UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

IDENTIFYING DEVELOPERS' HABITS AND EXPECTATIONS IN COPY AND PASTE PROGRAMMING PRACTICE

Máster Universitario en Investigación e Innovación en Inteligencia Computacional y Sistemas Interactivos

Autora: Luqi Guan

Directores: Silvia Teresita Acuña Castillo
John Wilmar Castro Llanos

Madrid
Julio de 2020

Acknowledgement

First of all, I want to thank tutor Dr. Silvia Teresita Acuña because of her huge help during this research work. Although I had a little communication barrier due to language problems, she still patiently communicated with me and guided my research. I really learned a lot of things from her. I thank her not only for her help in my study, but also for her concern for my life. She gave me confidence and made me feel warm in a foreign country. It was also with her encouragement that I was able to successfully complete my studies. During this research, she arranged for me every task to be completed next. The whole process was clear and rigorous. This is also a place worth learning.

In addition, I also want to express my gratitude to Dr. John Wilmar Castro because he always shares his knowledge on this area. It was his serious working attitude, detailed comments, patient guidance and strong support that my research was successfully completed. He really did a lot in this research. I admire his rigorous and serious professionalism, and his patience and care for students.

Abstract

Background: Both novice and experienced developers rely more and more in external sources of code to include into their programs by copy and paste code snippets. This behavior differs from the traditional software design approach where cohesion was achieved via a conscious design effort. Due to this fact, it is essential to know how copy and paste programming practices are actually carried out, so that IDEs (Integrated Development Environments) and code recommenders can be designed to fit with developer expectations and habits.

Objective: There are two main purposes of this study. The first one is to identify the role of copy and paste programming or code clone in current development practices and to know how developers use copy and paste. The second one is classifying secondary studies, which are with respect to copy and paste programming and to answer some questions about the quality of these studies and challenges of copy and paste programming.

Method: There are two Systematic Mapping Studies (SMSs) have been conducted, searching the main scientific databases. The first one is for the primary studies of the area of copy and paste programming. The search retrieved 1,271 citations and 39 articles were retained. The second one is for the secondary studies (systematic reviews) of the area of copy and paste programming. The search retrieved 65 citations and 5 articles were retained.

Results: The primary studies were categorized according to eight areas: General information of usage of clone, developer behavior, technologies and tools of code clone detection, technologies and tools of code clone reuse, patterns of cloning, clone evolution, effects of the code clone in the software maintenance and development, and tools of clone visualization. The secondary studies were categorized according to three areas: systematic review on clone detection, systematic review on clone evolution, and systematic review on software cloning.

Conclusions: The areas, techniques and tools of clone detection and developer behavior are strongly represented in the sample. The areas that have been least studied in the literature found in the SMSs are tools of clone visualization and patterns of cloning. The main challenges of copy and paste are the immaturity of existing clone detection technology or tools and clone management.

Keywords: Copy and Paste, Secondary Study, Tertiary Study

Table of Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

The framework of this study is copy and paste programming, which raises two Systematic Mapping Studies (SMSs) on the primary studies and secondary studies of copy and paste. In this chapter, we first outline the research topic. Secondly, it involves the field of research. Then, the research purpose and solution are presented separately. Finally, we introduce the structure and the contribution of the work.

1.1. Overview

The huge amount of available source code online has changed coding practices [Rouse, 2019]. Both novice and experienced developers rely more and more in external sources of code to include into their programs by copying and pasting code snippets [Yarmish and Kopec, 2007] [Pittenger, 2019]. In software engineering, it is basically a term used. It replicates the code and reuses the code by making several modifications or not in the existing code, which is a general activity in software engineering [Vashisht et al., 2018].

Copy and paste is usually done by novice or student programmers who think it difficult or annoying to write code from scratch and prefer to look for pre-written solutions or parts of solutions, which they could use to solve their own problem [Yarmish and Kopec, 2007]. Copy and paste is also completed by experienced software developers, who usually have their own libraries that are well tested, easy-to-use code fragments and general algorithms that can be easily adapted to certain tasks [Pittenger, 2019]. This behavior differs from the traditional software design approach where cohesion was achieved via a conscious design effort [Taylor et al., 2009]. It also differs from the code reuse attained through the usage of re-use repositories built for such specific purpose. We need to know how this copy and paste programming practices are actually carried out, so that IDEs (Integrated Development Environments) and code recommenders can be designed to fit with developer expectations and habits.

The research work aims to identify the role of copy and paste programming or code clone in current development practices, by identifying through a set of Systematic Mapping Studies [Petersen et al., 2008] the current knowledge about this topic in the existing literature, telling apart the works that have actually carried out some empirical study from those who have not.

1.2. Field of Research

In the domain of software engineering, there is not an appropriate definition of the terminology “code clone”. Baxter et al. [1998] have put forward a point of view:
“Code clone is a piece of code that is similar regarding some definitions of similarity.”

During the process of copy and paste programming, if bug is identified from partial code, it needs to be corrected in all copied sections. Therefore, all relevant segments must be identified in the source code. Several studies have shown that nearly 20-50% of large software systems is made up of cloning code [Baker, 1995]. For many reasons, code cloning is regarded as a problem. In the area of software maintenance, the cloned code may lead to higher maintenance expenses because inconsistent modifications of code may bring malfunctions and wrong program behavior [Shippey et al., 2012].

Novice software developers copy and paste code usually because they do not know how to write code themselves. In this way, the problem arises more because of their lack of experience and lack of programming courage than because of the copy and paste behavior itself. The code usually comes from different sources, such as the code of a friend or colleague, an Internet forum, the code offered by the professor/TA, and a computer science textbook. The outcome may be style conflicts, and there may be redundant code to solve problems that no longer require a new solution.

Another problem is that postulations and design decisions produced in specific resources may not be applicable in a new context, so that there would be errors in the process of copy and paste.

In fact, when people use code cloning, they often keep the names of variables, functions or classes in the original code segment inadvertently, even though the names represent something completely different in a new environment. Such code may actually be unconsciously confused [Yarmish and Kopec, 2007].

Copy and paste programming may also be caused by a lack of understanding of common functions (such as loop structures, functions, and subroutines) in computer languages [Müller et al., 2018].

1.3. Research Purpose

Firstly, we need to know how copy and paste programming practices are actually carried out, so that IDEs and code recommenders can be designed to fit with developer expectations and habits.

Secondly, we need to study those existing systematic reviews of the area of copy and paste programming, and get some useful information from them.

Therefore, two research problem have been proposed in this research work. The first one is conducting a secondary study of those studies of copy and paste to get a summary of the recent status in this domain. The secondary problem is conducting a tertiary study of the systematic reviews of copy and paste to supplement our research.

1.4. Research Method

In order to carry out this work, we have retrieved the publications related to the area of copy and paste. For this, we use a famous review method called Systematic Mapping Study (SMS). SMS allows review of documents in specific areas of interest [Kitchenham et al., 2011].

The first SMS is for the secondary study, which aims to reply the next questions:

(RQ1) What is the current status of copy and paste?
(RQ2) How do developers use copy and paste?

The search for studies was carried out in 3 digital databases: Scopus, ACM Digital Library, and IEEE Xplore. In addition, the retrieve period is set to start in January 2015 and end in October 2019. Obviously, the concept of copy and paste has been proposed for many years, and there are so many literatures related to this area. However, only the most recently published studies can represent the current level of the field. Therefore, we changed the search period to nearly five years and it starts at 2015.

The second SMS is for the tertiary study, which is with the goal of replying the next questions:

(RQ3) What are the major research areas in the secondary studies?
(RQ4) What are the measurements of the quality of the secondary studies?
(RQ5) What challenges of the practice of copy and paste are outlined in the published works?

Since there are not many works about the secondary study of this topic, all the published literatures are considered.

1.5. Work Structure

Our research introduces the different areas of the studies of copy and paste programming and the analysis of some systematic reviews of copy and paste programming. The chapters are shown below:

- Chapter 1 introduces the research work, which includes the research topic, purpose and method.
- The related works of the research, as well as the research method of the secondary study are presented in detail in the second Chapter.
- The third Chapter shows the analysis and results of secondary study.
- The fourth Chapter outlines the research method and the results of tertiary study.
- Chapter 5 indicates the discussion and validity threats.
- Chapter 6 draws a conclusion of the whole research and propose the possible future works.
- References.
- Appendix A lists the selected key words obtained from control group articles of the primary studies.

1.6. The Contribution of Work

Table 1.1 shows the contributions made by this research work. The task of this research is review of literature. It has been published as a conference paper in SEKE2020.

Table 1.1: The Contribution of work

Task	Contribution/Results	Type	State	Where
Review of Literature	Copy and paste is widely used by developers in software engineering. It is necessary to know how this copy and paste programming practices are actually carried out, so that IDEs and code recommenders can be designed to fit with developer expectations and habits.	Conference paper	P	SEKE2020

International Conference

- **Luqi Guan**, John W. Castro, Xavier Ferré and Silvia T. Acuña. (2020). Copy and Paste Behavior: A Systematic Mapping Study. In *Proceedings of the 32st International Conference on Software Engineering & Knowledge Engineering (SEKE'20)*. KSIR Virtual Conference, Pittsburgh (USA), pp. 463-466. DOI: 10.18293/SEKE2020-130.
  Quality Index: **Core B.**
  Relationship with Master's Dissertation: Chapter 2 and Chapter 3 shows the work.

CHAPTER 2

LITERATURE REVIEW

At the beginning, several literature reviews can provide guidance for our work. We can find some publications in this field through the literature review, and we can also refer to these literature review methods to do our research. This chapter presents some literature reviews of the research topic. Next, a secondary study has been proposed. In secondary study, we perform a Systematic Mapping Study of copy and paste, which includes the state of art of copy and paste and how developers use copy and paste.

In section 2.1., we introduce the studies related to our research. In section 2.2., we outline the research method of secondary study, which is a procedure of systematic mapping study.

2.1. Related Works

During the search for the primary studies of this research, I found that there are four non-systematic reviews related to copy and paste (code clone). The description of each of these literature reviews is presented below.

Wang et al. [2017] proposed various methods used by researchers to research clone evolution and summarized pros and cons of related studies on clone evolution in their literature review. Besides, they introduced two related studies: clone refactoring, and code clone quality assessment (dangerous code clone, code clone stability). Finally, they identified empirical research based on human clone evolution as a key area for future research. Table 2.1 shows the comparison of clone evolution research methods.

Table 2.1: Comparison of clone evolution research method

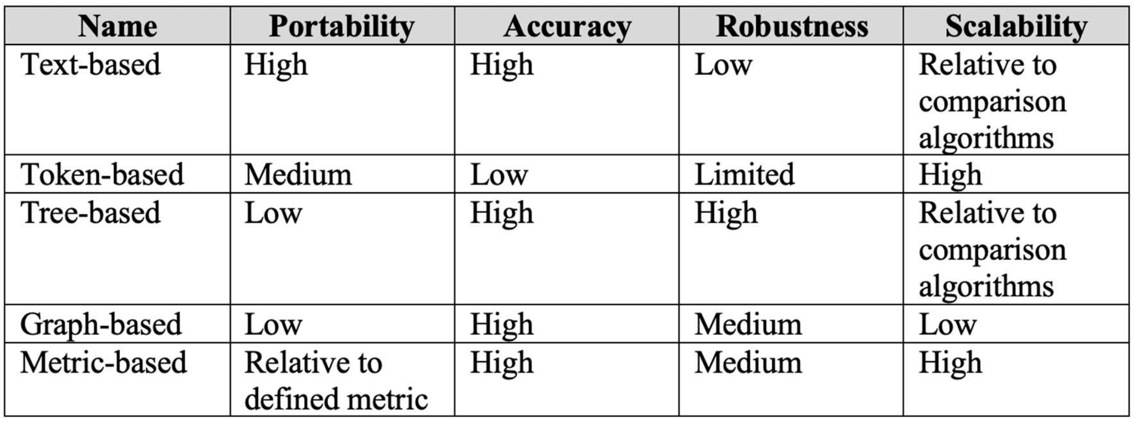
Clone evolution research method	Advantages	Disadvantages
Based on the modified log	Based on the actual clone changes to better correlate the bug trend.	Hard to research the clones increased by versions.
Based on text and position	High space and time efficiency; independent of a certain programming language; simple to extend to other languages; suitable for large scale software systems.	Comparative low detection accuracy.

Table 2.1: Comparison of clone evolution research method (Continuation)

Clone evolution research method	Advantages	Disadvantages
Based on the CRD (Clone Region Descriptors)	The formation of the mapping is not influenced by the location and annotation of the clone information, and it is easy to achieve the consistency change of the clone.	High ratio of false positives.
Based on the topic model	Reduce mapping problems from high-dimensional space to low-dimensional space by using source text and structure information.	High ratio of false positives.
Based on incremental and commit the transaction	Low time complexity, suitable for processing the version that the software has provided.	High space complexity.

Solanki and Kumari [2016] researched code cloning and several techniques for detecting code cloning in their literature review. An inclusive investigation was conducted into the area of code clone detection, focusing on the type of clone, its techniques of detection, and experience evaluation. The results of this research can be used as a guidance for potential users of cloning detection technology to guide them choosing the proper tools and technologies that suit their works. In addition, it could be helpful to identify novel combinations of technologies that are existed and other research questions. Table 2.2 presents the evaluation of different methods of clone detection [Solanki and Kumari, 2016].

Table 2.2: Assessment of clone detection methods



Name	Portability	Accuracy	Robustness	Scalability
Text-based	High	High	Low	Relative to comparison algorithms
Token-based	Medium	Low	Limited	High
Tree-based	Low	High	High	Relative to comparison algorithms
Graph-based	Low	High	Medium	Low
Metric-based	Relative to defined metric	High	Medium	High

Chatley et al. [2016] clarified various techniques for detecting cloning. They also introduced the reasons for cloning, its advantages and disadvantages, and the process of detecting cloning. The reasons for cloning are reuse mechanism, to be completed before the deadline, lack of explanation of requirements, tested code, and coincident problems. The advantages of cloning are fast process, template-based, encourage reuse and disadvantages are increased demand for resources, increased possibility of poor design, maintenance becomes a tedious task, rise in cost and time. The cloning detection process is pre-processing, transformation, matching detection, formatting, post-processing/filtering, and aggregation. Table 2.3 shows the methods for detecting code cloning.

Table 2.3: Methods of code clone detection

Methods	Description
Text-based	On basis of line-by-line comparison.
Abstract-syntax tree based	Convert codes into a tree-based algorithm or tree-based matching.
Token-based	Convert codes into tokens.
Graph-based	On basis of program dependency graph.
Metric-based	Calculate different code metrics. Metrics contain data about the strategy name, format, text, and project control. The parts of the code that show similar metric quality are regarded as clones.
Hybrid	Combine two or more clone techniques.

Saini et al. [2018] summarized a comparative overview of several clone detection technologies. These technologies are token-, metric-, graph-, text-, abstract-syntax tree based, and hybrid. So as to obtain the best results of clone detection, these technologies could be combined with some optimization algorithms. Compared with Type 3 (near missed clone) and Type 4 clone (semantic clone), Type 1 (exact clone) and Type 2 clone (rename/parameterized clone) are easy to identify. Therefore, tools and technologies that could effectively detect Type 3, 4 clones are needed. Besides, there are not many studies related to model clones and clone management, which could be considered as future work.

Most of these non-systematic literature reviews studies related to code clone detection and code clone evolution, they do not refer to developer behavior, techniques and tools of clone reuse, patterns of cloning, tools of clone visualization and effect of the code clone in the software maintenance and development. After analyzing papers that refer to those areas I mentioned before, I can confirm that there is no SMS on these areas of code cloning. Therefore, also it is necessary to study developer behavior, techniques and tools of clone reuse, patterns of cloning, tools of clone visualization and influence of the code clone in the software maintenance and development in code cloning.

2.2. Research Method of Secondary Study

The guidelines of Kitchenham and Charters [Kitchenham and Charters, 2007] given its representativeness in this type of software engineering studies will be used to perform the SMS. Specifically, the following activities will be carried out: (i) formulate the research questions, (ii) define the search strategy, (iii) select the studies, (iv) extract the data, and (v) perform data synthesis.

2.2.1. Formulate Research Questions

The research questions of our research are proposed below: (RQ1) What is the current status of copy and paste? and (RQ2) How do developers use copy and paste?

2.2.2. Define the Search Strategy

It is required to define the search string, the search period, and decide the search sources. Defining the search string is not a simple task and requires several iterations. For the definition of the chain, we will perform the following steps: (i) conformation of the control group (CG), (ii) identification and selection of the keywords, (iii) conformation of the search strings, and (iv) specification of the inclusion and exclusion criteria. Next, we describe each of these steps.

2.2.2.1. Conformation of the control group

According to Zhang et al. [Zhang et al., 2011], a CG is a group of relevant research that allow other similar studies to be established under criteria established in the research questions. From this perspective, the literature CG allows identifying and selecting the words commonly used in the context of the research topic, which constitute the basis for shaping the search chain. To form the CG, we conducted a traditional search to identify papers directly related to our research. After searching, we found a total of 10 papers: [Ahmed et al., 2015] [Chatterji et al., 2013] [Kapser and Godfrey, 2006] [LaToza et al., 2006] [Kim et al., 2004] [Stolee et al., 2009] [Vashisht et al., 2018] [Chatterji et al., 2012] [Zhang et al., 2012] [Balint et al., 2006].

2.2.2.2. Identification and selection of the keywords

In the articles of the CG, the words that appear most frequently must be identified. These papers make up the CG, from which the keywords that we will use in the construction of several search strings are obtained. These different search string options will be tested in Scopus to find the most suitable. The tests will be performed in Scopus because it is the largest database.

The keywords were obtained from a table with the frequency of all the words that appear in the articles of the CG. This table was generated with the help of the Atlas.ti software. The word selection process is made up of two phases. Firstly, three of the researchers in consensus selected the words considering that: (i) they are directly related to the objective and the research questions, (ii) have a higher frequency of use, and (iii) are present in at least 40% of the articles of the CG. In the second phase, each word obtained as a result of the previous phase, were assigned a weight ranging from 0 to 1, determined by the frequency of use of the words and their presence in the articles of the CG, where that word with the highest frequency of use and that was present in all the articles of the CG, had a weight equal to 1. Words that had a very low weight (less than 0.4) were eliminated in a second filter. Table 2.4 presents a fragment of the list of the words obtained as a result of the selection procedure described above (the whole list of words can be found in Appendix A - Table A.1). For each of the words, the percentage of presence in the different studies of the CG (Coverage), the frequency of use of the word (Frequency) and the weight assigned are reported.

Table 2.4: Fragment of the selected keywords list

Keywords	Coverage (%)	Frequency	Weight
Code	100	1,177	1
Clones	90	677	0.74
Clone	100	441	0.69
Software	100	327	0.64
Study	100	204	0.59

2.2.2.3. Conformation of the search strings

Once the keywords were identified, several options were built for the search string. For the construction of these strings the logical AND operator is used to include keywords that belong to different components and the OR operator to include synonyms for words that belong to the same component. We build a total 5 strings. For each of these strings, Table 2.5 shows the number of articles obtained from the Scopus database and the amount of papers of the CG that contains said string.

Table 2.5: Search strings defined

ID	Search string	No. papers found	No. papers CG found
1	(“copy and paste” OR “code snippets reuse” OR “code clone” OR “code cloning”) AND (study OR analysis OR patterns OR techniques OR studies OR design OR approach OR behavior OR context OR method OR tools) AND (systems OR engineering OR software OR development OR developer OR system OR programming OR program)	956	8
2	(“copy and paste” OR “code snippets reuse” OR “code clone” OR “code cloning”) AND (study OR analysis OR patterns OR techniques OR studies OR design OR approach OR behavior OR context OR method OR habits OR intent OR tools) AND (systems OR engineering OR software OR development OR developer OR system OR programming OR engineering OR program)	956	8
3	(“copy and paste” OR “code snippets reuse” OR “code clone” OR “code cloning”) AND (patterns OR techniques OR design OR approach OR behavior OR method OR habits OR intent OR tools) AND (“software system” OR development OR developer OR system OR programming)	729	8
4	(“copy and paste” OR “source code reuse” OR “code reuse” OR “code snippets reuse” OR “code clone” OR “code cloning”) AND (patterns OR techniques OR design OR approach OR behavior OR method OR habits OR intent OR tools) AND (“software system” OR development OR developer OR system OR programming)	1,691	8
5	(“copy and paste code” OR “source code reuse” OR “code reuse” OR “code snippets reuse” OR “code clone” OR “code cloning” OR “software clones”) AND (analysis OR design OR approach OR behavior OR habits OR intent OR research OR patterns OR “usage patterns” OR method OR techniques OR tools) AND (“software system” OR development OR developer OR system OR programming)	1,738	6

As mentioned earlier, the search string tests were performed in the Scopus database. This database contains 8 of the 10 papers of the CG. Search strings have 3 parts. The first part is about copy-and-paste, the second part is about the practice of copy-and-paste and its usage patterns, and the third corresponds to software development. The strings were constructed from: (i) the list of selected keywords, (ii) the habits and intent words suggested by one of the experts in the area, and (iii) the combinations “source code reuse” and “code reuse” also suggested by the expert. Strings 4 and 5 are the ones that have the expert’s suggestions and use the most keywords. Finally, we select string 4 because it is the string that uses the most keywords and finds the highest number of papers in the CG.

2.2.2.4. Specification of the inclusion and exclusion criteria

According to the characteristics and particularities of the articles of the CG, it is necessary to refine the preliminary inclusion and exclusion criteria. The criteria used to retrieve the primary studies are presented below.

Inclusion criteria:

- The paper is related to copy and paste behavior; OR
- The paper discusses aspects related to copy and paste patterns; OR
- The paper is related to code clones; OR
- The paper is related to code clone detection tools; OR
- The paper is about finding duplicated code.

Exclusion criteria:

- The paper is about traditional code reuse; OR
- The paper discusses about creating repository for future reuse; OR
- The paper is about programing for reuse; OR
- The paper is about managing duplicated code; OR
- The paper is a review; OR
- The paper is not written in English.

2.2.3. Select the Studies

The search for studies will be carried out in the 3 digital databases: Scopus, ACM Digital Library, and IEEE Xplore. Table 2.6 shows the different DBs and their search fields. The search fields were determined by the options offered by each database, due to the different query syntaxes [Ren et al., 2019].

Table 2.6: Search strings by DB

DBs	Search fields
Scopus	"Title OR Abstract OR Keywords"
ACM Digital Library	"Abstract"
IEEE Xplore	"Abstract"

Once the list of *retrieved articles* is obtained, it is necessary to eliminate duplicates between the databases and as a result of this first debug, the *candidate studies* are obtained. Then, a first filter must be made applying the assessment criteria according to the title, abstract and keywords of per *candidate study*. Articles obtained from the first filter will be evaluated again in a second filter. In this second filter, each researcher applies the assessment criteria to the full content of per study. As a result, the group of *primary studies* is obtained. Table 2.7 presents a summary for each digital database of the number of articles obtained in each of the groups (*retrieved articles*, *candidate studies*, *primary studies*). The search was conducted in November 2019.

Table 2.7: The number of papers obtained from each DB

Digital Database	Retrieved articles	Candidate studies	Primary studies
Scopus	626	138	28
ACM Digital Library	116	7	9
IEEE Xplore	529	18	2
TOTAL	**1,271**	**163**	**39**

2.2.4. Extract the Data

Once the *primary studies* are obtained, the relevant information is extracted to answer the research questions. When possible, it is needed that more than one researcher extracts the data separately. It is necessary to compare the data that come from different researchers, so that the consensus among researchers could be helpful to eliminate disagreements. Or it may also be a good method to find another researcher to offer a suggestion.

2.2.5. Perform Data Synthesis

The information extracted in primary studies should be consistent with the research questions. The answer should highlight the similarities and differences between the research results to facilitate further analysis.

CHAPTER 3

RESULTS OF THE SECONDARY STUDY

Figure 3.1 presents a summary of the 39 identified primary studies. The results have been segmented into two areas in Figure 3.1. On the left-hand side, there are two scatter (XY) plots with bubbles at the junctions of the year-type of publication categories (top) and topic-type of publication categories (bottom). The types of publication are conferences, workshops, articles, book chapter and symposia. The size of per bubble depends on the number of primary studies that are categorized by survey area. After analyzing the primary studies and papers belonging to the control group, we identified eight different research areas (see Figure 3.1): (i) general information of usage of clone, (ii) developer behavior, (iii), technologies and tools of clone detection, (iv) technologies and tools of clone reuse, (v) patterns of cloning (vi) clone evolution, (vii) effect of the code clone in the software maintenance and development, and (viii) tools of clone visualization. The graph on the right-hand side of Figure 3.1 presents the number of primary studies per year of publication, revealing that interest has been increasing since 2016.

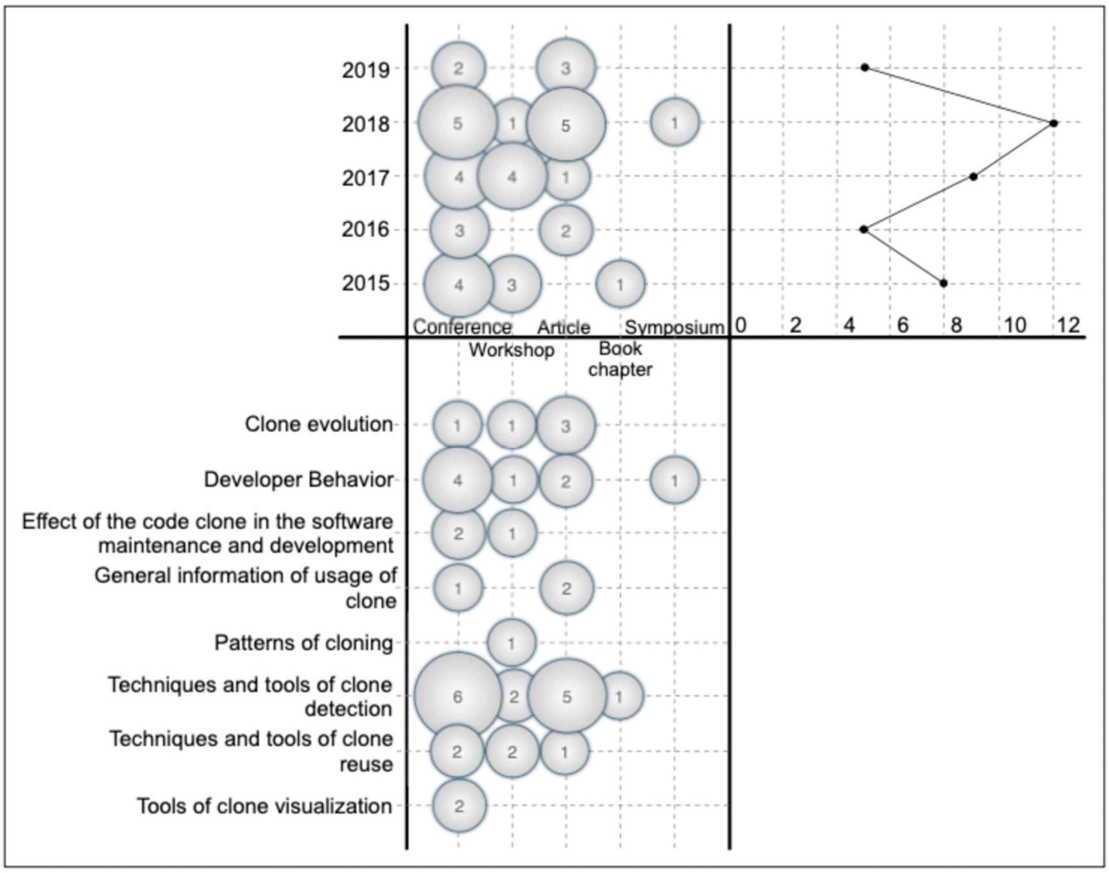


Figure 3.1: Mapping showing the primary study distribution

Table 3.1 describes each research area and lists the respective references. Note that Table 3.1 includes the primary studies and the control group papers. Next, we describe each of the research areas in detail.

Table 3.1: Summary of research areas

Research Area	Description	References
General Information of Usage of Clone	This area deals with clone usage patterns, as well as clone types.	[Chatterji et al., 2012] [Chatterji el al., 2016] [Islam et al., 2016] [Khan et al., 2018] [Kim et al., 2004] [LaToza et al., 2006] [Stolee et al., 2009] [Vashisht et al., 2018] [Zhang et al., 2012]
Developer Behavior	This area focuses on how developers address the use of clones (how they search for and embed clones in their code, etc.).	[Ahmed et al., 2015] [Balint et al., 2006] [Bharti and Singh, 2017] [Chatterji et al., 2012] [Chatterji et al., 2013] [Chatterji el al., 2016] [Ciborowska et al., 2018] [LaToza et al., 2006] [Müller et al., 2018] [Ohta et al., 2015] [Stolee et al., 2009] [Van Bladel et al., 2017] [Xu et al., 2019]
Technologies and Tools of Clone Detection	This area studies the techniques and tools for clone detection.	[Aktas and Kapdan, 2016] [Balint et al., 2006] [Gharehyazie et al., 2019] [Henderson and Podgurski, 2017] [Joshi et al., 2015] [Kamiya, 2015] [Kim et al., 2018] [Mondal et al., 2015] [Mubarak-Ali et al., 2014] [Priyambadha and Rochimah, 2018] [Reddivari and Khan, 2018] [Saini et al., 2016] [Sudhamani and Rangarajan, 2019] [Svajlenko and Roy, 2017] [Vashisht et al., 2018] [Wijesiriwardana and Wimalaratne, 2017]

Table 3.1: Summary of research areas (Continuation)

Research area	Description	References
Technologies and Tools of Clone Reuse	This area studies technologies and tools for clone reuse.	[Abid et al., 2017] [Lin et al., 2015] [Narasimhan et al., 2018] [Ohtani et al., 2015] [Zhang and Kim, 2018]
Patterns of Cloning	This area describes several patterns of cloning, such as forking, templating, and customization.	[Kanwal et al., 2017] [Kapser and Godfrey, 2006]
Clone Evolution	In this area, the cloning community focuses on how the cloning code has evolved over time.	[Chatterji et al., 2012] [Chatterji et al., 2016] [Mondal et al., 2018] [Kanwal et al., 2018] [Nguyen et al., 2018] [Zhang et al., 2017]
Effect of the Code Clone in the Software Maintenance and Development	This area studies the effect of the code clone. It deals with possible maintenance problems caused by cloned codes, as well as the clone display tools and clone patterns and refactoring recommendations to solve such problems.	[Kim et al., 2004] [Lerina and Nardi, 2019] [Mondal et al., 2017] [Wagner et al., 2016]
Tools of Clone Visualization	This area studies the tools of code visualization. These code clone visualization tools are used for checking code and analyzing code clones.	[Mondal et al., 2019] [Murakami et al., 2015]

3.1. General Information of Usage of Clone

This area deals with clone usage patterns, as well as clone types. There is a general classification of clone types:

- Type 1: The code snippets are the same, except for layout, comment changes and blanks.
- Type 2: The structure and syntax of code snippets, except for text, types, layout, identifiers and comment changes, are identical.
- Type 3: The code snippet is a copy with further modifications. Apart from changes in text, types, layout, comments, and identifiers, statements can also be added, deleted or changed.
- Type 4: More than one code snippet conducts the similar computation, albeit using different syntax variants.

This area also describes developer cloning practices from a technical, personal, and organizational perspective. The technical perspective refers to the need for more systematic approaches and better-automated tool cloning support. The personal perspective refers to the skills and experience of developers. The organizational perspective refers to organization and project management enhancement.

Chatterji et al. [2012] explored the extent of agreement on a common classification of clone types and the influence of the cloning proportion on code quality. The results show that there is general agreement on the definition of Type-1 clones, while there is a strong disagreement, in other cases, on the influence of the cloning proportion on the system.

Kim et al. [2004] built a classification of clone usage patterns (e.g., copy and paste the name of a method, a class or a method, where copy was to save what you type, or copy and paste a block or a method, where replication usually creates a clone of the structure and reflects the design decisions in the program). This classification was based on the analysis of why and how programmers use copy-and-paste operations.

LaToza et al. [2006] surveyed code duplication and reported six distinct types of code duplication: (i) repeated work clones, (ii) example clones, (iii) scattered clones, (iv) fork clones, (v) branch clones, and (vi) language clones. Each clone type can be described in terms of the method used for its creation, the reconstruction challenges for deleting the clones, whether the developers know they are creating clones and the size of the clones.

Stolee et al. [2009] identified several usage patterns, which they classed as elementary patterns —between, within—, and complex patterns —repeat, distribution, composition, isolation, relay—. These usage patterns describe how to transfer data in a desktop environment by recording clipboard interactions when end-users perform daily tasks.

Vashisht et al. [2018] included different types of code clones. Type-1, 2, 3 code cloning mechanisms are suitable for text content. Type-4 clone and several parts of Type-3 clone are suitable for functional formats.

Zhang et al. [2012] conducted industrial research on the practice of cloning in large-scale industrial developments from the perspective of technology, individuals, and organizations. Indeed, they found that cloning is not just a technical problem, it has to be addressed from the viewpoint of individuals, organizations, and history. In addition, the study also identified some adjustable factors and break points for further developing current cloning in industrial development practice. From a technological perspective, the adjustable factor is related to a more systematic approach and better automatic cloning tool support. From a human resources perspective, the adjustable factors are related to developer skills and experience. From an organizational perspective, adjustable factors are related to improvements in organizational and project management. They identify two break points in the code cloning life cycle for clone deletion. The first break point is when the tentative clone becomes the baseline clone. The second break point is when the third copy of the cloned code appears. Once these two critical points have been cleared, the clone will almost never be deleted from the system.

Khan et al. [2018] studied 11 different decent-sized web development projects (over 22K LOC on average) based on the same set of requirements, which were coded in Java, PHP, Ruby-on-Rails and C#. They analyzed simple clones and structural clones for evaluating different techniques under the number, coverage and size of clones, the reasons for creating clones, and the proportion of reconfigurable and non-reconfigurable clones. Their analysis shows that the frameworks with the most and least clones are C# and RoR, respectively. C# and RoR have the highest and lowest percentage of reconfigurable clones, respectively. PHP and RoR have the highest and lowest clone coverage, respectively.

There is no significant difference between projects within the same technology by clone size. The percentage of clones is determined by the size of the project, developer methods, and project architecture. The use of design patterns and frameworks helps to control clone generation.

Chatterji el al. [2016] addressed five questions. The first question explored whether the software system clone rate is a possible measure of system quality. Respondents were rather divided (45% said *No* and 55% said *Yes*). Questions 2-5 examined the extent to which respondents agreed on the clone type definitions. Although most respondents agreed on the cloning type definition, disagreement increases as we move up the classification (from Type 1 to Type 4). Opinions diverged most on if the proportion of cloned code is a measurement of system quality or not.

Islam et al. [2016] conducted empirical research on the error duplication according to Type 1 to 3 clones. Their survey of six different systems shows that a considerable part (10%) of clones may have error replication. Type 2 and 3 clones have more replicated errors than Type 1 clones. Therefore, Type 2 and 3 clones ought to be considered more in clone management. In the buggy clone class, there is usually a great deal of error duplication. Besides, they found that nearly half of code clone bugs were replicated errors. Their research shows that replication errors caused by cloning are common. Cloned fragments with if-conditions and method-calls ought to be refactored because such cloned segments may contain replication errors.

3.2. Developer Behavior

This area focuses on how developers address the use of clones (how they search for clones, how they embed clones in their code, etc.). Therefore, research looks at how developers use clones in both software systems development and maintenance. Responses to a survey by Chatterji et al. [2012] indicate that clone evolution information should be used to assist in long-term system quality maintenance tasks that have a broad effect, but is not necessary for short-term or relatively minor maintenance. Chatterji et al. [2013] believe that they can determine cloning intent from interviews with developers in order to develop a different clone classification, which could spawn further research on cloning management tools.

Ahmed et al. [2015] conducted empirical research on the copy-and-paste behavior of IDE users. They observed some distinctions between the copy-and-paste behavior of regular and IDE users. Their conclusions indicate that the Eclipse IDE needs a copy-and-paste support tool tailored for IDE users due to different usage methods. Elementary pattern (between, within, within and between, external paste) analysis shows that there is a clear difference between the behavior of regular and IDE users. They found that IDE users prefer to carry out copy-and-paste operations in the same file more frequently than copy-and-paste operations across different files. IDE users often perform copy and paste between different editor types. The analysis of complex patterns (repeat, distribution, relay, unknown) shows up the main differences between the copy-and-paste behavior of regular and IDE users: ordinary users use more distribution patterns, while IDE users use more relay patterns. In addition, clone detection technology should consider clone positioning between different file types rather than within the same file type.

Balint et al. [2006] associated code cloning with the programmer who made the change and with the modification time to detect how developers copied patterns from each other. They presented the Clone Evolution View tool to gain insight into the way that developers copy. They then conducted a clone analysis on three major case studies (Ptolemy2, ArgoUML, and Ant) to identify several clone patterns: consistent/inconsistent, line/block, with one/several authors.

LaToza et al. [2006] conducted a survey that showed the percentage of developers who made repeated changes, or refactored or eliminated duplicates in multiple locations within a week. Finally, the developers evaluated the trouble caused by each clone with respect to code base maintenance.

Stolee et al. [2009] identified several usage patterns (between and within, classed as elementary patterns; repeat, distribution, composition, isolation, relay, classed as complex patterns) that describe how to transfer data in a desktop environment by recording clipboard interactions when end-users perform daily tasks. Such patterns help to understand the behavior of end-users and points out areas where clipboard support tools could be improved.

Müller et al. [2018] conducted two studies with novice programmers to understand the general aspects of code reuse. They wanted to find out whether novice programmers understand their source code and whether they borrow from it. They summarize the searching approaches applied by novice programmers: based on its domain, based on its operation, from the teacher. They report that novice programmers often need some examples to understand the programming language, comprehend the problem, help them out of sticky situation and optimize the application. They also report how novice programmers use examples: as a reference and for code cloning (as design scavenging if it is simple, as code scavenging if it is complex). They also solved the communication problem. Additionally, they also found that source code reuse may affect programmer understanding of their source code and meta-communication.

Bharti and Singh [2017] highlighted four major problems: source of the copied code snippets, the reason for copy and paste, the type of clone, and cloning ratio. They examined the possible sources of the copied code snippets (the same program as is being coded, other modules of the same system, other software systems within the organization, internet, other sources), reporting that 68.8% of developers usually prefer the option of copying snippets from the same program, whereas the second option of 80.0% of developers is to copy from other modules. Most developers cited three main reasons for copy and paste: missing knowledge, module integration, and system requirements. They examined several types of code clones, reporting that 40.0% of developers extremely agreed with the use of the code snippet logic, however, 80.0% agreed with the use of code fragments with some modifications. They examined several copy-and-paste ranges (one line, multiple lines, function, module), reporting that the first option of 41.2% of developers was to copy only one line of the source code, whereas the second option of 55.6% was to copy more than one line, and the fourth option of 64.7% was to copy the entire module.

Chatterji et al. [2016] asked interviewees eight questions about the behavior/expectations of specific developers, as well as one question about the maintenance of code clones. The results indicate that clone evolution information should be used to assist maintenance tasks that affect long-term system quality. In contrast, short-term or relatively minor maintenance tasks do not need evolutionary information. Interviewees pointed out that the plan can be evolved independently if the cloned fragments have different contexts. They detailed a lot of potential research on the link between developer behavior and cloning. Many interviewees believe that further research is needed regarding the basic understanding of developer behavior using cloning awareness tools. Responses indicate that clone-aware tools and cloning information are most useful for finding and fixing defects and refactoring duplicates.

Ciborowska et al. [2018] studied the behavior of developers using the huge representative micro-interaction data set in IDE. Their analysis of developer behavior from the data set confirmed laboratory research observations that a large number of edits (in some cases quick undo) were made after reusing code from the web, and few tests were performed.

Ohta et al. [2015] introduced analysis and extraction methods for developer copy-and-paste behavior. They applied the code clone detection tool to extract actual source code (actual copy and paste), as well as code snippets (potential reuse) for reuse. They put forward copy-and-paste evaluation criteria. Their standards indicate that, based on a comparison of actual and potential copy-and-paste, actual copy-and-paste can be divided into three classes ("poor", "better" and "best"). They confirmed that 80% of the actual copy and paste in the case study still could be improved (i.e., is classed as "poor" or "better" copy and paste). Their research on actual and potential copy and paste offers a quantitative evaluation.

Xu et al. [2019] explored developer behavior with respect to library reuse and code re-implementation. They identified instances of these behaviors from multiple sources which they then surveyed. They supplemented this research by conducting a manual qualitative analysis on the submission log. The results of the experiment suggest that developers replaced the methods that they had implemented with external library methods mainly because they did not initially understand or lack the introduction of the library. However, developers tend to reuse well-maintained and tested libraries that meet their requirements. On the other hand, developers tend to re-implement the code themselves if the dependencies of library are complex, the library methods used constitute just a little of the total library, or the library methods are deprecated. Xu et al. also list a few points that could enhance the current code suggestion system: detect external code that is partially similar to user code (avoid repetition or re-implementation), tailor recommendations based on user preferences, categorize similar suggestions to help developers to select the suggestions they like, and do not recommend inferior libraries.

Van Bladel et al. [2017] did empirical research to study the number of clones changed during the evolution of the software and clone introduction trends. Generally, the analyzed projects tended to have a rather low cloning density. Besides, they found that developers used clones twice as often as deleting clones. Half of developers introduced few clones every ten times they submitted code. However, a quarter of developers never submitted clones, whereas the remaining 25% of developers preferred cloning.

3.3. Technologies and Tools of Clone Detection

This area studies the technologies and tools for code clone detection (CCD) and the use of clone-aware tools. Balint et al. [2006] described how to automatically detect duplicate fragments in multiple locations, and then described how to add developer information to the analysis. On the word of Vashisht et al. [2018], there are currently a variety of code clone detection technologies, including text-, tree-, token- and metric-based and PDG (program dependent graph) CCD technologies. The most popular CCD tools are Baker's Dup and CloneDR.

Gharehyazie et al. [2019] conducted empirical research on code clone in GitHub. They used a clone detection tool (Deckard) to identify clones of code snippets in the project. They applied network science and statistical methods and various case studies to study code fragment popularity and characteristics. By triangulating the discoveries from various research strategies, they observed that cross-project cloning is normal in GitHub. Besides, they found that ecosystem cloning is based on a concentric, where the first layer contains clones sourced from the same project, the next layer includes clones from projects in the same application space, and the last layer is composed of clones from projects in different areas. Based on these outcomes, they built a clone detection and tracking tool called CLONE-HUNTRESS operating on GitHub. It is built into GitHub, has an easy-to-understand interface that runs effectively on modern database frameworks.

Henderson and Podgurski [2017] proposed another algorithm for testing potential code clones on program dependence graphs employing unweighted random walks on a frequently connected subgraph lattice. As program graphs are hard to mine, the algorithm uses a greedy strategy to prune the subgraph matching search and reduce computational costs. As a result, the procedure can mine huge projects and detect clones fast enough for code inspection running on desktops or continuous integration systems (with at least 500,000 LOC). The proposed algorithm does not use heuristic techniques, nor does it place constraints on the size of the identified frequent subgraphs. Density-based clustering was also performed on the returned clones, and the analysis shows that this method returned significant clusters. They conclude that it is time to reconsider PDG-based clone detection as part of the overall clone management strategy and develop a clone management system that integrates multiple detection methods.

Sudhamani and Rangarajan [2019] presented metric-based methodologies to recognize code clone. They report two metric-based methodologies that identify code clone by looking at control statement (CS) and program features (PF). Method effectiveness was tested on two datasets. They found that these techniques can detect clones that perform similar functions. While the models explore the similarity between projects, they are capable of pinpointing similar text segments across program files.

Aktas and Kapdan [2016] put forward an approach to the structural CCD issue. They reported a new programming design which united various programming quality analysis tools estimating programming measurements for structural CCD. They experimented their approach and compared with the result that obtained by manual identification. The outcomes of two methods were similar. The result of the examination also demonstrates that a uniform structural CCD framework could be based on various programming quality instruments, where each took measures various object-oriented software metrics.

Reddivari and Khan [2018] presented a novel CCD approach which is based on topic modeling. They assumed that clone was contained by artifacts who have similar topic distributions. Then they experimented their method and got the results that the method had a good performance dealing with different clone types and was accurate enough for use in practical applications.

Priyambadha and Rochimah [2018] provided a framework for semantic clone detection. The framework is designed for CCD according to the code behavior, which is detected by observing input, output, and method effects. Approaches with the same input, output, and effect values will be semantically identical. However, input-, output- and effect-based detection methods are not applicable in void or parameterless methods. On the other hand, comprehensive detection is essential. Therefore, challenge is how to detect which variable in a method serves as input, output, and effect. In this case, program dependence graphs are used to detect variable input, output, and effects in void methods. They compare the experimental outcomes of the system with the test outcomes of experts to output the Kappa coefficient of the method proposed in this study. The result of calculating the Kappa coefficient is 0.2128, where the consistency between the system and the experts can be regarded as fair. They also compare the system results with previous methods using the average value of the Kappa coefficient for the previous method and 0.2128. The difference between the two methods is 0.08237. They conclude that the proposed method is more acceptable for experts.

Kamiya [2015] presented a method of CCD based on the execution semantics and arbitrary granularity model of code snippets. Control statements could be defined by developers themselves, i.e., lazy evaluation and lambda. The proposed approach detects instances of Type 3 clones where code snippets transcend procedure and module boundaries. The model could be used as a clone metric (for clone classification) on the basis of the content and context of code snippets in cloned classes. It also can be extended to a uniform approach for clone detection and code query. Preliminary experiments have shown that detection method performance is not yet reliable, as it is highly sensitive to inputs and parameters.

Mondal et al. [2015] proposed a tool named SPCP-Miner that can identify SPCP clones by detecting the history of code evolution in the software system. They conducted an empirical study and confirmed that this tool is valuable in recommending clones that are useful for refactoring and tracking. They believed that the tool could support the clone management.

Saini et al. [2016] presented SourcererCC, which is a token-based CCD tool for Type 1 to 3 clones and uses indexes to realize scalability for large inter-project repositories on standard workstations. SourcererCC utilizes an enhanced reverse index to rapidly search for potential clones. The tool can actively find, and non-intrusively report, method-level clones (between and within projects). They found from their experiments that the tool could effectively detect Type 1 to 3 clones.

Svajlenko and Roy [2017] introduced CloneWorks, a tool for large-scale CCD projects. CloneWorks users can fully customize the source code representation for clone detection, target specific clone types, or conduct custom clone detection experiments. CloneWorks uses improved Jaccard metrics to perform clone detection, and its partitioned partial index method effectively implements subblock filtering heuristics. It can expand Type-3 clones (where similar code snippets differ at statement level or statements are added, modified, and/or deleted) to 250MLOC input in just four hours, with good recall and accuracy.

Wijesiriwardana and Wimalaratne [2017] showed an experimental testing platform that included a group of clone detection components (CDC). CDC is a specific representation of tasks related to cloning detection projects, i.e., extracting the data, preprocessing, and clone detection. These CDCs can be utilized alone to represent easy tasks, or they can be combined to represent complicate tasks. The practicality of the testbed was evaluated on major clone detection experiments conducted on three open-source projects (Apache Wink, Apache Tomcat, and Apache Commons Lang).

Kim et al. [2018] proposed FaCoY, a novel static method that detects code segments that are semantically similar to the users' input code. This method is based on query substitution: the method searches Q&A systems for code segments with similar descriptions but potentially different implementation using a structured code query on basis of a summary of the structure code elements. Afterwards, alternate code queries are generated by utilizing resulting implementations. They conducted several experiments showing that: (i) Compared with online search engines, FaCoY is more effective; (ii) FaCoY is capable of detecting more Type-4 clone than existing technologies; (iii) static FaCoY is able to detect code fragments with similar execution behavior; and (iv) FaCoY is valuable for code or patch suggestions.

Joshi et al. [2015] used data mining techniques to study Type-1 and Type-2 functional clones. First, they collected indicators of all functions in the software system to create data sets. Second, they conducted the DBSCAN clustering algorithm to the data set to analyze each cluster and detect Type 1 and 2 functional clones. They observed that the proposed approach can retrieve a higher proportion of functional clones in the software system. The method was evaluated on the Bitmessage open-source software developed using Python. The case study outcomes show that the best results can be achieved with $\varepsilon=1$ because this value is small enough to maintain high accuracy and large enough to detect a sufficient number of clones. The functional clone detection method that they proposed effectively determined a satisfactory number of clones with higher accuracy values.

Mubarak-Ali et al. [2014] conducted an empirical evaluation of certain CCD tools and systematically organized a large amount of information in this respect. They selected the Java Code Clone Detector (JCCD) as the tool to be tested because it was the latest code clone detection technology and offered a systematic clone detection process. After the experiments conducted in three open-source applications, they conclude that the enhanced generic pipeline model is better than the normal one regarding the clone output and runtime performance.

3.4. Technologies and Tools of Clone Reuse

This area studies clone reuse technologies and tools, for example, where and how to modify the code pasting method, and the code segment merging method by creating correct abstractions, which is the best tool for implementing the developer's code generation method, etc. Lin et al. [2015] reported an interactive method based on cloning to suggest the way to change the pasted code. This method was used to build the proof-of-concept tool CCDemon whose effectiveness was evaluated. The outcomes indicate that this method is able to detect the majority of the positions to be modified in the pasted code and recommend numerous of corresponding modifications. Studies have confirmed that CCDemon supports more effective modification of the pasted code.

Narasimhan et al. [2018] put forward a method to automatically merge similar code segments by building appropriate abstractions. This method provides a variety of abstraction mechanisms that are selected based on research on general open-source repositories. In order to prove the practicality of this method, they proposed a prototype merge tool for C++ and assessed many code clones with slight variations in general open-source software packages. They indicated that maintainers considered that the abstraction created by the algorithm is much more helpful than the existing repeated code.

Abid et al. [2017] developed CodeEase. This prototype tool is an Eclipse plug-in and recommends methods according to the developers' code. Suggestions are on basis of CCD and the analysis of the method clone structure. According to the outcomes of user research, they found that the recommendation based on Type-2 cloning improved the code generation method and recommended a friend method based on MCS (Method Clone Structure). From qualitative data, it appears that users welcome recommendations for friend methods without having to write a definite query. The experimental data analysis found that the CodeEase tool reduced development time (70% users), as CodeEase developers successfully executed error-free code, whereas other developers gave up before they had managed to complete their programming tasks.

Zhang and Kim [2018] describe Grafter, a behavior comparison and test porting method for cloning. To reuse clones, Grafter transplants a clone into its copy by enacting a three-part process. First, it identifies changes in identifiers, types, and methods. Second, it resolves errors caused by changes through code conversion. Third, it inserts a stub code for inspection. Grafter provides difference test to check the differences of behavior between code clones. The report indicated that Grafter is able to well reuse tests and detect differences in behavior.

Ohtani et al. [2015] ran an experiment comparing three keyword-based code query technologies. The first technology recommended the method-level code. The second technology recommended code that had been reused in the past. The third technology suggested code based on past reuse, which was adapted to the recommended code range considering code blocks. They compared these techniques in respect of three points: i) accuracy of the reusable code suggested by the technology, ii) extent to which code reuse helps, iii) the time required to implement the code using these techniques. The third technique scored highest on points i) and ii), but ranked second with respect to implementation time. Experimental results show that method-level code recommendations can provide more accurate reusable code than reuse-level recommendations. However, reuse-level code recommendations are better than method-level recommendations for reusing larger codes.

3.5. Patterns of Cloning

This area describes several cloning patterns, such as forking, templating, and customization, and evaluates the benefits and disadvantages of using cloning, as well as code clone management methods. Kapser and Godfrey [2006] described several cloning modes (forking, templating, and customization) and found that code cloning can often be used beneficially. They described the forking, templating and customization cloning patterns and their purposes: i) forking patterns for larger portions of code with independent evolution of duplicates, with variations at hardware, platform and experimental variations; ii) templating for knowledge target behavior, with boiler-plating, API/language protocols, and general or algorithmic idioms, and iii) customization where there is existing code for similar problems with bug workarounds, and replicate and specialize methods. They discussed the benefits and disadvantages of using cloning and proposed ways to manage these code clones. For example, the API/library protocol pattern has the advantage that users can learn from other codes and reduce workload by copying the code. Its disadvantage is that developers may copy wrong or fragile code, thereby reducing the quality of the code. The model management is to strictly review duplicate items to guarantee the quality of cloned code. These findings support the idea that cloning could be a considerable design decision, and the long-term maintenance of replication should be kept in mind when developing tools.

Kanwal et al. [2017] examined the evolution of code cloning by studying the refactoring patterns applied to code cloning. The results show that a small number of code clones were rebuilt during the release. In most versions (five Java systems, namely JHotDraw, Guava, Jabref, JFreeChart, and Xerces_J), more than 40% of clones were consistently reconstructed. The consistent refactoring of clones means that developers are in many cases aware of clones in the system and deliberately apply this copy-and-paste method.

3.6. Clone Evolution

This area describes the evolution of cloned code by time. As the code changes, they show different patterns and characteristics. For example, a study of clone evolution could indicate which clones have a long lifespan and which clones are easy to change [Pate et al., 2013]. A survey by Chatterji et al. [2012] showed that most interviewees indicated that evolutionary information on cloning may be useful for certain tasks like checking what happened to clones or clone groups over a period of time, finding inconsistent clones, seeing how the code evolved, and so on.

Chatterji et al. [2016] discussed three open-ended questions about the evolution of clones: i) usefulness of clone evolution; ii) impact of system longevity on the clone evolution model; and iii) behavior of developers in propagating clone changes. In response, most interviewees indicated that cloning evolutionary information may be valuable for program understanding, although they did not point out tasks of development and maintenance that may need knowledge acquired by developers' applications. Interviewees considered that the age of the system will affect the clone model due to increased code re-use and increased inconsistency. Finally, survey respondents believe that developers can continue to propagate clones as long as they know the clones.

Mondal et al. [2018] conducted empirical research on the error tendency of different types of code cloning. Besides, they investigated whether cloning error tendencies are mainly related to late spread of code cloning. Based on the statistical test, the number of late spread clones undergoing error repair is much less than the number of non-post spread clones undergoing error repair. Their experimental results show that when considering the clone management, the highest priority should be given to Type-3 clones. They found that consistent changes in error-prone clones and the tendency to follow SPCP can be used to classify clones which are used for refactoring and tracking. Their work indicates that the late spread of code cloning has nothing to do with the error-proneness of code cloning. These findings present the relationship between late propagation and cloning error.

Kanwal et al. [2018] researched the evolution of different versions of structural clones (JHotDraw, Guava, and JFreeChart). They defined the evolutionary patterns of structural cloning and used these evolutionary patterns to extract the structural clone genealogy. Clone genealogies could be classified regarding evolutionary patterns (e.g., consistent/inconsistent genealogies) or life cycles (e.g., dead genealogies and surviving genealogies). The clone genealogy that remains in the system until the last release is called the survival genealogy. The clone genealogy that disappeared during the software development process is called the dead genealogy. They also compared the evolutionary characteristics of structural cloning and simple cloning. Their outcomes indicate that the live clone genealogy is more sustainable than the dead clone genealogy in simple cloning and structural cloning. In a simple clone, the average lifespan of the dead family tree is longer. However, the dead clone family tree represents unsustainable clones which do not live long in the software. In fact, the life span of the dead family at most three versions for structural clones and simple clones in any target system. The analysis reveals that structural clones have a lower change frequency than simple clones. Therefore, they are cheaper to maintain than simple clones. However, there are more maintenance costs in structural clone because of inconsistent changes. Structural clones represent similar levels of design in software, are larger in size, are involved in many software, and are inherently inconsistent. Consequently, they require smarter clone management than simple clones.

Nguyen et al. [2018] focused on clone changes between different versions. After studying the clone groups from seven open-sources applications, they concluded that the number of clones with comparable modifications in each group of revisions were not same. They studied whether there is any relation between the tendency of the project growth and the amount of comparable modifications clone sets. The answer is no.

Zhang et al. [2017] presented a method for clustering and analyzing clones based on the FCM method. They extracted some clone metrics to describe clone and clone evolution from clone detection results and clone genealogy. Besides, they generated the clone clustering vector for each clone, which can be easily clustered. They formulated four research questions and conducted empirical research on six open-source software implementations (DNSJava, jEdit, wget, conky, ProcessHacker, iTextsharp). The research was designed to show up the general relationships between clones and their evolution. Their findings can help developers understand clones. They found that: (i) most clones are short lived and have more clone patterns (inconsistent change, add, subtract, split) than the long-lived clones, especially for the object-oriented programming language; (ii) inconsistent changes occur frequently, especially in the clones that have fewer changes, suggesting that clones that have fewer changes require more attention; (iii) most clones are not coarse grained; and (iv) exact clones account for very few, short-lived clones, and the number of long-lived exact clones is insignificant (they disappear for some reason). This suggests that people should pay more attention to near-miss clones. These conclusions can provide some guidance for developers on clones in software development.

3.7. Effect of the Code Clone in the Software Maintenance and Development

This area studies the effect of code clones. It deals with the maintenance problems that clone codes can cause, as well as clone display tools and clone patterns and refactoring recommendations to solve such problems. Kim et al. [2004] identified software maintenance issues, including short-term maintenance tasks, where programmers immediately modified and pasted into specific parts of the code after copy and paste, and long-term maintenance tasks, where programmers reorganized (refactored) after frequent copy and paste. The general copy-and-paste usage patterns of the code may cause a lot of text, and they propose a set of tools (visualization, structure template extraction, warning/notification, refactoring suggestions) to solve such problems.

Wagner et al. [2016] clarified the relationship between code clones (inconsistent or Type 3 clones) and faults. They conducted an experiment firstly detecting clones. Then, as part of this study, they interviewed three developers. The outcomes showed that 17% of Type 3 clones contained defects. All Type 2 clones with defects evolve into Type 3 clones. They concluded that there are signs that the developers were aware of cloning in two cases, which may cause the fragile relation between the defects and Type 3 clones. Therefore, it is essential to assure that developers know about cloning, possibly with the help of novel tools. Future research needs to study whether it is reasonable to use the failure rate in Type 3 clones as an indicator of defect detection.

Mondal et al. [2017] conducted in-depth empirical research to study the maintenance work needed for non-cloned and cloned code. They implemented a prototype tool that can perform two tasks: (i) calculate how much work has previously changed in a specific approach, and (ii) predict how much work may need to be done for a particular method change in the future. They extracted and analyzed the entire evolutionary history of candidate software systems for the purpose of measurement and prediction. They applied the tool to six open-source systems to calculate the effort spent on non-cloned and cloned code. Regarding the outcomes, (i) the cloned code needs more work in maintenance than the non-cloned code, and (ii) the workload required for Type 2 (with identifier renamed to 'blindrename', dissimilarity threshold '0%') and Type 3 (identifier renamed to 'blindrename', the dissimilarity threshold '20%') is greater than for Type 1 (with an identifier renamed to "none", similarity threshold "0%") clones. According to the survey results, developers need to prioritize Type 2 and 3 clones when considering the clone management.

Lerina and Nardi [2019] set out to quantify the value of technical debt to see if the use of cloned code is higher than debt without cloned code. The Nicad clone detector detected different clone types —Type 1 (identifier renamed to 'none', dissimilarity threshold '0%'), Type 2 (identifier renamed to 'blindrename', dissimilarity threshold '0%'), and Type 3 (identifier renamed to 'blindrename', dissimilarity threshold '20%')—, and the SonarQube platform was used to evaluate the technical debt value. The experimental results revealed that the cloned code actually affects the value of technical debt: The technical debt of files with cloned code is much higher than files without cloned code. The type of cloning will result in different technical debt values, even if the outcomes of the four systems are different.

3.8. Tools of Clone Visualization

This area studies code visualization tools. Code clone visualization tools are used to check code and analyze code clones. Murakami et al. [2015] introduced the Eclipse plug-in called ClonePacker. This application helps programmers to change code fragments and check their clones. ClonePacker receives a group of source files and methods modified by the programmer, and it detects the clone from the source file. Finally, ClonePacker uses circle packing to visualize the inspection results. After comparing with another tool Libra, they proved that ClonePacker for reporting the clone location was much faster.

Mondal et al. [2019] proposed a tool of clone visualization to manage clones. This tool incorporates various zoomable views connected to information, where users could study clones through real-time interaction. They tested the tool on two open-source systems, Carol and Freecol. Natural visual selection allows the selection of a small group of cloned snippets that cover a large portion of clone change events. They observed that the cloned class contained 10 to 20 cloned snippets in several files, which were suitable for reconstruction. They studied the internal and cross-border relationships between the selected clones. They observed that clones with a high common change frequency were preferred in the boundary than the cross-boundary relationship. The parallel coordinate view reveals insights into the way that clones evolved. For example, in the Carol system, they observed that 12 fragments evolved together in four groups. They observed small groups of two to five cloned snippets, which had undergone close changes in many software versions. They also researched the distribution of clones. For Freecol, the file network view recommended a high degree of coupling between different modules. At the end, they checked the source code files covering the clones that changed a lot in order to determine the usage of clones.

CHAPTER 4

TERTIARY STUDY

During secondary study, we found some non-systematic reviews, systematic literature reviews, and systematic mapping studies about copy and paste (code cloning). Non-systematic reviews have been analyzed in section 2.1. However, the quality of the systematic studies has not been analyzed, and no combined studies have been conducted to derive answers about the current state of copy and paste. The intention of Chapter 4 is to classify secondary school studies that are with respect to the field of copy and paste programming and to critically analyze the quality of selected research. So as to achieve this aim, a tertiary study was conducted to classify the selected secondary studies.

In section 4.1., we describe the procedure of the tertiary study. In section 4.2, we analyze the results obtained.

4.1. Research Method of Tertiary Study

The tertiary study will be conducted in accordance with the instructions introduced by [Kitchenham and Charters, 2007] on systematic literature reviews in the area of software engineering. In order to conduct research, specific stages are proposed: plan, conduct and report. Planning is for determining the requirement of review, which includes objectives and research questions, determining search strategies that includes search strings and inclusion/exclusion criteria, which would be fully introduced along the sections below.

4.1.1. Objectives and Research Questions

Tertiary research can obtain information about literature studies on certain topics, and can also obtain information such as the quality, the number, and the emphasis of these publications on the research area. As mentioned above, the primary goal of a master's degree thesis is to classify secondary school research related to copy and paste practice. To this end, it is with the aim of solving the next research questions:

(RQ3) What are the major research areas in the secondary studies?
(RQ4) What are the measurements of the quality of the secondary studies?
(RQ5) What challenges of the practice of copy and paste are outlined in the published works?

4.1.2. Search Strategy

It is essential to define the search string, the search period and decide the search sources. The definition of the search string is not a simple task and requires several iterations. For the definition of the chain, we will perform the following steps: (i) conformation of the control group (CG), (ii) conformation of the search strings, and (iii) specification of the inclusion and exclusion criteria. Next, we describe each of these steps.

4.1.2.1. Conformation of the control group

According to [Zhang et al., 2011], CG is a group of related research, and other similar research can be established according to the criteria established in the research question. To form a CG, we conducted a traditional search to identify papers directly related to our research, which are systematic mapping study or review of copy and paste. As a result of the search, we found two papers: [Rattan and Kaur, 2016] and [Ain et al., 2019].

4.1.2.2. Conformation of the search strings

Table 4.1 presents the number of records obtained in the Scopus database and the amount of papers of the CG that obtains said string.

Table 4.1: Search strings defined

Search string	No. papers found	No. papers CG found
(“copy and paste code” OR “source code reuse” OR “code reuse” OR “code snippets reuse” OR “code clone” OR “code cloning” OR “software clones”) AND (“systematic literature review” OR “systematic review” OR “mapping study” OR “systematic mapping” OR “literature review” OR SLR)	13	2

This database contains all papers of the CG. Search strings have 2 parts. The first part is about copy and paste, the second part is about the systematic review. We can conclude that this string does well for our work.

4.1.2.3. Specification of the inclusion and exclusion criteria

So as to define which research ought to be involved in the tertiary study, inclusion and exclusion criteria are outlined. The assessment criteria are introduced as below:

Inclusion criteria: Studies mention an issue with respect to the defined search string; AND peer reviewed studies; OR secondary studies.

Exclusion criteria: The study is not marked as a secondary study; OR the study is a secondary study, but the subject does not present issue strictly with respect to programming practice of copy and paste; OR duplicated papers; OR the study is not written in English.

We adopted a peer review strategy. After I first searched the search string and selected the studies, the other two researchers and I conducted a second screening of the studies to ensure that the study was within the criteria we defined.

4.1.3. Quality Assessment

In the tertiary study, the quality of each publication will be assessed through the criteria used by [Verner et al., 2012] [Kitchenham et al., 2010]. The standard is based on the following five quality assessment questions (Table 4.2) [Curcio et al., 2019]:

Table 4.2: Quality assessment criteria of tertiary study

Quality Assessment Criteria	
1. Do the authors clearly define the inclusion and exclusion criteria?	**Inclusion and exclusion criteria** Y = yes, the inclusion and exclusion criteria are clearly stated, 1 point; P = partly, the inclusion and exclusion criteria are ambiguous, 0.5 points; N = no, the inclusion and exclusion criteria are not stated 0 points.
2. Is there evidence that huge efforts have been made to conduct each related research?	**Sufficiency of search** Y = yes, the researchers have searched at least 4 digital databases and incorporated other search strategies, or identified and cited each source that focus on this topic, 1 point; P = partly, the researchers have searched 3 or 4 digital databases without adopting an additional search strategy, or they searched a clear but limited set of sources and meeting records, 0.5 points; N = no, the researchers have searched up to 2 digital databases or a very limited set of sources, 0 points.
3. Whether the primary studies are correctly summarized?	**synthesis method** Y = yes, named the explicit synthesis method, and provide a reference to the method, 1 point; P = partly, a synthetic method is named, but no reference to the method is provided, 0.5 points; N = no, no synthesis method is named, 0 points.
4. Has the effectiveness of included studies been fully evaluated?	**Quality standard** Y = yes, the researchers have clearly elaborated quality standards and extracted them from one point of each major research, 1 point; P = partly, the research problem implicates the quality problem solved by the research, 0.5 points; N = no, there is no attempt to make a clear quality assessment of each basic research, or the author has defined quality standards but unused standards, 0 points.
5. Are the primary studies detailed?	**Information provided about primary studies** Y = yes, provide information about all studies, 1 point; P = partly, exclusively providing abstract about the paper, 0.5 points; N = no, unspecified the findings of a single study, 0 points.

4.1.4. Select the Studies

The search for studies will be carried out in the 3 digital databases: Scopus, ACM Digital Library, and IEEE Xplore. Table 4.3 shows the different DBs and their search fields. The search fields were determined by the options offered by each database, due to the different query syntaxes [Ren et al., 2019].

Table 4.3: Search strings by DB

DBs	Search fields
Scopus	"Title OR Abstract OR Keywords"
ACM Digital Library	"Abstract"
IEEE Xplore	"Abstract"

Once the list of *retrieved articles* is obtained, it is necessary to eliminate duplicates between the databases and as a result of this first debug, the *candidate studies* are obtained. Then, a first filter must be made applying the assessment criteria according to the title, abstract and keywords of per *candidate study*. Papers obtained from the first filter will be evaluated again in a second filter. In this second filter, each researcher applies the assessment criteria to all content of each study. Also, the quality assessment will be considered in next filter.

As a result, the group of *final studies* is obtained. Table 4.4 presents a summary for each digital database of the number of articles obtained in each of the groups (*retrieved articles*, *candidate studies*, *final studies*). The search was conducted in June 2020, and the received papers were published before June 2020.

Table 4.4: The number of papers obtained from each DB

Digital Database	Retrieved articles	Candidate studies	Final Studies
Scopus	13	9	5
ACM Digital Library	3	1	0
IEEE Xplore	49	1	0
TOTAL	**65**	**11**	**5**

4.2. Results of Tertiary Study

Along this section, we show the outcomes of the tertiary study, and we also answer the research questions that we proposed previously. Table 4.5 shows the classification of papers. There are five papers that are divided into three areas. They are systematic review on code clone detection, clone evolution, and software cloning.

Table 4.5: Specific areas for secondary studies

ID	Description	References
1	Systematic review on clone detection	[Rattan and Kaur, 2016] [Ain et al., 2019] [Rattan et al., 2013]
2	Systematic review on clone evolution	[Pate et al., 2013]
3	Systematic review on software cloning	[Shippey et al., 2012]

So as to solve the question: (RQ3) What are the major research areas in the secondary studies?, the study carried out in the specific areas is described below.

4.2.1. Systematic Review on Clone Detection

In work of [Ain et al., 2019], they reviewed the newest technologies and tools for detecting code clones. In particular, a systematic literature review (SLR) was conducted to study 54 papers related to CCD. Therefore, six classes are outlined based on relevance to merge these papers: text-, lexical-, tree-, metric-, semantic- based method and hybrid method. In addition, they pointed and studied 26 tools for CCD, inlucding 13 development/recommendation tools and 13 existing tools. In addition, 62 open source theme systems were introduced, and the source code was utilized for CCD. The conclusion is that there have been studies separately examining Type 1 to 4 clones. However, a novel method with complete tool support needs to be developed to jointly detect all types of clones. In addition, while dealing with Type 4 clone detection, more methods need to be introduced to make simpler the development of the program dependency graph (PDG).

In work of [Rattan and Kaur, 2016], they focused on the tools and metric-based clone detection technology. All selected studies are classified on basis of three aspects: metrics, tools, and match detection. These tools include Datrix, eMetrics, and so on, which are used for metric computation, clone deletion, and clone detection. The types of metric for CCD are process, product, project, and object-oriented. The categories of matching detection are clustering, fingerprint recognition, visualization, and classification algorithms.

[Rattan et al., 2013] did a systematic mapping study in the area of software cloning (especially software cloning detection). They studied 213 of the 2,039 articles. The selected papers on software cloning are roughly divided into various groups. These are: (i) an empirical assessment of clone detection tools/technologies, (ii) clone management, its benefits, and cross-cutting nature, (iii) the number of papers involving 9 different types of clones, and (iv) 13 intermediate representatives and 24 match detection technology.

4.2.2. Systematic Review on Clone Evolution

In work of [Pate et al., 2013], they systematically reviewed the studies related to clone evolution. They conducted a detailed research of 30 related studies determined according to their research plan. The review describes three issues: (i) methods of studying clone evolution, (ii) patterns of clone evolution, and (iii) evidence that clones have changed consistently during the period of software evolution. Generally, the outcomes of this review show that there are contradictions among the studies about the lifetime of clone generation and the consistency of changing clones during software evolution.

4.2.3. Systematic Review on Software Cloning

In work of [Shippey et al., 2012], they investigated the purpose, the CCD techniques and the dataset utilized in the research of code cloning between 2007 and 2011. Then they analyzed the state of art of the research on code cloning in order to find techniques for defecting prediction. They selected 220 studies to perform a mapping study. The outcome shows that the major focus of the research is code clone detection technology. The number of studies accepted in journals and conferences has increased by 71% during the past 4 years. The majority of dataset has been used only once, so the conclusion of one study report cannot have a comparison with that of another research report. There are few benchmark data sets that correctly identify clones. Few studies have applied code clone detection to defect prediction.

4.2.4. Classification According to Quality Criteria

This section describes question below:

(RQ4) What are the measurements of the quality of the secondary studies?

All 5 selected papers are considered into a quality assessment. The scores obtained by each assessment criteria are showed below (Table 4.6).

Table 4.6: Quality assessment of selected studies

References	Inclusion/ exclusion criteria	Adequacy of search	Synthesis method	Quality criteria	Information provided about primary studies	Final Score
[Rattan et al., 2013]	1	1	1	1	1	5
[Rattan and Kaur, 2016]	0.5	1	1	0	0.5	3
[Ain et al., 2019]	1	1	1	0.5	1	4.5
[Pate et al., 2013]	1	1	1	0.5	1	4.5
[Shippey et al., 2012]	1	0	1	0	0.5	2.5

From Table 4.6 we can see that the studies of [Rattan et al., 2013], [Pate et al., 2013], [Ain et al., 2019] have high quality. The study of [Rattan and Kaur, 2016] has medium quality with score 3. The study of [Shippey et al., 2012] has lowest quality with 2.5 score.

4.2.5. Challenges Described in Studies

This section describes question below:

(RQ5) What challenges of the practice of copy and paste are outlined in the published works?

The Table 4.7 shows the challenges that we found in 4 papers. There is no challenge mentioned in [Pate et al., 2013].

Table 4.7: Challenges of copy and paste

Challenge	References
The immaturity of existing clone detection technology or tools	[Rattan and Kaur, 2016] [Ain et al., 2019]
Clone management	[Rattan et al., 2013] [Shippey et al., 2012]

The work of [Rattan and Kaur, 2016] is based on studying clone detection techniques. They found that there are limited clone detection tools that can be used easily in an experiment. So, developing more simple tools for metric-based clone detection is necessary.

In work of [Ain et al., 2019], they found that it is very complex to detect the Type 4 clone and there are few studies dealing with the detection of this kind of clone. So, it is needed to develop new techniques and tools for detecting Type 4 clone.

In work of [Rattan et al., 2013], they think developers often need to deal with a lot of data and it is difficult to manage code clone. So, a scalable management tool which can help to understand the behavior of cloning patterns is necessary.

In work of [Shippey et al., 2012], they think the code clones may cause higher maintenance costs due to the error made in the process of copy and paste.

CHAPTER 5

DISCUSSION AND VALIDITY THREATS

The analysis reveals that the areas techniques and tools of clone detection and developer behavior are mainly represented in the sample. The area techniques and tools of clone detection is represented by 14 publications (35.9% of the total), while developer behavior is the second largest set of primary studies, with a total of 8 publications, that is, 20.5% of all of the primary studies retrieved in the SMS (39). The areas that have been least studied in the literature found in the SMS are tools of clone visualization and patterns of cloning. Because there is an increasement in the number of publications since 2016, the practice of copy and paste is of notable interest. However, the areas of tools of clone visualization and patterns of cloning that requires much more research effort.

The tertiary study is based on systematic literary reviews regarding the instructions introduced by [Kitchenham and Charters, 2007]. Among the first 65 papers selected from the well-known research database, 5 studies were remained according to a serious procedure, ranging from research selection to discussions to resolve the discussions conducted in pairs during the selection process. All 5 selected studies have undergone quality assessment. In addition to providing a systematic way that other researchers can replicate, the execution of the entire process also makes the results analysis more confident.

From the analysis of the results of tertiary study, there are 3 systematic reviews (60% of the total) on code cloning detection tools or techniques. There is 1 systematic review on code evolution and 1 systematic review on software clone. Therefore, the most important research area of secondary studies is also clone detection. The studies of [Rattan et al., 2013], [Pate et al., 2013], [Ain et al., 2019] have high quality. The study of [Rattan and Kaur, 2016] has medium quality with score 3. The study of [Shippey et al., 2012] has lowest quality with 2.5 score. The main challenges of copy and paste are the immaturity of existing clone detection technology or tools and clone management.

We identify as possible threats to validity: (i) Coverage of research questions (RQ), (ii) bias towards certain publications, (iii) quality of the evaluation, and (iv) lack of knowledge of the area. It is probable that the proposed RQs could partially cover the study theme, which we try to mitigate by defining a work objective and raising several RQs in consensus, with the purpose of making the objective attainable. It is possible that in an SMS the process will be directed towards a specific group of studies, which we avoid by forming a literature CG and by consensus building a search chain with explicit terms obtained from the CG. It is likely that the quality of the evaluation of the studies was not adequate due to ignorance of the research area, which we mitigate by including in the team an investigator with experience in the subject of code clone.

CHAPTER 6

CONCLUSIONS

This chapter review the contents of the previous chapters to briefly summarize the achievements. It also provides a global view of completed work and provides instructions for future works.

6.1. Conclusions

This work conducted both the secondary study and the tertiary study in order to reply the next research questions:

RQ1. What is the current status of copy and paste?

The research on copy and paste or code clone deals with eight areas: (i) *general information of usage of clone* [Chatterji et al., 2012] [Chatterji et al., 2016] [Islam et al., 2016] [Khan et al., 2018] [Kim et al., 2004] [LaToza et al., 2006] [Stolee et al., 2009] [Vashisht et al., 2018] [Zhang et al., 2012], (ii) *developer behavior* [Ahmed et al., 2015] [Balint et al., 2006] [Bharti and Singh, 2017] [Chatterji et al., 2012] [Chatterji et al., 2013] [Chatterji el al., 2016] [Ciborowska et al., 2018] [LaToza et al., 2006] [Müller et al., 2018] [Ohta et al., 2015] [Stolee et al., 2009] [Van Bladel et al., 2017] [Xu et al., 2019], (iii) *technologies and tools of clone detection* [Aktas and Kapdan, 2016] [Balint et al., 2006] [Gharehyazie et al., 2019] [Henderson and Podgurski, 2017] [Joshi et al., 2015] [Kamiya, 2015] [Kim et al., 2018] [Mondal et al., 2015] [Mubarak-Ali et al., 2014] [Priyambadha and Rochimah, 2018] [Reddivari and Khan, 2018] [Saini et al., 2016] [Sudhamani and Rangarajan, 2019] [Svajlenko and Roy, 2017] [Vashisht et al., 2018] [Wijesiriwardana and Wimalaratne, 2017], (iv) *technologies and tools of clone reuse* [Abid et al., 2017] [Lin et al., 2015] [Narasimhan et al., 2018] [Ohtani et al., 2015] [Zhang and Kim, 2018], (v) *patterns of cloning* [Kanwal et al., 2017] [Kapser and Godfrey, 2006], (vi) *clone evolution* [Chatterji et al., 2012] [Chatterji el al., 2016] [Kanwal et al., 2018] [Mondal et al., 2018] [Nguyen et al., 2018] [Zhang et al., 2017], (vii) *effect of the code clone in the software maintenance and development* [Kim et al., 2004] [Lerina and Nardi, 2019] [Mondal et al., 2017] [Wagner et al., 2016], and (viii) *tools of clone visualization* [Mondal et al., 2019] [Murakami et al., 2015].

Most primary studies and papers belonging to the CG, at 32.2%, deal with techniques and tools of clone detection area and followed by developer behavior (27.1%) area and general information of usage of clone (18.8%) area.

RQ2. How do developers use copy and paste?

There are two main kinds of patterns for using copy and paste have been defined. Elementary patterns include between, within, within and between, and external paste. Whereas complex patterns include repeat, distribution, relay, and unknown. For one thing, the elementary patterns consist of a single copy and paste incident involving one or more files. For another, complex patterns consist of more than two copy and paste interactions involving two or more files [Ahmed et al., 2015].

Among the areas of code clone research identified, one of the areas that most interest us is how developers face the use of clones (how they search, how they embed it in their code, etc.) in order to conduct an experiment with students from software engineering to see how they use them and compare quality, for example with the non-use of clones or with alternative ways of searching/using clones. Therefore, as future work will be deepened in the work on the behavior of developers and how and why they use copy and paste as well as in the empirical studies carried out to define an experimental design on how it affects the behavior of developers. They follow copy and paste practices in the quality of the software developed.

RQ3. What are the major research areas in the secondary studies?

The returned systematic reviews have been divided into 3 areas: (i) *Systematic review on clone detection* [Rattan and Kaur, 2016] [Ain et al., 2019] [Rattan et al., 2013], (ii) *Systematic review on clone evolution* [Pate et al., 2013], and (iii) *Systematic review on software cloning* [Shippey et al., 2012].

Secondary studies belong to clone detection area, at 60%, software clone (20%) area and clone evolution (20%) area.

RQ4. What are the measurements of the quality of the secondary studies?

Based on the criteria used in the tertiary study [Verner et al., 2012] [Kitchenham et al., 2010], the studies of [Rattan et al., 2013] (score 5), [Pate et al., 2013] (score 4.5), [Ain et al., 2019] (score 4.5) have high quality. The study of [Rattan and Kaur, 2016] has medium quality with score 3. The study of [Shippey et al., 2012] has lowest quality with 2.5 score.

RQ5. What challenges of the practice of copy and paste are outlined in the published works?

There are 4 papers discussing the challenges. First challenge is the immaturity of existing clone detection technology or tools [Rattan and Kaur, 2016] [Ain et al., 2019], second challenge is about clone management [Ain et al., 2019] [Shippey et al., 2012].

6.2. Discussion and Future Work

Among the areas of code clone research identified, one of the areas we identify as having a high interest is the study of how developers face the use of clones (how they search, how they embed it in their code, etc.), to experiment with students from software engineering to see how they use them and to compare their quality either with the non-use of clones or with alternative ways of searching/using clones. From tertiary study, we know that the main research question of secondary studies is clone detection. The main challenges of copy and paste are the immaturity of existing clone detection technology or tools and clone management. Therefore, we propose as future work firstly to further study how and why developers use copy and paste, and how the behavior of developers that follow copy and paste practices affect the quality of the software developed, secondly to develop more code clone detection and management tools in order to provide convenience to developers.

REFERENCES

Abid, S., Javed, S., Naseem, M., Shahid, S., Basit, H.A., and Higo, Y. (2017). “CodeEase: Harnessing method clone structures for reuse”, in *Proc. IEEE 11th International Workshop on Software Clones (IWSC’17)*. Co-located with SANER 2017. Klagenfurt, Austria, pp. 24-30.

Ahmed, T.M., Shang, W., and Hassan, A.E. (2015). “An empirical study of the copy and paste behavior during development”, in *Proc. IEEE/ACM 12th Working Conference on Mining Software Repositories*. Florence, Italy, pp. 99-110.

Ain, Q.U., Butt, W.H., Anwar, M.W., Azam, F., and Maqbool, B. (2019). “A systematic review on code clone detection”, *IEEE Access*, Vol. 7, pp. 86121-86144.

Aktas, M.S., and Kapdan, M. (2016). “Structural code clone detection methodology using software metrics”, *International Journal of Software Engineering and Knowledge Engineering*, Vol. 26, Issue 2, pp. 307-332.

Baker, B.S. (1995). “On finding duplication and near-duplication in large software systems”, in *Proc. 2nd Work. Conf. Reverse Eng.* Toronto, Ontario, Canada, pp. 86–95.

Balint, M., Marinescu, R., and Girba T. (2006). “How developers copy”, in *Proc. 14th IEEE International Conference on Program Comprehension (ICPC’06)*. Athens, Greece, pp. 1-10.

Baxter, I.D., Yahin, A., Moura, L., Sant’Anna, M., and Bier, L. (1998). “Clone detection using abstract syntax trees”, in *Proc. of the International Conference on Software Maintenance (Cat. No. 98CB36272)*. Bethesda, MD, USA, pp. 368-377.

Bharti, S., and Singh, H. (2017). “An industrial study on developers’ prevalent copy and paste activities”, in *Proc. International Conference on Next Generation Computing and Information Systems (ICNGCIS’17)*. Jammu, India, pp. 147-152.

Chatley, G., Kaur, S., and Sohal, B. (2016). “Software clone detection: A review”, *International Journal of Control Theory and Applications*, Vol. 9, Issue 41, pp. 555-563.

Chatterji, D., Carver, J.C., and Kraft, N.A. (2012). “Claims and beliefs about code clones: Do we agree as a community? A survey”, in *Proc. 6th International Workshop on Software Clones (IWSC’12)*. Zurich, Switzerland, pp. 15-21.

Chatterji, D., Carver, J.C., and Kraf, N.A. (2013). “Cloning: The need to understand developer intent”, in *Proc. 7th International Workshop on Software Clones (IWSC’13)*. San Francisco, CA, USA, pp. 14-15.

Chatterji, D., Carver, J.C., and Kraft, N.A. (2016). “Code clones and developer behavior: Results of two surveys of the clone research community”, *Empirical Software Engineering*, Vol. 21, Issue 4, pp. 1476-1508.

Ciborowska, A., Kraft, N.A., and Damevski, K. (2018). “Detecting and characterizing developer behavior following opportunistic reuse of code snippets from the web”, in *Proc. IEEE/ACM 15th International Conference on Mining Software Repositories (MSR’18)*. Gothenburg, Sweden, pp. 94-97.

Curcio, K., Santana, R., Reinehr, S., and Malucelli, A. (2019). "Usability in agile software development: A tertiary study", *Computer Standards & Interfaces*, Vol. 64, pp. 61–77.

Gharehyazie, M., Ray, B., Keshani, M., Zavosht, M.S., Heydarnoori, A., and Filkov V. (2019). "Cross-project code clones in gitHub", *Empirical Software Engineering*, Vol. 24, pp. 1538-1573.

Henderson, T.A.D., and Podgurski, A. (2017). "Rethinking dependence clones", in *Proc. IEEE 11th International Workshop on Software Clones (IWSC'17)*. Co-located with SANER 2017. Klagenfurt, Austria, pp. 66-74.

Islam, J.F., Mondal, M., and Roy, C.K. (2016). "Bug replication in code clones: An empirical study", in *Proc. IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER'16)*. Suita, Japan, pp. 68-78.

Joshi, B., Budhathoki, P., Woon, W.L., and Svetinovic, D. (2015). "Software clone detection using clustering approach", in: Arik S., Huang T., Lai W., Liu Q. (eds). *Neural Information Processing*. ICONIP 2015 (pp. 520-527). Lecture Notes in Computer Science, vol 9490. Springer.

Kamiya, T. (2015). "An execution-semantic and content-and-context-based code-clone detection and analysis", in *Proc. IEEE 9th International Workshop on Software Clones (IWSC'15)*. Montreal, QC, Canada, pp. 1-7.

Khan, A., Basit, H.A., Sarwar, S.M., and Yousaf, M.M. (2018). "Cloning in popular server side technologies using agile development: An empirical study", *Pakistan Journal of Engineering and Applied Sciences*, Vol. 22, pp. 1-13.

Kanwal, J., Inoue, K., and Maqbool, O. (2017). "Refactoring patterns study in code clones during software evolution", in *Proc. IEEE 11th International Workshop on Software Clones (IWSC'17)*. Co-located with SANER 2017. Klagenfurt, Austria, pp. 45-46.

Kanwal, J., Basit, H.A., and Maqbool, O. (2018). "Structural clones: An evolution perspective", in *Proc. IEEE 12th International Workshop on Software Clones (IWSC'18)*. Campobasso, Italy, pp. 9-15.

Kapser, C., and Godfrey, M.W. (2006). "Cloning considered harmful considered harmful", in *Proc. 13th Working Conference on Reverse Engineering (WCRE'06)*. Benevento, Italy, pp. 645-692.

Kim, M., Berman, L., Lau, T., and Notkin, D. (2004). "An ethnographic study of copy and paste programming practices in OOPL", in *Proc. International Symposium on Empirical Software Engineering (ISESE'04)*. Redondo, Beach, CA, USA, pp. 83-92.

Kim, K., Kim, D., Bissyandé, T.F., Choi, E., Li, L., Klein, J., and Traon, Y.L. (2018). "FaCoY: A code-to-code search engine", in *Proc. ACM 40th International Conference on Software Engineering (ICSE'18)*. Gothenburg, Sweden, pp. 1-12.

Kitchenham, B., and Charters, S. (2007). *"Guidelines for performing systematic literature reviews in software engineering"*. Tech. Rep., Keele University and Department of Computer Science, University of Durham.

Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O.P., Turner, M., Niazi, M., and Linkman, S. (2010). "Systematic literature reviews in software engineering - A tertiary study". *Information and Software Technology*, Vol. 52, pp. 792- 805.

B.A. Kitchenham, D. Budgen and O. Pearl Brereton. (2011). "Using mapping studies as the basis for further research - A participant-observer case study", *Information and Software Technology*, Vol. 53, Issue 6, pp. 638–651.

LaToza, T.D., Venolia, G, and DeLine, R. (2006). "Maintaining mental models: A study of developer work habits", in *Proc. 28th International Conference on Software Engineering (ICSE'06)*. Shanghai, China, pp. 492-501.

Lerina, A., and Nardi, L. (2019). "Investigating on the impact of software clones on technical debt", in *Proc. IEEE/ACM International Conference on Technical Debt (TechDebt'19)*. Montreal, QC, Canada, pp. 108-112.

Lin, Y., Peng, X., Xing, Z., Zheng, D., and Zhao, W. (2015). "Clone-based and interactive recommendation for modifying pasted code", in *Proc. 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'15)*. Bergamo, Italy, pp. 520-531.

Mondal, M., Roy, C.K., and Schneider, K.A. (2015). "SPCP-Miner: A tool for mining code clones that are important for refactoring or tracking", in *Proc. IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER'15)*. Montreal, QC, Canada, pp. 484-488.

Mondal, M., Roy, C.K., and Schneider, K.A. (2017). "Does cloned code increase maintenance effort?", in *Proc. IEEE 11th International Workshop on Software Clones (IWSC'17)*. Klagenfurt, Austria, pp. 1-7.

Mondal, M., Roy, C.K., and Schneider, K.A. (2018). "Bug-proneness and late propagation tendency of code clones: A Comparative study on different clone types", *Journal of Systems and Software*, Vol. 144, pp. 41-59.

Mondal, D., Mondal, M., Roy, C.K., Schneider, K.A., Wang, S., and Li Y. (2019). "Towards visualizing large scale evolving clones", in *Proc. IEEE/ACM 41st International Conference on Software Engineering: Companion (ICSE-Companion'19)*. Montreal, QC, Canada, pp. 302-303.

Mubarak-Ali, A.-F., Sulaiman, S., Syed-Mohamad, S.M., and Xing, Z. (2014). "Code clone detection and analysis in open source applications", *Open Source Technology: Concepts, Methodologies, Tools, and Applications*, Vol. 4, Issue 4, pp. 1112-1127.

Müller, L., Silveira, M.S., and de Souza, C.S. (2018). "Do I know what my code is saying?: A study on novice programmers' perceptions of what reused source code may mean", in *Proc. 17th Brazilian Symposium on Human Factors in Computing Systems (IHC'18)*. Belém, Pará, Brazil, pp. 1-10.

Murakami, H., Higo, Y., and Kusumoto, S. (2015). "ClonePacker: A tool for clone set visualization", in *Proc. IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER'15)*. Montreal, QC, Canada, pp. 33-39.

Narasimhan, K., Reichenbach, C., and Lawall, J. (2018). "Cleaning up copy–paste clones with interactive merging", *Automated Software Engineering*, Vol. 25, pp. 627-673.

Nguyen, T.L., Fish, A., and Song, M. (2018). "An empirical study on similar changes in evolving software", in *Proc. IEEE International Conference on Electro/Information Technology (EIT'18)*. Rochester, MI, USA, pp. 560-563.

Ohta, T., Murakami, H., Igaki, H., Higo, Y., and Kusumoto, S. (2015). "Source code reuse evaluation by using real/potential copy and paste", in *Proc. IEEE 9th International Workshop on Software Clones (IWSC'15)*. Montreal, QC, Canada, pp. 33-39.

Ohtani, A., Higo, Y., Ishihara, T., and Kusumoto, S. (2015). "On the level of code suggestion for reuse", in *Proc. IEEE 9th International Workshop on Software Clones (IWSC'15)*. Montreal, QC, Canada, pp. 26-32.

Pate, J. R., Tairas, R. and Kraft, N. A. (2013). “Clone evolution: A systematic review”, *Journal of Software: Evolution and Process*, Vol. 25, Issue 3, pp. 261-283.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008) “Systematic mapping studies in software engineering”, in *Proc. 12th International Conference on Evaluation and Assessment in Software Engineering (EASE’08)*. Bari, Italy, pp. 68-77.

Pittenger R. (2019). “Building ASP.NET web pages dynamically in the code-behind”. https://www.codeproject.com/Articles/25573/Building-ASP-NET-Web-Pages-Dynamically-in-the-Code. [Accessed 06/09/2020].

Priyambadha, B., and Rochimah, S. (2018). “Behavioral analysis for detecting code clones”, *Telkomnika*, Vol. 16, Issue 3, pp. 1264-1275.

Rattan, D., and Kaur, J. (2016). “Systematic mapping study of metrics based clone detection techniques”, in *Proc. International Conference on Advances in Information Communication Technology & Computing (AICTC’16)*. XBikaner, India, article 76, pp. 1-7.

Rattan, D., Bhatia, R., and Singh, M. (2013). “Software clone detection: A systematic review”, *Information and Software Technology*, Vol. 55, Issue 7, pp. 1165-1199,

Reddivari, S., and Khan, M.S. (2018). “A topic modeling approach for code clone detection”, in *Proc. 30th International Conference on Software Engineering and Knowledge Engineering (SEKE'18)*. San Francisco Bay, CA, USA, pp. 486-491.

Ren, R., Castro, J.W., Acuña, S.T., and de Lara, J. (2019). “Usability of chatbots: A systematic mapping study”, in *Proc. 31st International Conference on Software Engineering and Knowledge Engineering (SEKE’19)*. Lisbon, Portugal, pp. 479-484.

Rouse, M. (2019). “What is source code in programming and how does it work?” https://searchapparchitecture.techtarget.com/definition/source-code. [Accessed 06/09/2020].

Saini, V., Sajnani, H., Kim, J., and Lopes, C. (2016). “SourcererCC and SourcererCC-I: Tools to detect clones in batch mode and during software development”, in *Proc. IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C'16)*. Austin, TX, USA, pp. 597-600.

Saini, N., Singh, S., and Suman. (2018). “Code clones: Detection and management”, *Procedia Computer Science*, Vol. 132, pp. 718-727.

T. Shippey, D. Bowes, B. Chrisianson and T. Hall.(2013). “A mapping study of software code cloning”, in *Proc. 16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*. Ciudad Real, 2012, pp. 274-278.

Solanki, K., and Kumari, S. (2016). “Comparative study of software clone detection techniques”, in *Proc. Management and Innovation Technology International Conference (MITicon’16)*. Bang-San, Thailand, pp. 152-156.

Stolee, K.T., Elbaum, S., and Rothermel, G. (2009). “Revealing the copy and paste habits of end users”, in *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC’09)*. Corvallis, OR, USA, pp. 59-66.

Sudhamani, M., and Rangarajan, L. (2019). “Code similarity detection through control statement and program features”, *Expert Systems with Applications*, Vol. 132, pp. 63-75.

Svajlenko, J., and Roy, C.K. (2017). “Fast and flexible large-scale clone detection with cloneworks”, in *Proc. IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C’17)*. Buenos Aires, Argentina, pp. 27-30.

Taylor, R.N., Medvidovic, N., and Dashofy, E. (2009). *"Software Architecture: Foundations, Theory, and Practice"*. John Wiley & Sons, First Edition.

Van Bladel, B., Murgia, A., and Demeyer, S. (2017). "An empirical study of clone density evolution and developer cloning tendency", in *Proc. IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER'17)*. Klagenfurt, Austria, pp. 551-552.

Vashisht A., Sukhija, A., Verma, A., and Jain P. (2018). "A detailed study of software code cloning". *IIOAB Journal – Special Issue: Computer Science*, Vol. 9, Issue 2, pp. 20-32.

Verner, J.M., Brereton, O.P., Kitchenham, B., Turner, M., and Niazi, M. (2012). "Systematic literature reviews in global software development: A tertiary study". In *Proc. 16th International Conference Evaluation and Assessment Software Engineering (EASE'12)*. Ciudad Real, Spain, pp. 2-11.

Wagner, S., Abdulkhaleq, A., Kaya, K., and Para, A. (2016). "On the relationship of inconsistent software clones and faults: An empirical study", in *Proc. IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER'16)*. Suita, Japan, pp. 79-89.

Wang, K., Zhang, L., and Yann, S. (2017). "A study on code clone evolution analysis", in *Proc. IEEE 8th International Conference on Software Engineering and Service Science (ICSESS'17)*. Beijing, China, pp. 340-345.

Wijesiriwardana, C., and Wimalaratne, P. (2017). "Component-based experimental testbed to facilitate code clone detection research", in *Proc. 8th IEEE International Conference on Software Engineering and Service Science (ICSESS'17)*. Beijing, China, pp. 165-168.

Xu, B., An, L., Thung, F., Khomh, F., and Lo D. (2019). "Why reinventing the wheels? An empirical study on library reuse and re-implementation", *Empirical Software Engineering*, Vol. 25, pp. 755-789.

Yarmish, G., and Kopec, D. "Revisiting novice programmer errors". *ACM SIGCSE Bulletin*, Vol. 39, Issue 2, pp.131-137.

Zhang, T., and Kim, M. (2018). "Poster: Grafter: Transplantation and differential testing for clones", in *Proc. IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion'18)*. Gothenburg, Sweden, pp. 422-423.

Zhang, H., Ali, M., and Tell, P. (2011). "Identifying relevant studies in software engineering", *Information and Software Technology*, Vol. 53, Issue 6, pp. 625-637. Special Section: Best papers from the APSEC.

Zhang, G., Peng, X., Xing, Z., and Zhao, W. (2012). "Cloning practices: Why developers clone and what can be changed", in *Proc. 28th IEEE International Conference on Software Maintenance (ICSM'12)*. Trento, Italy, pp. 285-294.

Zhang, F., Su, X., Zhao, W., and Wang, T. (2017). "An empirical study of code clone clustering based on clone evolution", *Journal of Harbin Institute of Technology (New Series)*, Vol. 24, Issue 2, pp. 10-18.

APPENDIX A
LIST OF KEYWORDS

Table A.1 lists the selected keywords obtained from control group articles of the primary studies.

Table A.1: List of selected keywords

Keywords	Coverage (%)	Frequency	Weight
code	100	1,177	1
clones	90	677	0.74
clone	100	441	0.69
software	100	327	0.64
study	100	204	0.59
source	100	195	0.58
copy	100	193	0.58
system	100	187	0.58
patterns	100	181	0.58
paste	100	153	0.56
design	100	133	0.56
analysis	100	122	0.55
development	100	100	0.54
programming	100	90	0.54
research	100	82	0.53
systems	100	79	0.53
studies	100	77	0.53
approach	100	51	0.52
cloning	80	267	0.51
engineering	90	105	0.49
behavior	90	102	0.49
usage	90	89	0.49
tool	90	89	0.49
program	90	68	0.48
context	90	58	0.47
reuse	90	40	0.47
developer	80	142	0.46
method	80	58	0.42
techniques	70	81	0.38
snippets	40	22	0.21