: Open Source Development Meets Usability. In *Proceedings of the CHI'02 Extended Abstracts on Human Factors in Computing Systems*, CHI EA'02, pages 932–933, Minneapolis, MN, USA, April 2002. ACM.

[76] A. Fuggetta. Open Source Software–An Evaluation. *Journal of Systems and Software*, 66:77–90, April 2003.

[77] J. Galbraith. The Stages of Growth. *Journal of Business Strategy*, 3(1):70–79, 1982.

[78] D. M. German. GNOME: A Case of Open Source Software Development. In *International Workshop on Global Software Development*, (GSD'03), pages 39–43. Portland, Oregon, USA, May 2003.

[79] D. M. German. The GNOME Project: A Case Study of Open Source, Global Software Development. *Software Process: Improvement and Practice*, 8(4):201–215, October/December 2003.

[80] M. W. Godfrey and Q. Tu. Evolution in Open Source Software: A Case Study. In *Proceedings of the International Conference on Software Maintenance*, ICSM'00, pages 131–142, San José, CA, USA, October 2000. IEEE Computer Society.

[81] M. Göktürk and G. Çetin. Out of Box Experience Issues of Free and Open Source Software. In J. Jacko, editor, *Human-Computer Interaction: Interaction Design and Usability, part. I, HCII'07*, volume 4550 of *Lecture Notes in Computer Science*, pages 774–783. Springer-Verlag Berlin Heidelberg, Beijing, China, July 2007.

[82] K. Goodwin. *Getting from Research to Personas: Harnessing the Power of Data.* Available at http://www.cooper.com/journal/2008/05/getting_from_research_to_perso, November 2002. [Accesed September 2014].

[83] T. R. Gruber and D. M. Russell. Design Knowledge and Design Rationale: A Framework for Representation, Capture, and Use. *Technical Report KSL 90-45*, Knowledge Systems Laboratory, Computer Science Department, Stanford University, Stanford, California, USA, August 1991.

[84] S. Gujrati and E. Y. Vasserman. The Usability of Truecrypt, or How I Learned to Stop Whining and Fix an Interface. In *Proceedings of the 3rd Conference on Data and Application Security and Privacy*, CODASPY'13, pages 83–94, San Antonio, Texas, USA, February 2013. ACM.

[85] V. K. Gurbani, A. Garvert, and J. D. Herbsleb. A Case Study of a Corporate Open Source Development Model. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE'06, pages 472–481, Shanghai, China, May 2006. ACM.

[86] A. Hars and S. Ou. Working for Free? – Motivations of Participating in Open Source Projects. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, pages 1–9, Hawaii, USA, January 2001. IEEE Computer Society.

[87] A. Hars and S. Ou. Working for Free? Motivations for Participating in Open-Source Projects. *International Journal of Electronic Commerce*, 6(3):25–39, Spring 2002.

[88] H. R. Hartson and J. C. Castillo. Remote Evaluation for Post-Deployment Usability Improvement. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI'98, pages 22–29, L'Aquila, Italy, May 1998. ACM.

[89] H. Hedberg, N. Iivari, M. Rajanen, and L. Harjumaa. Assuring Quality and Usability in Open Source Software Development. In *Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development*, FLOSS'07, pages 1–5, Minneapolis, MN, USA, May 2007. IEEE Computer Society.

[90] L. G. R. Henderson. Requirements Elicitation in Open-Source Programs. *CrossTalk: The Journal of Defense Software Engineering*, 13(7):28–30, July 2000.

[91] J. D. Herbsleb and R. E. Grinter. Splitting the Organization and Integrating the Code: Conway's Law Revisited. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE'99, pages 85–95, Los Angeles, California, USA, May 1999. ACM.

[92] D. Hix and H. R. Hartson. *Developing User Interfaces: Ensuring Usability Through Product & Process*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[93] L. Ibáñez, W. Schroeder, L. Ng, and J. Cates. The ITK Software Guide and the Insight Software Consortium. *Guide*, New York, NY, USA, August 2003.

[94] IEEE. Std. 1074-2006: IEEE Standard for Developing a Software Project Life Cycle Process. *IEEE Computer Society*, July 2006.

[95] N. Iivari. Usability Innovations in OSS Development – Examining User Innovations in an OSS Usability Discussion Forum. In P. Ågerfalk, C. Boldyreff, J. M. González-Barahona, G. R. Madey, and J. Noll, editors, *Open Source Software: New Horizons (OSS'10)*, volume 319 of *IFIP Advances in Information and Communication Technology*, pages 119–129. Springer-Verlag Berlin Heidelberg, Notre Dame, IN, USA, May/June 2010.

[96] S. Jalali and C. Wohlin. Agile Practices in Global Software Engineering - A Systematic Map. In *Proceedings of the 5th International Conference on Global Software Engineering*, ICGSE'10, pages 45–54, Princenton, New Jersey, USA, August 2010. IEEE Computer Society.

[97] K. Johnson. A Descriptive Process Model for Open-Source Software Development. Master's thesis, Department of Computer Science, University of Calgary, Calgary, Alberta, June 2001.

[98] N. Jørgensen. Putting It All in the Trunk: Incremental Software Development in the FreeBSD Open Source Project. *Information Systems Journal*, 11(4):321–336, October 2001.

[99] P. T. A. Junior and L. V. L. Filgueiras. User Modeling with Personas. In *Proceedings of the 2005 Latin American Conference on Human-Computer Interaction*, CLIHC'05, pages 277–282, New York, NY, USA, October 2005. ACM.

[100] A. J. Kim. *Community Building on the Web: Secret Strategies for Successful Online Communities*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, April 2000.

[101] D. C. Lawrence. InterNetNews Server: Inside An Open-Source Project. *IEEE Internet Computing*, 2(5):49–52, September 1998.

[102] T. Lawrie and C. Gacek. Issues of Dependability in Open Source Software Development. *SIGSOFT Software Engineering Notes*, 27(3):34–37, May 2002.

[103] M. Learmonth. *Giving It All Away*. Available at http://www.metroactive.com/papers/metro/05.08.97/, February 1997. [Accesed March 2014].

[104] D. Lee and B. Kim. Motivations for Open Source Project Participation and Decisions of Software Developers. *Computational Economics*, 41(1):31–57, 2013.

[105] M. M. Lehman. Programs, Life Cycles, and Laws of Software Evolution. *IEEE Software*, 68(9):1060–1076, September 1980.

[106] F. Lelli and M. Jazayeri. Community Support for Software Development in Small Groups: The Initial Steps. In *Proceedings of the 2nd International Workshop on Social Software Engineering and Applications*, SoSEA'09, pages 15–22, Amsterdam, The Netherlands, August 2009. ACM.

[107] L. Lessig. *The Future of Ideas: The Fate of the Commons in a Connected World*. Random House, New York, NY, USA, 1st edition, 2001.

[108] M. Levesque. Fundamental Issues with Open Source Software Development. *First Monday*, 9(4), April 2004.

[109] M. Levesque and J. Montojo. Opening Up Open Source. *CrossTalk: The Journal of Defense Software Engineering*, 18(1):26–28, January 2005.

[110] J. R. Lewis and J. Sauro. The Factor Structure of the System Usability Scale. In *Proceedings of the 1st International Conference on Human Centered Design: Held As Part of HCI International 2009*, HCD'09, pages 94–103, Berlin, Heidelberg, 2009. Springer-Verlag.

[111] A. Loconsole, D. Rodriguez, J. Börstler, and R. Harrison. Report on Metrics 2001: The Science & Practice of Software Metrics Conference. *SIGSOFT Software Engineering Notes*, 26(6):52–57, November 2001.

[112] M. Loukides and A. Oram. *Programming with GNU Software*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, January 1997.

[113] C. Lyn Paul. Usability in Open Source Software. *The Usability Proffesional Association*, 10(1), February.

[114] C. Lyn Paul. A Survey of Usability Practices in Free/Libre/Open Source Software. In C. Boldyreff, K. Crowston, B. Lundell, and A. I. Wasserman, editors, *Open Source Ecosystems: Diverse Communities Interacting (OSS'09)*, volume 299 of *IFIP Advances in Information and Communication Technology*, pages 264–273. Springer-Verlag Berlin Heidelberg, Skövde, Sweden, June 2009.

[115] M. Maclachlan. *TechWeb: Panelists Describe Open Source Dictatorships*. Available at http://www.linuxtoday.com/developer/1999081201205PS, August 1999. [Accesed March 2014].

[116] K. Martin and B. Hoffman. An Open Source Approach to Developing Software in a Small Organization. *IEEE Software*, 24(1):46–53, January 2007.

[117] C. Martín Montero. Integración de Técnicas de Usabilidad en un Proceso de Desarrollo de Open Source Software. Thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Madrid, España, June 2014.

[118] B. Massey. Where Do Open Source Requirements Come from (and What Shold We Do About It)? In J. Feller, B. Fitzgerald, F. Hecker, S. A. Hissam, K. R. Lakhani, and A. Van der Hoek, editors, *Proceedings of the 2nd International Workshop on Open Source Software Engineering at ICSE 2002*, Orlando, Florida, USA, May 2002. IEEE Computer Society.

[119] B. Massey. Why OSS Folks Think SE Folks Are Clue-Impaired. In J. Feller, B. Fitzgerald, S. A. Hissam, and K. R. Lakhani, editors, *Proceedings of the 3rd Workshop on Open Source Software at ICSE 2003*, pages 91–97, Portland, OR, USA, May 2003. IEEE Computer Society.

[120] D. J. Mayhew. *The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[121] I. R. McChesney. Effective Coordination in the Software Process - Historical Perspectives and Future Directions. *Software Quality Journal*, 6(3):235–246, 1997.

[122] S. McConnell. Open-Source Methodology: Ready for Prime Time? *IEEE Software*, 16(4):6–8, July/August 1999.

[123] M. Michlmayr and B. M. Hill. Quality and the Reliance on Individuals in Free Software Projects. In J. Feller, B. Fitzgerald, H. Scott, and L. Karim, editors, *Proceedings of the 3rd Workshop on Open Source Software Engineering at ICSE 2003*, pages 105–109, Portland, OR, USA, May 2003. IEEE Computer Society.

[124] H. Mintzberg. Power and Organization Life Cycles. *The Academy of Management Review*, 9(2):207–224, April 1984.

[125] A. Mockus, R. T. Fielding, and J. Herbsleb. A Case Study of Open Source Software Development: The Apache Server. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE'00, pages 263–272, Limerick, Ireland, June 2000. ACM.

[126] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. *Transactions on Software Engineering and Methodology*, 11(3):309–346, July 2002.

[127] M. Monga. From Bazaar to Kibbutz: How Freedom Deals with Coherence in the Debian Project. In J. Feller, B. Fitzgerald, S. A. Hissam, and K. R. Lakhani, editors, *Proceedings of the 4th Workshop on Open Source Software Engineering at ICSE 2004*, pages 71–75, Edinburgh, Scotland, UK, May 2004. IEEE Computer Society.

[128] G. Moody. *Rebel Code: Linux and the Open Source Revolution.* Basic Books, Inc., 1st edition, July 2002.

[129] J. Muhlig and C. Lyn Paul. OpenUsability.org: Usability and Open Source Software. *Interfaces*, 66(Spring):18–21, January 2006.

[130] F. Mulazzani, B. Rossi, B. Russo, and M. Steff. Building Knowledge in Open Source Software Research in Six Years of Conferences. In S. A. Hissam, B. Russo, M. G. Mendonça Neto, and F. Kon, editors, *Open Source Systems: Grounding Research*, volume 365 of *IFIP Advances in Information and Communication Technology*, pages 123–141. Springer Berlin Heidelberg, Salvador, Brazil, October 2011.

[131] M. Müller-Prove. Community Experience at OpenOffice.Org. *Interactions*, 14(6):47–48, November/December 2007.

[132] M. Müller-Prove. User Experience for OpenOffice.org. *Interfaces*, 71(Summer):8–9, June 2007.

[133] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. Evolution Patterns of Open-Source Software Systems and Communities. In *Proceedings of the International Workshop on Principles of Software Evolution*, IWPSE'02, pages 76–85, Orlando, Florida, USA, 2002. ACM.

[134] B. A. Nardi, D. J. Schiano, M. Gumbrecht, and L. Swartz. Why We Blog. *Communications of the ACM*, 47(12):41–46, December 2004.

[135] D. M. Nichols, D. McKay, and M. B. Twidale. Participatory Usability: Supporting Proactive Users. In *Proceedings of the 4th International Conference NZ Chapter of the ACM's Special Interest Group on Computer-Human Interaction*, SIGCHI-NZ'03, pages 63–68, Dunedin, New Zealand, July 2003. ACM.

[136] D. M. Nichols and M. B. Twidale. The Usability of Open Source Software. *First Monday*, 8(1), January 2003.

[137] D. M. Nichols and M. B. Twidale. Usability Processes in Open Source Projects. *Software Process: Improvement and Practice*, 11(2):149–162, January/February 2006.

[138] J. Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[139] L. Nielsen and M. Bødker. To Do or Not to Do: Usability in Open Source Development. *Interfaces*, 71(Summer):10–11, June 2007.

[140] J. Noll and W. Scacchi. Supporting Software Development in Virtual Enterprises. *Journal of Digital Information*, 1(4):1–11, February 1999.

[141] S. O'Mahony. Guarding the Commons: How Community Managed Software Projects Protect Their Work. *Research Policy*, 32(7):1179–1198, July 2003.

[142] S. Osiński and D. Weiss. Introducing Usability Practices to OSS: The Insiders' Experience. In J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti, editors, *Open Source Development, Adoption and Innovation*, volume 234 of *IFIP The International Federation for Information Processing*, pages 313–318. Springer US, Limerick, Ireland, June 2007.

[143] E. Ostrom, R. Gardner, and J. Walker. *Rules, Games, and Common-Pool Resources*. The University of Michigan Press, Michigan, USA, May 1994.

[144] D. Pagano and W. Maalej. How Do Open Source Communities Blog? *Empirical Software Engineering*, 18(6):1090–1124, May 2013.

[145] C. Payne. On the Security of Open Source Software. *Information Systems Journal*, 12(1):61–78, January 2002.

[146] S. Pemberton. Scratching Someone Else's Itch: (Why Open Source Can't Do Usability). *Magazine Interactions*, 11(1):72, January/February 2004.

[147] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. Systematic Mapping Studies in Software Engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, University of Bari, Italy, June 2008. British Computer Society.

[148] V. Podtdar and E. Chang. Open Source and Closed Source Software Development Methodologies. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE'04, pages 105–109, Edinburgh, UK, May 2004. IEEE Computer Society.

[149] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey. *Human-Computer Interaction*. Addison Wesley, 1st edition, April 1994.

[150] J. Pruitt and T. Adlin. *The Persona Lifecycle: Keeping People in Mind Throughout Product Design.* Interactive Technologies Series. Morgan Kaufmann, San Francisco, CA, USA, 2006.

[151] J. Pruitt and J. Grudin. Personas: Practice and Theory. In *Proceedings of the 2003 Conference on Designing for User Experiences*, DUX'03, pages 1–15, New York, NY, USA, June 2003. ACM.

[152] M. Rajanen, N. Iivari, and E. Keskitalo. Introducing Usability Activities into Open Source Software Development Projects – A Participative Approach. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*, NordiCHI'12, pages 683–692, Copenhagen, Denmark, October 2012. ACM.

[153] E. S. Raymond. The Cathedral and the Bazaar. *First Monday*, 3(2), March 1998.

[154] E. S. Raymond. *Open Sources: Voices from the Open Source Revolution*, chapter 15. The Revenge of the Hackers, pages 96–101. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, January 1999.

[155] E. S. Raymond. *The Cathedral & the Bazaar: Musings on Linux and Open Source by and Accidental Revolutionary*, chapter 2. The Cathedral & the Bazaar. O'Reilly Media, Sebastapol, CA, USA, February 2001.

[156] E. S. Raymond. *The Art Of Unix Programming.* Addison-Wesley Professional, 1st edition, October 2003.

[157] E. S. Raymond. *The Luxury og Ingnorance: An Open-source Horror Story.* Available at http://www.catb.org/ esr/writings/cups-horror.html, April 2006. [Accesed February 2014].

[158] A. Raza and L. F. Capretz. Do Open Source Software Developers Listen to Their Users? *First Monday*, 17(3), March 2012.

[159] A. Raza, L. F. Capretz, and F. Ahmed. Improvement of Open Source Software Usability: An Empirical Evaluation from Developers' Perspective. *Journal Advances in Software Engineering - Special Issue on New Generation of Software Metrics*, volume 2010:1– 12 (article ID 517532), January 2010.

[160] A. Raza, L. F. Capretz, and F. Ahmed. An Empirical Study of Open Source Software Usability: The Industrial Perspective. *International Journal of Open Source Software and Processes*, 3(1):1–16, January/March 2011.

[161] A. Raza, L. F. Capretz, and F. Ahmed. An Open Source Usability Maturity Model (OS-UMM). *Journal Computers in Human Behavior*, 28(4):1109–1121, July 2012.

[162] A. Raza, L. F. Capretz, and F. Ahmed. Usability Bugs in Open-Source Software and Online Forums. *IET Software*, 6(3):226–230, June 2012.

[163] A. Raza, L. F. Capretz, and F. Ahmed. Users' Perception of Open Source Usability: An Empirical Study. *Journal Engineering with Computers*, 28(2):109–121, April 2012.

[164] C. R. Reis and R. P. de Mattos Fortes. An Overview of the Software Engineering Process and Tools in the Mozilla Project. In *Proceedings of the Open Source Software Development Workshop*, pages 155–175, Newcastle upon Tyne, UK., February 2002.

[165] E. Reitmayr, B. Balazs, and J. Mühlig. Integrating Usability with Open Source Software Development: Case Studies from the Initiative Open Usability. In *Proceedings of the Workshop on Governmental Educational, Usability and Legal Issues towards Open*

Source Software Adoption in an enlarged Europe, tOSSad'06, pages 65–72, Como, Italy, June 2006.

[166] D. Robey. *Designing Organizations: A Macro Perspective*. Irwin Professional Publishing, December 1982.

[167] P. Rooney. IBM Builds Dedicated Sales Channel for Red Hat, Novell Linux, publicado en December 2005, available at http://www.crn.com/news/applications-os/175002626/ibm-builds-dedicated-sales-channel-for-red-hat-novell-linux.htm.

[168] P. Runeson and M. Höst. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Journal Empirical Software Engineering*, 14(2):131–164, April 2009.

[169] W. Scacchi. Understanding the Requirements for Developing Open Source Software Systems. *IEE Software*, 149(1):24–39, February 2002.

[170] W. Scacchi. Free and Open Source Development Practices in the Game Community. *IEEE Software*, 21(1):59–66, January 2004.

[171] W. Scacchi. Free/Open Source Software Development: Recent Research Results and Emerging Opportunities. In *Proceedings of the 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers*, ESEC/FSE'07, pages 459–468, Dubrovnik, Croatia, September 2007. ACM.

[172] W. Scacchi. Understanding Requirements for Open Source Software. In K. Lyytinen, P. Loucopoulos, J. Mylopoulos, and B. Robinson, editors, *Design Requirements Engineering: A Ten-Year Perspective*, volume 14 of *Lecture Notes in Business Information Processing*, pages 467–494. Springer-Verlag Berlin Heidelberg, Cleveland, OH, USA, June 2009.

[173] W. Scacchi, C. Jensen, J. Noll, and M. Elliott. Multi-Modal Modeling of Open Source Software Requirements Processes. In *Proceedings of the First International Conference on Open Source Systems*, number 3, pages 1–8, Genova, Italy, July 2005.

[174] D. C. Schmidt and A. Porter. Leveraging Open-Source Communities to Improve the Quality & Performance of Open-Source Software. In J. Feller, B. Fitzgerald, and A. Van der Hoek, editors, *Proceedings of the 1st International Workshop on Open Source Software Engineering at ICSE 2001*, pages 1–5, Toronto, Ontario, Canadá, May 2001. IEEE Computer Society.

[175] W. Schroeder, M. Ken, and L. Bill. *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Kitware, New York, NY, USA, 4th edition, December 2006.

[176] C. M. Schweik and A. Semenov. The Institutional Design of Open Source Programming: Implications for Addressing Complex Public Policy and Management Problems. *First Monday*, 8(1), January 2003.

[177] A. Senyard and M. Michlmayr. How to Have a Successful Free Software Project. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, APSEC'04, pages 84–91, Busan, Korea, November/December 2004. IEEE Computer Society.

[178] W. Sewell. *Weaving a Program: Literate Programming in Web*. Van Nostrand Reinhold Computer, August 1989.

[179] S. Sharma, V. Sugumaran, and B. Rajagopalan. A Framework for Creating Hybrid-Open Source Software communities. *Information Systems Journal*, 12(1):7–25, January 2002.

[180] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, November 1997.

[181] G. L. Simmons and T. Dillon. Open Source Development and Agile Methods. In *Proceedings of the 7th IASTED International Conference on Software Engineering and Applications*, IASTED'07, pages 523–527, Marina del Rey, CA, USA, November 2003.

[182] K. G. Smith, T. R. Mitchell, and C. E. Summer. Top Level Management Priorities in Different Stages of the Organizational Life Cycle. *Academy of Management Journal*, 28(4):799–820, December 1985.

[183] S. Smith, D. Engen, A. Mankoski, N. Frishberg, N. Pedersen, and C. Benson. GNOME Usability Study Report. *Technical Report*, Sun Microsystems, July, 2001.

[184] I. Sommerville. *Software Engineering*. Addison Wesley, Boston, MA, USA, 9th edition, March 2010.

[185] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris. Code Quality Analysis in Open-Source Software Development. *Information Systems Journal*, 12(1):43–60, January 2002.

[186] I. Steinmacher, M. A. Gerosa, and D. F. Redmiles. Attracting, Onboarding, and Retaining Newcomer Developers in Open Source Software Projects. In *Workshop of Global Software Development in a CSCW Perspective at CSCW 2014*, 17th ACM Conference on Computer Supported Cooperative Work and Social Computing, pages 1–4 (Article 8), Baltimore, Maryland, USA, February 2014.

[187] SWEBOK. Guide to the Software Engineering Body of Knowledge (version 2004). *A project of the IEEE Computer Society Professional Practices Committee*, 2004.

[188] M. Terry, M. Kay, and B. Lafreniere. Perceptions and Practices of Usability in the Free/Open Source Software (FOSS) Community. In *Proceedings of the 28th Conference on Human Factors in Computing Systems*, CHI'10, pages 999–1008, Atlanta, Georgia, USA, April 2010. ACM.

[189] M. P. Thomas. *Why Free Software has Poor Usability, and How to Improve It*. Available at http://www.oreillynet.com/onlamp/blog/2008/08/matthew_paul_thomas_why_free_s_1.html_paul_thomas_why_free_s_1.html, 2008. [Accesed February 2014].

[190] Y. Tian. *Developing an Open Source Software Development Process Model Using Grounded Theory*. PhD thesis, Faculty of the Graduate College at the University of Nebraska - Lincoln, Nebraska, USA, August 2006.

[191] L. Torvalds. *Open Sources: Voices from the Open Source Revolution*, chapter 8. The Linux Edge, pages 51–55. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, January 1999.

[192] P. Trudelle. Shall We Dance? Ten Lessons Learned from Netscape's Flirtation with Open Source UI Development. In *Proceedings of the Open Source Meets Usability Workshop at CHI 2002*, pages 1–3, Minneapolis, MN, USA, April 2002. ACM.

[193] D. P. Truex, R. Baskerville, and H. Klein. Growing Systems in Emergent Organizations. *Communications of the ACM*, 42(8):117–123, August 1999.

[194] T. S. Tullis and J. N. Stetson. A Comparison of Questionnaires for Assessing Website Usability. In *Proceedings of the Usability Professionals Association*, UPA'04, pages 1–12, Minneapolis, MN, USA, 2004.

[195] M. B. Twidale and D. M. Nichols. Exploring Usability Discussions in Open Source Development. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, HICSS'05, pages 198–207, Hawaii, USA, January 2005. IEEE Computer Society.

[196] A. v. E. Van de Vliert and O. Janssen. *Groups at Work: Theory and Research*, chapter 9. Description, Explanation and Prescription of Intragroup Conflict Behaviour. Applied social research. L. Erlbaum Associates, Publishers, 2001.

[197] N. Viorres, P. Xenofon, M. Stavrakis, E. Vlachogiannis, P. Koutsabasis, and J. Darzentas. Major HCI Challenges for Open Source Software Adoption and Development. In D. Schuler, editor, *Online Communities and Social Computing (OCSC'07)*, volume 4564 of *Lecture Notes in Computer Science*, pages 455–464. Springer-Verlag Berlin Heidelberg, Beijing, China, July 2007.

[198] P. Vixie. *Open Sources: Voices from the Open Source Revolution*, chapter 7. Software Engineering, pages 47–50. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, January 1999.

[199] A. Ward. *The Leadership Lifecycle: Matching Leaders to Evolving Organizations*. Palgrave Macmillan, Basingstoke, November 2003.

[200] S. Williams. *Upside: Learning the ways of Mozilla*. Upside Today. Available at http://www.linuxtoday.com/sdeveloper/2000101100321PSDT. [Accesed February 2014].

[201] C. E. Wilson and K. P. Coyne. The Whiteboard: Tracking Usability Issues: To Bug or Not to Bug? *Magazine Interactions*, 8(3):15–19, May/June 2001.

[202] D. E. Wynn Jr. Organizational Structure of Open Source Projects: A Life Cycle Approach. In *Proceedings of the 7th Annual Conference of the Southern Association for Information Systems*, SAIS'04, pages 285–290, Savannah, Georgia, USA, February 2004.

[203] Y. Yamauchi, M. Yokozawa, T. Shinohara, and T. Ishida. Collaboration with Lean Media: How Open-Source Software Succeeds. In *Proceedings of the Conference on Computer Supported Cooperative Work*, CSCW'00, pages 329–338, Philadelphia, Pennsylvania, USA, December 2000. ACM.

[204] T. Zefferer and V. Krnjic. Towards User-Friendly e-Government Solutions: Usability Evaluation of Austrian Smart-Card Integration Techniques. In A. Kö, C. Leitner, H. Leitold, and A. Prosser, editors, *Advancing Democracy, Government and Governance (EGOVIS'12/EDEM'12)*, volume 7452 of *Lecture Notes in Computer Science*, pages 88–102. Springer-Verlag Berlin Heidelberg, Vienna, Austria, September 2012.

[205] L. Zhao and F. P. Deek. Improving Open Source Software Usability. In *Proceedings of the 11th Americas Conference on Information Systems*, AMCIS'05, pages 923–928 (paper 430), Omaha, Nebraska, USA, August 2005. Association for Information Systems Electronic Library (AISeL).

[206] L. Zhao and F. P. Deek. Exploratory Inspection – A Learning Model for Improving Open Source Software Usability. In *Proceedings of the CHI'06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA'06, pages 1589–1594, Montreal, Québec, Canada, April 2006. ACM.

[207] L. Zhao, F. P. Deek, and J. A. McHugh. Strategies for Improving Open Source Software Usability: An Exploratory Learning Framework and a Web-Based Inspection Tool. *International Journal of Open Source Software and Processes*, 1(4):49–64, January/March 2009.

[208] L. Zhao, F. P. Deek, and J. A. McHugh. Exploratory Inspection - A User-Based Learning Method for Improving Open Source Software Usability. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(8):653–675, December 2010.

[209] L. Zhao and S. Elbaum. A Survey on Quality Related Activities in Open Source. *SIGSOFT Software Engineering Notes*, 25(3):54–57, May 2000.

[210] M. Zhou and A. Mockus. What Make Long Term Contributors: Willingness and Opportunity in OSS Community. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE'12, pages 518–528, Zurich, Switzerland, 2012. IEEE Press.

ANEXOS

ANEXO A

ESTUDIOS PRIMARIOS SOBRE EL PROCESO DE DESARROLLO OSS

La Tabla A.1 muestra todos los estudios primarios encontrados en el SMS realizado para identificar las actividades del proceso de desarrollo seguido por la comunidad OSS.

Tabla A.1: Estudios Primarios del Proceso de Desarrollo OSS.

ID	Título de la Publicación	Autores	Año	Tipo de Public.	Ref.
SP1	The FreeBSD Project: A Replication Case Study of Open Source Development	T. T. Dinh-Trong, J. M. Bieman	2005	Revista	[57]
SP2	The Open Source Development Process - (Chapter 2)	S. Egan	2005	Capítulo de Libro	[58]
SP3	A Process That Is Not	H. Erdogmus	2009	Revista	[62]
SP4	Open Source Software Development: Expectations and Experience from a Small Development Project	A. Ezeala, H. Kim, L. A. Moore	2008	Conferencia	[63]
SP5	The Transformation of Open Source Software	B. Fitzgerald	2006	Revista	[73]
SP6	A Case Study of a Corporate Open Source Development Model	V. K. Gurbani, A. Garvert, J. D. Herbsleb	2006	Conferencia	[85]
SP7	A Descriptive Process Model for Open-Source Software Development	K. Johnson	2001	Tesis Máster	[97]
SP8	Putting It All in the Trunk: Incremental Software Development in the FreeBSD Open Source Project	N. Jørgensen	2001	Revista	[98]
SP9	Community Support for Software Development in Small Groups: The Initial Steps	F. Lelli, M. Jazayeri	2009	Seminario	[106]
SP10	An Open Source Approach to Developing Software in a Small Organization	K. Martin, B. Hoffman	2007	Revista	[116]
SP11	A Case Study of Open Source Software Development: The Apache Server	A. Mockus, R. T. Fielding, J. D. Herbsleb	2000	Conferencia	[125]

Tabla A.1: Estudios Primarios del Proceso de Desarrollo OSS (continuación).

ID	Título de la Publicación	Autores	Año	Tipo de Public.	Ref.
SP12	Two Case Studies of Open Source Software Development: Apache and Mozilla	A. Mockus, R. T. Fielding, J. D. Herbsleb	2002	Revista	[126]
SP13	From Bazaar to Kibbutz: How Freedom Deals with Coherence in the Debian Project	M. Monga	2004	Seminario	[127]
SP14	The Cathedral and the Bazaar - (Chapter 2)	E. S. Raymond	2001	Capítulo de Libro	[155]
SP15	An Overview of the Software Engineering Process and Tools in Mozilla Project	C. R. Reis, R. P. de Mattos Fortes	2002	Seminario	[164]
SP16	Free and Open Source Development Practices in the Game Community	W. Scacchi	2004	Revista	[170]
SP17	The Institutional Design of Open Source Programming: Implications for Addressing Complex Public Policy and Management Problems	C. M. Schweik, A. Semenov	2003	Revista	[176]
SP18	How to Have a Successful Free Software Project	A. Senyard, M. Michlmayr	2004	Conferencia	[177]
SP19	Open Source Development and Agile Methods	G. L. Simmons, T. Dillon	2003	Conferencia	[181]
SP20	Software Engineering - (Chapter 6)	P. Vixie	1999	Capítulo de Libro	[198]
SP21	Organizational Structure of Open Source Projects: A Life Cycle Approach	D. E. Wynn, Jr.	2004	Conferencia	[202]
SP22	Collaboration with Lean Media: How Open-Source Software Succeeds	Y. Yamauchi, M. Yokozawa, T. Shinohara, T. Ishida	2000	Conferencia	[203]

ANEXO B

ESTUDIOS PRIMARIOS SOBRE LA USABILIDAD EN EL PROCESO DE DESARROLLO OSS

La Tabla B.1 muestra todos los estudios primarios encontrados en el SMS realizado para identificar el estado actual de la usabilidad en los desarrollos OSS.

Tabla B.1: Estudios Primarios sobre el Estado de la Usabilidad en OSS.

ID	Título de la Publicación	Autores	Año	Tipo de Public.	Ref.
SU1	Usability in Open Source Software Development: Opinions and Practice	M. S. Andreasen, H. V. Nielsen, S. O. Schrøder, J. Stage	2006	Revista	[23]
SU2	FLOSS UX Design: An Analysis of User Experience Design in Firefox and OpenOffice.org	P. M. Bach, J. M. Carroll	2009	Conferencia	[28]
SU3	Professional Usability in Open Source Projects: GNOME, OpenOffice.org, NetBeans	C. Benson, M. Müller-Prove, J. Mzourek	2004	Conferencia	[35]
SU4	Enabling User Centered Design Processes in Open Source Communities	M. Bødker, L. Nielsen, R. N. Orngreen	2007	Conferencia	[38]
SU5	Interactions with Open Source Software: A Pilot Study on End Users' Perception	A. M. Braccini, C. Silvestri, S. Za	2010	Conferencia	[39]
SU6	An Analysis of Involvement of HCI Experts in Distributed Software Development: Practical Issues	G. Çetin, D. Verzulli, S. Frings	2007	Conferencia	[47]
SU7	Usability in Open Source Community	G. Çetin, M. Göktürk	2007	Revista	[43]
SU8	A Measurement Based Framework for Assessment of Usability-Centricness of Open Source Software Projects	G. Çetin, M. Göktürk	2008	Conferencia	[44]

Tabla B.1: Estudios Primarios sobre el Estado de la Usabilidad en OSS (continuación).

ID	Título de la Publicación	Autores	Año	Tipo de Public.	Ref.
SU9	Collaboration in Open Source Domains: A Perspective on Usability	G. Çetin, M. Göktürk	2009	Revista	[45]
SU10	Assessing Usability Readiness of Collaborative Projects	G. Çetin, M. Göktürk	2011	Revista	[46]
SU11	Usability Issues in Learning Management Systems (LMS)	M. I. de Porto Alegre Muniz, A. de Moraes	2012	Revista	[56]
SU12	StarOffice Cal v. MS Excel: Improving the Usability of an Open Source Spreadsheet Application	S. Eklund, M. Feldman, M. Trombley	2001	Reporte Técnico	[61]
SU13	Getting to Know You: Open Source Development Meets Usability	N. Frishberg, A. M. Dirks, C. Benson, S. Nickell, S. Smith	2002	Conferencia	[75]
SU14	Out of Box Experience Issues of Free and Open Source Software	M. Göktürk, G. Çetin	2007	Conferencia	[81]
SU15	The Usability of Truecrypt, or How I Learned to Stop Whining and Fix an Interface	S. Gujrati, E. Y. Vasserman	2013	Conferencia	[84]
SU16	Assuring Quality and Usability in Open Soruce Software Development	H. Hedberg, N. Iivari, M. Rajanen, L. Harjumaa	2007	Seminario	[89]
SU17	Fundamental Issues with Open Source Software Development	M. Levesque	2004	Revista	[108]
SU18	Opening Up Open Source	M. Levesque, J. Montojo	2005	Revista	[109]
SU19	Usability Innovations in OSS Development −Examining User Innovations in an OSS Usability Discussion Forum	N. Iivari	2010	Conferencia	[95]
SU20	User Experience for OpenOffice.org	M. Müller-Prove	2007	Revista	[132]
SU21	Participatory Usability: Supporting Proactive Users	D. M. Nichols, D. McKay, M. B. Twidale	2003	Conferencia	[135]
SU22	The Usability of Open Source Software	D. M. Nichols, M. B. Twidale	2003	Revista	[136]
SU23	Usability Processes in Open Source Projects	D. M. Nichols, M. B. Twidale	2006	Revista	[137]
SU24	To Do or Not to Do: Usability in Open Source Development	L. Nielsen, M. Bødker	2007	Revista	[139]
SU25	Introducing Usability Practices to OSS: The Insiders' Experience	S. Osiński, D. Weiss	2007	Conferencia	[142]
SU26	How Do Open Source Communities Blog?	D. Pagano, W. Maalej	2012	Revista	[144]

Tabla B.1: Estudios Primarios sobre el Estado de la Usabilidad en OSS (continuación).

ID	Título de la Publicación	Autores	Año	Tipo de Public.	Ref.
SU27	A Survey of Usability Practices in Free/Libre/Open Source Software	C. Lyn Paul	2009	Conferencia	[114]
SU28	Scratching Someone Else's Itch: (Why Open Source Can't Do Usability)	S. Pemberton	2004	Revista	[146]
SU29	Introducing Usability Activities into Open Source Software Development Projects - A Participative Approach	M. Rajanen, N. Iivari, E. Keskitalo	2012	Conferencia	[152]
SU30	Improvement of Open Source Software Usability: An Empirical Evaluation from Developers' Perspective	A. Raza, L. F. Capretz, F. Ahmed	2010	Revista	[159]
SU31	An Empirical Study of Open Source Software Usability: The Industrial Perspective	A. Raza, L. F. Capretz, F. Ahmed	2011	Revista	[160]
SU32	An Open Source Usability Maturity Model (OS-UMM)	A. Raza, L. F. Capretz, F. Ahmed	2012	Revista	[161]
SU33	Users' Perception of Open Source Usability: An Empirical Study	A. Raza, L. F. Capretz, F. Ahmed	2012	Revista	[163]
SU34	Do Open Source Software Developers Listen to Their Users?	A. Raza, L. F. Capretz	2012	Revista	[158]
SU35	Usability Bugs in Open-Source Software and Online Forums	A. Raza, L. F. Capretz, F. Ahmed	2012	Revista	[162]
SU36	Integrating Usability with Open Source Software Development: Case Studies from the Initiative Open Usability	E. Reitmayr, B. Balazs, J. Mühlig	2006	Seminario	[165]
SU37	Perceptions and Practices of Usability in the Free/Open Source Software (FOSS) Community	M. Terry, M. Kay, B. Lafreniere	2010	Conferencia	[188]
SU38	Shall We Dance? Ten Lessons Learned from Netscape's Flirtation with Open Source UI Development	P. Trudelle	2002	Seminario	[192]
SU39	Exploring Usability Discussions in Open Source Development	M. B. Twidale, D. M. Nichols	2005	Conferencia	[195]
SU40	Major HCI Challenges for Open Source Software Adoption and Development	N. Viorres, P. Xenofon, M. Stavrakis, E. Vlachogiannis, P. Koutsabasis, J. Darzentas	2007	Conferencia	[197]
SU41	The Whiteboard: Tracking Usability Issues: To Bug or Not to Bug?	C. E. Wilson, K. P. Coyne	2001	Revista	[201]

Tabla B.1: Estudios Primarios sobre el Estado de la Usabilidad en OSS (continuación).

ID	Título de la Publicación	Autores	Año	Tipo de Public.	Ref.
SU42	Towards User-Friendly e-Government Solutions: Usability Evaluation of Austrian Smart-Card Integration Techniques	T. Zefferer, V. Krnjic	2012	Conferencia	[204]
SU43	Improving Open Source Software Usability	L. Zhao, F. P. Deek	2005	Conferencia	[205]
SU44	Exploratory Inspection - A Learning Model for Improving Open Source Software Usability	L. Zhao, F. P. Deek	2006	Conferencia	[206]
SU45	Strategies for Improving Open Source Software Usability: An Exploratory Learning Framework and a Web-Based Inspection Tool	L. Zhao, F. P. Deek, J. A. McHugh	2009	Revista	[207]
SU46	Exploratory Inspection - A User-Based Learning Method for Improving Open Source Software Usability	L. Zhao, F. P. Deek, J. A. McHugh	2010	Revista	[208]

ANEXO C

DESCRIPCIÓN DE LOS MODELOS DEL PROCESO DE DESARROLLO OSS

El objetivo del presente anexo es describir cada uno de los 25 modelos de proceso de desarrollo OSS encontrados en la literatura. Recordemos que el número total de estudios primarios encontrados es de 22. Tres de estos 22 estudios primarios describen dos modelos de proceso cada uno. Por tal razón, el número de modelos de proceso de desarrollo OSS a estudiar es de 25. En las secciones siguientes, describiremos cada uno de estos modelos de proceso.

C.1. Proceso de Desarrollo de FreeBSD

El proceso de desarrollo del proyecto FreeBSD [57] está bien definido y documentado. FreeBSD es similar a Apache y Mozilla en complejidad técnica, tamaño de la comunidad de usuarios, y éxito continuado en su uso y actualización. La larga historia del proyecto FreeBSD permite examinar las actividades del proyecto en un periodo de 9 años, a diferencia del periodo de 3 años del proyecto Apache estudiado por Mockus et al. [126].

El proyecto FreeBSD mantiene dos versiones del código fuente base. La versión de desarrollo consiste de los proyectos que se están realizando y que necesitan ser probados y no están aún estables. La versión estable está madura y bien probada; las versiones liberadas están formadas de la versión estable. Las actividades del proceso de desarrollo de FreeBSD se describen a continuación.

1. **Determinar Roles y Responsabilidades**
   Los participantes del proyecto que contribuyen con el código fuente base desempeñan uno de los siguientes tres roles principales: miembro del núcleo del equipo, responsable del control de la configuración y contribuyente. En un proyecto de OSS que no está soportado comercialmente, cada desarrollador (incluyendo los miembros del núcleo del equipo) y contribuyente es un voluntario y probablemente tiene un trabajo remunerado. De este modo, muchos voluntarios contribuyen al proyecto FreeBSD a tiempo parcial, tal vez durante las noches y los fines de semana. El núcleo del equipo asigna los privilegios a los otros desarrolladores y resuelve los conflictos que puedan surgir entre ellos. Los miembros del núcleo del equipo también son desarrolladores que contribuyen al código del proyecto. Usualmente un miembro del núcleo del equipo puede también tener a su cargo la gestión de algunas otras áreas específicas tales como la documentación, la coordinación de las versiones, y el repositorio del código fuente.

Los responsables del control de la configuración son desarrolladores que tienen la autoridad de registrar los cambios del proyecto en el repositorio Concurrent Version System (CVS). De acuerdo a las normas del proyecto, un responsable del control de la configuración debe estar activo dentro de los últimos 18 meses. De no ser así, el núcleo del evaluador puede quitarle los privilegios al responsable del control de la configuración. Los contribuyentes son personas que desean contribuir al proyecto, pero no tienen los privilegios del responsable del control de la configuración. Los contribuyentes pueden probar el código, reportar problemas y también sugerir soluciones.

2. **Identificar el Trabajo a Realizar (Desarrollo de Nuevas Funcionalidades, Corregir Errores)**

   Hay dos tareas principales que necesitan ser realizadas en cualquier proyecto: el desarrollo de nuevas características y corregir errores. Aunque es la responsabilidad del núcleo del equipo decidir la dirección que tomará el proyecto, esto pasa raramente en la realidad, los responsables del control de la configuración usualmente determinan su propio proyecto, por ejemplo, agregar una característica. Algunas veces, los responsables del control de la configuración pueden formar equipos para trabajar en proyectos más grandes.

   A los errores reportados al proyecto por los contribuyentes se les realiza un seguimiento usando la base de datos GNATS. Hay tres formas de reportar un problema:

   - Usar el comando *send-pr* de FreeBSD,
   - Usar un formulario basado en web suministrado en la página web de FreeBSD,
   - Enviar un e-mail a FreeBSD-bugs@FreeBSD.org.

   Cada problema reportado tiene los siguientes campos: número de referencia, responsable del control de la configuración, fecha de registro, severidad, nombre de quien lo reporta, estado, y descripción. Un problema reportado puede estar en uno de los siguientes estados: abierto (solo ha sido enviado, ningún esfuerzo se ha realizado aún para corregirlo), analizado, retroalimentación, parcheado, suspendido, o cerrado (el error ha sido corregido o no pudo ser corregido). La página web de FreeBSD también suministra unas pautas para reportar problemas, y así hacer que las descripciones sean lo más informativas posibles.

3. **Asignar y Realizar el Trabajo de Desarrollo**
   Un responsable del control de la configuración puede buscar en la base de datos de reportes de errores los problemas que estén abiertos y asignárselos a él mismo o a otro responsable del control de la configuración que pueda estar capacitado para resolver el problema. Muchos problemas reportados tienen soluciones sugeridas por la persona que reporta el error. Los contribuyentes pueden buscar en la base de datos de problemas reportados y proponer soluciones a los que estén abiertos. Aunque los contribuyentes no tienen acceso a cambiar el código fuente base, ellos pueden probar las soluciones en su propia copia del código fuente y enviar la solución al correspondiente responsable del control de la configuración que la tenga asignada. Un contribuyente puede también enviar una solución a la lista de correo. El responsable del control de la configuración responsable del problema reportado puede discutir con todos los contribuyentes interesados el problema y las posibles soluciones. Un responsable del control de la configuración puede resolver el problema directamente o usar una solución que ha sido propuesta por un contribuyente. Luego de probar una solución propuesta el responsable del control de la configuración puede insertar la solución en el código fuente base actual.

Para implementar nuevas características, un responsable del control de la configuración escribe el código, lo prueba, y luego lo agrega al código fuente base actual. Antes de la fecha de liberación de una nueva versión estable, un responsable del control de la configuración puede decidir integrar su código nuevo con la versión estable.

4. **Realizar Pruebas**
   Los responsables del control de la configuración deben probar su propio código (con la ayuda de los contribuyentes interesados) antes de que ellos puedan entregar su código a la versión de desarrollo. El nivel de exigencia de las pruebas dependerá del criterio y la experiencia de un responsable del control de la configuración. Luego de desarrollar el nuevo código, los responsables del control de la configuración establecen un periodo de cuenta atrás y preguntan a los otros desarrolladores y contribuyentes quien desea probar el código. Si no se encuentran errores durante la cuenta atrás, un responsable del control de la configuración puede asumir que el código es aceptable.

   Una prueba del sistema puede ser considerada como otro tipo de prueba. Antes de liberar una nueva versión, la versión candidata es presentada a los responsables del control de la configuración y a los contribuyentes. La versión candidata es probada y corregida hasta que el equipo de control de versiones decide que el sistema está listo. Sin embargo, ningún responsable del control de la configuración es asignado como el evaluador; los voluntarios prueban la versión candidata.

5. **Inspeccionar el Código**
   Un responsable del control de la configuración puede necesitar entregar una unidad de código o componente que está bajo la responsabilidad de otro responsable del control de la configuración, es decir quien está actualmente realizando cambios. Esta persona debe revisar y aprobar el nuevo código antes de que sea agregado al código base.

   Un desarrollador puede saber quien está realizando cambios a una parte del código usando el comando log del CVS, que indica quien está actualmente cambiando el código. Se espera que un responsable del control de la configuración que planea realizar un cambio significativo en el código solicite a otros responsables del control de la configuración que revisen el código cambiado.

6. **Realizar la Gestión de Versiones**
   El equipo de control de versiones gestiona la liberación de versiones del proyecto FreeBSD. Este equipo de ingenieros tiene como jefe a un voluntario miembro del núcleo del equipo. Los otros miembros del equipo son voluntarios seleccionados de los responsables del control de la configuración. Una nueva versión de FreeBSD es liberada cada cuatro meses.

C.2. Proceso de Desarrollo OSS

El trabajo de investigación de Egan [58] presenta los pasos que deben ser seguidos por todo aquel que esté interesado en participar como voluntario en un proyecto OSS. Estos pasos en conjunto constituyen el proceso de desarrollo OSS. A continuación, se describen cada uno de ellos.

1. **Elegir un Proyecto**
   Tal vez la decisión más importante al iniciarse en el desarrollo OSS es elegir el proyecto donde se participará. El primer proyecto debe ser el de una aplicación que se use con regularidad y en la cual exista, por ejemplo, un interés o una necesidad por ampliar su funcionalidad de alguna manera. Algunos proyectos OSS tienen una lista del trabajo pendiente que necesita ser realizado. Así que si no hay un interés en desarrollar una funcionalidad particular, quizás uno de estos proyectos sea el más adecuado.

2. **Escribir el Primer Parche**
   Una vez se tiene un proyecto en mente, se debe decidir qué escribir. En muchos casos, esto es fácil. Si se necesita una funcionalidad determinada que aún no tiene la aplicación OSS, entonces es una buena oportunidad para empezar. Sin embargo, si se ha determinado en qué proyecto se desea participar pero no se tiene un objetivo específico, puede ser difícil decidir qué hacer.

3. **Corregir Errores**
   La mayoría de proyectos OSS tiene una lista de todos los errores que han sido reportados. Muchos desarrolladores afirman que corregir alguno de estos errores es la mejor manera de empezar a trabajar en un proyecto. Algunos errores son causados por errores triviales de programación que son fáciles de detectar y corregir sin necesidad de entender el contexto del código. A menos que se haga un esfuerzo intencional por aprender acerca del contexto del error o de la funcionalidad donde se presenta, un nuevo programador ganará poca experiencia al solucionar este tipo de errores.

   La mayoría de los otros errores son causados por la interacción inesperada entre partes completamente diferentes del programa. Esto significa que para corregir estos errores, se necesita primero comprender completamente cómo se espera que cada componente interactúe. Es difícil tener una comprensión real de esto sin tener experiencia previa de trabajo con el código fuente. Por lo tanto, los nuevos desarrolladores a menudo no pueden corregir los errores tan eficazmente como podrían hacerlo al añadir nuevas características.

4. **Implementar Nuevas Características**
   Implementar nuevas características es una forma de comenzar a familiarizarse con el código fuente. Estas características se deben ubicar en una parte particular de la aplicación, por lo que sólo es necesario aprender acerca de uno de los componentes y no cómo cada componente interactúa con los demás.

   Si se desea buscar una tarea que realizar, es una buena idea visitar el canal IRC del proyecto OSS y suscribirse a sus listas de correo. Revisar los foros de discusión también permite descubrir rápidamente desarrollos por realizar. Una buena práctica es elegir una tarea pequeña y preguntar a los desarrolladores las dudas que surjan. Esto permitirá darse a conocer dentro de la comunidad.

5. **Enviar el Trabajo**
   Todos los proyectos tiene un procedimiento para enviar parches. Por lo general, en la documentación del proyecto se encuentra descrito tal procedimiento. Usualmente, el parche debe ser enviado a un gestor de incidencias, una base de datos que contiene todos los errores, parches, peticiones de características, etc., del proyecto. Esto facilita a los desarrolladores la gestión de las contribuciones de los usuarios.

   Si se pide ayuda a un desarrollador sobre un parche determinado, es una buena práctica escribirle una breve nota de agradecimiento. Esto dará una buena impresión al desarrollador que revisará el parche. Es probable que el desarrollador que reciba la nota de agradecimiento preste más atención al parche que a otros, lo que asegura una retroalimentación constructiva para el nuevo desarrollador.

   Pueden ser necesarias un par de modificaciones antes de que el parche sea aceptado. Si es el primer parche para el proyecto, esto es normal. Conseguir una buena comprensión de cómo está diseñado el código o el estilo de programación tomará tiempo. Con cada revisión que se realice a los parches se aprenderá más sobre lo que el proyecto espera de cada participante.

C.3. Proceso de Desarrollo del Proyecto TikiWiki

El proyecto TikiWiki tiene una organización flexible, sin ninguna autoridad central que ejerza poder. La filosofía de desarrollo de software seguida en este proyecto implica la colaboración de los participantes a una escala mayor de lo habitual. La estrategia seguida se puede resumir como "reclutar temprano, reclutar a menudo". Esta estrategia fomenta la participación abierta de tantas personas como sea posible, y se aplica indistintamente al código, documentación, ideas, y todo lo que se necesite hacer [62]. A continuación, se describen las actividades del proceso de desarrollo del proyecto TikiWiki.

1. **Requisitos**
   La decisión de cómo crecerá el software y cuáles son las características prioritarias no radica en nadie en particular. En el portal de desarrollo existe una simple lista: cualquiera puede añadir una solicitud de una característica a la lista, y cualquier persona puede elegir cualquier elemento de la lista para implementarlo. El administrador del proyecto afirma que este enfoque no conduce a la duplicación de características similares. La cultura exige un ejercicio de comprobación de lo que ya se ha implementado antes de comenzar un nuevo desarrollo. Dicha comprobación es posible gracias a la completa documentación. Esta documentación es aportada por un gran grupo de personas descentralizadas. El resultado final es una aplicación muy rica en características incorporadas sin redundancia funcional entre ellas. Sin embargo, este resultado no es casual: desde el primer momento, TikiWiki estaba destinado a ser una aplicación con una gran cantidad de características.

2. **Diseño**
   TikiWiki no tiene un diseño central notable. Es una aplicación monolítica. Las características están directamente integradas en el núcleo: no hay una arquitectura plug-in, ni la capacidad para soportar características externas. Todas las contribuciones van a un repositorio central. A raíz de esto, el proyecto puede liberar una versión cada seis meses. Proyectos más pequeños, con una gran cantidad de características externas pueden tardar hasta un año. El proyecto anima fuertemente a la comunidad a que participe en las listas de correo, en las salas de chat y a pedir ayuda, pero eso es todo. Al parecer, este nivel de apoyo es suficiente, dado el gran número de contribuyentes.

   El código fuente es refactorizado sólo después de que ha existido durante mucho tiempo, una vez que los desarrolladores saben lo que se supone debe hacer y están convencidos que la refactorización es necesaria. Por lo tanto, la integridad del código fuente se preserva siguiendo la filosofía "si no está roto, no lo toque".

3. **Calidad (Corregir Errores y Documentación)**
   La calidad también es autorregulada por la cultura de la comunidad OSS. El código fuente base centralizado y la documentación son el punto central de toda actividad. Los usuarios no necesitan preocuparse de parches de terceras partes o de actualizaciones para decenas de plug-ins. Los contribuyentes no necesitan preocuparse de cumplir dependencias externas, con excepción de componentes de la plataforma estándar (MySQL y Apache).

   El arma principal del proyecto TikiWiki es la propuesta de Raymond [155] "con suficientes ojos todos los errores son superficiales". Cerca de 18.000 miembros registrados son un montón de ojos vigilantes. Una funcionalidad con algún error se descubre y corrige rápidamente, si es popular y es usada con bastante frecuencia. Sino es así, la funcionalidad y los errores que vienen con ella están condenados a morir por desuso. Si una funcionalidad nueva interfiere con las funciones más populares de la aplicación, la característica problemática es probablemente eliminada si no se puede corregir.

Entregas frecuentes y oportunas suministran una rápida retroalimentación. La alta actividad de entregas para corregir los errores indica que los errores efectivamente se arreglan. O por lo menos alguien es consciente y trabaja en ellos.

Gracias a un sistema de valoración los usuarios saben lo que obtendrán: la comunidad evalúa cada funcionalidad continuamente, y las puntuaciones cambian con el tiempo. Así, la calidad es visible en un buen nivel.

C.4. Proceso de Desarrollo del Proyecto Annada Event Management

Las siguientes actividades del proceso de desarrollo OSS están basadas en la experiencia de los autores del trabajo de investigación [63] cuando participan en el desarrollo de un pequeño proyecto OSS.

1. **Buscar y Seleccionar un Proyecto**
   Para desarrollar una aplicación de gestión de eventos, primero se realizó una búsqueda de un proyecto OSS que pudiera ser usado como el proyecto semilla. El objetivo era desarrollar una aplicación para organizar las funciones diarias de una persona dedicada a planificar eventos y/o asesorar bodas. Los planificadores de eventos usualmente guardan detalles de los eventos en muchas formas fragmentadas, tales como: hojas de cálculo, papel o mensajes de correo electrónico. Existía la necesidad de una herramienta que capturara y gestionara eficientemente los detalles de los eventos, en un único lugar centralizado. Tal herramienta debería asistir a los planificadores de eventos en la creación, seguimiento, y modificación de los eventos. El proyecto semilla debía tener una funcionalidad de planificación básica que pudiera ser extendida y diseñada en una forma modular.

   Durante el proceso de búsqueda se encontró que una búsqueda superficial, que involucró la lectura de la documentación, no fue suficiente para entender el proyecto y la aplicación. En muchos casos fue necesario descargar e inspeccionar el código fuente porque no había suficiente información acerca de la funcionalidad de la aplicación en su sitio Web. Luego de invertir una gran cantidad de tiempo, fue seleccionado el proyecto *phpMyAgenda* puesto que suministraba una funcionalidad básica y expandible para crear y modificar eventos, y estaba escrita en PHP.

2. **Interactuar con los Administradores del Proyecto/Desarrolladores Principales**

   El plan inicial consistió en comunicarse con los desarrolladores principales de *phpMyAgenda* para poder trabajar con ellos en la nueva funcionalidad y eventualmente publicar nuevas características como parte de una nueva versión de la aplicación. Sin embargo, los intentos de buscar ambos administradores del proyecto no fueron exitosos. Sin la cooperación de los desarrolladores principales, se decidió bifurcar un nuevo proyecto OSS desde el código fuente de *phpMyAgenda*.

   La documentación suministrada en el sitio Web del proyecto no decía nada acerca de los cambios que se debían hacer al momento de instalar la aplicación. Se encontró que la propia configuración de *phpMyAgenda* requería conocimientos de programación y administración de bases de datos. Si el usuario no es un programador o no está familiarizado con herramientas de desarrollo, esta configuración podía ser una tarea desalentadora. Esto muestra que sin una documentación detallada sobre cómo configurar y solucionar problemas del sistema, las aplicaciones OSS no pueden atraer el público general quienes están acostumbrados a manuales de usuarios completos, e interfaces de usuario visualmente agradables [108]. Esto se debe a que muchas aplicaciones OSS

son construidas por desarrolladores para otros desarrolladores, y el público en general es intimidado por aplicaciones que parecen estar reservadas para "la élite tecnocrática" [108].

3. **Análisis y Verificación de Requisitos**
   Los requisitos de la aplicación *phpMyAgenda* han sido encontrados en el sitio Web SourceForge y en la documentación. El sitio Web de *phpMyAgenda* estaba fuera de servicio, pero SourceForge incluía foros de discusión y un foro para la petición de características. Los desarrolladores principales habían escrito requisitos informales detallados para la liberación de la última versión en el 2006.

   Para verificar los requisitos, se ejecutó la aplicación y se comparó su comportamiento con los requisitos. En muchos casos, el comportamiento de la aplicación cumplía con los requisitos, pero en algunos casos se debieron hacer correcciones al código. Los nuevos requisitos fueron documentados formalmente. Cada requisito funcional y no funcional del módulo propuesto fue documentado en detalle en una especificación de requisitos.

4. **Diseño**
   El diseño de las nuevas funcionalidades fue realizado después de la funcionalidad original de *phpMyAgenda*. No existía ninguna documentación de los diseños de *phpMyAgenda*. Sin embargo, las nuevas funcionalidades adicionadas fueron formalmente documentadas en una especificación de diseño puesto que una documentación detallada y correcta es crítica para atraer a nuevos desarrolladores y para que comprendan la aplicación.

5. **Implementación**
   La implementación abarcó un tiempo de dos meses. Debido a la carencia de documentación, se invirtió una gran cantidad de tiempo en analizar y entender el código. Contrario a las experiencias reportadas por otros proyectos OSS, no se hizo uso de herramientas de control de versiones o herramientas de comunicación durante la implementación, puesto que este proyecto era pequeño y *stand-alone*. Se supo que algunas herramientas de comunicación fueron usadas en algún momento o en algún punto de las etapas tempranas de *phpMyAgenda*.

   Parece que estas herramientas tienen gran utilidad cuando algunos desarrolladores están involucrados en un proyecto, pero con solo dos desarrolladores estas herramientas generalmente no son muy usadas durante el desarrollo.

6. **Administración y Evolución del Proyecto**
   Durante el desarrollo ha existido la "meritocracia de habilidades". Los usuarios finales fueron quienes descargaron la aplicación. El rol de los usuarios finales se extendió tanto como su interés y capacidad de trabajar en el proyecto. En otras palabras, los roles de los participantes cambiaron con el progreso del proyecto. La liberación del trabajo se realizó bajo un nuevo proyecto con el nombre de *Annada Event Management Software*. El desarrollador que fue inicialmente un usuario llegó a ser el desarrollador principal/coordinador para el nuevo proyecto OSS.

   El nuevo proyecto solo podía crecer si atraía el interés de otros desarrolladores o usuarios. Pero este proyecto no generó un gran interés en los usuarios, porque los sistemas de gestión de eventos no atraen a los desarrolladores en la comunidad OSS. Parece que la gestión de eventos no es muy atractiva entre los desarrolladores debido a que no está directamente relacionada con su trabajo o sus principales intereses.

C.5. Proceso de Desarrollo FOSS

En el desarrollo de software tradicional, el ciclo de vida de desarrollo en su forma más genérica se compone de cuatro grandes fases: planificación, análisis, diseño e implementación.

En el desarrollo Free-Open Source Software (FOSS), estas fases tienden a ser configuradas de forma diferente. Las primeras tres fases de planificación, análisis y diseño se concatenan y realizan normalmente por un único desarrollador o un grupo pequeño de desarrolladores [73]. A continuación, se describen cada una de las cuatro fases que componen el proceso de desarrollo FOSS [73].

1. **Planificación**
   Probablemente la analogía de Raymond [155] que mejor resume la fase de planificación es la de un desarrollador que siente "una comezón que rascarse". La "comezón" conduce a la construcción de un prototipo inicial.

2. **Análisis**
   Teniendo en cuenta que un gran número de desarrolladores distribuidos globalmente, con diferentes niveles de habilidad y de conocimientos en el dominio deben ser capaces de realizar contribuciones, la fase de análisis de requisitos es reemplazada en gran medida. En el proceso de desarrollo FOSS, los requisitos son tomados en su acepción general y no necesitan de la interacción entre los desarrolladores y los usuarios finales. En este sentido, los desarrolladores FOSS son invariablemente los usuarios del software que está siendo desarrollado.

3. **Diseño**
   Las decisiones de diseño también tienden a ser realizadas por adelantado antes de que el mayor número de desarrolladores comience a contribuir. Las aplicaciones OSS son altamente modulares para permitir la distribución del trabajo y reducir la curva de aprendizaje para que los nuevos desarrolladores puedan participar (ellos pueden centrarse en los subsistemas particulares sin necesidad de considerar el sistema en su totalidad).

4. **Implementación**
   En el ciclo de vida del desarrollo FOSS, la fase de implementación consiste de diferentes sub-fases [98]:

   a) **Codificar:** Escribir el código fuente y enviarlo a la comunidad FOSS para revisión.

   b) **Revisar:** Un punto fuerte de FOSS es la revisión de pares independiente.

   c) **Pruebas Pre-Entrega:** Las consecuencias negativas de partir la construcción aseguran que las contribuciones se prueban cuidadosamente antes de ser entregadas.

   d) **Versión de Desarrollo:** Las contribuciones de código fuente pueden ser incluidas en la versión de desarrollo poco tiempo después de haber sido ser entregadas -esta rápida ejecución es un factor importante de motivación para los desarrolladores-.

   e) **Depuración Paralela:** La denominada Ley de Linus ("dados suficientes ojos, todo error es superficial") considera que el gran número de depuradores posibles en diferentes plataformas y configuraciones del sistema asegura que los errores sean encontrados y resueltos rápidamente.

   f) **Versión de Producción:** Una versión de producción depurada y relativamente estable es liberada.

   La gestión de este proceso varía mucho. Diferentes proyectos tienen diversos grados de formalismo en cuanto a cómo se toman las decisiones. A menudo, el fundador del proyecto inicial o un pequeño grupo principal de desarrolladores toma las decisiones clave de acuerdo con el proceso descrito en el ciclo de vida anterior.

C.6. Proceso de Desarrollo OSS 2.0

La iniciativa OSS ha triunfado y el emergente movimiento OSS 2.0 tiene una orientación comercial muy fuerte [73]. A continuación, se describen cada una de las actividades del proceso de desarrollo OSS 2.0 [73].

1. **Planificación**
   La naturaleza en gran medida voluntaria del proceso de desarrollo FOSS (discutido en la sección anterior) condujo a un vacío en relación con la planificación estratégica. En el ciclo de vida del proceso de desarrollo OSS 2.0, por el contrario, la planificación estratégica cobra relevancia. El principio de desarrolladores individuales que sienten "una comezón que rascarse" es sustituida, en el desarrollo OSS 2.0, por compañías que consideran la mejor manera de obtener una ventaja competitiva a partir del código abierto. Por ejemplo, Red Hat ha publicado una hoja de ruta de la arquitectura que detalla sus planes de mudarse al código abierto hacia herramientas de middleware y de gestión. Otras compañías también han visto el potencial estratégico del código abierto para modificar las fuerzas competitivas en su industria, tal vez para aumentar la cuota de mercado o debilitar a la competencia. Por ejemplo, IBM es un firme defensor de Linux, ya que erosiona la rentabilidad del mercado de sistemas operativos y afecta negativamente a competidores como Sun y Microsoft.

2. **Análisis y Diseño**
   Las aplicaciones FOSS han sido dirigidas principalmente a infraestructuras horizontales donde las necesidades y los problemas de diseño son en gran medida parte de la sabiduría establecida, facilitando así una base global de desarrolladores. La mayoría de aplicaciones comerciales, sin embargo, existe en dominios verticales donde el análisis eficaz de requisitos plantea problemas reales. Los estudiantes y desarrolladores sin experiencia en el área del dominio de la aplicación, carecen de los conocimientos necesarios para obtener requisitos precisos. En el proceso de desarrollo OSS 2,0, por lo tanto, las fases de análisis y diseño requieren mejor esfuerzo.

   Dada la naturaleza cada vez más comercial del proceso de desarrollo OSS 2.0, se requiere de una gestión más rigurosa del proyecto para lograr un producto profesional, lo que está ocasionando un cambio. Mediante este cambio, la gestión del proceso de desarrollo es cada vez menos del estilo bazar. Este resultado es ya evidente en las reuniones formales que realizan un pequeño grupo de proyectos OSS populares (por ejemplo, las conferencias de Apache en los Estados Unidos y Europa, y las conferencias anuales del proyecto GNOME) [78]. En estas reuniones los desarrolladores coordinan y planifican los futuros desarrollos.

3. **Implementación**
   En el proceso de desarrollo OSS 2.0, la fase de implementación es similar a la del proceso de desarrollo FOSS (ver subsección anterior), pero el proceso general de desarrollo es menos del estilo bazar.

C.7. Proceso de Desarrollo Open Source Corporativo

El modelo de desarrollo está caracterizado por tres distintas fases [85]. La primera fase -Fase de Desarrollo- abarcó desde abril de 2000 hasta noviembre de 2001. La siguiente fase -Asociación *Ad Hoc* y Solicitudes de Cambio de los Usuarios- desde noviembre de 2001 hasta abril de 2004. Entre mayo de 2004 y mayo de 2005, el software entró en su última, pero más crucial fase, la fase de desarrollo OSS. A continuación, se describirán cada una de estas fases.

1. **Fase I: Desarrollo Inicial**
   El trabajo inicial de desarrollo del software fue conducido por uno de los coautores del trabajo de investigación [85]. El desarrollo fue principalmente un esfuerzo dirigido por el autor del código y un desarrollador adicional. El objetivo del proyecto era la construcción de un servidor de telecomunicaciones. El servidor debía ser una fiel implementación del proyecto *Internet Engineering Task Force (IETF)*. Una vez que el código tenía implementadas suficientes características, se aseguró que cumplía con el protocolo.

2. **Fase II: Asociación *Ad Hoc* y Solicitudes de Cambio de los Usuarios**

   a) **Asociación *Ad Hoc*:** Como el código creció de una manera estable y logró una paridad de características con la funcionalidad especificada en el protocolo, el desarrollo principal empezó a distribuir el binario a una amplia audiencia dentro de la compañía. Un sitio web interno avisaba de las nuevas liberaciones de binarios del servidor a otros dentro de la compañía, para que lo descargaran y experimentaran con él.

   Como el interés en el servidor crecía, se realizaron demostraciones a ciertos grupos de la compañía. Por ejemplo, el desarrollador principal amplió la capacidad de programación del servidor. Usando esta mayor capacidad de programación, el servidor fue vinculado a un *framework* de colaboración que era el centro de la investigación en otros grupos de la compañía [49]. Asociaciones de este tipo benefician a muchos proyectos de investigación dentro de la compañía.

   b) **Solicitudes de Cambio de los Usuarios:** Como el servidor maduró, éste pasó de ser solo un proyecto de investigación a ser producido como parte de un estándar de la compañía. En un principio, a pesar de que ciertos grupos dentro de la empresa tenían acceso al código fuente, no habían contribuciones de los usuarios más allá de la presentación de reportes de su experiencia al desarrollador principal. La mayoría de los usuarios internos simplemente descargaban la versión compilada del servidor y la usaban en su trabajo. Extender la clase de usuarios de esta forma creaba un ciclo de retroalimentación positiva en el que el desarrollador principal del código original implementaba nuevas características que estos usuarios necesitaban. El desarrollador principal animaba a otros usuarios dentro de la compañía a usar el software y a reportar la retroalimentación y solicitudes de nuevas características. Esta comunicación se realizó en una forma *ad hoc*, principalmente a través de correo electrónico y de una pagina Web. Las solicitudes de nuevas características eran ordenadas de acuerdo a las necesidades del grupo que producía el servidor y a los intereses de investigación del desarrollador principal.

   El código fuente del servidor fue estudiado ampliamente por otros grupos de la compañía. Las solicitudes comenzaron a llegar permitiendo al servidor evolucionar hasta convertirse en un *framework* para muchos grupos dentro de la compañía.

3. **Fase III: Estableciendo el Proyecto de Desarrollo OSS**

   a) **Contribución de Código e Ideas por Parte de los Usuarios:** Casi al mismo tiempo que las solicitudes de cambios específicos del producto comenzaron a crecer, otros dentro de la compañía comenzaron a contribuir con código e ideas al desarrollador principal. La etapa estaba lista para entrar en el modelo tradicional de desarrollo OSS, aunque dentro de un entorno industrial.

   b) **Verificar que el Código Cumple con la Arquitectura:** El desarrollador principal del código original asumió el rol de un "dictador benevolente" que controla el código fuente base para asegurar que las contribuciones y las características que otros grupos proponen se implementen siguiendo los principios de arquitectura del software.

c) **Refactorizar el Código:** El autor refactorizó las principales partes del código del servidor para crear una biblioteca de transacciones que pudiera ser usada en cualquier proyecto dentro de la compañía. Trabajando en estrecha cooperación con otros dos proyectos, las APIs e interfaces entre el administrador de transacciones y los usuarios de la transacción fueron definidas para que pudiese fluir información entre el administrador y el usuario de la transacción. La refactorización del software de esta manera fue muy útil y permitió la rápida creación de agentes de usuario [26], que se ejecutaban en la parte superior del administrador de transacciones. Puesto que los agentes de usuario usaban los servicios del administrador de transacciones que ya estaba implementado y probado, los programadores de estos agentes de usuario podrían concentrarse en la tarea de implementar el comportamiento específico del agente de usuario en sí mismo, en lugar de preocuparse por los detalles del manejo de las transacciones y otras minucias relacionadas con el protocolo.

d) **Gestión de la Configuración:** Una vez refactorizado, el código fuente base evolucionó de la siguiente manera. La versión inicial de la biblioteca de transacciones fue desarrollada en un archivo CVS. Como el código pasó a la fase III, fue necesario crear una rama de construcción para permitir que otros desarrolladores ayudaran en el desarrollo y evolución del producto. Las contribuciones en este momento consistían de extensiones al producto base, así como de APIs y de interfaces en torno a ellas.

Esta estructura fue eficaz al principio, puesto que las modificaciones eran relativamente independientes y el número de desarrolladores era limitado. Sin embargo, en etapas posteriores a la fase III, la implementación de las características, la madurez del producto, el número de contribuyentes, y la experiencia de los colaboradores, llevaron a la necesidad de un modelo diferente para evolucionar el código fuente base.

Es muy importante señalar que en un desarrollo de software corporativo, cada proyecto tiene una afinidad con un cierto conjunto de herramientas. El grupo de contribuyentes adiciona características al código de la forma que ya estaba acostumbrado en el entorno de desarrollo de su organización. Por lo tanto, algunas organizaciones hicieron una copia del archivo CVS y lo replicaron en su ambiente de software local para modelar de cerca lo que los desarrolladores estaban acostumbrados. Por supuesto, cuando ninguna de las organizaciones usaba CVS para controlar el código fuente, los archivos de código fuente fueron colocados bajo el sistema de control de código que la organización particular conocía. Al mismo tiempo, al CVS original y a la rama inicial de CVS se le seguía brindando soporte. Fue en este momento que nació el concepto de un repositorio de código fuente común e independiente. Para coordinar este repositorio se constituyó formalmente un grupo open source. Este grupo se denominó Common SIP Stack (CSS).

El objetivo del grupo CSS es doble: por una parte mantener el repositorio de código fuente común e independiente de manera que todos los proyectos dentro de la empresa tomen sus entregables del grupo CSS. Esto no es una tarea fácil. El grupo CSS no solamente debe mantener tal repositorio, sino también arbitrar qué características serán implementadas y asegurar que todas las nuevas características no dañen las funcionalidades existentes pertenecientes a un proyecto diferente. Además, el proyecto CSS debe tener una visión de hacia donde debe evolucionar el código.

El segundo objetivo de CSS es evangelizar la tecnología y la implementación mediante la creación de concienciación de los recursos dentro de la compañía. En este sentido, fue establecido un Centro de Excelencia (COE) que actúa como un sitio web central desde el que otros proyectos dentro de la compañía obtienen información sobre los elementos compartidos e instrucciones sobre cómo descargar,

compilar y ejecutar el código fuente. El COE actúa como una ventanilla única para todas las necesidades que surjan en cualquiera de los proyectos de la compañía.

C.8. Proceso Descriptivo para el Desarrollo OSS

El modelo de proceso descriptivo para el desarrollo OSS [97] consta de tres vistas:

1. **Vista de Estado**
   La vista de estado de un proceso de software cubre varias etapas del desarrollo del producto. Éstas incluyen tareas y etapas relacionadas con el diseño, codificación y pruebas del producto. Las características de esta vista pueden identificarse respondiendo a la pregunta ¿Cómo se produce el trabajo?

   Los proyectos OSS inician con un prototipo cerrado, ya sea desarrollado desde cero o basado en algún producto más viejo existente. Luego de una extensa liberación de versiones, los voluntarios empiezan a evolucionar gradualmente esta versión inicial a través de iteraciones rápidas, mientras que concurrentemente se gestionan tantas actividades de diseño, construcción y pruebas como sea posible. Los requisitos están dirigidos al usuario y los proyectos confían en una revisión de pares a gran escala para eliminar los errores. Este enfoque puede ser dividido en cinco características:

   a) **Prototipado Cerrado:** Una persona o un pequeño grupo desarrollan una versión inicial del producto. Ésta es usada para presentar una promesa creíble y establecer un diseño conceptual. Una vez el prototipo está listo, es liberado en Internet.

   b) **Mejoras Iterativas e Incrementales:** El prototipo, o "construcción 0" evoluciona incrementalmente a través de una serie de iteraciones regulares. Los incrementos son pequeños y las iteraciones frecuentes. La mayoría de los proyectos tienden a realizar un grupo pequeño de cambios. El ciclo de vida de una petición de cambio típica, es la siguiente:

      1) *Voluntario Corrige Error/Hace Mejora.* Un voluntario acepta la responsabilidad de una tarea dada, normalmente corregir un error o hacer una mejora.
      2) *Copiar el Código Fuente.* Obtener una copia funcional del código fuente.
      3) *Implementar el Cambio.* Realizar los cambios en la copia funcional modificando el código fuente. Debido a que la copia funcional está aparte, no hay ninguna interferencia.
      4) *Crear el Parche.* Crear un parche que represente las diferencias entre la copia vieja y una nueva versión.
      5) *Entregar el Código Fuente.* Colocar el parche y una breve descripción que explique su relevancia en la lista de correo de los desarrolladores.

   c) **Desarrollo Concurrente en Muchos Niveles:** En los proyectos OSS muchas actividades de diseño, construcción y pruebas se realizan simultáneamente. No hay fases distintas. En lugar de ello, los participantes tienen la flexibilidad de trabajar en cualquier tarea que les parezca interesante. Algunos escriben código, mientras que otros depuran cambios o discuten nuevas características.

   El desarrollo se lleva a cabo concurrentemente en muchos niveles. Muchos contribuyentes iteran a través de una serie común de acciones. Ellos descubren un requisito, identifican una solución, desarrollan y prueban con su propia copia local del código fuente, y entregan un cambio. En cualquier momento, diferentes desarrolladores pueden estar trabajando paralelamente en un rango de tareas, cada una en diferentes fases de realización.

Para sincronizar los cambios, en algún punto un ciclo iterativo superpuesto finaliza y todos los parches pendientes son combinados en la distribución principal para ser liberada. Las pruebas se intensifican así como los errores son corregidos y una nueva ronda de características es implementada, iniciando un nuevo ciclo.

d) **Revisión de Pares a Gran Escala:** Los cambios están sujetos a revisión por una base de usuarios diversa y muy motivada. La revisión de pares está implícita en el desarrollo OSS. El código fuente está disponible para todos, y las comunicaciones técnicas, como por ejemplo los reportes de error, son llevadas a cabo en público. Esto anima a los desarrolladores a ser cuidadosos y a pensar dos veces antes de liberar un código fuente con fallos. Aún así, se asume que el código liberado no está libre de defectos. Más bien, el producto continuamente es mejorado a través de revisiones y correcciones generadas por un gran número de usuarios involucrados activamente.

e) **Requisitos Dirigidos por el Usuario:** Los requisitos son tácitamente entendidos por los desarrolladores quienes son ellos mismos los usuarios del producto. En el desarrollo OSS no hay especificaciones formales. En cambio, los proyectos intentan minimizar la invasión de características debatiendo públicamente los cambios propuestos, ya sea a través de un grupo de noticias o de una lista de correo.

2. **Vista Organizacional**
   La vista organizacional de un modelo de proceso software direcciona los aspectos sociales del desarrollo. Esto incluye factores relacionados con la comunicación y la coordinación, roles y responsabilidades clave y la motivación. Las características de esta vista pueden ser identificadas respondiendo a la pregunta ¿cómo es organizado el trabajo?

   La estructura organizacional de un proyecto OSS es descentralizada. Los participantes se autorregulan y escogen el trabajo que desean realizar de acuerdo a sus intereses. Ellos se comunican asincrónicamente. Un líder del proyecto coordina todos los desarrollos, mientras que los líderes secundarios son los responsables de especificar subsistemas específicos. Los líderes controlan la integración, y su autoridad está basada en la confianza ganada por su buen trabajo. Este enfoque puede ser dividido en cuatro características:

   a) **Colaboración Descentralizada:** El equipo tiene un líder que coordina las tareas específicas y los líderes secundarios tienen la responsabilidad de definir subtareas. La solución de problemas continúa siendo una actividad de grupo, pero la implementación es dividida entre subgrupos.

   b) **Confianza en el Liderazgo:** La jerarquía del control está construida sobre una red personal de confianza. La autoridad y la responsabilidad se mueven hacia aquéllos que demuestran ser los más competentes.

   c) **Motivación Interna:** Motivaciones externas como compensaciones económicas son secundarias. Las personas contribuyen por diferentes razones, como por ejemplo, la comunidad y el estatus.

   d) **Comunicación Asíncrona:** La distribución geográfica hace que la comunicación síncrona sea impracticable. Gran parte de la comunicación se hace a través del correo electrónico.

3. **Vista de Control**
   La vista de control de un modelo de proceso software se centra en la dirección. Más específicamente, tiene que ver con los mecanismos para guiar el desarrollo. Esto incluye la planificación, aprobación, recolección de datos, soporte y documentación. Las características de esta vista puede ser identificadas respondiendo a la pregunta ¿cómo es controlado el trabajo?

En el desarrollo OSS, el control está implícito. No hay reglas o pautas escritas. Los voluntarios trabajan en las tareas hasta que ellos se sienten cansados, y como resultado los plazos de entrega estrictos no son viables. La integridad del código base es mantenida a través de un modelo participativo basado en competencias. Esto es soportado normalmente por una arquitectura modular, que ayuda a promover una alta división del trabajo. Este enfoque puede ser dividido en cinco características:

a) **Planificación Informal:** No hay un plan o visión concreta. La única meta a largo plazo es mejorar el producto.

b) **Participación Escalonada:** Los participantes trabajan en diferentes niveles, lo que refleja una inclinación natural de competencia y compromiso.

c) **Arquitecturas Diseñadas para Modularidad:** El diseño modular reduce las interdependencias, permitiendo que el desarrollo se lleve a cabo más limpiamente. La modularidad es enfatizada fuertemente en el desarrollo OSS. Esto ayuda a los participantes a contribuir de manera efectiva, al mismo tiempo que comprenden sólo un subconjunto del sistema, y se reducen los problemas asociados con las entregas de baja calidad. Esto es importante porque los proyectos OSS no siguen siempre la práctica del diseño limpio o elegante. Un diseño limpio en los proyectos OSS es secundario a la ejecución del código.

   La mayoría de los participantes están interesados en mejoras o modificaciones específicas. Los cambios se realizan en pequeños incrementos y todas las decisiones de diseño que sean posibles se posponen hasta que el código ha tenido algún uso real. Los proyectos OSS siguen diseños altamente modulares porque una estructura organizacional descentralizada requiere este enfoque.

   En general, el diseño modular en el desarrollo OSS es impulsado por una estructura organizativa descentralizada. Existe una alta división del trabajo en la mayoría de los proyectos y una arquitectura modular hace esto más fácil.

d) **Herramientas de Soporte Unificadas:** Las herramientas están libremente disponibles y son razonablemente consistentes en todos los proyectos, disminuyendo la barrera de entrada a la que se enfrentan los nuevos colaboradores.

e) **Espacio de Información Compartido:** Los sitios Web suministran fácil acceso a los recursos de información, como por ejemplo: la documentación, los foros de discusión y las bases de datos de reportes de problemas.

C.9. Proceso de Desarrollo de Software en el Proyecto FreeBSD

La Figura C.1 ilustra el Ciclo de Vida por Cambios seguido en el proyecto OSS FreeBSD [98]. En general, cada etapa es obligatoria.

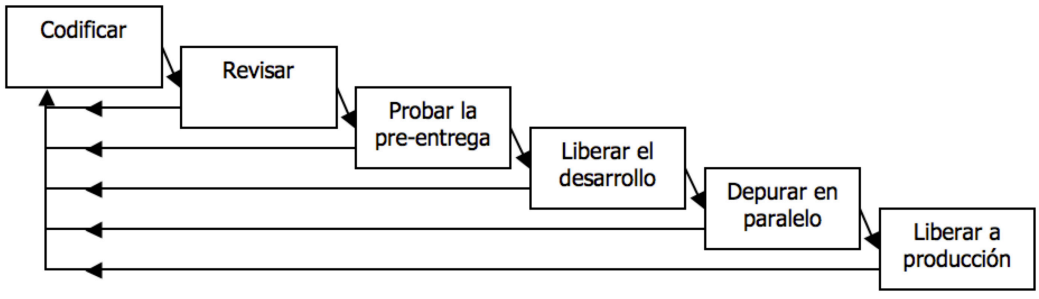


Figura C.1: Ciclo de Vida por Cambios.

1. **Codificar**
   El núcleo, las utilidades y los "ports wrappers" están escritos casi exclusivamente en lenguaje C. La guía de estilo para el código fuente describe el uso correcto del lenguaje. Esta guía de estilo es complementada con una guía dedicada a la seguridad, que explica los problemas de seguridad básicos y esboza la manera de solucionarlos.

   El aspecto más importante del proceso de codificación es quizás lo sencillo de supervisar siempre y cuando se sigan las guías de programación. Cuando se realiza una entrega, automáticamente es enviado un mensaje de correo. La visibilidad instantánea del código entregado permite monitorizar la calidad del código y motiva a los desarrolladores a mejorar sus habilidades.

2. **Revisar**
   El proyecto sugiere fuertemente (pero no es absolutamente necesario) que toda contribución sea revisada. Para solicitar retroalimentación, el código es distribuido típicamente a través de e-mail. El principal obstáculo es la carencia de retroalimentación, particularmente para los documentos de diseño y el código complejo.

3. **Probar la Pre-Entrega**
   En el corazón del modelo de desarrollo de FreeBSD está el requisito de que un delegado debe probar la pre-entrega para evitar que se rompa la estructura en la rama de desarrollo. La entrega es requerida para comprender no solo los archivos nuevos o modificados, sino cualquier archivo independiente. En realidad, esto implica que el delegado es responsable de integrar sus cambios antes de entregarlos.

   Para un delegado, la construcción del sistema trae consigo ciertos requisitos de hardware y de infraestructura. El delegado debe tener acceso a una copia privada de la última versión que se encuentra en la rama de desarrollo, en la que él pueda insertar cambios sin interferir con los otros desarrolladores. La alta frecuencia de contribuciones implica que la copia de un delegado puede rápidamente desactualizarse.

4. **Liberar el Desarrollo**
   El delegado tiene la autoridad para decidir si un cambio está listo, para ser liberado. Él también lleva a cabo la tarea de liberar el cambio, que será una nueva versión en la rama de desarrollo. Existe un proceso bien definido para el caso en el que, después de una entrega, resulta que no se ha llegado a un consenso, a pesar de la revisión de los requisitos.

5. **Depurar en Paralelo**
   Cuando un cambio ha sido entregado a la rama de desarrollo, es sometido a la depuración en paralelo. La depuración es realizada principalmente por los delegados y los contribuyentes externos de código fuente. Para aquellos que tengan dificultades o inconvenientes al descargar y construir el código fuente, el proyecto suministra los ficheros binarios.

   Existe una herramienta basada en Web para la creación de los reportes de problemas (RPs) y el envío de los mismos a la base de datos correspondiente. Al realizar una depuración en paralelo en la rama de desarrollo, se obtiene una gran cantidad de retroalimentación para los colaboradores. Sin embargo, la retroalimentación generada por esta depuración tiene poco valor para los delegados.

6. **Liberar a Producción**
   Cuando existen muchos cambios, la versión de producción se realiza en dos pasos. El primero, es combinar los cambios de la rama de desarrollo con los de la rama de producción. Un cambio es incorporado por el delegado cuando él cree que ha sido suficientemente probado en la rama de desarrollo. Sin embargo, el ingeniero de versiones

puede requerir finalmente que un cambio sea retirado de la rama de producción. El segundo paso hacia la versión de producción es un periodo de congelamiento del código. Durante el congelamiento del código, el ingeniero de versiones tiene la autoridad de rechazar todos los otros cambios que no sean correcciones de errores. El congelamiento del código en la rama de producción finaliza cuando el ingeniero de versiones declara que está lo suficientemente estable como para constituir la siguiente versión menor.

Los cambios que son parte de una nueva característica principal no son incorporados a la rama de producción. En su lugar, estos cambios permanecen dentro de la rama de desarrollo, hasta que eventualmente hagan parte de la siguiente versión principal de producción. Luego de un congelamiento del código de aproximadamente un mes, el ingeniero de versiones declara la versión residente en la rama de desarrollo como la nueva versión principal de producción.

Las versiones principales son uno de los pocos eventos para los que hay una fecha límite: el proyecto FreeBSD intenta suministrar una versión principal aproximadamente cada 18 meses. Esto establece un objetivo común para todo el proyecto. Aunque, normalmente no hay una fecha límite para cambios individuales.

C.10. Proceso para Formar una Comunidad OSS

En el trabajo de investigación de Lelli y Jazayeri [106], los autores describen paso a paso un método para que el administrador de un proyecto OSS maximice la posibilidad de formar una comunidad en torno a un proyecto de desarrollo. Además, los autores sugieren algunas pautas para comprender mejor la comunidad OSS con el fin de minimizar la posibilidad de malentendidos. A continuación, se describen cada uno de estos pasos.

1. **Todo Inicia con una Buena Idea y una Prueba de Concepto**
   Todos los proyectos OSS deben comenzar con una idea razonable. La comunidad OSS debe ver el proyecto como una "promesa plausible", aunque la versión inicial de la aplicación OSS esté incompleta y tenga errores al principio. Los desarrolladores OSS tienen que ver que el proyecto tiene potencial, o no contribuirán. Definir una visión, misión y estrategia para el proyecto juega un papel importante en este paso:

   a) **Definición de la Visión**. El objetivo del proyecto debe ser escrito de una manera comprensible y concisa. En otras palabras, el líder del proyecto debe ser capaz de describir el propósito de su proyecto en una frase.

   b) **Definición de la Misión**. Debe ser fácil de entender lo que el proyecto quiere lograr. Una vez más, una frase debería ser capaz de capturar el objetivo macro del proyecto.

   c) **Definición de la Estrategia**. Una vez que se han definido los objetivos del proyecto, es necesario describir cómo se pondrá en práctica la misión.

2. **Escoger una Licencia**
   La licencia que se seleccione para distribuir la aplicación OSS influirá en la manera en que el código será utilizado por otros y el tipo de voluntarios que contactará con el líder del proyecto. Podemos clasificar las licencias OSS en dos categorías diferentes:

   - **Libre Genérica:** Estas licencias no restringen ningún uso del código. Por tanto, el código del proyecto puede ser incluido en software propietario. Ejemplos de estas licencias incluyen MIT y BSD.
   - **GNU GPL:** Ningún software propietario puede usar parte del código liberado con esta licencia.

3. **Infraestructura del Proyecto**
   Es necesario seleccionar, instalar y configurar algunas herramientas de desarrollo para el proyecto. Esta fase del proyecto no requiere mucho tiempo porque la mayoría de estas herramientas que conforman la infraestructura de los proyectos OSS están disponibles en plataformas como *SourceForge* o la *Free Software Foundation*. Las herramientas y servicios más importantes son:

   - **Sitio Web:** Ésta es la principal fuente de información del proyecto hacia el público. El sitio Web también puede servir como una interfaz administrativa para coordinar la actividad del proyecto y debería incluir una wiki y un blog.
   - **Listas de Correo:** Por lo general, es el medio de comunicación más activo del proyecto y proporciona una manera de registrar las conversaciones. En las fases iniciales del proyecto OSS probablemente la mayoría de la comunicación se realizará a través de correos electrónicos, pero si la comunidad crece entonces el tráfico en las listas de correo se incrementará lentamente.
   - **Control de Versiones:** Sirve para dos propósitos diferentes: (i) permitir a los desarrolladores gestionar los cambios del código adecuadamente, y (ii) permitir que cualquiera observe que es lo que está ocurriendo con el código.
   - **Seguimiento de Errores:** Los desarrolladores pueden realizar un seguimiento del trabajo que realizan, coordinando con los demás, y planificando los lanzamientos. Además, la herramienta para el seguimiento de errores permite a todos los miembros de la comunidad consultar el estado de los errores y registrar información.
   - **Web Analytics y Search Engine Analytics:** Estas herramientas pueden ayudar a entender quién está accediendo al sitio Web del proyecto y qué sitios Web están referenciando el proyecto. Algunas empresas como Google ofrecen estos servicios de forma gratuita.

4. **Anunciar el Proyecto en Canales Open Source**
   Las actividades 1 y 3 descritas anteriormente deben ser realizadas por los líderes del proyecto y no implican directamente ningún tipo de interacción con la comunidad OSS. En esta etapa, los objetivos del proyecto deben ser claros y la infraestructura del proyecto debe estar configurada correctamente. Por lo tanto, éste es el momento de comenzar a comunicar las ideas del proyecto dentro de la comunidad OSS. Uno de los recursos más populares para anunciar los proyectos OSS es el sitio Web de SourceForge. Este sitio no solamente suministra la infraestructura básica descrita en la actividad 3, sino que también ofrece servicios para la comunidad OSS en general, como por ejemplo foros de discusión.

5. **Hacer el Código Entendible por Otros**
   El artefacto creado en la actividad 1 se ha utilizado para validar la visión del proyecto y ha contribuido en la definición de una estrategia. Por lo tanto, puede tener errores y/o no estar escrito correctamente. Uno de los objetivos principales de la construcción de una comunidad es beneficiarse de la contribución de los voluntarios y los pasos iniciales, por lo general, representan el mayor obstáculo para los voluntarios. Mucha gente puede cambiar de opinión y decidir contribuir con otros proyectos si se encuentra con dificultades al principio. Por lo tanto, hay que reducir tanto como sea posible la curva de aprendizaje del código fuente del proyecto. Es necesaria una refactorización mayor del código y la metáfora que debe conducir a la siguiente versión es: "usted está escribiendo código para la comunidad y no para usted". Es recomendable, por ejemplo, que el código fuente sea modular, fácil de instalar y esté debidamente documentado.

   Al final de la refactorización, las decisiones arquitectónicas y de diseño deben ser públicas y documentadas en las páginas wiki del proyecto y el sitio Web debe contener

toda la información presentada en la actividad 3. La versión del software resultado de esta fase del proyecto debe estar disponible para su descarga. Esta versión no tiene todas las características planificadas pero debe ser auto-consistente. Una persona debe entender cuál es el propósito principal del software y cómo utilizarlo en poco tiempo. Un pequeño demo ayudará también a cualquier miembro de la comunidad OSS a entender el propósito del software. Por lo tanto, debe ser producido y documentado un demo público en el sitio Web del proyecto. La liberación de versiones del software con frecuencia dará una mayor visibilidad al proyecto dentro de la comunidad OSS.

6. **Listar las Características del Proyecto**
   En esta etapa del proyecto podemos asumir que la primera versión del software ha sido realizada. Por lo tanto, los líderes del proyecto deben tener algunas ideas con respecto a las características que faltan y a los errores conocidos. El administrador del proyecto debe listar estas características y errores en la herramienta de seguimiento respectiva. Para cada punto debe proporcionarse una pequeña descripción de la tarea. En la descripción los voluntarios esperan encontrar una estimación del tiempo y las habilidades necesarias para terminar la tarea. Esta actividad servirá para crear una especie de hoja de ruta pública del proyecto que puede ser compartida con los nuevos miembros potenciales. Al planear las características, los líderes deben tener en cuenta que el proyecto se encuentra en sus fases iniciales. Por lo tanto, la mayoría de los voluntarios se pondrán en contacto con el administrador porque han quedado impresionados positivamente por la visión del proyecto (ver actividad 1) y están interesados en tener más información. Con el tiempo, es posible que decidan hacerse cargo del desarrollo de una característica o proponer una nueva. En esta fase los líderes no deben asumir que todos los voluntarios podrían realizar con éxito una tarea. Por lo tanto, es recomendable dividir las características en dos categorías:

   - **Críticas:** El éxito del proyecto depende de éstas.
   - **Opcionales:** El proyecto podría vivir sin esta característica, pero su implementación mejorará la calidad general del software.

7. **Delegar a los Voluntarios**
   Los líderes personalmente realizan un seguimiento de las características que pertenecen a la categoría de críticas y deberían estar felices al delegar a los voluntarios las tareas opcionales. Tener un nuevo miembro en la comunidad que realiza una característica opcional da la oportunidad de establecer una conexión personal, conocer la calidad de su trabajo y luego decidir si se involucra en otras actividades del proyecto (ver actividad 9).

8. **Buscar Apoyo en la Comunidad OSS**
   Los líderes deben ahora buscar apoyo en la Comunidad OSS. Al observar las características que faltan por implementar, se pueden definir los perfiles de los desarrolladores. Se recomienda no limitar solamente las solicitudes a personas con conocimientos en programación, un proyecto exitoso necesita traductores, editores de documentación, administradores para el sitio Web, probadores, responsables de la gestión de la configuración, investigadores y también empresas que buscan la posible explotación del proyecto. Deben crearse diferentes anuncios para cada distinto perfil. Al escribir los anuncios se recomienda utilizar un estilo que se encuentre en el límite entre el email a un amigo y un verdadero anuncio de trabajo. Finalmente, debe proporcionarse un enlace para obtener más información sobre el proyecto, así como la información de contacto de una persona de referencia. Estos anuncios pueden ser publicados, por ejemplo, en los foros de SourceForge.

9. **Finalizar la Liberación de la Versión**
   En esta etapa del proyecto se espera que los miembros de la comunidad OSS se comuniquen con el líder del proyecto para solicitar más información y discutir su posible contribución. Algunos de los miembros contribuyen con ideas, mientras que otros esperan ser guiados en la realización de una tarea. Se recomienda una actitud positiva y transparente y evolucionar la documentación de acuerdo con las solicitudes de aclaración provenientes de los voluntarios. El proceso de formar y mantener una comunidad que soporte el software será parte de los desarrollos del proyecto durante su ciclo de vida. Esta etapa concluye la fase inicial de la creación de una comunidad en torno a un producto software determinado. La comunidad está en su fase embrionaria y hay una dependencia total del líder. Si él/ella decidiera abandonar el proyecto, probablemente una comunidad nunca se establecería correctamente. Los desarrolladores junior y senior se pondrán en contacto con el administrador del proyecto y esperarán una respuesta rápida y precisa a sus preguntas. La visión del proyecto y la calidad de la información que están disponibles en la web influirá en la cantidad de personas que entrarán en contacto con el administrador del proyecto. Las habilidades técnicas del lider y su capacidad de comunicar las ideas del proyecto también influirán en el número de voluntarios que decidan contribuir al proyecto. Mientras que la comunidad se encuentra en esta fase, todos los voluntarios consultan al líder del proyecto para cualquier decisión y/o problema [196] y él puede decidir seguir una de las siguientes estrategias generales:

   - **Mantener una comunicación personal:** La información se dirige siempre a la persona que la solicita.
   - **Intentar abrir la comunicación:** Tratar de usar listas de correo o cualquier otra forma de comunicación abierta tanto como sea posible.

C.11. Proceso de Desarrollo OSS para una Organización Pequeña

Kitware es una compañía que ha pasado de tener dos desarrolladores a tener 30. Kitware produce una mezcla de productos comerciales closed-source y open source. Algunos productos son internos a la compañía, pero otros tienen colaboradores ubicados alrededor del mundo. Del mismo modo, algunos proyectos son pequeños con uno o dos desarrolladores, pero otros tienen 30 o más. Por lo tanto, cualquier enfoque de desarrollo que se utilice debe tener costes generales bajos que permitan utilizarse con proyectos pequeños, pero también ser capaz de manejar grandes proyectos de miles de archivos de código fuente. Todos estos requisitos, además de la necesidad de desarrollar para Windows, Linux y Mac OS X, sugieren que la solución debe ser muy flexible. El enfoque de desarrollo de software de Kitware trata algunos de estos aspectos. Este enfoque ha sido refinado mientras la compañía trabajaba en diferentes proyectos de desarrollo, y tiene su origen en el año 1999 cuando la compañía desarrolló un kit de herramientas OSS para *US National Library of Medicine (NLM)* [93][175].

El proceso de desarrollo de Kitware consta de cinco partes: comunicación y documentación, control de revisiones, gestión de la construcción, pruebas y creación de la versión o proceso de liberación [116]. El proceso ha sido diseñado para ser ágil y permitir el desarrollo rápido, teniendo como requisito que la mayoría de las herramientas usadas sean OSS. A continuación, se describirán cada una de las cinco partes que componen el proceso de desarrollo de Kitware [116].

1. **Comunicación y Documentación**
   Podría decirse que los aspectos más importantes de un proceso de desarrollo de software son una buena comunicación y documentación. En el proceso de desarrollo de Kitware

se usan diferentes técnicas para fomentar la comunicación y documentación constructiva durante el desarrollo y mantenimiento del software.

- **Listas de Correo:** Las listas de correo son una forma indispensable de comunicación entre los desarrolladores de software y los usuarios. La herramienta Web *Mailman* permite crear y mantener listas de correo de una manera simple. Cada proyecto tenía normalmente una lista de desarrolladores y una lista de usuarios. La lista de los desarrolladores contiene discusiones detalladas acerca de los problemas de diseño. La lista de usuarios suministra un lugar donde los usuarios finales exponen sus inquietudes acerca de cómo usar el software. Los usuarios a menudo se ayudan entre sí, aunque los desarrolladores comúnmente se suscriben a la lista de los usuarios para contrarrestar cualquier mal consejo que pudiera aparecer. La lista de correos es también beneficiosa porque da lugar a archivos de búsqueda, que proporcionan una fuente de documentación y pueden contener la información más reciente acerca de cómo usar un proyecto.
- **Wiki:** Un excelente complemento a la lista de correos es una wiki, es decir, una página Web que la comunidad puede editar rápidamente. Esto permite la rápida documentación de las discusiones en las listas de correo.
- **Doxygen:** En el proceso de desarrollo se incorporaron las ideas de *Literate Programing*, que consiste en combinar documentación y código fuente en un solo archivo [178]. La herramienta *Doxygen* emplea un lenguaje de marcado de comentarios simple en C o C++. Esta herramienta genera una extensa documentación de código fuente. Como la herramienta genera automáticamente la documentación del código fuente, puede ser actualizada automáticamente cada noche para que coincida con la copia más reciente del código fuente en el sistema de control de versiones.
- **Seguimiento de Incidencias:** Otra herramienta poderosa en el proceso de desarrollo de software es el gestor de incidencias o errores. Un gestor de incidencias permite a los desarrolladores y usuarios reportar los problemas del software y proveer un lugar para almacenar las peticiones de características con el objetivo de que no sean olvidadas. El sistema web escogido fue *phpBugTracker* porque es fácil de instalar, tiene una interfaz de usuario clara, y puede fácilmente manejar múltiples proyectos. Como los desarrolladores y los usuarios crean incidencias en el sistema, éstas tienen un único identificador, al cual se le puede asignar una severidad y prioridad. El administrador del gestor de incidencias puede dar a los desarrolladores privilegios extras para que ellos puedan asignar el estado de sus incidencias. El sistema también permite anexar archivos a las incidencias.

2. **Control de Revisiones**
   Un sistema de control de revisiones permite a los desarrolladores realizar un seguimiento de los cambios que sufre el software a través del tiempo. La mayoría de software de control de revisiones trabaja con un repositorio central para los archivos bajo control de revisión. El repositorio central, usualmente almacena los archivos como un histórico de cambios. Cada vez que un desarrollador cambia un archivo, él o ella puede "registrar" el cambio en el repositorio, haciendo que esté disponible a los otros desarrolladores.

   La herramienta más popular para el control de revisiones es CVS, construida en una arquitectura cliente-servidor. El servidor almacena los archivos en el proyecto, y los clientes pueden conectarse al servidor y echar un vistazo a las copias del software. El cliente puede solicitar cualquier versión de un archivo en el sistema.

   Cuando llega el momento de crear una versión para un proyecto software, un desarrollador es asignado como el administrador de la versión. El administrador informa a todos los desarrolladores que dejen todas las modificaciones finales en el repositorio.

Luego, el repositorio es bloqueado y solamente el administrador de la versión tiene permisos de escritura. El administrador de la versión entonces prueba el sistema y, cuando las características deseadas están presentes, crea una rama para la liberación. El administrador de la versión a continuación aplica correcciones sólo a la rama. La herramienta CVS se puede configurar para que otros desarrolladores no puedan realizar cambios en una rama determinada. Luego, el resto de los desarrolladores puede volver al árbol principal y comenzar a desarrollar la siguiente versión del software.

3. **Gestión de la Construcción**
   El proceso de construcción toma un grupo de archivos de código fuente y los transforma en bibliotecas y aplicaciones de usuario final. Un paquete de software grande suele emplear un conjunto de herramientas que consta de compiladores. Coordinar todas estas herramientas de forma consistente de un ordenador a otro y de un desarrollador a otro desarrollador puede ser difícil y consumir tiempo. Estas herramientas y la forma en que son usadas varían significativamente de una plataforma a otra, por lo que gestionar el proceso de construcción a través de Windows, Unix y Mac es difícil.

   Los desarrolladores de Kitware usan la herramienta *CMake*. Esta herramienta toma como entrada scripts sencillos escritos en el lenguaje *CMake*. La sintaxis del lenguaje *CMake* es sencilla, permitiendo que cualquier desarrollador construya fácilmente los ejecutables y las bibliotecas requeridas por el proyecto que se está desarrollando.

4. **Pruebas**
   Las pruebas son una herramienta clave para la producción y mantenimiento de software robusto.

   - **Tipos de Pruebas:** Las pruebas de un paquete software pueden tomar muchas formas. En el nivel más básico están las pruebas que simplemente verifican que el software compila. Aunque esto pueda parecer simple, con la amplia variedad de plataformas y configuraciones disponibles, estas pruebas capturan muchos más problemas que cualquier otro tipo de pruebas.

     Más allá de estas pruebas básicas están las pruebas específicas como las de regresión, caja negra y caja blanca. Cuando una prueba de regresión falla, una rápida mirada a los cambios recientes del código puede usualmente identificar al culpable. Sin embargo, las pruebas de regresión suelen requerir de un mayor esfuerzo.

     El último tipo de pruebas es la del cumplimiento del estándar del software. Mientras que los otros tipos de pruebas se centran en determinar si el código funciona apropiadamente, estas pruebas intentan determinar si el código se adhiere a los estándares de codificación del proyecto.

   - **Método de Prueba:** Kitware dispone de un simple framework en el que se agregan las pruebas a los proyectos y cuenta con un sencillo mecanismo para ejecutar estas pruebas y ver sus resultados. Para proyectos pequeños o de corta vida, las pruebas a menudo están limitadas a las pruebas básicas de compilación. Para proyectos más grandes, se invierte más esfuerzo en probar y crear pruebas de regresión específicas para ejecutar funciones claves.

     Una de las herramientas para realizar las pruebas es *CTest*, un programa de líneas de comando desarrollado y construido con *CMake*. *CTest* lee la descripción de los archivos de prueba normalmente generado por *CMake* y permite al usuario ejecutar todas o algunas de las pruebas definidas para un proyecto. Una secuencia de desarrollo normal consiste en modificar el código fuente, compilar y ejecutar *CTest* para verificar que las pruebas existentes sigan pasando y luego aplicar los cambios al control de revisiones.

El método de pruebas automatizado de Kitware ejecuta *CTest* en el cliente, pero en lugar de reportar los resultados en la línea de comandos, envía los resultados de las pruebas a *Dart*, una herramienta Web OSS de reporte de pruebas. *CTest* se ejecuta por las noches y envía los resultados de las mismas a *Dart*. *Dart* acepta entradas continuamente, y los proyectos más grandes usan esta característica para construir reportes continuamente. Para proyectos OSS que serán compilados por muchas personas es necesaria una amplia variedad de plataformas de prueba.

5. **Proceso de Liberación**
   El proceso para gestionar la liberación de una versión involucra chequear en la misma máquina tanto el árbol principal como la rama de la versión del software. Luego, el administrador de la versión periódicamente combina parches de la rama al tronco (pueden usarse herramientas como *WinMerge* para realizar esto). Los desarrolladores envían un e-mail al administrador cuando ellos desean mover cambios a la rama. En algún momento, la rama se marca de nuevo y es realizada una nueva liberación (incremental).

   Tener una rama estable del código fuente para trabajar es solo una parte del problema, se debe luego empaquetar el software para liberarlo. Existen dos tipos de liberación: fuente y binarios. En Kitware la liberación se realiza en las tres principales plataformas: Windows, Unix y Mac. Cada una tiene una forma diferente de empaquetar las fuentes y los binarios. Kitware está desarrollando una nueva herramienta llamada *CPack* que abstrae la creación de los diferentes tipos de distribuciones binarias.

C.12. Proceso de Desarrollo del Servidor Apache

Aunque no hay un único proceso de desarrollo, cada desarrollador de Apache itera a lo largo de una serie común de acciones mientras trabaja en el código fuente del software [125]. Las acciones incluyen:

1. Descubrir la existencia de un problema.
2. Determinar si un voluntario trabajará en un problema determinado.
3. Identificar una solución.
4. Desarrollar y probar el código fuente.
5. Entregar parche a CVS.
6. Revisar los parches entregados.
7. Liberar la versión.

Dependiendo del alcance del cambio, el proceso anterior puede involucrar muchas iteraciones antes de concluir, aunque se prefiere generalmente que el conjunto completo de cambios necesarios para resolver un problema particular sea aplicado en una sola entrega. A continuación, se describirán cada una de las acciones anteriores.

1. **Descubrir la Existencia de un Problema**
   Existen muchas formas para descubrir problemas. Los problemas son reportados en la lista de correos de desarrollo, en la herramienta de reporte de problemas BUGDB y en el grupo de noticias de Apache USENET. Los problemas en la lista de correos tienen la prioridad más alta. Puesto que quien realizó el reporte, probablemente es un miembro de la comunidad de desarrollo, el reporte generalmente contiene información suficiente para analizar el problema. Estos mensajes reciben la atención de todos los

desarrolladores activos del proyecto. Los problemas mecánicos comunes, tales como problemas de compilación o de estructura, son normalmente encontrados primero por uno de los desarrolladores principales y son o bien corregidos inmediatamente o bien reportados y gestionados en las listas de correo. Para mantener la traza del estado del proyecto, una agenda es almacenada en cada uno de los repositorios del producto. Esta agenda contiene una lista de problemas con prioridad alta, problemas abiertos y los planes de liberación de versiones.

La segunda área para descubrir problemas está en el BUGDB del proyecto, que permite a cualquiera con acceso a la Web o e-mail entrar y categorizar los problemas por severidad y tema. Una vez registrado el problema, es colocado en una lista de correos separada y anexado a un e-mail, o editado directamente por los desarrolladores principales. Desafortunadamente, debido a algunas características molestas de la herramienta BUGDB, muy pocos desarrolladores mantienen pendientes de los problemas reportados allí. El proyecto se apoya en uno o dos desarrolladores interesados en desarrollar un seguimiento periódico de los nuevos reportes: eliminando reportes erróneos o respondiendo reportes que pueden ser contestados rápidamente y reenviando los problemas a la lista de correos de desarrollo si ellos son considerados críticos. Cuando un problema en cualquier código fuente es solucionado, se busca en BUGDB los reportes asociados con este problema para que puedan ser incluidos en el reporte de cambios y ser cerrados.

Otra fuente para descubrir problemas está en el grupo de noticias de USENET. Sin embargo, el nivel de ruido percibido en estos grupos es tan alto que solamente unos pocos desarrolladores de Apache tienen tiempo para leer las noticias. En general, el Grupo Apache (GA) confía en que la comunidad y los voluntarios interesados reconozcan cuando un problema real es descubierto y se tomen el tiempo necesario para reportarlo en BUGDB o reenviarlo directamente a la lista de correos de desarrollo. En general, solamente los problemas reportados en versiones liberadas del servidor Apache son registrados en BUGDB.

2. **Determinar si un Voluntario Trabajará en un Problema Determinado**
   Una vez un problema ha sido descubierto, el siguiente paso es encontrar un voluntario que trabaje en este problema. Los desarrolladores intentan trabajar en problemas que son identificados en áreas que son familiares para ellos. Algunos trabajan en los servicios principales del producto, mientras que otros trabajan en características particulares que ellos desarrollaron. La arquitectura del software Apache está diseñada para separar la funcionalidad principal del servidor de las características que están ubicadas en módulos que pueden ser selectivamente compilados y configurados. Los desarrolladores obtienen una implícita "propiedad del código" de las partes del servidor que ellos conocen. Aunque la propiedad del código no les da a los desarrolladores ningún derecho especial sobre el control de cambios, los otros desarrolladores tienen un respeto mayor por las opiniones de aquellos con experiencia en el área que están cambiando. Como resultado, los nuevos desarrolladores intentan centrarse en áreas donde el ex-desarrollador responsable ya no está interesado en trabajar o en el desarrollo de nuevas arquitecturas y características que no tienen ninguna demanda preexistente.

3. **Identificar una Solución**
   Luego de decidir trabajar sobre un problema, el siguiente paso es intentar identificar una solución. En general, la dificultad principal en esta etapa no es encontrar una solución sino decidir cuál de las diferentes soluciones es la más apropiada. Aún cuando el usuario da una solución que funciona, ésta puede tener características que son indeseables como una solución general o puede que no sea portable a otras plataformas. Cuando existen diferentes soluciones alternativas, el desarrollador usualmente reenvía las alternativas a

la lista de correos para obtener retroalimentación del resto del grupo, antes de desarrollar una solución.

4. **Desarrollar y Probar el Código Fuente**
   Una vez una solución ha sido identificada, el desarrollador realiza los cambios en una copia local del código fuente, prueba los mismos sobre su propio servidor y envía los cambios directamente o produce un "parche" y lo coloca en la lista de correo de desarrollo para revisión.

5. **Entregar Parche a CVS**
   Si es aprobado, el parche puede ser enviado al código fuente por cualquiera de los desarrolladores, aunque en muchos casos es preferible que quien originó el cambio también desarrolle la entrega.

6. **Revisar los Parches Entregados**
   Cada entrega realizada a CVS genera un resumen de los cambios. Estos cambios son colocados automáticamente en la lista de correos de CVS-Apache e incluyen el log del envío y un parche con los cambios. Todos los desarrolladores principales son responsables de revisar la lista de correos del CVS-Apache para asegurar que los cambios son apropiados. Además, puesto que cualquiera puede suscribirse a la lista de correos, los cambios son revisados por muchas personas aparte de la comunidad de desarrolladores principales, lo que a menudo resulta en una útil retroalimentación antes de que el software sea formalmente liberado como un paquete.

7. **Liberar la Versión**
   Cuando el proyecto está cerca de liberar una versión del producto, uno de los desarrolladores principales voluntariamente es el administrador de la versión, y el responsable de identificar los problemas críticos, determinar cuando estos problemas han sido solucionados y el software ha alcanzado un punto estable, y controlar el acceso al repositorio para que los desarrolladores inadvertidamente no cambien cosas que deberían no ser cambiadas justo antes de la liberación. El administrador de la versión genera un efecto que obliga a que muchos de los reportes de problemas pendientes sean identificados y cerrados. En esencia, esto equivale a "agitar el árbol antes de recoger las hojas". El rol del administrador de versiones es rotado entre los desarrolladores principales con más experiencia dentro del proyecto.

C.13. Proceso de Desarrollo de Apache

El proceso de desarrollo de Apache es el resultado, tanto de la naturaleza del proyecto como del *background* de los líderes del proyecto [126]. A continuación, se describirán cada una de las actividades del proceso de desarrollo de Apache.

1. **Definir Roles y Responsabilidades**
   El AG, la organización informal de personas principales responsable de guiar el desarrollo del proyecto Apache, consiste completamente de voluntarios, cada uno de los cuales tiene por lo menos otro trabajo "real". Por esta razón, ninguno de los desarrolladores puede dedicar grandes cantidades de tiempo al proyecto de una forma planificada o consistente, por tanto se requiere de un proceso de toma de decisiones y de desarrollo que haga énfasis en espacios de trabajo descentralizados y con comunicación asíncrona. AG usa listas de correo exclusivamente para comunicarse con los demás, y un sistema de votos de quórum para resolver los conflictos.

   La selección y roles de los desarrolladores principales es descrita en [72]. Los miembros de AG son personas que han contribuido durante un gran periodo de tiempo, usualmente más de 6 meses y son nominados por los miembros del equipo y luego votados por los

miembros existentes. AG inició con 8 miembros (los fundadores), tuvo 12 durante la mayor parte del tiempo y ahora tiene 25.

Cada miembro de AG puede votar sobre la inclusión de cualquier cambio en el código, y tiene acceso al CVS para hacer entregas (si ellos lo desean). Los votos son generalmente reservados para los cambios principales que puedan afectar a otros desarrolladores que estén adicionando o cambiando alguna funcionalidad.

2. **Identificar el Trabajo a Realizar**
Hay muchas formas a través de las cuales la comunidad del proyecto Apache puede reportar problemas y proponer mejoras:

   a) **Peticiones de Cambios:** Las peticiones de cambios son reportados en la lista de correos de los desarrolladores, el sistema de reporte de problemas BUGDB y los grupos de noticias USENET asociados con los productos de Apache. La lista de discusión de los desarrolladores es donde son discutidas las nuevas características y los parches de los errores. Las peticiones de cambios en las listas de correo tienen la prioridad más alta. Puesto que quien realiza el reporte probablemente es un miembro de la comunidad de desarrollo, entonces el reporte es más probable que contenga la información suficiente para analizar la petición o que contenga un parche que resuelva el problema. Estos mensajes reciben la atención de todos los desarrolladores activos. Los problemas mecánicos comunes, tales como problemas de compilación o de construcción, son encontrados normalmente primero por uno de los desarrolladores principales y son o bien corregidos inmediatamente o bien reportados y gestionados en la lista de correos.

   b) **Reporte de Problemas:** La segunda área para reportar problemas o solicitar mejoras está en la herramienta BUGDB del proyecto, que permite a cualquiera con acceso a la Web o e-mail entrar y categorizar las solicitudes por severidad y tema. Una vez registrada la solicitud es colocada en una lista de correos separada y anexada a un e-mail o editada directamente por los desarrolladores principales. Desafortunadamente, debido a algunas características molestas de la herramienta BUGDB, muy pocos desarrolladores mantienen pendientes de las solicitudes reportadas en tal herramienta. El proyecto se apoya en uno o dos desarrolladores interesados en realizar un seguimiento periódico de los nuevos reportes: eliminando reportes erróneos o respondiendo reportes que pueden ser contestados rápidamente y remitiendo solicitudes a la lista de correos de desarrollo si ellos las consideran críticas. Cuando un problema en cualquier código fuente es solucionado, se busca en BUGDB los reportes asociados con este problema para que puedan ser incluidos en el reporte de cambios y ser cerrados.

   c) **Consultar Grupo de Noticias:** Otra fuente para reportar problemas y solucionar mejoras está en el grupo de noticias de USENET. Sin embargo, el nivel de ruido percibido en estos grupos es tan alto que solamente unos pocos desarrolladores de Apache tienen tiempo para leer las noticias. En general, AG confía en que los voluntarios interesados reconozcan las mejoras prometedoras y los problemas reales y se tomen el tiempo necesario para reportarlo en BUGDB o reenviarlo directamente a la lista de correos de desarrollo. En general, solamente los problemas reportados en versiones liberadas del servidor Apache son registrados en BUGDB.

Para que un cambio propuesto realmente sea hecho, un miembro de AG debe finalmente ser convencido de si tal cambio es necesario o deseable. Se trata de problemas que son lo suficientemente serios (a la vista de una mayoría de miembros de AG) para que una versión no pueda seguir adelante hasta que sean resueltos. Otros cambios propuestos son discutidos en la lista de correos de desarrollo y si un miembro de AG está convencido de que es importante, se hará un esfuerzo para realizar el trabajo.

3. **Asignar y Realizar el Trabajo de Desarrollo**
   Una vez un problema o mejora ha encontrado la ayuda del AG, el siguiente paso es encontrar un voluntario que trabaje en este problema. Los desarrolladores principales intentan trabajar en problemas que son identificados en áreas que le son familiares. Algunos trabajan en los principales servicios del producto, mientras que otros trabajan en características particulares que ellos desarrollaron. La arquitectura del software Apache está diseñada para separar la funcionalidad principal del servidor de las características que están ubicadas en módulos que pueden ser selectivamente compilados y configurados. Los desarrolladores principales obtienen una implícita "propiedad del código" de las partes del servidor que conocen. Aunque la propiedad del código no les da a ellos derechos especiales sobre el control de cambios, los otros desarrolladores tienen un mayor respeto por las opiniones de aquellos con experiencia en el área que están cambiando. Como resultado, los nuevos desarrolladores intentan centrarse en áreas donde el ex-desarrollador responsable ya no está interesado en trabajar o en el desarrollo de nuevas arquitecturas y características que no tienen ninguna demanda preexistente.

4. **Identificar una Solución**
   Luego de decidir trabajar sobre un problema, el siguiente paso es intentar identificar una solución. En general, la dificultad principal en esta etapa no es encontrar una solución sino en decidir cuál de las diferentes soluciones es la más apropiada. Aún cuando el usuario da una solución que funciona, ésta puede tener características que son indeseables como una solución general o puede que no sea portable a otras plataformas. Cuando existen diferentes soluciones alternativas, el desarrollador usualmente reenvía las alternativas a la lista de correos para obtener retroalimentación del resto del grupo, antes de desarrollar una solución.

5. **Realizar Pruebas Pre-Liberación**
   Una vez una solución ha sido identificada, los desarrolladores hacen los cambios en una copia local de código fuente y prueban estos cambios sobre su propio servidor. Este nivel de pruebas es más o menos comparable a las pruebas de unidad, aunque la minuciosidad de las pruebas depende del juicio y pericia del desarrollador. No hay pruebas adicionales (por ejemplo, regresión, pruebas de sistema) antes de la liberación de una versión, aunque se requiere de una revisión antes o después de entregar los cambios.

6. **Inspecciones**
   Luego de las pruebas de unidad, los desarrolladores principales o entregan los cambios directamente o producen un parche y lo colocan en la lista de correos de los desarrolladores para revisión. En general, los cambios para una versión estable requieren de revisión antes de ser entregados, mientras que los cambios de las entregas de desarrollo son revisadas luego de que el cambio sea entregado. Si es aprobado, el parche puede ser entregado al código fuente por cualquiera de los desarrolladores, aunque en muchos casos se prefiere que el originador del cambio también realice la entrega.

   Cada entrega realizada a CVS genera un resumen de los cambios. Estos cambios son colocados automáticamente en la lista de correos de CVS-Apache e incluyen el log del envío y un parche con los cambios. Todos los desarrolladores principales son responsables de revisar la lista de correos del CVS-Apache para asegurar que los cambios son apropiados. De hecho, muchos desarrolladores revisan todos los cambios. Además, puesto que cualquiera puede suscribirse a la lista de correos, los cambios son revisados por muchas personas aparte de la comunidad de desarrolladores principales, lo que a menudo resulta en una útil retroalimentación antes de que el software sea formalmente liberado como un paquete.

7. **Administrar las Versiones**
   Cuando el proyecto está cerca de liberar una versión del producto, uno de los desarrolladores principal voluntariamente es el administrador de la versión, y el responsable de identificar los problemas críticos, determinar cuando estos problemas han sido solucionados y el software ha alcanzado un punto estable, y controlar el acceso al repositorio para que los desarrolladores inadvertidamente no cambien cosas que deberían no ser cambiadas justo antes de la liberación. El administrador de la versión genera un efecto que obliga a que muchos de los reportes de problemas pendientes sean identificados y cerrados. En esencia, esto equivale a "agitar el árbol antes de recoger las hojas". El rol del administrador de versiones es rotado entre los desarrolladores principales con más experiencia dentro del proyecto.

C.14. Proceso de Desarrollo de Mozilla

Inicialmente, el número de voluntarios que participaban en el proyecto Mozilla no alcanzó el nivel esperado. Pero hubo un importante grupo de voluntarios que aportó contribuciones importantes al proyecto desde mucho antes de que el código estuviera listo para usarse. Luego de un año, uno de los líderes del proyecto abandonó por la falta de interés externo, la arquitectura engorrosa, la ausencia de un producto funcional y la falta de un soporte por parte de Netscape. Sin embargo, luego de que la documentación fue mejorada, los tutoriales fueron escritos y las herramientas de desarrollo y los procesos fueron refinados, la participación empezó a incrementarse. Además, el hecho de que las herramientas de desarrollo fueran exportadas para ser usadas en proyectos de software comerciales como HP, Oracle y Sun Microsystems [200], evidencia su alta calidad y escalabilidad. Mozilla desarrolló una sustancial documentación sobre la arquitectura, las tecnologías usadas y las instrucciones para la construcción y las pruebas. A continuación, se describirán cada una de las actividades del proceso de desarrollo de Mozilla [126].

1. **Definir Roles y Responsabilidades**
   El proyecto Mozilla es gestionado actualmente por el Grupo Mozilla.org que coordina y guía el proyecto, suministra procesos y realizan alguna codificación. Solo cerca de 4 de los miembros principales ha invertido una parte importante de su tiempo escribiendo código para la aplicación. Otros tienen roles dedicados a aspectos tales como aseguramiento de calidad, liberación de versiones por hitos, herramientas y mantenimiento del sitio Web, y herramientas como Bugzilla que ayudan a los desarrolladores. Aunque la participación externa (más allá de Netscape) se ha incrementado con los años, aún algunas personas externas (por ejemplo, de Sun Microsystems) trabajan tiempo completo por un salario en el proyecto.

   La autoridad para la toma de decisiones de varios módulos es delegada a los individuos en la comunidad de desarrollo más cercanos al código fuente correspondiente. Las personas con un historial comprobado de buena calidad en su código fuente pueden intentar obtener acceso al repositorio CVS para realizar entregas. Los directorios y archivos de un módulo particular pueden ser adicionados o cambiados obteniendo el permiso del propietario del módulo. Adicionar un nuevo módulo requiere los permisos de Mozilla.org. Si bien gran parte de la responsabilidad es delegada a través del acceso de entregas distribuidas y la propiedad del módulo, Mozilla.org tiene la autoridad para tomar decisiones en última instancia y se reserva el derecho de designar y retirar los propietarios de los módulos, y resolver todos los conflictos que surjan.

2. **Identificar el Trabajo a Realizar**
   Mozilla.org mantiene un documento de roadmap [59] que especifica qué será incluido en futuras versiones, así como las fechas para la liberación de las versiones. Mozilla.org determina el contenido y el calendario, pero realiza considerables esfuerzos para asegurar que la comunidad de desarrollo sea capaz de comentar y participar en estas decisiones.

   Cualquiera puede reportar errores, o pedir mejoras, en la misma forma. El reporte de errores y el proceso de petición de mejoras se realiza a través de la herramienta de reporte de problemas Bugzilla y exige a quien realiza la petición crear una cuenta en el sistema. Bugzilla también tiene herramientas que permite a quien reporta los errores ver los errores más recientes y si se desea buscar en la base de datos completa de reporte de errores. Bugzilla suministra una forma detallada para reportar los problemas o describir las mejoras deseadas.

3. **Asignar y Realizar el Trabajo de Desarrollo**
   Los miembros de Mozilla.org quienes escriben el código del navegador Web se centran en las áreas donde ellos tienen experiencia y donde el trabajo es más necesario para soportar la próxima versión. La comunidad de desarrollo puede explorar Bugzilla para identificar los errores o las mejoras que quieren realizar. Las correcciones son a menudo enviadas como anexos al reporte de problemas de Bugzilla. Los desarrolladores pueden marcar ítems de Bugzilla con la palabra clave *"helpwanted"* si consideran que un ítem es digno de hacer pero no tienen los recursos o toda la experiencia necesaria. Las discusiones pueden también ser encontradas en los grupos de noticias de Mozilla, que pueden dar a los miembros de la comunidad de desarrollo ideas sobre dónde contribuir. Los miembros de Mozilla.org pueden usar las páginas Web de Mozilla para destacar áreas específicas donde se necesita ayuda. Cuando trabajan en un ítem particular de Bugzilla, los desarrolladores son animados para registrar este hecho en Bugzilla con el fin de evitar la duplicación de esfuerzos.

4. **Realizar Pruebas Pre-Liberación**
   Mozilla.org desarrolla una compilación diaria y ejecuta como mínimo una *"smoke test"* diariamente sobre las principales plataformas, con el objetivo de asegurar que la compilación sean lo suficientemente estable para permitir que el trabajo de desarrollo siga adelante. Si *"smoke test"* identifica errores, estos se publican diariamente para que los desarrolladores estén al tanto de cualquier problema serio.

   Mozilla actualmente tiene 6 equipos de pruebas en el área de producto que tienen la responsabilidad de probar varias partes o aspectos tales como cumplimiento de estándares, noticias/correos de clientes, e internacionalización. El personal de Netscape está fuertemente representado entre los equipos de prueba, pero los equipos también incluyen personal de Mozilla.org, y muchos otros. Los equipos de pruebas mantienen casos y planes de pruebas, así como otros materiales tales como directrices para verificar errores y guías para solucionar problemas.

5. **Inspecciones**
   Mozilla realiza inspecciones de código en dos etapas. En la primera, los propietarios del módulo revisan el parche en el contexto del módulo. En la segunda, un grupo revisa el parche para su interacción con el código base como un todo antes de que sea entregado.

6. **Administrar las Versiones**
   Mozilla ejecuta un proceso de compilación continuo (Tinderbox) que muestra qué partes del código tienen problemas bajo ciertas plataformas. El proceso también genera los ficheros binarios por las noches y los problemas "hitos", aproximadamente cada mes. La liberación de "hitos" de versiones involucra más que Tinderbox. Estas liberaciones involucran decisiones del administrador del proyecto, usualmente un congelamiento de

código por unos pocos días, una rama "hito", eliminar errores críticos y un mantenimiento perfectivo. La decisión de cuando una rama está lista para ser liberada como un "hito" es un proceso humano, no un proceso automatizado de Tinderbox. Estas decisiones de "hitos" son realizadas por un grupo designado conocido como *"drivers@mozilla.org"*.

C.15. Proceso de Desarrollo de Debian

El proceso de desarrollo de software del proyecto Debian se lleva a cabo dentro de una comunidad cooperativa de voluntarios que comparten un objetivo común. Las características de esta comunidad son: las personas se unen a la comunidad de forma voluntaria, y no esperan que se les pague por su trabajo; los miembros de la comunidad están de acuerdo en un objetivo final ambicioso; los miembros comparten una conciencia civil y aceptan que su trabajo esté regulado por reglas explícitas establecidas por una democracia directa. A continuación, se describen las actividades del proceso de desarrollo de Debian [127].

1. **Estructura de Debian**
   Cualquiera puede ser miembro del proyecto Debian. Para ser aceptado en el proyecto se tiene que demostrar el control de las habilidades básicas necesarias para gestionar paquetes de software y entender "las Pautas del Software Libre de Debian" y el "Contrato Social de Debian".

   Al unirse al grupo, se da el consentimiento de contribuir al proyecto de acuerdo a la Constitución de Debian [1]. La Constitución define una organización ágil, con un Líder de Proyecto (DL), un Secretario de Proyecto (DS), un Comité Técnico (TC) y Desarrolladores Individuales. El DL, DS y el Presidente del TC son tres personas diferentes. El trabajo es completamente voluntario: nadie está obligado a hacer algo y cada uno escoge libremente realizar la tarea que encuentre útil o interesante. Un nuevo DL es nombrado cada año por unas elecciones generales donde participan todos los desarrolladores individuales. El DL puede tomar decisiones urgentes y nombrar el DS y junto con el TC renueva los miembros del TC. El TC está compuesto por hasta 8 miembros, con un mínimo de 4 personas. El TC decide las políticas técnicas y soluciona los desacuerdos entre los desarrolladores. Los desarrolladores individuales pueden anular cualquier decisión de DL y TC mediante la emisión de una resolución general con una mayoría cualificada. El DS es nombrado cada año por el DL y el DS anterior. El DS está encargado de la gestión de las elecciones y de otras convocatorias de votación y resuelve cualquier disputa sobre la interpretación de la Constitución. Las propiedades y las actividades financieras son gestionadas por el "Software in the Public Interest" (SPI), en la que cada miembro de Debian puede ser un miembro votante.

   La consecuencia de esta estructura organizacional es que ningún individuo puede tomar el control personal de proyecto. Aún mejor, "cualquier Desarrollador Individual puede tomar cualquier decisión técnica o no técnica en relación con su propio trabajo" [1].

2. **Distribuciones Debian**
   Una distribución del sistema Debian está compuesta por un programa de instalación y un conjunto de paquetes de software.

   a) **Programa de Instalación:** El programa de instalación es capaz de configurar el sistema desde cero en un gran número de diferentes configuraciones de hardware. Los paquetes de software se pueden recuperar de un grupo de CD's, un disco duro local de la red.

b) **Producción de Paquetes:** Todo el esfuerzo de desarrollo de Debian está centrado en la producción de paquetes. Un paquete es la mínima unidad que puede ser instalada o eliminada de un sistema. En consecuencia, cada Desarrollador de Debian (DD) es responsable de uno o más paquetes y es quien realiza su mantenimiento. Cuando un desarrollador ha creado su paquete, lo carga en un repositorio público desde donde los usuarios de Debian de todo el mundo intentan instalarlo en sus sistemas. Dado que hasta ahora el paquete ha sido probado solamente en la máquina de los DD's, su estado debe ser considerado como *alpha-testing* y el repositorio es llamado *distribución inestable*. Si un paquete reside en la distribución inestable durante diez días sin ningún tipo de error crítico, este paquete es automáticamente cargado a otro repositorio más estable, cuyo estado es beta-testing. Este repositorio es conocido como la distribución de prueba. Aproximadamente cada año, es nombrado por el DL un administrador de versiones y se congela un conjunto de paquetes de la distribución de pruebas. Esto significa que ningún nuevo paquete puede ser adicionado al conjunto, incluyendo solo los paquetes que evolucionan por corrección de errores, y eventualmente, cuando todos los errores críticos de la versión son corregidos, una nueva distribución estable es liberada al público. La distribución estable es la que usualmente es considerada la distribución oficial de Debian e incluye paquetes que son actualizados solamente para corregir vulnerabilidades de seguridad.

3. **Coordinación**
   En el proceso de desarrollo de Debian la coordinación del proyecto tiene dos componentes clave. En primer lugar, la distribución de los programas que componen el proyecto. En segundo lugar, las políticas de desarrollo que definen cómo se realizará tal distribución. A continuación, se describen cada uno de estos componentes.

   a) **Distribución:** El objetivo de obtener una distribución coherente donde todos los programas pueden interactuar sin problemas es muy complejo. El problema parece sin solución si la distribución se obtiene mediante la agregación de miles de paquetes producidos por cientos de desarrolladores en docenas de configuraciones de sistemas diferentes. Sin embargo, los sistemas Debian fueron capaces de obtener una satisfacción general del usuario bastante buena, como demuestran los diferentes premios que ganó en el año 2003. De hecho, el principal esfuerzo realizado por DD está enfocado en garantizar que sus paquetes sean totalmente compatibles con las políticas de Debian.

   b) **Políticas de Desarrollo:** Las políticas son fundamentales en el enfoque de Debian para la distribución del software. La libertad de los DD's es ilimitada, siempre y cuando cumplan con las políticas acordadas colectivamente. Las políticas a menudo están basadas en estándares internacionales o de la comunidad (por ejemplo, el Estándar de Jerarquía de Archivos del Sistema [10]) y se refieren a todas las cuestiones globales que afectan la coherencia de un sistema, es decir, despliegue de bibliotecas, variables de ambiente, servicios compartidos, lenguajes de scripting.
   La aplicación de las políticas se lleva a cabo en diferentes niveles a fin de aprovechar la validación cruzada para reducir al mínimo los paquetes inconsistentes:
   - *Durante el ensamblaje del paquete*: La mayor parte de políticas están asociadas a una herramienta (llamada colectivamente *"debhelpers"*) que asegura la correcta aplicación.
   - *Durante las pruebas del paquete*: Existen varias herramientas para comprobar el cumplimiento de las políticas antes de cargar el paquete en el repositorio público.
   - *Durante la implementación del paquete*: Todos los usuarios que detectan una incoherencia pueden reportar un error con un procedimiento automatizado.

C.16. Proceso de Desarrollo Bazar

Raymond, en su trabajo de investigación [155], describe el modelo de proceso de desarrollo de un proyecto OSS a partir de sus experiencias en la participación del proyecto *Fetchmail* e inspirado en el proceso de desarrollo de Linux. Este modelo de proceso lo nombró "Bazar" y lo contrapuso al modelo de procesos de la IS tradicional al que etiquetó como "Catedral". Raymond se limita a describir este modelo mediante una serie de cláusulas sencillas, en las que no incluye ninguna metodología específica ni rigurosa, ya que esto iría en contra del modelo de Bazar mismo. A partir de estas cláusulas [155], se han obtenido las actividades del proceso de desarrollo OSS que son descritas a continuación.

1. **Surge una Necesidad o una Idea**: A partir de una idea o de la necesidad personal de un programador, surge el desarrollo de un producto software. En este desarrollo inicialmente están involucrados una o dos personas.

2. **Obtener Diseño Inicial**: De ser posible, obtener el diseño inicial del producto software de uno similar ya construido.

3. **Implementar Primera Versión**: Implementar la primera versión del producto software, partiendo del diseño inicial.

4. **Liberar Rápida y Frecuentemente**: En las primeras versiones del producto software, es normal liberar una nueva versión diariamente.

5. **Ampliar Lista de Voluntarios**: Ampliar la lista de voluntarios, incorporando a todos aquellos que contacten para conocer más sobre el producto software que se está desarrollando.

6. **Anunciar la Nueva Versión Liberada**: Anunciar a todos los voluntarios cada vez que se libera una nueva versión, animando a las personas a participar. Para poder construir una comunidad de desarrollo se necesita atraer gente, interesarla en lo que se está haciendo y mantenerla a gusto con el trabajo que se está desarrollando.

7. **Los Usuarios Usan el Software**: Un grupo de usuarios utiliza el software y algunos se adhieren a la comunidad de desarrollo. Una comunidad grande y diversa es muy importante para la evolución de un producto software así como también para su depuración.

8. **Los Usuarios Reportan Errores**: Los usuarios proporcionan retroalimentación a los desarrolladores y notifican los errores encontrados, solicitando además el desarrollo de nuevas funcionalidades.

9. **Generar una Nueva Versión**: Algunos usuarios deciden contribuir diseñando o desarrollando las funcionalidades que más les interesan.

10. **Los Usuarios Contribuyen con Diseños o Desarrollando**: Generar una nueva distribución del producto software con errores corregidos, nuevas funcionalidades y un diseño evolucionado, que se obtiene cuando el código va mejorando y se va simplificando. Un diseño perfecto se alcanza no cuando ya no hay nada que agregar, sino cuando ya no hay algo que quitar.

11. Finalmente, volver a la actividad 4.

C.17. Proceso de Desarrollo del Navegador Web Mozilla

El proyecto Mozilla tiene un extenso proceso de desarrollo de software, que abarca muchos aspectos diferentes del desarrollo. En el trabajo de investigación [164] se discuten los aspectos más relevantes en este proceso de desarrollo. A partir de estos aspectos, se han obtenido las actividades del proceso de desarrollo de Mozilla, que se discuten a continuación.

1. **Elaborar los Requisitos**
   A menudo, uno de los aspectos más controvertidos de los proyectos OSS [90][122] es el proceso de requisitos, que en el proyecto Mozilla también es algo peculiar. Los requisitos de alto nivel son dados por el administrador de mozilla.org, pero ya que estos son pocos y muy abstractos, muchas de las decisiones sobre la inclusión y cambios de funcionalidades son discutidas gradualmente por la comunidad y por los propietarios de los módulos, a través de los comentarios de los grupos de noticias y los reportes de error. Aunque algunas áreas, tales como la interfaz de usuario y las APIs, están razonablemente documentadas y detalladas [5][32], muchas áreas evolucionan poco a poco.

   Una modificación en los requisitos comienza como lo hacen los cambios propuestos del código fuente. La discusión sobre la relevancia del cambio es iniciada a través de una de las siguientes rutas:

   - Una cadena de mensajes se inicia en un grupo de noticias público, con respecto a un cambio en la funcionalidad. Otras personas usualmente comentan su relevancia y discuten las ventajas y desventajas del tema en cuestión. La esencia del debate es a menudo bastante técnica y generalmente culmina en un error archivado sobre el cambio o la idea es abandonada.
   - Cuando una persona tiene una idea específica para un cambio de un error con frecuencia se presentará directamente sin la discusión del grupo de noticias. La discusión ocurre a menudo en el error en sí, y el sistema de votación, los miembros de la comunidad, el propietario del módulo, y en última instancia el personal de mozilla.org ayudará a determinar si se trata de un cambio relevante o no.

   Una vez que un cambio se ha decidido, no hay garantía de que será implementado pronto (o nunca). Corresponde a la comunidad y a la gestión de mozilla.org ver que los desarrolladores realicen un seguimiento de la tarea y propongan los cambios de código fuente que implementarán la funcionalidad solicitada. Aún cuando un cambio es implementado, hay una remota posibilidad de que no sea aceptado en el código base.

   Es difícil decir que el proceso de requisitos es generalmente inadecuado, pues la comunidad tiene una activa participación en la adopción de las características propuestas, y cualquiera es libre de implementar un cambio deseado y enviarlo para su aprobación. El hecho de que los propietarios de los módulos y el grupo de mozilla.org sean las autoridades finales que determinan la relevancia de un cambio, sí presenta una barrera para la adopción de características controvertidas, pero esto crea precedentes y parece una consecuencia del nivel de control que requiere el proceso de desarrollo de Mozilla.

2. **Diseñar**
   El actual proceso de diseño de la arquitectura de software de Mozilla es difícil de abstraer debido a dos problemas importantes. Primero, el diseño heredado en parte de la experiencia del Netscape original, por lo que no fue completamente desarrollado a la vista del público. Segundo, porque las discusiones de diseño son inherentemente difíciles de capturar [83] y usualmente sus registros son escasos.

El diseño y las especificaciones tanto de la arquitectura original [4] como de la nueva arquitectura [3][14] están documentados y existen esfuerzos de extender la documentación del diseño y del código a través de un análisis y un proceso automatizado.

Como se refleja en el proceso de requisitos *ad-hoc*, muchos problemas de diseño son abordados cuando llegan: los errores son archivados para cambiar el diseño o el API de un cierto componente o clase, y la discusión se sigue sobre los comentarios de estos errores. La posibilidad constante de cambios en el diseño requiere que los propietarios de los módulos y los revisores superiores tengan un buen conocimiento del diseño actual. Ambos son los responsables finales de juzgar y autorizar que el diseño evolucione en una cierta dirección.

Un problema de estos constantes cambios producidos es la pesada tarea de mantener la documentación al día, y los documentos en mozilla.org están considerablemente desactualizados.

Debido al "ajuste continuo del diseño", hay una constante necesidad de refactorizar el código legado. La refactorización permite a los desarrolladores simplificar las APIs, remover el código que no se usa, y generalmente mejorar la modularidad y legibilidad del código.

3. **Desarrollar de Forma Distribuida**
Una de las premisas del proyecto Mozilla estuvo basada en que la comunicación cara a cara podría no ser requerida para el desarrollo, que es estrictamente la regla para muchos, sino todos, los proyectos OSS [90][203]. De este modo, todo el código debería ser diseñado, implementado, probado e integrado sin contar con el contacto personal para resolver los problemas. Esto plantea muchas situaciones difíciles y requiere de herramientas de soporte y planificación [91], pero en el proyecto este objetivo es actualmente posible.

   Todos los desarrolladores trabajan usando un sistema de control de versiones (CVS) sobre un código base común y centralizado, que permite que los cambios sean desarrollados concurrente e independientemente. Existe una única imagen del código, y en todo momento cualquier desarrollador puede fácilmente recuperar la información de cual es la última versión del código fuente de Mozilla. Esto asegura, que todos los desarrolladores vean todos los cambios realizados por los demás, y que las pruebas de regresión puedan ser realizadas sobre el último código integrado.

   El hecho de que los desarrolladores raramente se reúnan personalmente, tiene algunas consecuencias que son a menudo pasadas por alto:

   - Los participantes son forzados a comunicarse claramente, escribiendo en inglés, usando e-mail, chats, grupos de noticias y comentarios en los reportes de errores.
   - La carencia de documentación es una barrera importante para que se involucre cualquier participante novato, pero el hecho de que está disponible el código fuente compensa esto un poco.
   - La comunicación en tiempo real suministra un mecanismo importante para educar a los desarrolladores y clarificar el código. Debido a que muchos de los líderes de desarrollo están a menudo disponibles y en línea, preguntas y revisiones de diseño informales pueden ser rápidamente realizadas.
   - La carencia de herramientas para facilitar la comunicación y la visibilidad en el proceso puede obstaculizar seriamente el progreso del proyecto, por lo que su disponibilidad, calidad y usabilidad son de gran importancia.

4. **Realizar Revisiones Formales**
   Quizás una de las más sorprendentes características del proceso de desarrollo de Mozilla es el estricto sistema de aprobación y revisión de código al que están sujetos todos los cambios antes de su integración. Mientras otros proyectos incluyen la revisión como una de sus actividades base, el proyecto Mozilla es uno de los primeros en implementar sistemáticamente herramientas que soporten la revisión formal. La revisión de código fue tempranamente instituida en Mozilla como una forma de evitar tener que integrar código incorrecto en el repositorio, que era una de las principales causas de retraso. Luego se realizaron excelentes revisiones que involucraron ingenieros senior muy familiarizados con el código [60], para dar soporte a las evaluaciones de diseño más críticas. El proceso de revisión es el siguiente:

   - Un desarrollador que está trabajando en un cambio para un error produce un parche, que es un archivo de texto que describe línea por línea las diferencias entre la versión local del desarrollador y la última versión en el repositorio del código fuente.
   - Este parche es entonces anexado al error en el sistema Bugzilla, y el desarrollador solicita revisión.
   - Un revisor, que puede ser el propietario de un módulo o cualquier otro familiarizado con el mismo, leerá críticamente el código y aceptará la revisión o solicitará cambios.

   La revisión consiste en realizar comentarios al error. Estos comentarios describen los cambios que podrían ser realizados para mejorar la calidad del código, preguntas acerca de una sección no clara, o recomendaciones sobre otros aspectos del parche (impacto en el desempeño, dependencias, problemas relacionados). Si no hay más problemas, el revisor pondrá su sello de aprobado adicionando "r= (nombre revisor)" a un comentario del error.

5. **Aseguramiento de Calidad y Seguimiento de Errores**
   Aunque muchos proyectos incluyen en su proceso procedimientos de Aseguramiento de Calidad (QA) [209], el proyecto Mozilla fue un pionero en el uso del término en sí mismo para describir un proceso explícito. El QA en Mozilla es desarrollado por diferentes clases de personas, que van desde ingenieros contratados por Netscape y otros patrocinadores comerciales a voluntarios informales dispuestos a probar y realizar seguimiento de problemas.

   El equipo QA en Mozilla tuvo la misión de "encontrar y reportar constructivamente los errores relevantes" [6]. De esta misión, queda claro que el trabajo de QA está íntimamente comprometido con probar el código y usar las herramientas de desarrollo de Mozilla. La realidad confirma esto, ya que muchas de las actividades que realiza QA se reflejan en la manera en que estas herramientas operan. La principal actividad de QA es ser responsable de las pruebas y del seguimiento de errores. Existen muchas actividades de pruebas en Mozilla:

   - Las *Smoketests* son bastantes diferentes en Mozilla como para justificar una explicación aparte. Asa Dotzler, un miembro del staff de mozilla.org encargado de QA, explica que los planes *smoketest* de Mozilla son más completos de lo que se considera generalmente un *smoketest*, pero no tan completos como una completa prueba funcional. El *Smoketesting* está vinculado íntimamente a los procesos "de construcción nocturna":

- Cada día, la última versión del explorador Mozilla es compilada en varias de las plataformas soportadas.
- La estructura de los binarios es ubicada en un servidor FTP público.
- Un grupo de testers entonces descarga estos binarios y procede a ejecutar una serie documentada de casos de prueba que es requerida para declarar que la versión del explorador es estable.
- Si las pruebas fallan, las personas responsables de los cambios que ocasionaron estos fallos son contactados para o bien corregir el error o deshacer los cambios. Esto permite al desarrollo continuar en un código base que garantiza no incluir importantes regresiones.
- El *smoketesting* es desarrollado cada día para las tres plataformas más importantes de Mozilla: Windows, Mac OS y Linux, y se considera que es una herramienta fundamental que permite el desarrollo concurrente y no ocasiona que el código falle separadamente debido a las regresiones.

- El seguimiento de errores involucra:
  - Trabajar sobre cientos de errores almacenados en Bugzilla, reproduciendo y asignando problemas.
  - Invalidar errores cuando "el error" descrito es un comportamiento intencionado, o cuando hay una carencia de pasos reproducibles.
  - Trabajar para distribuir mejor la carga de correcciones entre los desarrolladores disponibles.
  - Los voluntarios QA, junto con los propietarios de los módulos, ayudan a definir lo que será implementado en la siguiente liberación de la versión estable.
- Al invitar voluntarios a desempeñar tareas de QA, el proyecto Mozilla tuvo efectivamente la atención de una gran población de testers, quienes reportaron errores y siguieron la resolución de los mismos. Se anima a los usuarios a que reporten errores debido a la naturaleza abierta de Bugzilla, y porque ellos sienten que los desarrolladores les están prestando una atención que es fructífera.

Tener un foro público para reportar problemas y obtener retroalimentación de los desarrolladores sobre ellos mismos, es una de las ventajas más importantes identificadas en todo el proceso QA. No sólo previene que los problemas no sean ignorados u olvidados, sino que también hace viable un análisis histórico de los datos almacenados.

C.18. Proceso de Desarrollo OSS en la Comunidad de los Videojuegos

Según el trabajo de investigación de Scacchi [170], el proceso de desarrollo OSS consta de cinco actividades. Scacchi describe brevemente cada una de las actividades y afirma que estas actividades ocurren al mismo tiempo y no en un orden estricto como en el modelo del ciclo de vida tradicional o parcialmente ordenado como en un modelo de proceso en espiral. A continuación, se describen cada una de las cinco actividades de proceso de desarrollo OSS de la comunidad de los videojuegos.

1. **Análisis y Especificación de Requisitos**
   Los estudios realizados hasta la fecha aún no han encontrado registros de educción, captura, análisis y validación de requisitos, en ninguna de las cinco comunidades OSS: Videojuegos en red, infraestructura Web/Internet, bio-informática, software para la educación superior y software militar [169]. En general, estos registros no se encuentran en los sitios Web de los proyectos OSS o en documentos de "especificación de requisitos". Lo que los estudios han observado y encontrado es diferente.

Los requisitos OSS tienen la forma de mensajes hilados o de discusiones en sitios Web que están disponibles para revisión, elaboración, refutación o refinamiento por parte de cualquier miembro de la comunidad. El análisis y especificación de requisitos son actividades implícitas. Estas actividades surgen habitualmente como un subproducto del discurso de la comunidad acerca de lo que el software debe o no debe hacer y quién será el responsable de contribuir con funcionalidades nuevas o modificadas al sistema. Los requisitos aparecen como afirmaciones en foros de discusión públicos y privados, en artefactos de software *ad-hoc* (como fragmentos de código fuente incluidos en un mensaje) y en actualizaciones de contenido que surgen continuamente [169][193]. Más convencionalmente, el análisis, especificación y validación de requisitos no son realizados como una tarea necesaria que produce obligatoriamente un entregable de requisitos. En su lugar, se encuentran prácticas generalizadas que implican la lectura y *sense-making* de los contenidos disponibles en línea. Existen "webs" de discurso interrelacionado que efectivamente trazan, condensan, y toman forma de requisitos de software en retrospectiva. Al mismo tiempo, el proyecto es accesible globalmente por los participantes antiguos o nuevos del proyecto OSS.

En resumen, los requisitos toman estas formas porque los desarrolladores OSS implementan sus sistemas y luego afirman que ciertas características son necesarias. Los requisitos no se derivan de las necesidades escritas explícitamente por los usuarios representativos o por los estrategas de marketing del producto.

2. **Control de Versiones Coordinado, Construcción del Sistema y Revisión-Liberación Incremental por Etapas**

   Las herramientas de control de versiones como Concurrent Versions System (CVS) [74], son ampliamente utilizadas en las comunidades OSS. Estas herramientas sirven como un mecanismo centralizado para la coordinación del desarrollo OSS y como un lugar para controlar qué mejoras, extensiones o actualizaciones serán realizadas al software. Si han sido verificadas estas actualizaciones, estarán disponibles para la comunidad como parte de las versiones alfa, beta, candidata o versión oficial.

   El control de versiones como parte de una actividad de administración de la configuración de software es recurrente y requiere de coordinación, pero permite estabilizar y sincronizar los desarrollos dispersos. Esta coordinación es necesaria porque los contribuyentes de código y los revisores descentralizados pueden contribuir independientemente con actualizaciones o revisiones que es posible se solapen, entren en conflicto con otras o generen efectos secundarios no deseados.

   Cada equipo del proyecto o administrador del repositorio CVS debe decidir el código fuente que puede ser subido al CVS y quien puede o no hacerlo. Algunos proyectos hacen estas políticas explícitas a través de un esquema de votación [72] y en otros proyectos siguen siendo informales, implícitas y objeto de negociación con el propietario del módulo o de la versión. En cualquier caso, el equipo debe coordinar las actualizaciones de versión y la correspondiente liberación. Posteriormente, las actualizaciones que realizan los desarrolladores al repositorio compartido de la comunidad están basadas principalmente en las discusiones online en forma de mensajes en foros [203], en lugar de tener que lidiar con comités de control de configuración del sistema o con plazos de entrega arbitrarios. Por lo tanto, la utilización conjunta del control de versiones, comunicación online y herramientas de transferencia y búsqueda de archivos median en el proceso de control de versiones, construcción del sistema, liberación y revisión.

3. **Mantenimiento como Redesarrollo, Reinvención y Revitalización Evolutiva**

   El mantenimiento del software -adicionar y eliminar funcionalidades del sistema, depurar, reestructurar, ajustar, convertir (por ejemplo, la internacionalización) y migrar entre plataformas- es un proceso extenso y recurrente en las comunidades OSS. Tal vez

esto no es de extrañar, teniendo en cuenta que el mantenimiento es generalmente visto como la principal actividad asociada con un sistema software a través de su ciclo de vida. Sin embargo, la etiqueta tradicional de mantenimiento de software no encaja del todo con lo que ocurre en las diferentes comunidades OSS. En su lugar, podría ser mejor caracterizar la dinámica evolutiva general de OSS como reinvención. La reinvención se produce a través de intercambiar, examinar, modificar y redistribuir los conceptos y técnicas que han aparecido en los sistemas de código cerrado, trabajos de investigación, conferencias y libros de texto.

Los sistemas OSS parecen evolucionar a través de mejoras menores o de mutaciones que son recombinadas y redistribuidas a través de muchas versiones con ciclos de vida cortos. Los usuarios finales OSS que actúan como desarrolladores continuamente producen estas mutaciones. Las modificaciones o actualizaciones son expresadas como versiones alfa, beta o versiones finales capaces de sobrevivir a la redistribución y revisión. Luego, estas versiones pueden ser recombinadas y reexpresadas con otras mutaciones produciendo una nueva versión estable que será liberada. Como resultado, estas mutaciones articulan y adaptan un sistema OSS a lo que los usuarios-desarrolladores quieran que haga, mientras se reinventa el sistema.

Los sistemas OSS co-evolucionan con sus comunidades de desarrollo; una evolución depende de otras. Es decir, un proyecto con pocos desarrolladores no produce y mantiene un sistema viable hasta que el equipo llegue a tener una masa crítica de entre 5 y 15 desarrolladores principales. Si esto sucede, el sistema podría ser capaz de crecer en tamaño y complejidad a un ritmo exponencial sostenido, desafiando las leyes de la evolución del software que se han mantenido durante décadas [105].

4. **Gestión de Proyectos y Desarrollo Profesional**
   Los equipos de desarrollo OSS pueden tomar la forma organizacional meritocrática de capas entrelazadas que funcionan como una empresa virtual ligeramente acoplada pero organizada dinámicamente [140]. Una meritocracia en capas [72] es una forma jerárquica de organización que centraliza y concentra ciertos tipos de autoridad, confianza y respeto por la experiencia y por los logros dentro del equipo. Sin embargo, no implica una sola autoridad, porque la toma de decisiones puede ser compartida entre los desarrolladores principales que actúan como compañeros. En cambio, las meritocracias tienden a adoptar innovaciones incrementales, como mutaciones evolutivas a una base de código fuente existente, por encima de cambios radicales. Los cambios radicales consisten en la exploración o adopción de funcionalidades del sistema no probadas. Una minoría de contribuyentes de código que desafíen el estatus de los desarrolladores principales podría abogar por cambios radicales. Sin embargo, su éxito por lo general implica crear y mantener por separado una versión independiente del sistema y perder potencialmente una masa crítica de otros desarrolladores OSS. Así que las mutaciones incrementales tienden a ganar con el tiempo.

   Los participantes en el proyecto que se encuentran más arriba en la meritocracia tienen una mayor autoridad que aquellos que están más abajo. Pero estas relaciones sólo son eficaces si todos están de acuerdo en su composición y legitimidad. Administrar o coordinar los conflictos que no pueden resolverse pueden llegar a dividir o bifurcar una nueva versión del sistema. Entonces, los participantes en conflicto deben asumir la responsabilidad de mantener esa nueva versión reduciendo su participación en el proyecto en curso o simplemente cediendo la posición en conflicto.

   Los desarrolladores OSS tienen motivos complejos para estar dispuestos a distribuir su tiempo, habilidades y esfuerzo a la evolución de los sistemas OSS. Ellos simplemente pueden pensar que el trabajo es divertido, gratificante, o un medio para el ejercicio y mejora de sus competencias técnicas de una manera que no pueden llevar a cabo en sus puestos o campos de trabajo formales [87]. Algunos desarrolladores OSS crean

modificaciones de juegos de ordenador que circulan ampliamente y generan ingresos por ventas considerables a los propietarios del juego, quienes algunas veces comparten los beneficios [48]. Además, al ser un nodo central en una red de desarrolladores de software que interconectan múltiples proyectos OSS no sólo aportan capital social y reconocimiento de sus compañeros. También permite que los sistemas OSS independientes se mezclen con otros más grandes aumentando la masa crítica de desarrolladores, para crecer aún más y atraer a mayores comunidades de usuarios-desarrolladores. Así, es sorprendente que más del 60 % de los desarrolladores OSS encuestados en un estudio reciente [87] reportaron haber participado en 2 a 10 proyectos OSS. Esto no solamente interconecta proyectos de sistemas independientes en una arquitectura de sistemas más grandes, sino que también articula su meritocracia y control social. Esto permite que el sistema y la comunidad crezcan más sólidamente en conjunto.

5. **Transferencia de Tecnología y Licencias del Software**
   La transferencia de tecnología del software es un proceso importante y descuidado a menudo en la comunidad de IS académica. Sin embargo, la difusión, adopción, instalación y uso rutinario de los sistemas OSS y sus activos basados en la Web son fundamentales para la evolución de los sistemas. La transferencia de tecnología OSS desde sitios Web para la práctica organizacional es un proceso de construcción de equipos del proyecto y de la comunidad [100]. Los desarrolladores OSS publicitan y comparten sus proyectos adoptando y usando para ello sitios Web. Estos sitios Web pueden ser construidos usando sistemas de gestión de contenido OSS (como PhP-Nuke) y usando servidores Web OSS (Apache), sistemas de bases de datos (MySQL) o servidores de aplicación OSS (JBoss).

   La transferencia de tecnología OSS no es una ingeniería de procesos, al menos no todavía. Es más bien un proceso sociotécnico que implica el desarrollo de relaciones sociales constructivas; acuerdos sociales negociados de manera informal; y disposición para buscar, navegar, descargar y probar los activos OSS. Es también un compromiso de participar continuamente en debates públicos, basados en la Web. La construcción en comunidad y la participación sostenida son actividades esenciales y concurrentes que permiten que OSS persista sin centros de desarrollo y sin una planificación centralizada.

   Una parte general y esencial de lo que permite la transferencia y la práctica de desarrollo OSS y lo que la distingue de la IS tradicional es el uso y la reiteración de licencias públicas OSS. Más de la mitad de los 60.000 proyectos OSS registrados en sourceForge utilizan la licencia pública general GNU. La licencia pública general mantiene y reitera las creencias y las prácticas de compartir, examinar, modificar y redistribuir los sistemas OSS y los activos como derechos de propiedad para la libertad colectiva.

C.19. Ciclo de Vida de Proyectos de Desarrollo OSS

El trabajo de investigación de Schweik y Semenov [176] presenta un resumen del ciclo de vida de los proyectos OSS basado en la literatura existente que está centrada ampliamente en proyectos OSS como Linux y el servidor Web Apache. El ciclo de vida de los proyectos OSS involucra tres grandes etapas: Iniciación, Hacia lo Abierto y Crecimiento o Decadencia del Proyecto. A continuación, se describen cada una de estas etapas.

1. **Iniciación del Proyecto**

   a) **Surge un Problema:** Los proyectos OSS son iniciados porque una o más personas comprenden que existe un problema de computación o un desafío sin cumplir, y por una o más razones, ellos deciden asumirlo [80]. Esto describe la historia del

sistema operativo Linux, donde Linus Torvalds, un estudiante graduado finlandés, decidió que era necesario un sistema operativo para PC basado en Unix barato y se propuso construirlo él mismo [103][128]. Los tres componentes importantes de esta etapa son: la motivación, el núcleo y un diseño modular.

b) **Desarrollar la Versión Inicial:** El desarrollo de una versión inicial del producto es realizada para que otros puedan construir sobre ésta -lo que podemos llamar el núcleo del proyecto o "*kernel*"-. Por ejemplo, Torvalds desarrolló el *kernel* del sistema operativo Linux principalmente solo y, cuando él sintió que ya estaba listo para ser compartido, dejó disponible el código fuente del *kernel* en Internet, y animó a otros a ayudar a mejorarlo [103].

c) **Realizar un Buen Diseño:** Otro componente crítico en esta etapa del desarrollo OSS es un buen diseño y el concepto de modularidad, sin el cual no es posible tener a varias personas trabajando en paralelo [191]. Modularidad significa que el *kernel* está organizado alrededor de pequeñas piezas manejables. Por ejemplo, el software del Servidor Web Apache tiene cerca de 40 módulos y Perl sobre los 1000 módulos [66]. Con un diseño modular, múltiples programadores pueden estar trabajando en la construcción de nuevas funciones en el mismo módulo. Este enfoque paralelo estimula la innovación y puede llevar a un proceso de desarrollo rápido. La modularidad también permite que el desarrollo continúe, pero evita situaciones donde el impacto de las mejoras que realiza una persona a un módulo ocasione problemas con el trabajo en algún otro módulo. Además, la modularidad permite al administrador de contenido del proyecto, mantener un mejor control a medida que progresa el trabajo y cuando el producto se vuelve más complejo [191].

Muy cercana a la modularidad está el buen diseño del proyecto (o por lo menos una visión articulada de hacia donde va el proyecto). La manera más fácil de coordinar el comportamiento de una gran multitud semi-organizada es que ellos conozcan el objetivo [2].

2. **Hacia lo Abierto**

a) **Seleccionar una Licencia Particular:** La segunda etapa del ciclo de vida de los proyectos OSS, es el momento en el que los fundadores deciden hacer el proyecto "abierto". Ser abierto significa que los fundadores del proyecto deciden seguir los principios de las licencias del desarrollo OSS [9], y seleccionan una licencia particular para el producto [8]. Pero además de seleccionar la licencia, en esta coyuntura debe ser considerados cinco componentes adicionales: (1) credibilidad del proyecto/producto; (2) los sistemas de comunicación adecuados; (3) los sistemas de control de versiones apropiadas; (4) las estrategias de captación de voluntarios; y (5) las estructuras de gobierno del proyecto. Se describirán a continuación cada una de estos cinco componentes.

b) **Credibilidad del Producto y del Proyecto:** La credibilidad del *kernel* y del proyecto es un aspecto importante en el momento que el proyecto pasa a ser abierto [153]. Raymond y los autores de los "Documentos Halloween" [2] postulan algunos criterios claves para que un proyecto sea considerado creíble por otros:

- Es necesario que haya al menos unos cuantos "desarrolladores principales" entusiastas ya interesados en el proyecto.
- El proyecto tiene "una promesa plausible" tanto técnica como sociológicamente.
- El proyecto o producto es algo con un interés atractivo y es innovador.
- El proyecto es importante y de despliegue para un (futuro) gran número de desarrolladores.
- La cantidad correcta del problema ya ha sido resuelta antes de que el proyecto sea "abierto". Esto significa que debe haber bastante del producto desarrollado

que suministre un adecuado marco de trabajo para coordinar el trabajo futuro. Pero, si por el contrario, demasiado del producto es completado antes de que sea abierto, se corre el riesgo de que otros colaboradores que podrían unirse en el esfuerzo se conviertan en *"testers"* -una tarea que muchos programadores encuentran poco interesante- [7].

c) **Seleccionar los Mecanismos de Comunicación Adecuados:** Los proyectos OSS utilizan herramientas estándar de Internet para la comunicación: correo electrónico, listas de correos y listas de discusión ubicadas en sitios Web [33]. En este aspecto, la Web ha mejorado la habilidad para comunicar y compartir documentos y módulos. Diferentes sistemas Web están ahora disponibles para soportar la colaboración en proyectos OSS (por ejemplo, sourceforge.net).

d) **Considerar el Sistema de Control de Versiones:** En el núcleo de cualquier proyecto OSS exitoso hay un sistema para la administración de los desarrollos que se van haciendo del producto –también conocido como sistemas de control de versiones-. El más popular de estos en la comunidad OSS es el CVS [33][74].

e) **Escoger las Estrategias para Anunciar el Proyecto:** Otro motivo de preocupación importante para un proyecto es escoger las estrategias para anunciar el proyecto y para reclutar participantes adicionales. Para los proyectos OSS, esto ahora es mucho más fácil gracias al establecimiento de sitios Web "centrales" para *hosting* de proyectos OSS, como por ejemplo, surgeforge.net y freshmeat.net.

f) **Realizar el Diseño del Gobierno:** Como Bezroukov [36] señala: "*... en cada proyecto [OSS] particular, existen sistemas políticos con sus correspondientes y algunas veces difusas estructuras jerárquicas*". Estos conceptos incluyen variables críticas que pueden asegurar el éxito o fracaso del proyecto.

Para analizar los componentes institucionales de los proyectos OSS, utilizamos un framework de análisis institucional propuesto por Ostrom et al. [143]. El centro de este framework es el concepto de "Reglas de Trabajo", que definen qué acciones son requeridas, prohibidas o permitidas y qué sanciones están autorizadas si alguien rompe estas reglas [143]. Éstas pueden ser reglas formalmente escritas o normas de comportamiento entendidas y compartidas por una comunidad. Un aspecto importante de las reglas de trabajo es que ellas son seguidas, por lo menos la mayoría de las veces. Las reglas formalmente escritas a las que nadie presta atención no son, en este contexto, consideradas reglas de trabajo.

g) **Revisión de Pares:** La revisión de pares es un componente vital de los proyectos OSS. La revisión de pares es un proceso donde otros revisan las mejoras enviadas [115]. La revisión de pares es una práctica bien entendida dentro de los proyectos OSS.

h) **Resolución de Conflictos:** En muchos proyectos OSS, la reputación dentro del grupo es una importante motivación y en un enfoque de desarrollo paralelo pueden existir conflictos entre dos soluciones alternativas a un problema. La selección de una solución sobre otra puede llevar a un serio debate entre miembros del grupo. El peor escenario de conflicto en un proyecto OSS es una clase de motín intelectual conocido como *"forking"*. En estos casos los miembros del grupo deciden separarse del proyecto OSS existente para formar un nuevo proyecto rival porque ellos no están de acuerdo con las decisiones de diseño que el líder está tomando [107]. Debido a las reglas de licencias en OSS, ellos tienen acceso garantizado al código fuente y son libres de desarrollarlo en la forma que deseen. Feller y Fitzgerald [66] notan que en algunos proyectos OSS los desarrolladores principales que tienen un merecido respeto en la comunidad toman el rol de árbitros. Existen mecanismos para la resolución de conflictos en proyectos OSS exitosos –incluso dictatoriales- aunque ellos pueden ser simplemente normas sociales y no mecanismos formales.

3. **Crecimiento o Decadencia del Proyecto**

   a) **Generar Entusiasmo:** Una vez un proyecto se convierte oficialmente en OSS, la esperanza es que genere entusiasmo y tiente a programadores y a usuarios en la comunidad global de Internet a utilizar el producto existente y contribuir a su desarrollo a través de la programación, pruebas, o suministrando documentación.

   b) **Crecimiento del Proyecto:** En esta etapa, los proyectos pueden crecer con nuevos miembros.

   c) **Dejar el Código Disponible en Internet:** Dejar el código fuente disponible en Internet suministra acceso a la lógica programación de otros. Nuevos programadores o programadores interesados en aprender una nueva tecnología tienen el lujo de estudiar la infraestructura actual del programa –los detalles de la propiedad intelectual de otros en el grupo- y de esto pueden participar en una especie de aprendizaje a distancia.

   d) **Retroalimentación:** En situaciones donde el programador envía una contribución a la comunidad, el componente de revisión de pares del proceso OSS suministra una forma de obtener retroalimentación importante sobre la calidad del trabajo de cada uno.

   e) **Muerte del Proyecto:** Cuando los participantes pierden el interés en el proyecto, éste finalmente desaparece o muere.

C.20. Ciclo de Desarrollo de un Proyecto de Software Libre

El trabajo de investigación de Senyard y Michlmayr [177] describe estructuralmente el ciclo de vida de desarrollo de un proyecto de software libre. Los autores emplean la terminología de "Catedral" y "Bazar" propuesta por Raymond [155], pero en lugar de ver estos enfoques como diametralmente opuestos, son vistos como fases que se complementan dentro del mismo ciclo de vida. La Figura C.2 ilustra las tres fases básicas del ciclo de desarrollo de un proyecto de software libre, según la tesis defendida por Senyard y Michlmayr [177]. A continuación, se describirán cada una de estas fases.

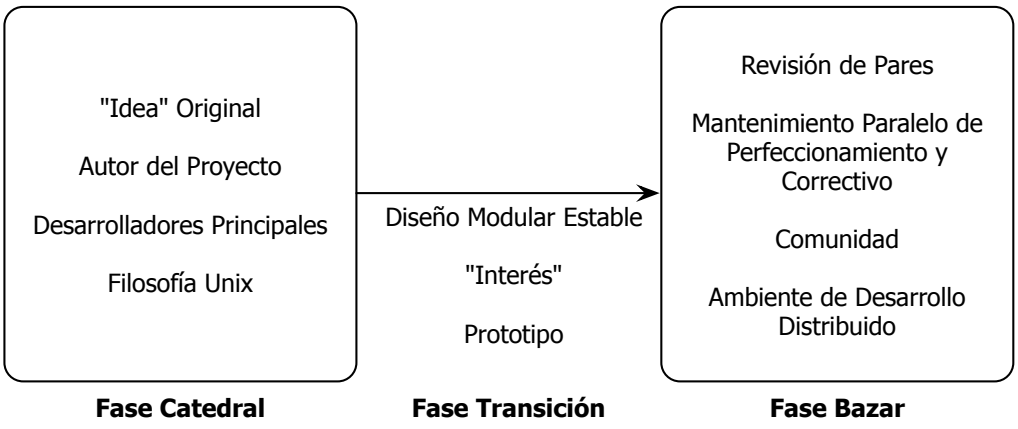


Figura C.2: Ciclo de Desarrollo de un Proyecto de Software Libre.

1. **Fase Catedral**
   Una de las principales diferencias entre el software cerrado (tradicional) y el software libre es la propiedad del código. En entornos tradicionales, un grupo de individuos conduce el desarrollo, mientras que los usuarios ni contribuyen ni tienen acceso al código.

En el software libre, potencialmente cualquiera tiene el derecho a acceder y modificar el código fuente de una aplicación. Creemos que un sistema libre típico presenta una fase de catedral en la primera parte de su historia evolutiva [42].

La fase inicial de un proyecto de software libre no opera en el contexto de una comunidad de voluntarios. Todas las características del estilo catedral (educción de requisitos, diseño, implementación y pruebas) están presentes en esta fase, y también en el estilo de construcción típico de una catedral, es decir, el trabajo lo realiza un individuo o un pequeño grupo de desarrolladores aislados de la comunidad. Este proceso de desarrollo muestra un control estricto y una planificación centralizada y realizada por el autor principal, que ha sido denominada "prototipado cerrado".

a) **Requisitos:** La única condición a ser satisfecha para continuar con la fase investigativa es que un usuario tenga algún requisito que pueda ser satisfecho por el software. Iniciar un nuevo proyecto de software libre es una tarea seria e involucra un compromiso a largo plazo y mucha energía. Así, los proyectos de software libre no pueden ser iniciados a menos de que exista un fuerte incentivo para hacerlo. Para un individuo que comienza el proceso de preparar un nuevo proyecto, esta motivación es generada por un requisito específico. En palabras de Raymond, un proyecto comienza para "rascar una comezón" [155].

b) **Investigar:** Para iniciar un nuevo proyecto es necesario estar seguro que no existe un proyecto que pueda satisfacer las necesidades de los usuarios. Por otra parte, un proyecto conveniente pudo haber sido identificado pero los usuarios no desean participar en el proyecto.

La fase de investigación asegura que ningún proyecto ya ha satisfecho las necesidades del individuo. De este modo, el individuo inicia una investigación de los proyectos existentes para encontrar si alguien más ha completado o ha estado trabajando en una herramienta que satisfaga sus necesidades. El escenario ideal para el individuo es descubrir un proyecto que hace exactamente lo que desea. El proyecto también se beneficia puesto que atrae a nuevos usuarios que probarán el software a través del uso, sugiriendo posiblemente nuevos requisitos y publicitando el proyecto.

c) **Iniciar:** Existen dos condiciones que necesitan ser satisfechas para pasar a la etapa del prototipo. La primera, es que haya sido realizado algún análisis de requisitos, riesgos y planificación, consciente o inconscientemente. La segunda, es que el desarrollador esté suficientemente motivado por la tarea de implementación, para proceder.

Los individuos que desean empezar su propio proyecto deben emprender (consciente o inconscientemente) el análisis de riesgos con respecto a un proyecto exitoso, el alcance de los requisitos y la creación del cronograma.

1) *Análisis de Riesgos:* Primero, el análisis de riesgos debe dirigir la motivación del individuo y el compromiso para un esfuerzo a largo plazo. Segundo, el análisis de riesgos debe direccionar si el nuevo proyecto puede competir con alguno existente y atraer voluntarios.
2) *Definir el Alcance de Requisitos:* El alcance de los requisitos debe definir la extensión del proyecto e identificar cómo el nuevo proyecto puede diferenciarse en sí mismo de proyectos existentes.
3) *Creación del Cronograma:* La definición del alcance de los requisitos es beneficiosa si el autor del proyecto tiene una idea poco elaborada del cronograma de desarrollo. Esto permite a otros desarrolladores principales (si existen) ayudar con la implementación inicial.

Todo este trabajo se realiza de manera informal porque en los proyectos de software libre no existen directrices oficiales para analizar los riesgos, determinar el alcance y crear el cronograma. Esto solo es necesario para que el individuo tenga algunos indicios positivos de los análisis, alcance y del cronograma.

d) **Prototipo:** La condición del prototipo que debe ser satisfecha para comenzar la fase de transición es que éste implemente los requisitos del autor del proyecto y sea extensible, permitiendo a múltiples desarrolladores trabajar en el proyecto simultáneamente [25][185].

Una vez el individuo ha decidido que vale la pena continuar, es relativamente fácil preparar el proyecto. No es necesario ningún equipo especial, con PCs normales es suficiente como plataforma de desarrollo (en muchos casos). El software necesario para desarrollar y probar el software, tal como compiladores, depuradores y entornos de programación, fueron algunas de las primera piezas de software libre creadas y están ampliamente disponibles con mínimos costos de distribución.

Una vez la infraestructura del proyecto ha sido establecida, se siguen las etapas tradicionales de requisitos, diseño, implementación y pruebas, y el individuo toma el rol de desarrollador. En este punto, no hay una estructura rígida para las etapas y actividades de desarrollo. Algunos desarrolladores siguen las etapas linealmente, mientras que otros prefieren un estilo de prototipado más circular en el que repiten todas o algunas etapas varias veces. Escoger un modelo de desarrollo en este punto es responsabilidad del desarrollador [25].

Solamente cuando el prototipo está finalizado y los desarrolladores deciden pasar al estilo de desarrollo Bazar, se vuelve importante el obtener retroalimentación de la comunidad. Las propiedades del diseño y la implementación del prototipo deben motivar a otros para participar en el proyecto. Un diseño modular, simple y claro facilita la labor.

1) *Requisitos del Prototipo*: La condición para pasar al diseño es simplemente que el desarrollador tenga alguna idea de los requisitos del software a ser diseñado. Los requisitos para el prototipo surgen de las necesidades específicas del desarrollador [118][185][198]. Estos requisitos y el deseo de programar suministran la motivación intrínseca para seguir adelante. El desarrollador tiene un buen entendimiento de lo que ellos necesitan, que es refinado durante las etapas de investigación e implementación. Requisitos adicionales pueden provenir de otros requisitos de los usuarios que son encontrados en la etapa de investigación y en las experiencias con otros programas.
2) *Diseño del Prototipo*: La condición del diseño para proceder a la implementación es que éste pueda ser extendido -esto se logra a través de la modularidad y la simplicidad-. El proceso usado dentro de la etapa de diseño no es importante con tal de que el diseño tenga un número de propiedades clave. En combinación con las prácticas del individuo, la filosofía y cultura Unix suministran un marco para el diseño que contribuye al éxito de los proyectos de software libre. Dividir un sistema software en subsistemas con interfaces y comunicación clara permite a los voluntarios contribuir al proyecto sin una estrecha coordinación [123].
3) *Implementación del Prototipo*: La condición de la implementación para continuar con las pruebas es que ésta sea técnicamente adecuada y muestre una "promesa creíble" [155]. Existe una considerable infraestructura del proyecto disponible para la implementación. Esto es debido al énfasis de la filosofía Unix en rehusar el software y las herramientas de soporte [156]. Como el software libre es a menudo diseñado e implementado en forma modular, el software habitualmente es dividido en diferentes bibliotecas. Estas bibliotecas pueden ser usadas para implementar las funcionalidades

requeridas en la aplicación inicial. Adicionalmente, durante la implementación, las herramientas estándares usadas en el software libre tales como *autoconf* de GNU y *make* de GNU, facilitan la construcción de software [112].

4) *Pruebas del Prototipo*: La única condición necesaria para que inicie la fase de transición es que alguna promesa creíble haya sido alcanzada con la implementación inicial y esto haya sido reforzado a través de las pruebas. Antes de que la implementación sea liberada a la comunidad, el desarrollador realiza algunas pruebas. Es decisión del desarrollador cómo realiza este proceso. Usualmente, herramientas o metodologías de pruebas específicas, como las pruebas de regresión, no son involucradas [27][119]. Más bien, el desarrollador decide si el programa funciona. Si los desarrolladores no están satisfechos con lo que han producido, pueden regresar a la fase de diseño o implementación. Por otro lado, si están convencidos de que el programa está en un estado donde puede ser compartido con otras personas y atraer a más desarrolladores, pueden liberar la implementación inicial y comenzar la transición a la fase Bazar.

e) **Distribuir:** En la fase catedral, preguntas acerca de la distribución y hosting del proyecto son lo menos importante. Mientras que un medio efectivo de distribución es crucial cuando se realiza la transición a la fase Bazar, la implementación inicial es usualmente desarrollada por un solo desarrollador o un grupo pequeño. Compartir el código a través del correo electrónico con otros desarrolladores o colocar éste en un sitio Web es suficiente para la distribución. Es posible usar sitios de hosting para proyectos de software libre públicos, como sourceForge o savannah.

2. **Fase de Transición: Nuevas Vías de Desarrollo**
El marco teórico representado en la Figura C.2 asigna un papel fundamental a la fase de transición, dado que requiere de una drástica reestructuración del proyecto, especialmente en la manera en la que es gestionado. Un aspecto importante es comenzar la fase de transición en el momento correcto. Este paso es crucial y muchos proyectos no logran superar este obstáculo [74]. Dado que durante la fase de transición es cuando hay que atraer a los voluntarios, el prototipo tiene que ser funcional pero a la vez todavía debe necesitar de desarrollo [25][97][155].

   Si el prototipo no tiene suficiente estabilidad o funcionalidad, los voluntarios potenciales puede que no se unan al proyecto. Por otro lado, si el prototipo está demasiado avanzado, los nuevos voluntarios no tienen demasiados incentivos para unirse al proyecto porque el código ya existente es complejo o las características que estos desarrolladores requieren han sido implementadas ya. En los dos casos, añadir funcionalidades futuras de desarrollo al sistema puede proporcionar a los potenciales desarrolladores vías para el desarrollo del proyecto.

   Cuando los nuevos desarrolladores se unen a un proyecto, tienden a trabajar en módulos nuevos más que en módulos viejos. Como consecuencia de esto, los desarrolladores principales deben expandir el sistema original hacia nuevas direcciones para proporcionar nuevo código sobre el que trabajar: esto fomentaría el reclutamiento de desarrolladores nuevos y facilitaría la fase de transición.

   La fase de transición requiere de una drástica reestructuración del proyecto, especialmente en la forma en que el proyecto es administrado. Existen muchas preguntas que deben ser consideradas al realizar una transición exitosa:

   a) **Distribución:** La primera pregunta es cómo el código debe ser distribuido. Esta pregunta está también relacionada con los aspectos de infraestructura del proyecto como la selección del *hosting*.

b) **Infraestructura:** En segundo lugar, la pregunta de cuál licencia usar influye en la probabilidad de que los contribuyentes se involucren en el proceso de desarrollo formando así un bazar.

c) **Estilo de Administración:** En tercer lugar, el estilo de administración debe ser establecido, incluso si se trata de una decisión implícita.

3. **Fase Bazar**
El objetivo de muchos proyectos de software libre es alcanzar una etapa en la que una comunidad de usuarios pueda contribuir de manera activa al desarrollo posterior del proyecto. Algunas de las características clave de la fase de bazar se ilustran en la Figura C.3 y pueden resumirse así:

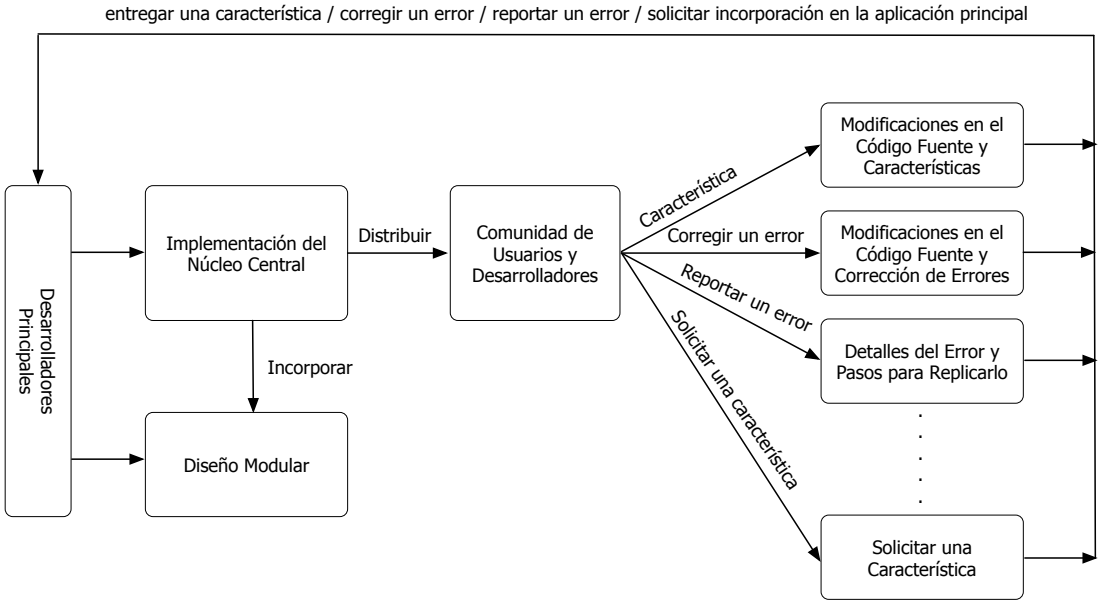


Figura C.3: Detalle de la Fase Bazar.

- *Contribuciones*: El estilo bazar hace que el código fuente esté disponible públicamente, y las contribuciones son fomentadas activamente, sobre todo las personas que utilizan el software. Las contribuciones pueden venir de muchas maneras y en momentos diferentes. Los usuarios sin perfil técnico pueden sugerir nuevos requisitos, escribir documentación y tutoriales, o poner de manifiesto problemas de usabilidad.
- *Calidad del Software*: Las inspecciones exhaustivas y en paralelo del código proporcionan altos niveles de calidad. Estas inspecciones las realiza una comunidad grande de usuarios y desarrolladores. Estos beneficios son consistentes con los principios de la IS: el proceso de depuración de un proyecto de software libre es sinónimo de la fase de mantenimiento del ciclo de vida tradicional de un proyecto de software.
- *Comunidad*: una red de usuarios y desarrolladores revisa y modifica el código asociado con un sistema software. El viejo dicho "el trabajo compartido es más llevadero" describe las razones por las que algunos proyectos de software libre tienen éxito [126].

Los proyectos de software libre, una vez han alcanzado la fase de mantenimiento y pueden acceder a una comunidad de codesarrolladores, son mucho más productivos que los proyectos tradicionales con recursos de mantenimiento limitados. Las actividades que facilitan la fase Bazar son las siguientes:

a) **Revisión de Pares:** La revisión de pares (incluyendo inspecciones) ha sido establecida como parte de las mejores prácticas de desarrollo y da lugar a software de gran calidad [64]. Los proyectos de software libre emplean revisión de pares descentralizada permitiendo a cualquiera el acceso completo del código. Mientras que sólo un número limitado de desarrolladores pueden trabajar en la implementación de un programa [41], no hay un límite en el número de personas que pueden inspeccionar el código y buscar defectos. Sólo es posible corregir un error una vez que se han identificado las condiciones asociadas. Los proyectos de software libre exitosos se benefician de una gran base de usuarios y desarrolladores que inspeccionan el código e identifican errores.

b) **Desarrollo Concurrente:** Las actividades dentro de la fase Bazar pueden ser desarrolladas concurrentemente, porque no hay una separación clara entre actividades [97]. Existen dos tipos principales de paralelismo de desarrollo en la fase bazar. En primer lugar, varios tipos diferentes de desarrollo pueden ser desarrollados concurrentemente, como agregar una característica y corregir un error. Los desarrolladores voluntarios pueden desarrollar sus actividades favoritas siempre que deseen. En segundo lugar, múltiples desarrolladores pueden desarrollar en paralelo. Esto permite que numerosas características y correcciones de errores sean adicionadas rápidamente. Otras tareas, tales como la documentación y las pruebas también requieren menos coordinación y pueden ser desarrolladas en paralelo [123][174]. Sin embargo, existen factores que limitan la cantidad de paralelismo en la implementación que puede favorecer al proyecto [41]. Este factor limitante depende de la modularidad del diseño de la aplicación inicial. Un segundo problema es la divergencia gradual del código desarrollado a partir del código base principal que requiere una sincronización regular.

c) **Apertura de los Requisitos:** La fase bazar se caracterizada por ser un proceso abierto en el que la entrada de voluntarios define la dirección del proyecto, incluyendo los requisitos. La implementación inicial está basada principalmente en los requisitos del autor del proyecto. En la fase bazar, los proyectos se benefician de la participación de una amplia gama de desarrolladores (con diferentes requisitos) que trabajan juntos para incrementar la funcionalidad del software. Esto conduce al problema de arrastrar características. Para prevenir esto, el encargado del mantenimiento debe decidir si una característica específica está en línea con el diseño y el alcance global del proyecto. Si se rechaza una característica, el voluntario se desanimará de usar y contribuir al proyecto porque no tiene una característica que deseaba. Por otro lado, si todas las características son incorporadas, el exceso de características asociadas puede provocar un sistema grande e imposible de mantener. Este problema puede superarse teniendo bien claro y definido el alcance del proyecto. El alcance del proyecto es establecido en la fase de transición.

d) **Implementación en Paralelo y Depuración:** Puesto que muchas personas diferentes pueden contribuir con código al proyecto de software libre, deben haber pautas que describan los estándares y el estilo de codificación. El proyecto GNU tiene un documento detallado que describe las mejores prácticas y la mayoría de proyectos grandes tienen una documentación similar o se refieren a estándares bien conocidos [98]. Ésta es la tarea del encargado del mantenimiento, asegurar que los parches están conformes con tales estándares. Los parches pueden provenir de diferentes personas y pueden variar enormemente en su tamaño. En muchos

proyectos, hay un número pequeño de colaboradores que hacen la mayoría de las implementaciones, mientras que un grupo más grande entrega parches más pequeños que corrigen errores.

La depuración requiere un cierto nivel de especialización técnica por parte del usuario. En el pasado, aquellos que usaban software libre eran usuarios avanzados y normalmente programadores, lo que contribuyó a la práctica en la que los usuarios participarán en el proceso de inspección del software. Esto ha cambiado a medida que el software libre ha ganado popularidad. Si bien todavía hay muchos usuarios técnicos, existe un número creciente de usuarios que no poseen habilidades técnicas para inspeccionar el código o corregir defectos, pero sí pueden enviar reportes tradicionales de errores. La ventaja del software libre sobre el software propietario es que cualquiera puede inspeccionar el código y completar un reporte de error. No hay ninguna dependencia en el encargado del mantenimiento del software, para corregir el error. Más bien, puede haber un número grande de desarrolladores cuya tarea es actuar como una capa entre el encargado del mantenimiento y el usuario. Cada vez más, esta tarea es desarrollada por el proveedor que reúne las distribuciones completas del software basadas en software libre.

e) **El Diseño y la Importancia de la Modularidad:** Como resultado de los requisitos y del código base realizado, se hace necesario adaptar el diseño. Las alternativas de diseño pueden ser discutidas y probadas en paralelo. Los desarrolladores, motivados por el deseo de crear el diseño e implementación más elegante [145], compiten para producir los mejores diseños.

Un error de adaptación del diseño es que la complejidad del mismo aumentará. En programas que evolucionan la complejidad aumenta, a menos que exista un trabajo específico realizado para reducirlo [105]. La única manera de facilitar la adición de características y evitar el incremento de la complejidad del diseño, es con un diseño modular desde la implementación inicial [25][185]. Un diseño claro y modular de la implementación inicial permite a la comunidad de desarrolladores refinar los requisitos y diseñar dentro de un alcance específico y crucial para el éxito del proyecto [191].

Sin embargo, en algún punto puede ser técnicamente deseable reemplazar un sistema con una versión simplificada más que continuar con el sistema existente.

En la mayoría de los casos, una reimplementación y rediseño completo es realizada de una manera similar a la creación de la implementación inicial (la fase catedral). Un equipo principal (compuesto por los mejores desarrolladores encontrados en la fase bazar) trabajará para crear un diseño y una implementación totalmente nuevas. Esta nueva implementación inicial debe ser más modular y adaptable que el software previo. En resumen, la propiedad clave de la implementación inicial de la fase catedral es un diseño modular y extensible.

C.21. Desarrollo de Software Open Source

El trabajo de investigación de Simmons y Dillon [181] presenta una visión global del proceso de desarrollo OSS. Según Simmons y Dillon [181], en el nivel más básico OSS puede pensarse como software que está libremente disponible tanto en forma de archivos ejecutables como de código fuente. Los proyectos OSS exitosos estimulan el interés de la comunidad de desarrollo solo si se ve que tienen potencial y que son útiles a los desarrolladores. El desarrollo OSS no está restringido por tiempo o dinero y los clientes son también los desarrolladores. Este rol dual de desarrollador y cliente significa que las dificultades de comunicación asociadas con la Ingeniería de Requisitos en el desarrollo de software tradicional no es principalmente un problema en el desarrollo OSS. Documentar los requisitos no es más que bosquejar “una

lista de deseos" de las potenciales características del producto. A continuación, se describirá el proceso de desarrollo OSS propuesto por Simmons y Dillon [181].

1. **Requisitos Dirigidos por el Desarrollador**
   Puesto que la carencia de conocimiento del dominio es frecuentemente un problema en proyectos de software grandes [55], una de las principales fuentes de error es eliminada cuando los expertos del dominio escriben el código. Los proyectos OSS normalmente no producen especificaciones de requisitos formales, más bien los requisitos están continuamente cambiando y son debatidos en público a través del uso de la tecnología de Internet, como por ejemplo las listas de correo. Como los requisitos son afirmados más que educidos, la invasión de características puede ser un problema, por consiguiente una vez los requisitos son afirmados deben ser justificados al líder del proyecto antes de ser aprobados.

2. **Prototipo Cerrado**
   Los proyectos OSS raramente inician en la fase conceptual. La gran cantidad de proyectos disponibles en SourceForge categorizados en las fases de planificación o pre-alpha que no evolucionan a otras fases sirven como una advertencia a los desarrolladores OSS. De hecho, Raymond [155] observa que "*es bastante claro que no se puede codificar de cero en el estilo bazar. Es posible probar, depurar y mejorar en el estilo bazar, pero sería muy difícil originar un proyecto en el modo bazar. La naciente comunidad de desarrolladores necesita tener algo ejecutable y probable para jugar*". Por esta razón los proyectos OSS tienden a ser iniciados por un individuo o grupo de individuos de una forma cerrada.

   Las siguientes actividades son completadas generalmente antes de que el proyecto sea liberado a la comunidad OSS:

   a) Definir la visión del proyecto

   b) Realizar el diseño inicial

   c) Implementar el prototipo

   Existen también muchos proyectos OSS a gran escala que son donados a la comunidad OSS como aplicaciones maduras para seguir desarrollándolas y que fueron desarrolladas y vendidas previamente como sistemas propietarios.

3. **Desarrollo Concurrente Iterativo**
   Incrementos pequeños e iteracciones rápidas tipifican el desarrollo OSS. Este ciclo de liberaciones rápidas mantiene a la comunidad de desarrolladores comprometida a medida que se van viendo los beneficios tangibles de sus contribuciones. Los desarrolladores deciden en lo que quieren trabajar y obtienen una copia del código, normalmente a través de un sistema de control de versiones (como CVS), luego hacen los cambios planeados que luego son encapsulados como un parche y sometidos a la revisión de pares. Este ciclo de desarrollo es repetido continuamente y llevado a cabo independientemente de otros desarrolladores.

4. **Revisión de Pares**
   Los proyectos OSS no realizan pruebas formales como norma. Mas bien las pruebas se suelen dejar a la comunidad de usuarios, quienes encuentran errores y posteriormente envían estos como un reporte de error. David Lawrence [101] observa: "*los errores obtienen una rápida atención en los proyectos OSS porque los participantes tienden a tener un significativo interés en los paquetes que desarrollan, que no suele ser el caso de la programación comercial. Como los usuarios finales tienen mucha experiencia en el mundo real y conocen como desean que el programa funcione, pueden identificar las características que faltan y señalar los comportamientos extraños y molestos. Cuando los*

usuarios ven un problema potencial, responden rápidamente". La carencia de pruebas automatizadas o formalizadas puede conducir a problemas cuando el código necesita una reestructuración importante y puede hacer que técnicas como la refactorización sean muy difíciles de lograr.

5. **Participación Escalonada**
   Muchos proyectos OSS implementan un sistema escalonado de participación, aunque cualquiera puede descargar el código, solicitar características, enviar un reporte de error o incluso sugerir una solución. Solamente al círculo más interno le es permitido enviar cambios al código base. El paso de un escalón al siguiente es juzgado por el líder del proyecto y por lo general es una recompensa por la persistencia y demostración de competencia durante un periodo de tiempo.

6. **Diseño Modular**
   Los proyectos OSS evolucionan continuamente con los requisitos que son constantemente adicionados en una forma aparentemente caótica y los desarrolladores deciden trabajar en cualquier problema que sea de su preferencia. Un problema común con el desarrollo evolutivo es que debido a los frecuentes cambios del sistema, llegan a tener una pobre estructura y son difíciles de mantener [184], no obstante muchos autores han observado que los proyectos OSS se escalan bastante bien [80]. Una posible razón para esto es que los proyectos OSS utilizan una arquitectura modular para ayudar a las futuras pruebas de la aplicación y minimizar los riesgos de corrupción de la base de código.

C.22. Proceso de Ingeniería OSS

El trabajo de investigación de Vixie [198] ofrece una visión global de la IS, con el objetivo de dar alguna motivación y contexto para que los desarrolladores OSS se interesen en esta área. Dentro de esta visión, el autor describe el proceso de desarrollo seguido por la comunidad OSS. A continuación, se describen las actividades de este proceso.

1. **Marketing de Requisitos**
   Los voluntarios de la comunidad OSS tienden a construir las herramientas que necesitan o desean tener. A veces esto ocurre junto con el trabajo diario y muchas veces se trata de alguien cuyo trabajo principal está relacionado con la administración de sistemas más que con la IS. Si después de varias iteraciones, un sistema de software alcanza una masa crítica y adquiere vida propia, éste será distribuido a través de Internet y a otros usuarios que empezarán a solicitar características o simplemente se sentarán e implementarán éstas, para luego enviarlas.

   El Documento de Requisitos (DR) OSS es usualmente una lista de correos o un grupo de noticias, con usuarios y desarrolladores bromeando de un lado a otro. Lo habitual, es que cualquiera de los desarrolladores recuerde o acepte lo reportado en la lista de correos o en el grupo de noticias. La falta de consenso a menudo da lugar a una "división de código", donde otros desarrolladores comienzan a liberar sus propias versiones. El equivalente al DR para OSS puede ser muy rico, pero la resolución de conflictos no es posible algunas veces (o no se intenta).

2. **Realizar el Diseño del Nivel-Sistema**
   Normalmente no existe un diseño del Nivel-Sistema para un esfuerzo OSS sin financiación. O bien el diseño del sistema está implícito o evoluciona con el tiempo (como el propio software). Por lo general, por la Versión 2 o 3 de un sistema OSS hay un diseño del sistema, aún cuando no se escriba en ninguna parte.

Se puede compensar la falta de un DR formal o incluso el aseguramiento de calidad teniendo realmente buenos programadores (o usuarios amigables), pero si no hay ningún diseño del sistema (aún cuando solo esté en la cabeza de alguien), la calidad del proyecto será en sí misma limitada.

3. **Realizar el Diseño Detallado**
   Otra consecuencia de no tener financiación es el diseño detallado. Algunas personas encuentran divertido trabajar en el Documento de Diseño Detallado (DDD), pero estas personas generalmente consiguen toda la diversión que pueden soportar escribiendo los DDD durante sus trabajos diarios. El diseño detallado termina siendo un efecto colateral de la implementación. "Yo sé que necesito un *parser*, así que lo escribo". Documentar el API en la cabecera de los archivos o escribir manuales son tareas opcionales y puede que no ocurran si el API no está pensado para ser publicado o ser usado fuera del proyecto.

   Esto es un incoveniente, ya que una gran cantidad de buen código reutilizable está oculto de esta manera. Incluso, módulos que no son reutilizables limitan el proyecto cuando sus APIs no son parte de los entregables. Realmente es necesario tener manuales que expliquen qué hacen y cómo usar las APIs. Es muy útil para las otras personas que quieren mejorar el código ya que tienen que empezar por leer y entenderlo.

4. **Implementar**
   La implementación es lo que los programadores aman más; es lo que les mantiene hasta tarde "picando" cuando pudieran estar durmiendo. La oportunidad de escribir código es la principal motivación para casi todos los esfuerzos de desarrollo OSS.

   En los proyectos OSS es donde muchos programadores experimentan con nuevos estilos, ya sean de indentación o de identificación de variables, o "técnicas para intentar ahorrar memoria" o "intentar ahorrar ciclos de CPU". Y hay algunas piezas de código de gran belleza, donde algún programador intentó por primera vez un estilo y funcionó.

   Un esfuerzo OSS no financiado puede tener tanto rigor y consistencia como se desee. Los usuarios ejecutarán el código si es funcional y a la mayoría de las personas no les importa si el desarrollador cambia de estilo tres veces durante el proceso de implementación. Los desarrolladores generalmente cuidan el código, o aprenden a cuidarlo luego de un tiempo.

   La principal diferencia en la implementación de los proyectos OSS sin financiación, es que la revisión es informal. No siempre existe un mentor o par que revise el código antes de que se libere. Usualmente, no se realizan pruebas unitarias ni de regresión.

5. **Realizar la Integración**
   La integración de un proyecto OSS por lo general consiste en escribir algún manual, asegurar su estructura en cada tipo de sistema al que el desarrollador tiene acceso, limpiar el fichero *Makefile* para eliminar cualquier cosa que se haya heredado de la fase de implementación, escribir un README, comprimir todo esto en un fichero, subirlo en alguna parte de un FTP anónimo y publicar una nota en alguna lista de correo o grupo de noticias donde los usuarios interesados puedan encontrarlo.

   Por lo general, no se realizan pruebas de unidad ni se cuenta con planes de pruebas a nivel de sistema. Existen excepciones como Perl y PostgreSQL. Sin embargo, esta falta de pruebas previas a la liberación no es una debilidad como se explica en la siguiente actividad.

6. **Realizar las Pruebas**
   Los proyectos OSS sin financiación disfrutan de las mejores pruebas de nivel de sistema de la industria.La razón es simplemente, que los usuarios tienden a ser mucho más amistosos cuando a ellos no se les está cobrando nada y los usuarios avanzados (a menudo los propios desarrolladores) son mucho más útiles cuando pueden leer y corregir el código fuente de algo que están ejecutando.

La esencia de las pruebas es su falta de rigor. Lo que la IS está buscando a partir de sus pruebas de campo son patrones de uso que son inherentemente imprevisibles en el momento en el que se diseña y construye el sistema –en otras palabras, experiencias del mundo real de usuarios reales-. Los proyectos OSS sin financiación son insuperables en esta área.

7. **Revisión de Pares**
   Una ventaja adicional que disfrutan los proyectos OSS es la "revisión de pares" de decenas o cientos de otros programadores que buscan errores leyendo el código fuente en lugar de simplemente correr los paquetes ejecutables. Algunos de los lectores buscarán fallos de seguridad y algunos de los fallos encontrados no serán reportados, pero este peligro no elimina la ventaja global de tener innumerables extraños que lean el código fuente. Estos extraños realmente pueden tener a un desarrollador OSS siempre trabajando de una manera que ningún gerente o mentor jamás podría.

8. **Dar Soporte**
   "*Oops, lo siento!*" es lo que usualmente se dice cuando un usuario encuentra un error, o "*Oops, lo siento, y gracias!*" si ellos también envían un parche. "*Eh, funciona para mí*" es la manera como los desarrolladores OSS ocultan un error. Si esto parece caótico, lo es. La falta de apoyo puede impedir a algunos usuarios estar dispuestos a ejecutar aplicaciones OSS sin financiación, pero esto además crea oportunidades para que consultores o distribuidores de software vendan contratos de soporte y/o versiones comerciales y/o mejoradas.

   Cuando la primera comunidad de vendedores de Unix encontró un fuerte deseo de sus usuarios para enviar preempaquetado el software open source con su sistema base, la primera reacción fue "*muy bien, OK, pero nosotros no vamos a darle soporte a esto*". El éxito de compañías como Cygnus ha llevado a examinar de nuevo tal posición, pero el choque cultural es bastante profundo. Las casas de software tradicional, incluyendo vendedores Unix, no pueden planificar o presupuestar el costo de ventas de un negocio de soporte, cuando existen cambios que no son revisados y que son aportados por extraños a la compañía.

   A veces la respuesta es hacer que el software sea interno, ejecutándolo a través del proceso normal de aseguramiento de calidad que incluye pruebas de unidad y del sistema, análisis de código, y así sucesivamente. Esto puede involucrar reingeniería a fin de obtener los DR y DDD para proporcionar algo de aseguramiento de calidad (es decir, qué funcionalidades probar). Otras veces la respuesta es reescribir los términos del acuerdo de soporte para "mejorar los esfuerzos" más que "garantizar resultados". Finalmente el mercado del soporte de software será ocupado por quien pueda conseguir la influencia de todos esos extraños, ya que muchos de ellos son excelentes escribiendo buen software y la cultura OSS es más efectiva en muchos casos en la generación del nivel de funcionalidad que los usuarios realmente quieren.

C.23. Ciclo de Vida de los Proyectos OSS

El trabajo de investigación de Wynn Jr. [202] examina la estructura de los proyectos OSS a medida que evolucionan con el tiempo. A continuación, se describirán cada una de las fases del ciclo de vida en el contexto de los proyectos OSS.

1. **Introducción**
   La fase de introducción está compuesta por cuatro actividades: surge una necesidad, producir una versión inicial, crear la estructura inicial del equipo de trabajo y registrar el proyecto en Internet.

a) **Surge una Necesidad:** Para comenzar un desarrollador generalmente encuentra un vacío entre una necesidad de software relacionada con su trabajo o con sus intereses personales y las aplicaciones disponibles que satisfagan tal necesidad o interés. Esto da como resultado la motivación inicial para crear un proyecto OSS en el que se desarrolla una aplicación para llenar este vacío.

b) **Producir una Versión Inicial:** Un único desarrollador o un pequeño grupo de desarrolladores trabaja para producir una versión inicial de la aplicación.

c) **Crear la Estructura Inicial del Equipo de Trabajo:** Normalmente el desarrollador fundador es el responsable de crear la estructura inicial y reclutar a otros miembros del equipo [124]. El grupo principal negocia una estructura informal que consiste de roles generales por cada miembro. El rol del líder del grupo en esta fase consiste en dirigir los esfuerzos del resto del equipo [77] y comunicar la misión y objetivos del proyecto [30]. Debido a la naturaleza de las organizaciones OSS, no hay una asignación directa de trabajo por parte del líder; todos los miembros del equipo son libres de escoger sus propias tareas [179].

d) **Registrar el Proyecto en Internet:** El fundador también es responsable de asegurar los medios necesarios para registrar el proyecto en Sourceforge.net u otros sitios Web que dan a otros usuarios la posibilidad de descargar y experimentar con la aplicación, pues la base de usuarios es típicamente muy pequeña.

2. **Crecimiento**
   El proyecto crece a medida que más usuarios se dan cuenta de la existencia de una aplicación que les proporciona una solución a un vacío observado en sus necesidades. Como resultado, las exigencias administrativas del proyecto se incrementan.

   a) **Solicitud de Características, Reporte de Errores:** Estas exigencias administrativas se deben, por ejemplo, al crecimiento de la retroalimentación de los usuarios con respecto a las solicitudes de nuevas características, reporte de errores, peticiones de ayuda, etc.

   b) **Usuarios Contribuyen con Código:** Muchos de estos usuarios son también desarrolladores y contribuyen con código para resolver los problemas que puedan tener con el proyecto.

   c) **Evaluar el Código Contribuido:** El equipo de administradores tiene la responsabilidad de evaluar los fragmentos de código contribuido en cuanto a calidad, con la posibilidad de incluir el código adicional en versiones posteriores del software (y dar el crédito apropiado a quien corresponda).

   d) **Agregar Miembros a los Equipos de Administradores y de Desarrolladores:** Debido a que las necesidades se han incrementado, el equipo de administradores añade algunos de estos usuarios a su propio equipo y al de desarrolladores.

   e) **Liberación de Versiones:** El producto en sí mismo experimenta diferentes versiones, numerosos parches y corrección de errores a lo largo de la fase de crecimiento.

   f) **Formalizar la Estructura:** Debido al incremento de tamaño y al número de miembros del equipo (especialmente en roles no principales), surge la necesidad de una estructura más formalizada. El administrador asume el rol de un "acelerador", que emplea sistemas y estructuras para que el proyecto siga creciendo y para administrar su crecimiento [199]. La confianza en herramientas tecnológicas como CVS, grupos de discusión y listas de correos llegan a ser importantes para coordinar los esfuerzos de la comunidad OSS. A medida que la fase avanza, el aumento en la cantidad de trabajo permite a los miembros seleccionar roles más especializados,

como probador de código, administrador de versiones, diseñador de interfaces, administrador de soporte, escritor de documentación, y corrector de errores. La estructura resultante es todavía relativamente centralizada con los desarrolladores principales manteniendo todo el control del proyecto, pero las funciones menores son delegadas fuera del grupo principal.

3. **Madurez**
   En esta tercera fase el proyecto alcanza la masa crítica. El número de usuarios y desarrolladores crece al tamaño máximo. Las actividades en esta fase son las siguientes:

   a) **Evaluar el Código:** Debido al tamaño aún mayor de la comunidad OSS, los administradores invierten una cantidad significante de tiempo ejecutando políticas, evaluando el código de los demás y realizando otras funciones que no son de desarrollo.

   b) **Delegar:** Lo anterior, lleva a incrementar los niveles de delegación a los miembros de la comunidad.

   c) **Liberar Múltiples Versiones:** En algunos casos, el código es lo bastante grande para garantizar múltiples versiones y liberaciones del proyecto. Aunque la base de usuarios crece, un porcentaje de los usuarios son más pasivos que otros.

   d) **Coordinar y Controlar la Estructura del Equipo:** El foco central del grupo principal administrativo en esta fase es mantener el proyecto [199]. Debido al tamaño de los proyectos grandes, hay una necesidad organizacional de coordinar y controlar la estructura divergente [166]. En organizaciones tradicionales, esto lleva al control burocrático. Sin embargo, los proyectos OSS son muy resistentes a la burocracia por naturaleza, por lo que ésta es una respuesta inaceptable. Hay un incremento en el nivel de formalidad y rigidez, pero la estructura general sigue siendo bastante flexible y orgánica durante la vida del proyecto. Los administradores deben tener cuidado en centrarse en toda la comunidad y no solamente en un conjunto de roles. También deben centrarse en los efectos a largo plazo y no solamente en soluciones a corto plazo, especialmente para proyectos con un horizonte de tiempo más largo [182].

   e) **Resolver Conflictos:** Aunque muchas decisiones son alcanzadas por consenso [179], algunas veces surgen conflictos entre partes dentro de la comunidad, que el administrador (quien tiene la única responsabilidad del código y las versiones) está bien posicionado para resolver o mediar. Para ayudar en la resolución de conflictos y en el control de la comunidad, el grupo principal (de acuerdo con las normas de OSS) establece reglas y normas.

   f) **Mantener la Moral y Motivación Individual:** El gran número de miembros activos, el alto grado de delegación y la autogestión conduce a niveles muy altos de especialización de tareas. Es también en este punto que el desgaste y rotación se hace más significativo ya que los desarrolladores y otros miembros activos pierden interés en el proyecto. Por tal razón, otra tarea de administración subestimada es mantener la motivación y moral del individuo [30].

4. **Decadencia (o Renacimiento)**
   Como los usuarios encuentran otras soluciones a sus necesidades y los desarrolladores pierden interés en continuar el crecimiento del producto, el proyecto entra en la fase final marcada por la disminución de los usuarios y desarrolladores. Hay menos descargas del producto porque los usuarios están menos interesados en el proyecto. Hay un grupo más pequeño de administradores y desarrolladores restantes, que puede no incluir a los desarrolladores fundadores.

a) **Soporte y Mantenimiento:** El objetivo principal de la comunidad restante es el soporte y mantenimiento de las funcionalidades existentes.

b) **Renacimiento** El algunos casos, hay un renacimiento de la comunidad del proyecto en respuesta a una nueva versión, cambio de ambiente o condiciones del mercado (liberación de nuevos sistemas operativos), o simplemente un descubrimiento tardío por un grupo de desarrolladores motivados. Este renacimiento puede ocasionar que el proyecto entre en una nueva etapa de crecimiento o madurez dependiendo de la magnitud.

c) **Decadencia:** Encontrar el líder sucesor del proyecto se convierte en un problema ya el desarrollador original pudo haber cumplido sus necesidades originales (técnicas o personales) y perder el interés. En este punto, se produce una sucesión si otro miembro está interesado en asumir el rol del líder [155]. Si no es así, el proyecto no es dirigido o bien el fundador original simplemente cesa de innovar. Hay casos donde un desarrollador se separa del proyecto original para formar un nuevo proyecto de desarrollo basado en el código fuente. En algunos casos, los proyectos simplemente son declarados inactivos y disueltos.

C.24. Proceso de Desarrollo del Proyecto FreeBSD Newconfig

En el trabajo de investigación de Yamauchi et al. [203], los autores estudian el proceso de desarrollo de dos proyectos OSS: FreeBSD Newconfig y GCC. En esta sección se describirá el proceso de desarrollo del proyecto FreeBSD Newconfig y en la siguiente sección se describirá el proceso del proyecto GCC. La Figura C.4 ilustra cada una de las actividades del proceso de desarrollo del proyecto FreeBSD Newconfig. A continuación, se describirán cada una de estas actividades.

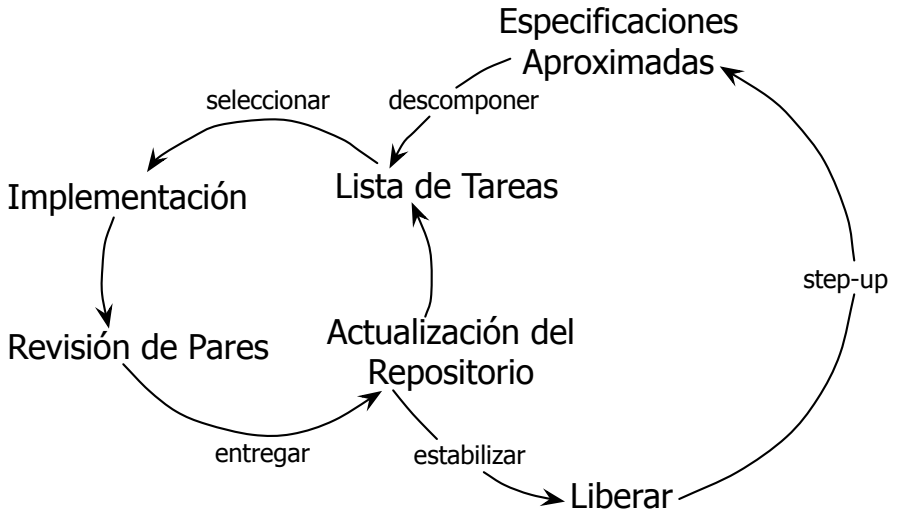


Figura C.4: Proceso de Desarrollo del Proyecto Newconfig Project.

1. **Especificaciones Aproximadas**
   Los desarrolladores inician discutiendo las especificaciones del producto tales como el comportamiento de los mecanismos de control, las estructuras de datos, las mejoras potenciales, y los hitos de desarrollo. Esta discusión, sin embargo, contiene una idea aproximada en lugar de una estructura detalladamente documentada del producto. El objetivo principal es compartir y combinar diferentes ideas de los miembros de la comunidad.

2. **Lista de Tareas por Hacer**
   El resultado de la discusión es una lista de tareas por hacer. En el proceso de desarrollo, esta lista de tareas juega un rol crucial. Para facilitar el trabajo, la forma de la lista de tareas es muy elaborado. En primer lugar, cada elemento que representa cada tarea tiene un indicador del nivel de prioridad marcado desde la A+ hasta la C-, donde una la prioridad A+ es la más urgente. En segundo lugar, cada elemento tiene un nivel de dificultad que va desde la A+ hasta la C-. Por último, una breve descripción es adjuntada a cada elemento. Es importante destacar que la longitud de las descripciones se limita a dos líneas. Esto significa que las descripciones no son para especificar el contenido de la tarea sino para explicar el tipo de tarea. La forma de realizar tareas es decidida por los miembros individuales del equipo.

3. **Implementación**
   Los programadores seleccionan un ítem de la lista y lo implementan. Sorprendentemente, los desarrolladores no suelen declarar qué tarea van a realizar antes de comenzar. Lo primero que realiza el desarrollador al comenzar la tarea es copiar el código fuente principal del repositorio en su espacio de trabajo. Los desarrolladores modifican el código y prueban la modificación en su copia local.

4. **Revisión de Pares**
   Una vez terminada la tarea, los desarrolladores envían el parche a la lista de correos del proyecto. Otro miembro del proyecto que tenga tiempo y conocimientos revisa el parche leyendo el código.

5. **Actualización del Repositorio**
   Si el revisor da el OK, el trabajo es incluido en el repositorio de código compartido.

6. **Liberación**
   Las versiones oficiales liberadas son anunciadas cuando el producto software es estable y proporciona bastante funciones.

C.25. Proceso de Desarrollo del Proyecto GCC

El proyecto GNU Compiler Collection (GCC), es uno de los proyectos más reconocidos de GNU y tiene su origen en el compilador de C creado por Richard Stallman. Ahora, el proyecto mantiene varios compiladores y bibliotecas, que incluyen C++, Pascal y Fortran aunque inicialmente el proyecto fue llamado GNU C Compiler (GCC). Debido al alto rendimiento y estabilidad los compiladores, estos fueron ampliamente usados para la creación de programas.

El proceso de desarrollo del proyecto GCC es sencillo ya que implica mantener el software más no crearlo [203]. La mayoría del trabajo del proyecto se originó a partir del uso diario del programa, como se ilustra en la Figura C.5. A continuación, se describirán cada una de las actividades del proceso de desarrollo del proyecto GCC.

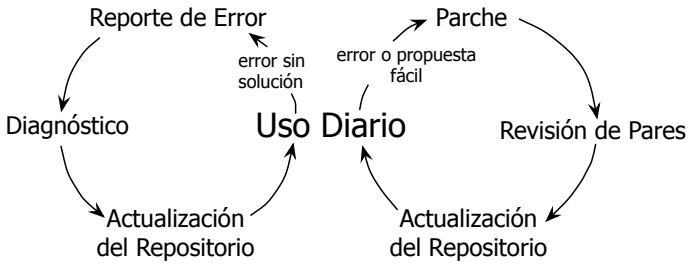


Figura C.5: Proceso de Desarrollo del Proyecto GCC.

1. **Uso Diario**
   El uso diario de los compiladores permitió encontrar errores y desarrollar las correspondientes correcciones.

2. **Reporte de Errores**
   Los errores encontrados deben ser cuidadosamente reportados de acuerdo a las guías para el reporte de errores, disponibles en el sitio Web del proyecto. Esta guía solicita a los usuarios que consulten los manuales, las preguntas frecuentes y la lista de errores conocidos antes de reportar un nuevo error. El reporte de error debe incluir la versión de GCC, el tipo de sistema, las opciones pasadas al compilador, y la salida del preprocesador. Estas instrucciones estaban destinadas a minimizar los errores duplicados y garantizar que se haya proporcionado la información suficiente para replicar los errores.

3. **Diagnosticar y Corregir los Errores**
   Los reportes de error implican un diagnóstico y posterior corrección. La corrección de los errores se da en forma de parches.

4. **Enviar el Parche**
   En el caso de pequeños errores o propuestas, los usuarios envían parches después de corregir los errores o implementar las propuestas.

5. **Revisión de Pares**
   La revisión de pares se realiza de la misma manera que en el proyecto FreeBSD Newconfig. Una vez el parche es entregado a las listas de correo del proyecto, otro miembro de la comunidad revisa el parche leyendo el código fuente entregado.

6. **Actualización del Repositorio**
   Si la revisión es exitosa, el código fuente del parche es incluido en el repositorio de código compartido.

ANEXO D

PROBLEMAS DE USABILIDAD EN OSS POR ESTUDIO PRIMARIO

El presente anexo tiene por objetivo listar los problemas de usabilidad que hemos identificado en los estudios primarios. La Tabla D.1 presenta para cada uno de los estudios primarios el fragmento del artículo donde se describe el problema. Nótese que no todos los estudios primarios se encuentran en la tabla debido a que estos estudios no aportan nuevos problemas.

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario.

Estudio Primario	**Descripción del Problema Dado por el Autor**
SU1	... some developers see usability as a trivial task which is not interesting nor intellectually stimulations; "... hackers code for fun, and sure it is more fun to add support for some protocol feature than fixing a dialog for grandma". ... who have been more interested in improving the functionality of the software than in improving its usability.
	... Usability experts are only advisors. Though most OSS contributors wanted a higher degree of usability in their software, they were reluctant to include usability experts directly in the development process. OSS contributors clearly stated that they were afraid that direct involvement of usability experts, especially in decision making, would overrule the democratic way of OSS, since it would be difficult to have a democratic debate against the only expert on the matter.
	Despite the appreciation of the knowledge of usability professionals and the usability reports, we sensed a gap between the technically minded contributors and those with a usability background.
	Trust is crucial in the development process. Co-operation with the OSS community is based on trust and both developers and usability professionals contributing to OSS need to be prepared to build a rapport with other contributors. For instance Relevantive experienced that almost all problems faced when working with OSS developers were grounded in lack of trust, which made developers ignore suggestions from usability professionals.
	... The nature of OSS, where contributors rarely meet in person, makes it necessary to judge others based on past merits. It can be difficult for a usability expert to display merits, since usability improvements are more difficult to measure than the programming of a new feature.

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU2	Noting this trend, past research has identified some of the challenges inherent in addressing usability in FOSS development. For example, the distributed, largely voluntary nature of FOSS development makes it difficult to engage in holistic design methods since developers tend to work on individual application components in isolation.
	Free/open source software architectures are often highly modular in design. However, usability concerns cut across the entire application. As a consequence, it can be difficult for individuals to make small, incremental improvements to a software's usability, a significant issue for software largely developed by volunteers in their spare time.
	Similarly, the lack of dedicated infrastructure for usability activities and design artifacts makes it difficult to coordinate usability work in distributed development environments.
	Finally, since this merit-based culture has traditionally valued source code as its primary currency, it has sometimes been difficult for UX practitioners to join and make contributions that are perceived as valuable.
SU3	Noting this trend, past research has identified some of the challenges inherent in addressing usability in FOSS development. For example, the distributed, largely voluntary nature of FOSS development makes it difficult to engage in holistic design methods since developers tend to work on individual application components in isolation.
	Free/open source software architectures are often highly modular in design. However, usability concerns cut across the entire application. As a consequence, it can be difficult for individuals to make small, incremental improvements to a software's usability, a significant issue for software largely developed by volunteers in their spare time.
	... typically there are no resources for UCD in OSS development.
	... no UCD methodology is typically employed, because this can be seen as being in contrast with the 'open source philosophy'; it is assumed that in OSS development there is no possibility for systematic UCD or formal process models.
	Typically, the developers do not have knowledge about the non-developer users, their goals and their contexts of use, and they do not necessarily have any interest in learning about them either.
	... communicating usability problems to the development has proven to be difficult for the non-developer users.
	Furthermore, usability specialists would be very useful in improving OSS usability, but typically there is a lack of usability specialists in OSS projects.
SU6	Interaction issues: In a distributed environment, limited and asynchronous information flow leads to a problem of low performance among developers, compared to collocated teams. In such a team, most of the communication is ideally conducted electronically (e-mails, phone, teleconferencing, emails, etc.) –sometimes all team members meet in a predefined location with changing periods. For example, most KDE developers meet at least twice in a year, but some less technical and low-profile projects' developers may find it unnecessary and expensive even to meet annually.

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU6 (continuac.)	... It's convenient to add the following items about why usability experts face problems while taking an active part in F/OSS development. 1. With usability experts taking an active part in the project, the time-to-market is slowed down. 2. A developer should cope with other roles – now he is not the only decision maker. 3. The developer without knowing its user base assumes that the requirements for the new application are the same as the requirements in all other applications. 4. The necessity of a user interface developer is questioned after introduction, adoption and exploitation of interface design software by non-designers.
	Focusing on usability reporting tools. Current tools are not convenient for reporting usability issues with the following reasons: - They don't have a mechanism to interactively record, upload, show, maintain and comment on user submitted videos, images and voice. - There's no way to merge a note to an attachment to show the submitters' and developers' opinion, annoyance and feedback. Trying to spot a minor usability issue may not be explained verbally, and hence needs a graphical representation. Unfortunately, not all computers are bundled with a painting and drawing application. - Current bug reporting tools have an increased complexity which is trying to spot all kinds of problems on the direction of the developer, ignoring the mental model of the user. - An average bug reporting tool requires to fill a considerable amount of information, some of which are not immediately pertinent to the end user or HCI expert, making it sometimes impossible to submit a bug thus leave the HCI expert out of the scope of the project. Under the light of the facts above, the lack of a suitable usability reporting interface results in some issues. First, number of reporters and reports thereof decreases. Most of the critical bugs never go into the bug database, rendering it unusual to increase the quality of the code. Second, usability reports are handled by the mailing lists and forums instead of a database, which is hard to follow, fix and give proper feedback to the reporter. And finally, since the aim of the bug reporting tool is often misunderstood, the end user (sometimes the usability expert) starts to discuss about an issue and/or report a problem he cannot solve, mostly ending up with closing the bug because of misusing the bug reporting tool.
SU8	Free and Open Source Software (F/OSS) developers tend to build feature-centric projects rather than following a user centred design, ignoring the necessity of usability inside the resulting product, ever since. As there are many reasons behind this, main cause can be attributed to the lack of awareness of usability from developers' point of view and little interaction of project stakeholders with Human-Computer Interaction (HCI) studies.
	Lack of common usability design guidelines and methods of communication between usability experts and F/OSS developers resulted in F/OSS software with relatively low level of usability.

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU8 (continuac.)	On the other hand, previous research suggests that main driving factor for F/OSS development is motivation of project stakeholders, and usability has not been seen as the primary objective within the project aims. Moreover, primary driving factor of developing a F/OSS product is the freedom of the movement itself, where, this freedom does not enforce, imply or mean that the resulting application will be highly usable.
	User centric design has not been the first priority of open source projects developed by people geographically distributed in all parts of the world. Thus, it can be argued that usability awareness and representation has been neglected for a long time.
	The lack of clear and well defined usability requirements, awareness of user centered design and a social collaborative tool to discuss usability issues usually result in a poor evaluation for F/OSS products.
	... few projects (28.3 %) have a specific component dedicated to usability inside the bug tracker, a situation which has the potential to lead to issues like inability to track usability bugs and pitfalls during a search in bug database.
	... We have shown that there are many issues and solid barriers of communication between usability experts and developers. There's a clear boundary between these entities apart from the blurry links between developer and user. It can be concluded that a web page should have a project web site, but more importantly, it needs to deploy connective links between users and developers. Providing a content management system and communication tools for project stakeholders is a helping tool to ensure a positive cooperation and information flow.
	It's convenient to list the issues and concerns of communication paths between usability experts and F/OSS developers here, derived from answers to open ended questions. 1. Developers are uninformed about contextual inquiry and user centered requirements process, resulting in a lack of immediate experience of observing product use in the field. 2. Usability experts do not have a detailed understanding of how F/OSS projects work, usually criticizing excessively, rather than contributing, patching to the codebase or solving issues in bug database. While part of this is perception, other part is the attitudes of usability experts have the inability or willingness to develop software. 3. Developers do not have the benefits and outcomes of working with a usability expert. Personal contact between developer and usability expert is often ignored, leaving little or no space to get any form user feedback and to show that the product does actually not work the way the developers thought it would. 4. Lack of scientific research in this field results in a lack of respect in the validity of the field of usability by software developers, and use of scientific usability metrics the developers can relate to, in order to assess their applications' user interface.

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU16	... The field of HCI emphasizes the need of trained usability specialists to contribute to the development. The users can not act as usability specialists, because they are not trained for developing and ensuring usability, even thought they encounter the usability problems while using the system. However, problematic is that usability specialists do not typically participate in the OSS development, and the OSS developers do not have the knowledge and skills needed.
	... if usability is acknowledged, this typically happens too late in the process.
SU17	User interface design: ... For some reason, Open Source projects seem to have a lot of trouble with user interface design. I suspect that there isn't one single reason for the poor quality of user interfaces, but here are some explanations I've heard roaming the Open Source circles: geeks value integrity over beauty; the gender gap in Open Source communities; it's intuitive to the programmers so why would they fix it? (ser Programming for the self), the belief that a pretty user interface can always be designed later once they're done the real work, the belief that user interface design isn't real work, and several others.
	Documentation: ... Open Source projects tend to have a major problem with providing decent documentation — if they provide any documentation at all. Because they don't have a contractual responsibility to provide this documentation, it's usually intended to be a general guide rather than a complete manual that you could hand to a novice. ... ... anyone who has ever had to debug a problem in Open Source software knows that the answers don't lie in Open Source software documentation: they're found in Usenet articles, bulletin boards and chat logs. Users who can't figure out how to do something runs to *alt.projectx.devel* and asks the same question as hundreds of users before them. Some expert takes pity on their plight and responds to their question, but never documents this answer. So when the next user hits the same problem, the process has to be repeated. Additionally, this is making the fundamentally flawed assumption that users are capable of finding these alternative streams of communication [133], or that they're patient enough and care about the product enough, to bother. Without adequate documentation, Open Source projects are inherently at a disadvantage.
	Feature-centric development: ... With so much emphasis on features and geek cred, fundamental aspects of a programming project go missing... Sometimes it's a lack of focus on the user interface (see User interface design), documentation, coding standards, security, project direction, specified target audience (see Programming for the self), etc.
	Programming for the self: ... A very common problem among software developers (and not just ones working on Open Source projects) is the fallacy that intuitiveness is problem–specific rather than audience–specific: what is easy to them will naturally be easy to everyone else [154]. Therefore, they don't bother simplying anything that they regard as intuitive. This problem is particularly potent in Open Source projects due to members of the community investing time only in building what features they themselves would want to use (see Feature– centric development), and the

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU17 (continuac.)	fact that audiences are often not as well defined as they are in commercial software (because you don't have to literally sell to a specific group). Also, Open Source projects don't have the usability experts that commercial software projects may employ. The result is that Open Source projects are made by programmers for programmers, who then can't understand why the general public would bother with proprietary software when this Open Source tool is working so well for them. Meanwhile, the rest of the world begins to associate "Open Source" with software that's only accessible to the technocratic elite.
	Religious blindness: ... Since the beginning of the hacker age, programmers of all types have been "unfortunately intolerant and bigoted on technical issues" [13]. So when it comes to devout Open Source programmers, there's a strong tendency to immediately reject all proprietary software and anything to do with non–Open Source programs. Because of the philosophical and sociological issues behind the Open Source movement, this resistance is particularly stubborn. While this has the advantage of increasing Open Source software usage amongst programmers themselves, unfortunately it has the side effect of preventing the Open Source community from learning what proprietary software has to teach. Concepts invented in the world of proprietary software are automatically rejected on the assumption that there's nothing that could possibly be learned from those who are competing with their movement.
SU19	Imitation of other applications is not appreciated in this OSS project, as can be concluded from numerous messages criticizing it in the discussion forum.
SU22	Developers are not typical end-users: ... for many more advanced OSS products, developers are indeed users, and these esoteric products with interfaces that would be unusable by a less technically skilled group of users are perfectly adequate for their intended elite audience. Indeed there may be a certain pride in the creation of a sophisticated product with a powerful, but challenging to learn interface.
	Open source projects lack the resources to undertake high quality usability work: OSS projects are voluntary and so work on small budgets. Employing outside experts such as technical authors and graphic designers is not possible. As noted earlier there may currently be barriers to bring in such skills within the volunteerist OSS development team. Usability laboratories and detailed large scale experiments are just not economically viable for most OSS projects. Discussion on the K Desktop Environment (KDE) Usability mailing list has considered asking usability laboratories for donations of time in which to run studies with state of the art equipment. Recent usability activity in several open source projects has been associated with the involvement of companies, e.g. Benson et al. [34], although it seems likely that they are investing less than large proprietary software developers. Unless OSS usability resources are increased, or alternative approaches are investigated (see below), then open source usability will continue to be constrained by resource limitations.

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU22 (continuac.)	Usability experts do not get involved in OSS projects: ... Good usability design draws from a variety of different intellectual cultures including but not limited to psychology, sociology, graphic design and even theatre studies. Multidisciplinary design teams can be very effective, but require particular skills to initiate and sustain. As a result, existing OSS teams may just lack the skills to solve usability problems and even the skills to bring in 'outsiders' to help. The stereotypes of low hacker social skills are not to be taken as gospel, but the sustaining of distributed multidisciplinary design teams is not trivial. ... There are several possible explanations for the minimal or non-participation of HCI and usability people in OSS projects: - There are far fewer usability experts than hackers, so there are just not enough to go around. - Usability experts are not interested in, or incentivised by the OSS approach in the way that many hackers are. - Usability experts do not feel welcomed into OSS projects. - Inertia: traditionally projects haven't needed usability experts. The current situation of many technically adept programmers and few usability experts in OSS projects is just an historical artifact. - There is not a critical mass of usability experts involved for the incentives of peer acclaim and recruitment opportunities to operate.
	The incentives in OSS work better for improvement of functionality than usability: Are OSS developers just not interested in designing better interfaces? As most work on open source projects is voluntary, developers work on the topics that interest them and this may well not include features for novice users. The importance of incentives in OSS participation is well recognised [66][86]. These include the gaining of respect from peers and the intrinsic challenge of tackling a hard problem. Adding functionality or optimising code provide opportunities for showing off one's talents as a hacker to other hackers. If OSS participants perceive improvements to usability as less high status, less challenging or just less interesting, then they are less likely to choose to work on this area. The voluntary nature of participation has two aspects: choosing to participate at all and choosing which out of usually a large number of problems within a project to work on. With many competing challenges, usability problems may get crowded out.
	OSS development is inclined to promote power over simplicity: 'Software bloat' is widely agreed to be a negative attribute. However, the decision to add multiple alternative options to a system may be seen as a positive good rather than an invidious compromise. We speculate that freedom of choice may be considered a desirable attribute (even a design aesthetic) by many OSS developers. The end result is an application that has many configuration options, allowing very sophisticated tailoring by expert users, but which can be bewildering to a novice [138]. ... Thus there is a tendency for OSS applications to grow in complexity, reducing their usability for novices, but with that tendency to remain invisible to the developers who are not novices and relish the power of sophisticated applications.

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU22 (continuac.)	Usability problems are harder to specify and distribute than functionality problems: Functionality problems are easier to specify, evaluate and modularize than certain usability problems. These are all attributes which simplify decentralized problem solving. Some (but not all) usability problems are much harder to describe and may pervade an entire screen, interaction or user experience. Incremental patches to interface bugs may be far less effective than incremental patches to functionality bugs. Fixing the problem may require a major overhaul of the entire interface — clearly not a small contribution to the ongoing design work. Involving more than one designer in interface design, particularly if they work autonomously, will lead to design inconsistency and hence lower the overall usability. Similarly, improving an interface aspect of one part of the application may require careful consideration of the consequences of that change for overall design consistency. This can be contrasted with the incremental fixing of the functionality of a high quality modularised application. The whole point of modularisation is that the effects are local. Substantial (and highly desirable) refactoring can occur throughout the ongoing project while remaining invisible to the users. However, many interface changes are global in scope because of their consistency effects.
	OSS has an even greater tendency towards software bloat than commercial software: ... The process of 'release early and release often' can lead to an acceptance of certain clumsy features. People invest time and effort in learning them and create their own workarounds to cope with them. When a new, improved version is released with a better interface, there is a temptation for those early adopters of the application to refuse to adapt to the new interface. Even if it is easier to learn and use than the old one, their learning of the old version is now a sunk investment and understandably they may be unwilling to re-learn and modify their workarounds. The temptation for the project maintainer is to keep multiple legacy interfaces coordinated with the latest version. This pleases the older users, creates more opportunities for development, keeps the contributions of the older interfaces in the latest version, and adds to the complexity of the final product.
	Commercial software establishes state of the art so that OSS can only play catch-up: Regardless of whether commercial software provides good usability, its overwhelming prominence in end user applications creates a distinct inertia with respect to innovative interface design. In order to compete for adoption, OSS applications appear to follow the interface ideas of the brand leaders. Thus the Star Office spreadsheet component, Calc, tested against Microsoft Excel in [61] was deliberately developed to provide a similar interface in order to make transfer learning easier. As a result it had to follow the interface design ideas of Excel regardless of whether or not they could have been improved upon. ... the underlying code of a commercial system is proprietary and hidden, requiring any OSS rival to do a form of reverse engineering to develop. This activity can inspire significant innovation and extension.

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU22 (continuac.)	By contrast, the system's interface is a very visible pre-existing solution which might dampen innovation — why not just copy it, subject to minor modifications due to concerns of copyright? One might expect in the absence of other factors that open source projects would be much more creative and risk-taking in their development of radically new combinations of functionality and interface, since they do not suffer short-termist financial pressures.
	Design for usability really ought to take place in advance of any coding: ... We speculate that while an OSS project's members may share a strong sense of vision of the intended functionality (which is what allows the bypassing of traditional software engineering advance planning), they often have a much weaker shared vision of the intended interface. Unless the initiator happens to possess significant interaction design skills, important aspects of usability will get overlooked until it is too late. As with many of the issues we raise, that is not to say that CSS always, or even frequently, gets it right. Rather we want to consider potential barriers within existing OSS practice that might then be addressed.
SU23	The lack of HCI expertise in OSS projects: ... Open source projects that follow the archetype of Raymond's [153] developers 'scratching an itch' typically have not included members with significant usability expertise [136][154]. Possible explanations include a shortage of usability engineers and differences in culture between the disciplines of programming and usability [136]. Even large open source projects have had problems in this area: one of the 'lessons learned' in the Mozilla project was to 'ensure that UI [user interface] designers engage the Open Source community' [192]. The lack of human–computer interaction (HCI) expertise in OSS projects is in part due to the volunteer nature of the communit.
	Complexity: Usability bug reports and their subsequent analyses and re-design discussions have the potential to be particularly complex and contentious [195]. We believe that this additional complexity and contentiousness, compared to many functionality bugs, may occur for a number of reasons: - The very existence of certain usability bugs may be viewed as subjective and hence unreliable: ... A given interface element can be confusing or ambiguous to some people but not to others – something we term 'subjective' usability bugs. ... - Usability bugs can cluster in complex ways: Often a usability bug can have similar manifestations in other parts of the interface, and this is important to note if within-application consistency is to be maintained. - Discussions need to involve complex design rationales, multiple, perhaps contradictory goals and design trade-offs: Usability bug analysis and fix discussions can become long, and it can be hard to keep track of all their elements. ... The discussion also touches on a wider OSS functionality-usability-consensus problem, namely, how alternate design solutions are frequently resolved by providing both as alternate options, and this in turn leading to ever more complex configuration options and interfaces.

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU23 (continuac.)	Reporting a usability problem: Even the basic act of reporting a usability bug is more complex than reporting a functionality bug. In the case of the former, almost by definition the user is confused about what happened or what to do next (unless reporting a potential problem that a third party might have). Articulating that confusion can understandably be difficult. Additionally, current bug reporting software, such as Bugzilla, seems to be inadequate for supporting effective usability discussions for various reasons: - Difficulties that a User May Experience with a Graphical User Interface May Not be Easy to Describe Textually. - Some Usability Confusions Have a Dynamic Element. - Proposed Design Solutions are Often Best Explained and Justified Using Non-Textual Means.
	Human Interface Guidelines: HIGs are one way to guide usability work, both design and discussions, about candidate design alternatives. ... In GNOME the HIG can provide an external source of authority to help a group of developers achieve a consensus; observations of usability bugs in the Mozilla Bugzilla suggest that the lack of a well-used set of interface guidelines may contribute to extended discussions. ... A problem with the interface can be legitimated by referral to the HIG. Any disagreement can really only be about whether it does not in fact deviate from the HIG. By contrast, in projects lacking clearly specified guidelines, a problem has to be noted as deviated from the reporter's notion of usability norms, and both the problem and the norm are subject to debate.
SU27	User research: Every project reported that the lack of user research was an issue in achieving better usability. The GIMP, TV Browser, and TYPO3 case studies provided the most descriptive accounts of the effects of user research on their projects. The GIMP project failed to identify clear project goals and a user focus early on. As a result, developers focused on user feature requests which resulted in irrelevant "feature creep". This increased the complexity of the tool without providing an increased overall value. The project was forced to re-engineer the entire application and redefined the purpose and scope of the application. TV Browser reported not knowing enough about their users in order to make informed design decisions. This problem was identified early in the project due to their proactive inclusion of designers. A series of user surveys were conducted to help fill in gaps of information about the users. TYPO3 experienced the effect of a lack of community interest and participation. The usability team was organized and attempted to gather user research data through stake holder surveys to help create user groups and personas. They received poor participation from the community and were unable to complete the activity and so design efforts continued without valuable user research. They speculated that reasons for a lack of participation from the community were that the culture of getting work done may have surpassed the need for understanding their users needs and goals.

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU27 (continuac.)	The other projects also discuss the needs or effects of user research and usability. The GNOME, KDE and OpenOffice.org projects reported missing a shared vision of project goals or audience. The Mozilla case study stresses the importance of user research. The project had failed to gather both functional and user requirements which resulted in wasted resources. NetBeans questions if mailing lists are representative of their true user population, and if not, who their primary users really are.
	Communication channels: FLOSS development is often distributed and most communication occurs over mailing lists, forums, and Internet relay chat (IRC). Design over these mediums is difficult because of the iterative and visual nature of design and the textual nature of the communication protocols. GNOME was dubious as to how useful mailing list and IRC discussions could be. Also, they thought that these communication methods, in addition to a complex bug database, may intimidate non-technical users (such as usability engineers). This can be especially true when developers begin discussing technical details and implementation. Mozilla experienced community and content problems with having a too-open (open-to-public) or too-closed (invite-only) mailing lists. NetBeans reported the problem of fragmented discussions between bug reports and mailings lists, which can also happen across different mailing lists.
	Culture: FLOSS contributors are primarily developers, and it is easy for developers to assess the skill and competency of another developer through experience. It is difficult for usability engineers to prove themselves to developers in the same way since developers do not always understand the processes or results of usability activities [129]. As a result, there is often tension and communication problems when usability engineers and developers try to work together. Trust must be established between the usability engineer and developer in order to have a successful usability relationship [75][129]. There is also a cultural issue of what it means to "do" and what activities are considered useful contributions. The TYPO3 project reported a failure in conducting a user research activity and speculated that the "culture of doing" was stronger than the "abstract notion" of understanding their users (which is a key component in creating usable products). Developers, who can count "doing" by lines of code or number of commits, simply did not see the value of abstract activities such as discussion and research.
	Usability engineers: It can be difficult to get started with user research and the ability to ask the right questions and draw useful conclusions requires experience. Defining user groups and personas takes guidance and experience. Usability engineers are necessary to help conduct user research studies such as surveys and interviews, and to help make sense of the data by summarizing it for developers in user groups and personas. However, participation by usability engineers is rare [132] [165]. There are few incentives for usability engineers to get involved if they are aware of FLOSS at all [136]. Organizations such as OpenUsability were founded to help address these issues by acting as a meeting place where usability engineers and developers can meet on equal ground [17][129].

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU28	... stated that it was an obstacle for OSS's mainstream acceptance that developers with programming backgrounds did not understand the problems of ordinary users.
SU37	First, for mature projects, such as 3DA, BG, and VG, their software already offers sophisticated functionality. For them, the challenge is often providing this functionality in a way that seamlessly integrates with users' workflows. Second, project members engage in an ongoing dialogue with their core users about the software, its functionality, how it is used, and how it should be improved. Similarly, projects continually receive bug reports and feature requests in mailing lists, IRC, and bug tracking databases. These continual streams of user input likely lessen the perceived need to actively discover user needs, especially since the existing bugs and feature requests often outstrip a project's resources. Finally, the limited attention paid to discovering user needs may also be due to a lack of market pressures, coupled with the volunteer nature of the projects. That is, projects are not driven to compete in the marketplace in the same way commercial products are. We examine this issue in greater detail later when discussing motivations for usability.
	... in our study a number of developers reported that they do not regularly use the software they produce, and often lack relevant domain expertise. D8 illustrates this finding: "One of the problems is that we don't really use our own program. We're working on the program because it's an interesting challenge to design an image manipulation application, but we don't usually do a lot of graphics. This is a problem because we just implement something, test it, then maybe never use it again. There is this detach between people using BG daily and us".
SU39	Bug Reporting: Some usability problems can easily be explained textually. However, not all usability bugs can easily be reported textually and graphics can be an invaluable supplement [88]. If not included in the initial report, developers can request reporters to provide a screenshot in order to understand the issue. ... sometimes even a screenshot is not sufficient to uniquely identify the problem.
	Subjective usability bugs: ... In user testing, one frequently encounters many subjective usability bugs, where one test user has a problem, but another does not. In some cases a single user error result is sufficient to persuade a designer that there is indeed a usability bug that needs fixing, but often stronger evidence is needed before a designer will be convinced, especially when others, perhaps a majority, encounter no confusion.
	Heterogeneity in usability discussion: ... the conventional bug report design may be insufficient. A linear temporal sequence of comments may be perfectly adequate for cases where there are relatively few comments or when the comments document a straightforward temporal workflow... In the case of some usability bugs, the process of describing and analyzing the bug may be much more complex and more contested, with different kinds of evidence proposed and debated.

Tabla D.1: Listado de los Problemas de Usabilidad Identificados por Estudio Primario (continuación).

Estudio Primario	Descripción del Problema Dado por el Autor
SU39 (continuac.)	Scarcity of expertise: ... Systems initially developed for use by expert users were gradually adapted for use in more contexts and by people less willing to learn an unduly complex interface. At times we also see frustration in trying to manage discussions between the two very different worldviews of expert developers and end user advocates. This is made harder when usability expertise is scarce in a project. Without a certain critical mass, it can be difficult to establish the legitimacy of usability arguments against countervailing expert-user functionality-centric claims. The usability advocate can feel outnumbered and give up the fight, or be crowded out in discussions, particularly OSS online discussions where ideally a consensus emerges, but where consensus-based interface design may not always be possible or even desirable. With limited numbers and isolation, it is likely that the progress of usability in any given project will depend crucially on the approach (both in analysis and design, but also in rhetoric) of the few usability advocates involved.
SU41	... comparing usability issues in the context of programming bugs is a risk, since a bug that makes the system crash obviously will get a high priority compared to a 'cosmetic' problem in the user interface.
	... They also mention that current bug databases do not have sufficient categories to fully describe usability issues.
SU43	... typically there are no resources for UCD in OSS development.
	... no UCD methodology is typically employed, because this can be seen as being in contrast with the 'open source philosophy'; it is assumed that in OSS development there is no possibility for systematic UCD or formal process models.
	Typically, the developers do not have knowledge about the non-developer users, their goals and their contexts of use, and they do not necessarily have any interest in learning about them either.
	... communicating usability problems to the development has proven to be difficult for the non-developer users.
	Furthermore, usability specialists would be very useful in improving OSS usability, but typically there is a lack of usability specialists in OSS projects.
SU44	... typically there are no resources for UCD in OSS development.
	... no UCD methodology is typically employed, because this can be seen as being in contrast with the 'open source philosophy'; it is assumed that in OSS development there is no possibility for systematic UCD or formal process models.
	Zhao and Deek highlight the problem of reporting bugs by OSS users. They observe that when an average user wants to report an error, particularly related to usability, s/he does not know how to do it effectively.

ANEXO E

TÉCNICAS DE USABILIDAD INCORPORADAS EN OSS POR ESTUDIO PRIMARIO

El presente anexo tiene por objetivo listar las técnicas de usabilidad incorporadas por OSS que hemos identificado en los estudios primarios. La Tabla E.1 presenta para cada uno de los estudios primarios el nombre de la técnica usado por los autores OSS.

Tabla E.1: Listado de Técnicas de Usabilidad por Estudio Primario.

Estudio Primario	**Nombre de la Técnica Usado por los Autores OSS**
SU1	User interface guidelines
	Usability evaluation in laboratory
	Inspection by usability expert
	Synchronous remote usability evaluation
	Follow common usability
SU3	Writting specifications
	User interface specification document
	Human interface guidelines
	Conference
	Discussion
	Design worskhops
SU4	Usability heuristic
	Personas
SU10	Get feedback from early adopters
	Heuristic evaluation
	Expert review
	Focus groups
	Interface guidelines
	Usability inspections
	Paper prototyping
	Usability tests
	Hunting seasons
	Personas

Tabla E.1: Listado de Técnicas de Usabilidad por Estudio Primario (continuación).

Estudio Primario	**Nombre de la Técnica Usado por los Autores OSS**
SU15	Think aloud
	Rapid prototype
	Cognitive walkthrough
SU16	Users assumed to report bugs
SU20	Competitive analysis
	Dialogue and interaction design
	Site visits
	Usability test
SU23	User human interface guidelines
	Report usability problems
	Design-by-blog
SU24	Usability heuristic
	Personas
SU25	Prototyping of the new design
	Usability testing of the new design
	Brainstorming
	Evaluation of the original design
SU26	Blogs about user interface & usability
SU29	Heuristic evaluations
	Audio recording
	Video recording
	Interviewed the test persons after the usability testing sessions
	Cognitive walkthrough
	Descriptions of the usability problems in the wiki
	Laboratory usability testing
	The student usability teams
SU36	Surveys
	Usability heuristic
	User and task observations
	Identify user profiles
SU37	Through surveys of user base
	Directed program asking for user
	Interviews of users
	Drawing up specifications
	Via expert reviews, performed remotely by UX members
	Via dedicated UX people
	By developing and applying UI design patterns
	Use of human interface guidelines
	By finding inconsistencies in the “rules” of the interface
	By conducting think-aloud studies
	Defining a target user group
	Creating scenarios of user to guide design
	By performing usability studies in controlled settings
	Via “reference users”, “bleeding edge” users of nightly builds, and professional users

Tabla E.1: Listado de Técnicas de Usabilidad por Estudio Primario (continuación).

Estudio Primario	**Nombre de la Técnica Usado por los Autores OSS**
SU37 (continuac.)	By giving tutorials on the software
	Annual, monthly, weekly, or *ad hoc* meetings
	By paying attention to what is asked, discussed, or requested in IRC, mailing lists, and forums
	Through informal observations of friends and family
	Creating personas of users
	By discovering what doesn't work for them as they develop the software
	Through bug reports
	Seeking feedback on mock-ups, prototypes
	Conference
	Interface brainstorming wiki
	Bloggins about designs, getting feedback from user base
	Through open content projects
	Open content projects
	Participation in the season of usability
	Discussions on IRC and mailing lists
	Running design clinics at conferences
	By fixing things with solutions that are "logically" better
	Reliance on the larger community of users to fill in gaps in expertise
	Through usability "champions" in the projects
SU38	Design before coding

ANEXO F

CATÁLOGO DE TÉCNICAS IPO

El objetivo de este anexo es presentar el catálogo de técnicas de usabilidad de la IPO utilizado para estructurar nuestro análisis de las técnicas usadas en los desarrollos OSS. Las Tablas F.1, F.2 y F.3 listan las técnicas de usabilidad de la IPO relacionadas con ingeniería de requisitos, diseño y evaluación, respectivamente.

Para cada técnica se especifica la actividad de desarrollo de la IS con la que está relacionada, el nombre genérico de la técnica, el nombre dado por los diferentes autores en la literatura de la IPO (en itálica las variantes de la técnica genérica) y las referencias respectivas. Nótese que las actividades marcadas con un asterisco (*) no son propias de la IS, sino actividades de la IPO incluidas para ofrecer una visión estructurada de las técnicas que pueden aplicarse en educción y análisis de requisitos y diseño.

Tabla F.1: Técnicas IPO Relacionadas con Actividades de Ingeniería de Requisitos en el Proceso de Desarrollo de Software (adaptada de [68]).

Tipo de Actividad IS		**Nombre Genérico de la Técnica en IPO**	**Nombre Dado por los Autores IPO**	**Ref.**
Educción y Análisis de Requisitos		Análisis Competitivo	Análisis Competitivo	[50]
		Análisis de Impacto Financiero	Análisis de Impacto Financiero	[50]
		Investigación Contextual	Investigación Contextual	[92][149]
			Entrevistas Contextuales	[120]
		Diagramas de Afinidad	Diagramas de Afinidad	[120]
		Observación Etnográfica	Etnografía	[149]
			Observación Etnográfica	[180]
		JEM	JEM (Joint Essential Modeling)	[50]
		Card Sorting	Card Sorting	[50][138]
	* Análisis de Usuarios	Perfiles de Usuario	Perfiles de Usuario	[92]
			Características de Usuario Individuales	[138]
			Perfiles de Uso	[180]
			Modelo Estructurado de Roles	[50]
			Cuestionarios Perfiles de Usuario	[120]
		Mapa Roles de Usuario	Mapa de Roles de Usuario	[50]
		Modelo Operacional	Modelo Operacional	[50]
			Capacidades y Restricciones de Plataforma	[120]
		Personas	Personas	[52]
	* Análisis de Tareas	Casos de Uso Esenciales	Casos de Uso Esenciales	[50]
		HTA	HTA (Hierarchical Task Analysis)	[149]
		Familia de Modelos GOMS	GOMS (Goals, Operations, Methods and Selection Rules)	[138][149] [180]
		Modelo de Interfaz Objeto-Acción	Modelo de Interfaz Objeto-Acción	[149]
		Escenarios de Tareas	Escenarios de Tareas	[120]
		Task Sorting	Task Sorting	[120]
	* Desarrollo del Concepto del Producto	Escenarios y Storyboards	Escenarios, Storyboards e Instantáneas	[149]
			Escenarios	[180]
			Escenarios y Storyboards	[50]
		Tormenta de Ideas Visual	Tormenta de Ideas Visual	[149]
	* Prototipado	Prototipado	Prototipado	[92][138]
			Prototipos Escenario	[138]
			Prototipos de Papel	[50][120][149]
			Prototipos Activos	[50][120][149]
			Prototipos Guiados	[149]
			Prototipos Mago de Oz	[149]

Tabla F.1: Técnicas IPO Relacionadas con Actividades de Ingeniería de Requisitos en el Proceso de Desarrollo de Software (adaptada de [68]) (continuación).

Tipo de Actividad IS	**Nombre Genérico de la Técnica en IPO**	**Nombre Dado por los Autores IPO**	**Ref.**
Especificación de Requisitos	Especificaciones de Usabilidad	Especificaciones de Usabilidad	[92][149]
		Objetivos de Usabilidad	[120][138]
Validación de Requisitos	Evaluación Heurística	Evaluación Heurística	[50][92][120] [138][149] [180]
	Inspecciones	*Inspecciones de Conformidad con Estándares*	[50][120][149]
		Revisión de Guías	[120][180]
		Inspecciones de Consistencia	[50][120][149] [180]
		Inspecciones de Usabilidad Colaborativas	[50]
	Recorridos Cognitivos	Recorridos Cognitivos	[50][120] [149][180]
	Recorrido Pluralístico	Recorrido Pluralístico	[50][120][138] [149]

Tabla F.2: Técnicas IPO Relacionadas con Actividades de Diseño en el Proceso de Desarrollo de Software (adaptada de [68]).

Tipo de Actividad IS		**Nombre Genérico de la Técnica en IPO**	**Nombre Dado por los Autores IPO**	**Ref.**
Diseño	* Diseño de la Interacción	Representaciones de Pantallas	Escenarios y Representaciones de Pantallas	[92]
		Guía de Estilo del Prod.	Guía de Estilo del Prod.	[120]
		Gramáticas	Gramáticas	[180]
		UAN	UAN (User Action Notation)	[92][180]
		TAG	TAG (Task-Action Grammars)	[180]
		Arboles de Menús	Arboles de Menús	[180]
		Diagramas de Transición de Estados de la Interfaz	Diagramas de Transición de Estados de la Interfaz	[92][180]
		Diagramas de Estados de Harel	Diagramas de Estados de Harel	[180]
		Modelo del Contenido de la Interfaz	Modelo del Contenido de la Interfaz	[50]
		Mapa de Navegación	Mapa de Navegación entre Contextos	[50]
		Prototipado	Prototipado	[92][138]
			Prototipos Escenario	[138]
		Diseño Integrador	Diseño Integrador	[50]
		Diseño Paralelo	Diseño Paralelo	[138]
		Análisis de Impacto	Análisis de Impacto	[92][138][149]
		Organización de la Ayuda según Casos de Uso	Organización de la Ayuda según Casos de Uso	[50]

Tabla F.3: Técnicas IPO Relacionadas con Actividades de Evaluación en el Proceso de Desarrollo de Software (adaptada de [68])

Tipo de Actividad IS	**Nombre Genérico de la Técnica en IPO**	**Nombre Dado por los Autores IPO**	**Ref.**
Evaluación por Expertos	Evaluación Heurística	Evaluación Heurística	[50][92][120] [138][149] [180]
	Inspecciones	*Inspecciones de Conformidad con Estándares*	[50][120][149]
		Revisión de Guías	[120][180]
		Inspecciones de Consistencia	[50][120] [149][180]
		Inspecc. Usab. Colaborat.	[50]
	Recorridos Cognitivos	Recorridos Cognitivos	[50][120] [149][180]
	Recorrido Pluralístico	Recorrido Pluralístico	[50][120] [138][149]
Test de Usabilidad	Pensar en Voz Alta	Toma del Protocolo Verbal Concurrente	[92]
		Pensar en Voz Alta	[50][138][149]
		Test Formales de Usab. (en las etapas iniciales)	[120]
		Interacción Constructiva	[138]
		Test Retrospectivo	[50][92][138] [149]
		Toma de Incidentes Crít.	[92]
		Método de Entrenamiento	[138]
	Medición del Rendimiento	Tareas de Referencia	[149]
		Métricas de Rendimiento	[50]
		Test Formales de Usab. (en etapas avanzadas)	[120]
	Información Post-Test	Información Post-Test	[50]
	Test de Usabilidad en Laboratorio	Test en Laboratorio	[50][92]
		Laboratorios de Usabil.	[138]
		Test de Usab. y Lab.	[180]
	Test de Campo	Test de Campo	[50][92]
	Grabación Vídeo	Grabación Vídeo	[92][138][149]
	Grabación Audio	Grabación Audio	[92]
		Protocolo Verbal	[149]
	Registro del Uso	Instrumentación Interna de la Interfaz	[92]
		Registro del Uso	[138]
		Registro Software	[149]
		Registro Continuo del Rendimiento del Usuario	[180]
		Registro de Pulsaciones en el Tiempo	[149]
		Registro de la Interacción	[149]
	Eval. por Control Remoto	Eval. por Control Remoto	[120]
	Test Remoto por Video-Conferencia	Test Remoto por Video-Conferencia	[120]

Tabla F.3: Técnicas IPO Relacionadas con Actividades de Evaluación en el Proceso de Desarrollo de Software (adaptada de [68]) (continuación).

Tipo de Actividad IS	**Nombre Genérico de la Técnica en IPO**	**Nombre Dado por los Autores IPO**	**Ref.**
Estudios de Seguimiento de Sistemas Instalados	Observación Directa	Observación Directa	[138][149]
		Observación Aleatoria	[120]
	Cuestionarios y Encuestas	Cuestionarios	[138]
		Cuestionarios y Encuestas	[149]
		Encuestas	[180]
	Entrevistas	Entrevistas	[138][149] [180]
		Entrevistas Estructuradas	[92][149]
		Entrevistas Flexibles	[149]
	Focus Groups	Focus Groups	[138][180]
	Registro del Uso	Instrumentación Interna de la Interfaz	[92]
		Registro del Uso Real	[138]
		Registros Software	[149]
		Registro Continuo del Rendimiento del Usuario	[180]
		Evaluación Remota Instrumentada	[120]
		Registros de Pulsaciones en el Tiempo	[149]
		Registro de la Interacción	[149]
		Monitores Software	[120]
	Retroalimentación del Usuario	Retroalimentación del Usuario	[138]
		Buzón de Sugerencias o Reporte de Errores en Línea	[180]
		Servicion de Atención al Usuario en Línea	[180]
		Foros	[180]
		Revistas y Conferencias para Usuarios	[180]
		Evaluación Remota Semi-Instrumentada	[120]