EMPIRICAL STUDIES OF CLONE MUTATION AND CLONE MIGRATION IN CLONE GENEALOGIES

by

Shuai Xie

A thesis submitted to the
Department of Electrical and Computer Engineering
in conformity with the requirements for
the degree of Master of Applied Science

Queen's University
Kingston, Ontario, Canada
September 2013

Abstract

Duplications and changes made on code segments by developers form code clones. Cloned code segments are exactly the same or have a particular similarity. A set of cloned code segments that have the same similarity with each other become a clone group. A clone genealogy contains several clone groups in different revisions and time periods. Based on different textual similarities, there are three clone types, *i.e.,*Type-1, Type-2, and Type-3. Clone mutation contains the changes of clone types in the clone evolutions. Clone migration is known as moving cloned code segment to another location in the software system.

In this thesis, we build clone genealogies by clone groups in two empirical studies. We conduct two studies on clone migration and clone mutation in clone genealogies. We use three large open source software systems in both studies.

In the first study, we investigate if the fault-proneness of clone genealogies is affected by different patterns of clone mutation and different evolution patterns of distances among clones in clone groups. We conclude that clone groups mutated between Type-1 and Type-2 and between Type-1 and Type-3 clones have higher risk for faults. We find that modifying the location of a clone increases its risk for faults.

In the second study, we study if the fault-proneness of migrated clones is affected by clone mutation with different changes on clone types. We examine if the length of

time interval between clone migration and the last change of the cloned code has an impact on the faultiness of migrated clones. Our results show that the clone migration associated with clone mutation is more fault-prone than the clone migration without clone mutation. We find that a longer time interval between clone migration and the last change makes the migrated clones more fault-prone.

Co-Authorship

The empirical study introduced in Chapter 4 is based on a published paper [1], which is co-authored with Dr. Ying Zou, my supervisor, and Dr. Foutse Khomh, an Assistant Professor from Polytechnique Montréal, QC, Canada.

I am the primary author for both studies. Dr. Ying Zou supervised all of my work and provided feedbacks to both papers. Dr. Foutse Khomh gave me suggestion about two studies and polished the papers.

[1] S. Xie, F. Khomh, and Y. Zou, "An empirical study of the fault-proneness of clone mutation and clone migration," in *Proc. 10th Working Conference on Mining Software Repositories (MSR)*, pp. 149-158, 2013.

Acknowledgments

First of all, I want to thank to my supervisor, Dr. Ying Zou. She provided valuable suggestions and detailed feedbacks for my thesis research. I also want to thank her for the good opportunity that she provided for me to finish my master degree in her lab. It is such a honor to be one of her students.

I want to thank the other research members in the Software Reengineering Research Group for their help and support throughout my studies. I would like to acknowledge Dr. Foutse Khomh, who assisted my studies, Feng Zhang, who provided advices for the problems I faced, and Lilian Barbour, who helped me learn the approach to process data. I want to acknowledge Dr. Jim Cordy, who addressed my questions about using NiCad, and Weiyi Shang, who assisted me in using J-Rex.

Moreover, I appreciate the hard work from my committee members: Dr. Thomas Dean, Dr. Hossam S. Hassanein and Dr. Ying Zou.

Last but not least, I want to thank my mother Jianpeng Shi and my father Wande Xie for their great love and consistent support during my academic lifetime. Finally, I appreciate the care and support from my wife, Yaokun Zhao.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Copying and pasting activities frequently occur in software systems. Developers perform these activities for specific purposes or by accident. Those activities make some code segments become exactly the same or have a certain similarity with other code segments. This forms a clone group that contains a set of clones. Since there are a lot of changes during the entire lifetime of a software system, the clone groups evolve in different revisions or periods. The evolution of a clone group is defined as a clone genealogy. We mine the software repositories and detect clones and fault fixes for addressing different research questions. In the following two sub-sections, we introduce the software repositories, clones and clone evolution in details.

1.1.1 Software Repository

A software repository contains all the changes and source code for the entire lifetime of a software system. When more than one developer make changes to a software system, the software repository saves a new revision right after the latest change

and keeps all other changes. Thus a developer can track the snapshot of a file in a particular revision. A snapshot can capture the source code in a specific period or in a specific revision for a file in the software system.

Developers upload changes to the central repository of a software system after making a request and getting a latest copy of the file. A comment can be added by developers for a change to clarify their actions or intentions. For example, a comment containing "fix" is helpful for others to recognize a fault fixing change.

There are three most popular software repositories, *i.e.,* Concurrent Versioning System (CVS) [2], Subversion (SVN) [3], and Git [4].

1.1.2 Clones and Their Evolutions

Clones

Clones are made by duplicating of several lines of code segments to another method or file in a software system. While after the first duplication, developers are required to make consistent changes for all clones to avoid the future fault fixes. More clones in a clone group make it more difficult to make consistent changes for developers. The activity with multiple duplication of different files is more likely to introduce defects for software systems. Since it is difficult for developers to cover all clones in a fault fixing change.

Since there are different textual and functional similarities for cloned code, clones can be classified into four types by a previous research [5]:

- Type-1: The code segments are exactly the same.
- Type-2: The code segments are exactly the same except identifiers and literals.

Table 1.1: Example of Different Types of Clones

Clone Types	Clone A	Clone B

Type-1	for(int j = 0; j < 5; j ++){
	sum = sum + a[j];
	}
		for(int j = 0; j < 5; j ++){
		sum = sum + a[j];
		}

Type-2	for(int j = 0; j < 5; j ++){
	sum = sum + a[j];
	}
		for(int **id** = 0; **id** < 5; **id** ++){
		sum = sum + a[**id**];
		}

Type-3	for(int j = 0; j < 5; j ++){
	sum = sum + a[j];
	}
		for(int **id** = 0; **id** < 5; **id** ++){
		sum = sum + a[**id**];
		d = sum * c;
		}

Type-4	for(int j = 0; j < 5; j ++){
	sum = sum + a[j];
	}
		int j = 0;
		while(j < 5){
		sum = sum + a[j];
		j ++;
		}

- Type-3: The code segments have more modifications than Type-2 clones, such as changing, adding or removing statements.
- Type-4: The code segments perform the same function but with different syntactic variants.

The examples of different types of clones are shown in Table 1.1. For Type-1 clones, Clone A and Clone B are exactly the same. Code segments with only differences in identifiers are Type-2 clones. Adding a line of code (statement) for

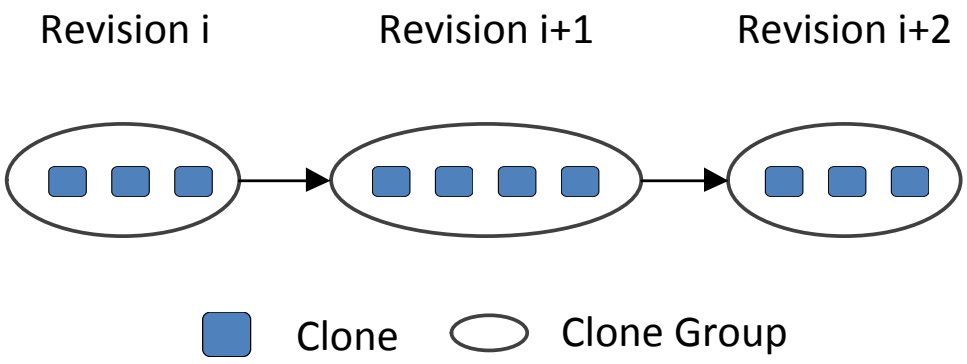


Figure 1.1: Example of Clone Genealogy

Type-2 clones refers to the Type-3 clones. The Type-4 clones perform the same function but have different code structures.

In these definitions, the changes for whitespace, layout and comments are not considered as differences among clone types. A higher clone type (*i.e.,* Type-3) has a lower similarity compared with a lower clone type (*i.e.,* Type-1). Usually, there are more clones detected when detecting clones with a lower similarity. In this thesis, we only study the Type-1, Type-2, and Type-3 clones.

Clone detection tools use several different clone detection approaches, *i.e.,* text-based, abstract syntax tree-based, program dependency graph-based, and metrics-based techniques. We use NiCad [6] with both text-based and abstract syntax tree-based techniques to detect the Type-1, Type-2, and Type-3 clones.

Clone Genealogies

After the duplication actions form several clones in a clone group, these clones evolve in the following revisions of a software system by experiencing more modifications. These modifications can change the clone type of the clone group. The clones in a clone group can be added, removed, and modified in the evolution of the software system. This phenomenon is defined as clone genealogy for clone groups. Both maintenance and development tasks of software systems will lead to changes for the clones in clone groups.

Figure 1.1 shows an example of clone genealogy for clone groups. Each revision has a clone group, which contains three or four clones. In Revision i+1, a clone is added to the clone group due to a duplication action. But in next revision (Revision i+2), a clone is removed from the clone group. Therefore, the clone genealogy captures the changing and movement of clones in clone groups.

Clone Mutation

When clones of a clone group evolve in the subsequent revisions of the clone genealogy, there could be one or more changes made on each of those clones. In those changes, some of them may modify the clone type of the clone group due to the change of the similarity between every two clones in that clone group. Changing code segments leads to the changes of similarities, identifiers and clone types for clones in clone groups. The clone type for clone groups can be changed to different directions, *i.e.,*to a higher (*i.e.,* Type-3) or lower type (*i.e.,* Type-1).

Figure 1.2 shows an example of clone mutation for clones and clone groups in a clone genealogy with three revisions. All clones in a clone group are changed to

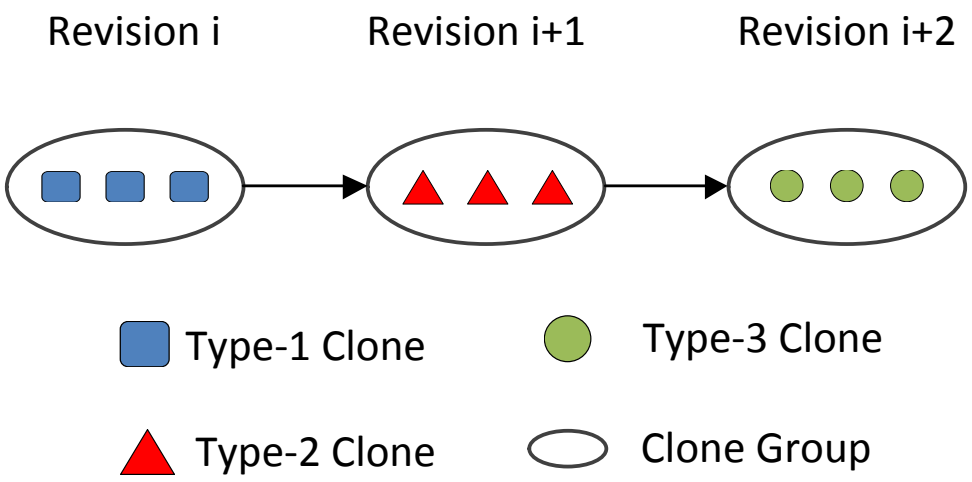


Figure 1.2: Example of Clone Mutation

Type-2 clones from Revision i to Revision i+1 due to some modifications on these clones. In the last revision (Revision i+2), more modifications make the clones of that clone group become Type-3 clones. These two revisions present clone mutation in clone genealogies.

Clone Migration

A modification can change the location of the file of a clone in a software system. We define this activity as clone migration that moves a cloned code to a new location in a software system. Migrated clones are changed for locations in software systems but maintain the file names. Clone migration made from developers by accident or to implement an intended movement decision. Since being moved to a new location, migrated clones may experience more fault fixes in the following period after the

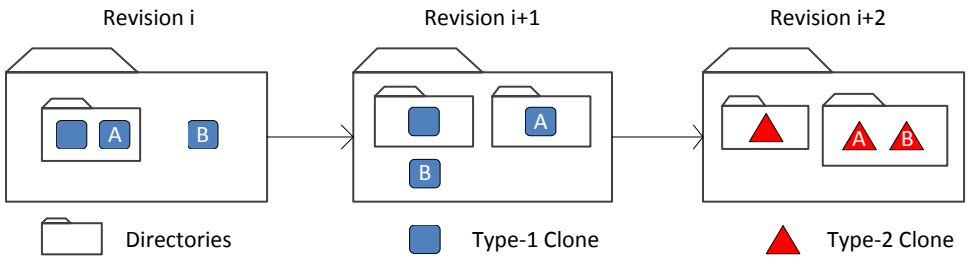


Figure 1.3: Example of Clone Migration

migration action.

Clone migration can be associated with clone mutation, which means clone migration made on a code segment changes the clone type for the clone group when developers move it to new location. Clone mutation occurs concurrently with clone migration when changing and movement actions are made by developers in one modification.

Figure 1.3 shows an example of clone migration among three revisions in a clone genealogy. From Revision i to Revision i+1, Clone A is moved to another directory. We use the highest distinct directory level in the shorter path to measure the distance between two clones. Thus moving Clone A to another directory increases the distance between each two clones in the clone group of Revision i+1, which may increase the efforts of making more changes for these clones.

The example of the clone migration associated with clone mutation is also shown in Figure 1.3. From Revision i+1 to Revision i+2, Clone B is moved to the same directory with Clone A and the clone group in Revision i+2 is changed to Type-2. Subsequent maintenance on these clones may need more efforts, since the movement action and the modification action are required to change a lot of contents.

Distance Between Cloned Code

We define the distance between every two clones of a clone group using the code directory structure defined by Kamiya *et al.* [7]. More specifically, we define the distance between two cloned codes as the distance between two different locations (paths) in the source code directory structure .

We define a model to compute the distance between two locations in software directories. In a clone group G, for two clone segments A and B, f_1 and f_2 are the files containing A and B respectively. The distance between A and B, $d_{dir}(A, B)$ is equal to $d_{dir}(f_1, f_2)$. $d_{dir}(A, B)$ is the value of the highest (first) level where two paths have different folder names in the smaller (shorter) one of the paths for two clones. The value of the highest distinct level is counted in the smaller path of two clones. The first folder in the directory is identified as the highest level [1]. We define the lowest level of a path as level 1. For example, if A and B are included in the same file, $d_{dir}(A, B) = 0$.

As shown in Figure 1.4, clone A and clone B are in a large directory, which is the highest one and defined as Level 3 (L3 in Figure 1.4). Since clone A has the shorter path, the level of directory is counted based on the path of clone A. Clone A and Clone B are in different directories of the second level (L2), thus the distance between A and B is 2 ($d_{dir}(A, B) = 2$).

1.2 Research Statement

Developers may forget their previous copying activities in the subsequent revisions of software systems, where they should make consistent changes on the same files or clones to avoid introducing defects. Clone migration and clone mutation would make

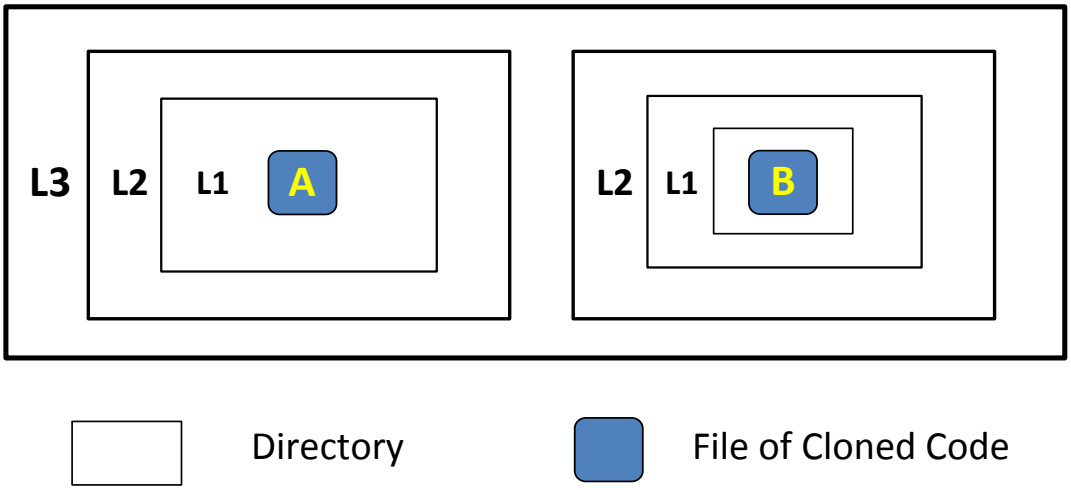


Figure 1.4: Example of Distance Between Clones

the clones more dangerous since more changes are required in the following evolution of a software system. However, some specific patterns of clone mutation and clone migration may experience more fault fixes. Because they are difficult for developers to keep consistent changes for the clones that follow those patterns.

When clone mutation and clone migration occur at the same time, it may make consistent changes more difficult to be implemented for clones. Moreover, the time interval between the clone migration and the last change of the cloned code is expected to affect the risk for faults of migrated clones. Because the clones that have not been touched for longer time will increase the difficulty for developers to make consistent changes on migrated clones.

In this thesis, we investigate the effects from clone mutation and clone migration on the fault-proneness of clone genealogies in two empirical studies.

In the first empirical study, we use different evolution trends for the distances

between paths of each two clones in clone groups to classify the clone genealogies into different patterns. We use the changes of distances to measure clone migration in clone genealogies. We use different evolution trends of distances among clones to identify the impact of clone migration on the risk for faults of clone genealogies. We identify clone mutation by identifying different clone mutation patterns and compare their impacts on fault-proneness of clone genealogies.

In the second study, we identify the fault-proneness of cloned code experiencing both clone migration and clone mutation at the same time. We aim to identify the risk for faults of clone groups and clone genealogies with different frequencies of clone migration. Moreover, we investigate the fault-proneness of cloned code based on different time intervals between clone migration and the last change of the cloned code.

1.3 Organization

The subsequent chapters are organized as follows:

- **Chapter 2 Related Work:** We discuss the related work about clone detection, fault in clones, and clone evolution.
- **Chapter 3 Building Clone Genealogies from Software Systems:** We describe the approach that processes data in software repositories, builds clone genealogy, and identifies clone migration. Then we introduce the statistical techniques used in our two studies.
- **Chapter 4 An Empirical Study of the Fault-Proneness of Clone Mutation and Clone Migration:** We present our first study to identify the impact

of clone mutation and the change of distances among clones in clone groups on the fault-proneness of clone genealogies.

- **Chapter 5 An Empirical Study on the Fault-Proneness of Clone Migration in Clone Genealogies:** We discuss our second study that investigates the impact of the clone migration associated with clone mutation on the fault-proneness of migrated clones. Moreover, we investigate the impact of clone migration made in different time intervals after the last change of the cloned code on risk for faults of clone genealogies.

- **Chapter 6 Conclusion:** We present the contribution, conclusion, recommendation, threats to validity and our future work for the thesis.

Chapter 2

Related Work

In this chapter, we discuss previous studies about clone detection, fault in clones, and clone evolution.

2.1 Clone Detection

In this section, we summarize the clone detection techniques that are commonly used in recent studies. We classify the clone detection tools into four categories based on the different approaches used in these tools. Roy *et al.* [5] conduct a comprehensive study about clone detection in a technical report.

Token-based: Token-based clone detection tools parse the source code to a set of tokens with a specific sequence before detecting clones on the code segments. This process normalizes all the identifiers and literals in the source code by certain criteria [7]. The clone detection mainly compares different sequences of tokens extracted from source code. We can define the minimum length of tokens before detecting clones. But token-based tools can only detect clones for certain programming languages. A typical example of Token-based tool is CCFinder [7].

Text-based: Text-based tools [8, 9, 10] do not process the source code as Token-based tools. While this technique converts all the lines of source code into strings and compare these strings within the source code to identify the same or similar code fragments. To detect Type-2 clones with changed identifiers, Token-based tools normalize the source code before comparing the strings. Text-based tools can handle many languages and allow developers to adjust the configurations, *i.e.,* line of clones (LOC) and similarity, to meet different requirements. For example, SIMIAN [11] is a text-based clone detection tool for commercial application.

Tree-based: Same as token-based tools, tree-based clone detection tools [12, 13, 14] include the parsing process before matching clone segments. A tree-based tool normalizes the source code and builds a parse tree or an abstract syntax tree for the source code. When detecting clones, the tool locates similar sub-trees among all the trees extracted from parsing process. While the tree-based technique is language dependent. In this thesis, we use NICAD [6], a clone detection tool based on both text and abstract tree techniques. NICAD is still evolving and it now has more functions than the first version. We use it because its less memory and time consumption compared to other tools.

Metrics-based: The metric-based approach [15, 16, 17, 18] divides the source code into small units in line, method, or class level and extracts metrics for each one of these units. After comparing those extracted metrics, the tool identifies two code segments as the clones when they have the same value for those metrics. The defined metrics of the metric-based tools will decide the supported programming languages. Some studies [16, 17] perform this technique for identifying duplicated web pages. SeByte [19] can detect semantic (Type-4) clones using new approach

based on traditional pattern-based detection. SeByte is a metric-based tool including set theory and pattern matching and uses Java bytecode (binary) as input.

2.2 Fault in Clones

Researchers find that code clones are harmful and a lot of studies have demonstrated this point. Many previous studies [20, 21, 22] have concluded that code clones increase the maintenance efforts for developers, and code clones should be analyzed and removed.

However, Krinke [23] concludes that cloned code is more stable than non-cloned code. He performs a study to compare the differences on ages of both cloned and non-cloned code segments. Using SIMIAN to detect clones for three large open source software systems written in JAVA, the study finds that cloned code is older than non-cloned code on average. This conclusion supports other studies concluding that cloned code is more stable than non-cloned code.

Lozano *et al.* [22] present a study on the change rate of software systems containing code clones. They develop a tool called CLONETRACKER to identify the number and density for the changes in clone-containing methods of JAVA files. Then they compare these results with other methods without cloned code. They find that cloned code has more changes than non-cloned code. This is probably because the copied code segments are required to be changed consistently after the creation of code clones.

Kasper and Godfrey [24] conduct a case study on the harmfulness of clones. To identify the usage of cloning in software systems, this study defines eleven cloning patterns extracted from large software systems. Performing a case study on two medium-sized open source software systems, they aim to identify the cloning patterns

with benefits for software quality. They find that refactoring is properly not the good method for all cloning patterns. The study concludes that code clones can be useful and positive for software systems when developers have a good design and consistent maintenance on these clones for a long term.

Kim *et al.* [25] perform a study by observing how programmers perform coding tasks and copy and paste activities. They aim to identify the common usage patterns for copy and paste activities and to understand and solve the problems developers have experienced. They summarize the reasons for copy and past activities from the view of maintenance. The study introduces some tools that can be used by developers to reduce the maintenance efforts and problems from copy and paste activities in the software systems.

Same as most of these studies, we identify the fault fixes in code clones. We consider code clones will lead more defects to software system, which is proved by our studies. We use a tool called J-Rex [26] to locate the fault fixing changes from commit logs.

2.3 Clone Evolution

In this section, we introduce the related works about clone evolution, which is the clone genealogy introduced in our two studies.

Kim *et al.* [27] conduct a study on clone evolution for the first time. They build clone genealogies for clone groups from two Java systems and perform CCFinder tool to detect clones. They examine whether refactoring clones can solve the issues that are related to the harmfulness of clones. They find that refactoring is not helpful for reducing the harmfulness of the long-lived and consistently changing clones. In

this thesis, we use a different method to generate clone genealogies and we use the clone groups for building clone genealogies. We aim to understand the risky patterns of clone genealogies to reduce the maintenance efforts of developers, but not to change the clones.

Aversano et al. [20] perform a study on clone genealogies for two JAVA open source systems to analyze the cochanges and how clones are maintained in their evolutions. They perform the SIMSCAN duplication detection tool to detect the clones and define several patterns for clone genealogies of clone groups. They conclude that most of the clone groups are maintained consistently and the late propagation is a risky pattern for clone genealogies.

Same as [20], Thummalapenta *et al.* [28] investigate the clone evolution in software systems. They conduct a empirical study on clone genealogies for four large-scale open source software systems written in C and JAVA. The study concludes that the clone evolution of late propagation pattern is more fault-prone. They find that clones are consistently changed, thus the clone genealogies are worth being studied in depth.

Barbour *et al.* [29, 30] investigate the fault-proneness of clone evolution by identifying different evolutionary patterns in software systems. They define different fault-prone states and transitions in the clone evolution. They define several patterns and metrics for clone pairs to predict faults in code clones. This thesis performs the same approach for processing data and a similar approach for building clone genealogies as their study. However, our two studies apply a different clone detection tool (NICAD) and our study includes Type-3 clones. Moreover, we build clone genealogies based on clone groups.

Duala-Ekoko *et al.* [31] try to track code clones in evolving softwares to inform

developers of the modification on clones. Based on the concept of clone region descriptors (CRDs), they build the CloneTracker clone tracking system for locating clones. CloneTracker manages clone detection outputs and tracks clone regions to help developers to make subsequent changes on all related clones. The case study demonstrates that CloneTracker can track most of clone regions in software systems. This study aims to help developers locate the clone regions in software systems. While our studies try to identify clone mutation and clone migration in clone genealogies.

Göde [32] performs a study on modeling clone evolution from changes on source code between two continuous versions by using nine open source systems. Without making definition of patterns for clone evolution, the study concludes that the proportion of clones is decreased as the evolving of the software system. Our study builds clone genealogies to capture the clone evolution by mapping clones at the revision level but not versions level. And we define several patterns for classifying clone genealogies.

Göde and Koschke [33] performs another study to extract clone evolution by using clone detection techniques. They present an algorithm for incremental clone detection to detect clones based on the analysis of previous versions. This new method, which is embedded into iClone, maps clones in the consecutive versions when detecting clones on different versions of a software system. They evaluate the performance of iClone by comparison with another tool clones and conclude that their tool is highly effective for repeating clone detection whenever a file is changed.

Alali *et al.* [34] perform a initial study to improve the accuracy of the evolution for couplings, which is similar to clone evolution. Evolutionary couplings refer to the

co-changing patterns for two times in different versions of software systems.

In our study, we identify clone mutation and clone migration to classify the clone genealogies. Different from those studies that only involving with Type-1 and Type-2 clones, our work examines clone genealogies containing all three types of clone groups.

Saha *et al.* [35] conduct an exploratory study about the clone evolution focusing on Type-3 clones in six subject systems. To understand the evolution of Type-3 clones in softwares systems, they identify the lifetimes, change frequencies, change patterns and changes of clone types for the Type-3 clones. Similar to our study, they study all three types of clones on class (group) level and they use NiCad to detect clones. However, different from our studies, they apply gCad for building clone genealogies in different releases of software systems. They conclude Type-3 clone classes experience more consistently changes and they need more consideration from developers. Moreover, a large number of Type-1 and Type-2 clones are transformed to Type-3 clones, which is verified in our studies.

We investigate Type-3 clones but we also study two other types of clones (Type-1 and Type-2 clones), rather than only focus on Type-3 clones. Saha *et al.* [35] use gCad to build clone genealogies, which is a better method for us to build clone genealogies in the future studies. They use several metrics for identifying Type-3 clones but we mainly investigate the evolutionary patterns for three types of clones and study clone mutation and clone migration in the clone genealogies.

2.4 Summary

In this chapter, we introduce the related works about clone detection techniques, fault in clones, and clone evolution (genealogies).

Chapter 3

Building Clone Genealogies from Software Systems

In this chapter, we introduce the overall procedure that processes the data from software repositories. We discuss the approach for identifying changes for fault fixing, detecting clones, building clone genealogies, and identify clone mutation and clone migration in the entire lifetime of software systems. We also describe the subject systems and statistical techniques used in our studies.

3.1 Approach Overview

Figure 3.1 gives an overview of our approach. We follow the identical procedures as Barbour *et* al. [29] to process data to identify faults fixes. We use J-Rex [26] to mine the repositories of three Java subject systems for both studies in this thesis. J-Rex identifies all the revisions experiencing code changes and outputs the snapshots of the source code files changed at those revisions. We remove the test files and process the results to detect clones.

For the next step, we use NiCad [6], a near-miss clone detection tool, to detect clones on the Java files extracted from the results of J-Rex. Then we build clone genealogies from those clone groups identified by NiCad. Finally, we identify clone

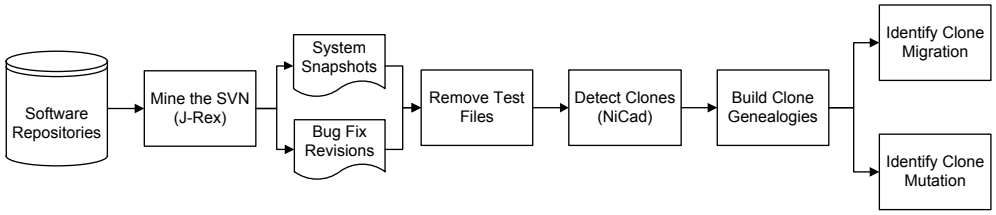


Figure 3.1: Overview of the Analysis Process

mutation and clone migration in the clone genealogies to address research questions. More details about our case study are introduced in the following sections for each step.

3.1.1 Identifying Faults

Similar to the approach in previous studies [29, 30], we employ J-Rex to extract snapshots of all the revisions experiencing changes in the software systems. We extract all the methods that contain modifications in the software history. We use J-Rex to identify fault fixing changes from the commit messages in software repositories.

J-Rex is a tool that can identify the keywords by using the heuristics presented by Mockus *et al.* [36]. It is reported to have an accuracy with 87% in the study of Barbour *et al.* [29]. A previous study from Hassan [37] demonstrates that J-Rex is as good as professional developers for performing identification of fault fixes. The correlation between developers and J-Rex is more than 0.8.

3.1.2 Detecting Clones

Before detecting clones, we remove the test files. Because test files are primarily used to test the functions of the software systems, developers often copy and modify them

Table 3.1: Parameters for NiCad

Clone Types	Identifier Renaming	Similarity Threshold
Type-1	None	100%
Type-2	Blind-rename	100%
Type-3	Blind-rename	80%

for different testing scenarios. Therefore, test files contain a lot of clones but they are not related with normal executions in the software systems. We remove all the files containing the keyword "test" in their file names or folder names and manually verify the removed files. This procedure is also adopted by Barbour *et al.* [29]. After the removing step, we extract methods from all the remaining source code files and save each method snapshot into a single file. Finally, we use NICAD to detect clones in all these files.

In this thesis, we use NICAD [6] as the clone detection tool. NICAD is a flexible TXL-based text comparison software. It can detect clones in many languages, *i.e.,* C, C#, JAVA, PYTHON and WSDL. Roy *et al.* [6] have demonstrated that NICAD has a high accuracy for detecting both exact (*i.e.,* Type-1) and near-miss (*i.e.,* Type-2 and Type-3) clones. We detect clones on all source code files in each revision of three software systems in one shot. We use revision level to detect clones since each revision presents a change and we can capture more detailed clone evolution in revision level.

We select NICAD as the clone detection tool because NICAD consumes less time and less memory for large-scale detection. Moreover, since there is no other tool can help us detect Type-3 clones better than NICAD, we choose NICAD as our clone detection tool to help us detect all three types of clone.

NICAD is used in previous studies (*i.e.,* Zibran *et al.* [38]) for the study of clone genealogies. We deploy NICAD and use the same parameters as Zibran *et al.* [38].

Table 3.1 shows the parameters of NiCad for us to detect clones. Since Type-3 clones can be defined in different similarities, there are different similarities that can be used to detect Type-3 clones. To prevent the impact from choosing specific similarity, we use six similarities (*i.e.,* 70%, 75%, 80%, 85%, 90% and 95%) to conduct our study. We use the version 3.4 of NiCad to conduct each of two studies. We process the clone detection results to identify valid clones that co-exist during the same time period.

3.1.3 Building Clone Genealogies

Before building clone genealogies, we identify the unchanged clones by matching clone detection results with the results from J-Rex and remove the unchanged clones. Since clones will be changed during the clone genealogies, unchanged clones can not have an evolution. We remove the clone groups only containing clones that belong to the same method, because it is not clone but just code segments for different revisions of a method. Then we remove the clone groups that have less than two clones. After the cleaning activities, we connect the clone groups from the results of NiCad over all the revisions of the software system to build clone genealogies for each subject system. We use a similar approach as Barbour *et al.* [29] for generalizing clone genealogies.

At first, we extract the modification date of each change from the outputs of J-Rex for each of the changed methods. We go over each modified method to check if the clone segments inside the method are modified. This process is repeated for each revision of the software history. We identify the valid clone groups from clone detection results and connect these clone groups from the entire history of the software system. Each two related clone groups are connected by identifying whether they have at least one shared clone and in a reasonable time order. Finally, we extract the clone

genealogies by mapping each two related clone groups to build the longest evolution.

3.1.4 Identifying Clone Mutation

Since NiCad can help us identify Type-3 clones by certain configuration, we have clones for three clone types after detecting clones using NiCad. Then we have the clone type for each cloned code and clone group to identify clone mutation in clone genealogies. We use different clone mutation patterns among clone groups to categorize the clone genealogies. More specifically, we check how the clone type is changed between every two continuous clone groups in a clone genealogy. We extract the evolution of clone types for all clone groups in a clone genealogy.

Moreover, we identify the changes of clone type that occur with clone migration. This means when a clone is migrated to a new location, we identify whether the clone type is changed as well. The clones that belong to a clone group have the identical clone type.

3.1.5 Identifying Clone Migration

We perform two different approaches to measure clone migration in two empirical studies. For the first approach, we track the evolution trend of the median distance among each two clones in clone groups and the size of clone group of clone genealogies. Thus we use the evolution trends of distance and group size to measure clone migration. Then we identify the fault fixes in these clone genealogies with different evolution patterns.

While for the second approach, we locate clone migration by identifying each clone migration that occurs between two clones in two continuous clone groups respectively.

Clone migration is identified by changed location (in the source code directory structure) with maintained file names of clones. After finding clone migration, we track fault fixing changes in the migrated clones in the subsequent revisions after clone migration. Therefore, we have the migrated clones in the clone genealogies and their fault fixes data that can be used for addressing research questions.

3.2 Subject Systems

Table 3.2: Overview of the Subject Systems

Systems	# LOC	# Revisions	# Clone Genealogies
JBoss	1.6M	109K	1.7K
Apache-Ant	2.3M	1.0M	23K
ArgoUML	3.1M	18K	15.6K

We process the software repositories of three large open source subject systems, *i.e.,* JBoss, Apache Ant, and ArgoUML. All three software systems are written in Java, while they have different sizes (line of code) and belong to distinct application areas. Some basic statistics for the three software systems are shown in Table 3.2. We use the same three subject systems for the two studies introduced in Chapter 4 and Chapter 5 respectively.

JBoss is an application server and a division of Red Hat that started in 1999. JBoss contains 109K revisions and about 1.7M LOC (lines of code). We perform our case studies for the code snapshots of JBoss from April 2000 to December 2010.

Apache-Ant is a tool to compile, assemble, test and run applications written in programing languages like Java, C, and C++. Apache-Ant has 1.0M revisions and over 2.3M LOC in the history of its software repository. It was created early to

January 2000 and it maintains active now. We identify code snapshots of APACHE-ANT during the period ranging from January 2000 to November 2010.

ARGOUML is a UML-modeling application. It has functions that model systems, generate the corresponding code skeletons, and reverse-engineer diagrams. ARGOUML is also applied in previous studies [20, 29] about clone evolution. Created in January 1998, ARGOUML is still under evolution now. Our data repository for ARGOUML contains 18K revisions and over 3.1M LOC. Each revision presents a change. We use the code snapshots from January 1998 to November 2010.

3.3 Statistical Tests Techniques

We compute odds ratio (OR) [39] to compare the fault-proneness of different clone genealogy patterns and use the Chi-Square test [39] to verify the significance of our results. We introduce these two techniques in the following two subsections:

3.3.1 Odds Ratio

The odds ratio (OR) is a statistical technique for identifying the possibility of the occurrence for an event. For the calculation of OR, there are two groups of data, *i.e.,* a control group and an experimental group. By comparing the possibilities of the events occurring in the experimental group and in the control group, we get the OR value for the likelihood of an event occurs in experimental group. Thus OR is calculated as the following equation:

$$OR = \frac{p/(1-p)}{q/(1-q)} \tag{3.1}$$

In Equation (3.1), p presents the possibility of an event occurs in experimental

group and q presents the possibility of an event occurs in control group. Moreover, (1-p) equals q and (1-q) equals p. Equation (3.1) is used for comparing the odds of occurring for experimental and control group. We use the same approach introduced in Sheskin [39] to compute odds ratio. Barbour *et al.* [29] also use this equation to compute odds ratio. Therefore, we can have the following conclusions for different OR results:

- OR=1: an event equally likely occurs in both groups;
- OR>1: an event more likely occurs in the experimental group; or
- OR<1: an event more likely occurs in the control group.

3.3.2 Chi-Square Test

The Chi-Square test is a statistical test used to identify the existence of non-random associations between two groups of variables [39]. If the Chi-Square test gives us a statistically significant result, we can conclude that the study results are reasonable but not random from our data. We will not make a firm conclusion when the Chi-Square test has no statistically significant result. We perform Chi-Square test calculations for our input for OR computation to verify the statistical significance of those data. We identify whether there are enough samples for the input of OR computation. If the Chi-Square test result is lower than our standard, we can prove the significance our results. We define the threshold value for lowest *p-value* as 0.05 to test the statistically significance of the OR computation results.

3.4 Summary

In this chapter, we introduce the overview of our study design for building clone genealogies, identifying clone mutation and clone migration in software systems. At first, we implement a tool called J-Rex to mine the software repository and process its outputs including snapshots of changed methods and the commit logs with fault fixes. For these outputs, we use NiCad to detect clones for the entire history of the software system. Then we connect the processed clone groups from clone detection results to build clone genealogies. Finally, we introduce the statistical techniques used in our empirical studies.

Chapter 4

An Empirical Study of the Fault-Proneness of Clone Mutation and Clone Migration

In this chapter, we define seven patterns to classify the clone genealogies through identifying the clone mutation patterns of clone groups in clone genealogies. We generate ten clone migration patterns for clone genealogies based on the evolution trends of distances between paths of every two clones in clone groups and the sizes of clone groups in clone genealogies. We identify the frequency of the existence for both phenomenons in clone genealogies in the first research question and discuss the two phenomenons in the following two research questions respectively.

We conduct three research questions based on the clone genealogies we build by applying the approach described in Section 3.1.3. Overall, we implement these research questions to examine the impacts of different clone mutation and clone migration patterns on the fault-proneness of clone genealogies. We introduce two different methods to identify patterns for clone genealogies in following two sections.

Table 4.1: Clone Mutation Patterns of Clone Genealogies

Patterns	List of Clone Types in the Genealogy	Possible Dominant Clone Types
G<1>	Type-1	Type-1
G<2>	Type-2	Type-2
G<3>	Type-3	Type-3
G<1, 2>	Type-1, Type-2	Type-1, Type-2
G<1, 3>	Type-1, Type-3	Type-1, Type-3
G<2, 3>	Type-2, Type-3	Type-2, Type-3
G<1, 2, 3>	Type-1, Type-2, Type-3	Type-1, Type-2, Type-3

4.1 Clone Mutation Patterns

In Section 1.1.2, we introduce the definition of clone mutation. In this section, we discuss how we use clone mutation to classify the clone genealogies into different patterns. We define seven clone mutation patterns for clone genealogies based on the existence of three types of clones. In the patterns that contain multiple clone types, we identify the effects of different dominant clone types on the fault-proneness of clone genealogies.

To prove the existence of different clone types in clone genealogies, we use G<> to represent to clone genealogies. For example, G<1,2,3> stands for clone genealogies that contain all three clone types (*i.e.,* Type-1, Type-2, and Type-3) of clone groups in different revisions of software history. We define the patterns for all the combinations of three clone types in clone genealogies.

We organize clone genealogies in seven patterns as presented in Table 4.1. As shown in the second column of Table 4.1, each pattern of clone genealogy is characterized by the list of clone types involved in clone groups of clone genealogies. For example, G<1,2> pattern represents the clone mutation pattern where clones are transitioned between Type-1 and Type-2 in a clone genealogy. For G<1,2> pattern,

a Type-1 clone group can be changed to a Type-2 clone group in software history. In one of the subsequent revisions, a Type-2 clone group can be changed back to a Type-1 clone group.

The dominant clone type is the clone type for most clone groups in a clone genealogy. When there are more Type-1 clone group than Type-2 clone groups, we define Type-1 as the dominant clone type. The last column of Table 4.1 presents the possible dominant clone type for each pattern in clone genealogies. For example, each one of Type-2 and Type-3 can become the dominant clone type for G<2,3> pattern of clone genealogies. For each one of the first three single type patterns, *i.e.,* G<1>, G<2>, and G<3>, there is only one clone type, thus we do not identify the dominant type for these three patterns.

4.2 Clone Migration Patterns

In Section 1.1.2, we define clone migration. In this section, we introduce the measurement for clone migration in clone genealogies and define different patterns for clone genealogies. We define ten patterns for clone genealogies to identify the impact of clone migration on the fault-proneness of clone genealogies. We use the combination of evolution trends of median distances among clones and sizes of the clone groups to define clone migration patterns.

To capture clone migration cross different directories in a software system, we compute the distance between every two clones of a clone group by applying the method introduced in Section 1.1.2.

We use two factors to define clone migration patterns. One factor is the evolution trend of the median value of the distances between each two code segments in the

Table 4.2: Clone Migration Patterns of Clone Genealogies

Pattern	Description	
	Evolution trend of the	
	median distance of the clone group	size of the clone group
Constant	Constant	Constant
Wave Stable	Constant	Increase, Decrease, Wave Increase, or Wave Decrease
	Wave Constant	Constant, Increase, Decrease, Wave Increase, or Wave Decrease
High Density Strong Up	Increase	Increase or Wave Increase
Low Density Strong Up	Increase	Constant, Decrease, or Wave Decrease
High Density Wave Up	Wave Increase	Increase or Wave Increase
Low Density Wave Up	Wave Increase	Constant, Decrease, or Wave Decrease
High Density Strong Down	Decrease	Decrease or Wave Decrease
Low Density Strong Down	Decrease	Constant, Increase or Wave Increase
High Density Wave Down	Wave Decrease	Decrease or Wave Decrease
Low Density Wave Down	Wave Decrease	Constant, Increase or Wave Increase

clone group and another one is the evolution trend of the sizes for clone groups. These evolution trends are shown in the right two columns of Table 4.2. Based on the different combinations of two evolution trends, we categorize the clone genealogies into ten patterns. Those ten clone migration patterns are shown in the left column of Table 4.2, each of them contains one or more combinations of two evolution trends.

Each row in Table 4.2 shows a clone migration pattern and the description for that pattern in terms of the evolution trends detected respectively from the median

Table 4.3: Examples for Clone Migration Patterns in Clone Genealogies

Patterns	If Clone Migration Exists	Evolution of Median Distance Among Clones in Clone Group (values of median distances)	Evolution of Size of Clone Group (sizes of clone groups)
Constant	No	$1 \rightarrow 1 \rightarrow 1 \rightarrow 1$	$6 \rightarrow 6 \rightarrow 6 \rightarrow 6$
High Density Strong Up	Yes	$0 \rightarrow 2 \rightarrow 4$	$2 \rightarrow 4 \rightarrow 3$
		$1 \rightarrow 3$	$5 \rightarrow 6$

value of the distances among all code segments in a clone group, and from the sizes of the clone groups in clone genealogies. For example, the second row of Table 4.2 presents the Constant pattern that contains a constant trend for the evolution of the median distance among the code segments in a clone group, and a constant trend for the evolution the number of code segments in a clone group (size of clone group).

For the two factors, we use five types of evolution trends to describe all variation situations for median distance of the clone group and the size of the clone group, *i.e.,* Constant (Wave Constant), Increase, Wave Increase, Decrease, and Wave Decrease. The "Constant" presents the evolution trend where the value is not changed in a clone genealogy. While "Increase" ("Decrease") stands for the evolution trend where the values are continuously increased (decreased) or unchanged. Similarly, the "Wave Increase" ("Wave Decrease") stands for evolution trend where the values are continuously increased (decreased) associated with temporary decrease (increase), where the final value is greater (respectively lower) than the initial value. We also use "Wave Constant" to measure the Constant pattern. This means there are constant values in the beginning and ending but changed values in the middle.

Table 4.3 shows two examples of the ten patterns. The "Constant" pattern shown in the second row has both unchanged median distances (value: 1) and unchanged

size of clone group (value: 6). There is no clone migration for this pattern. While the "High Density Strong Up" pattern shown in Table 4.3 contains clone migration. Since this pattern includes a continuous increasing evolution for median distances and a wave increasing evolution for the size of clone group. For instance, the median distance in clone group is increased continuously, *i.e.,*from 0 to 2 then from 2 to 4. The size of clone group is increased from 2 to 3, but the size is changed to 4 in the middle, which is a wave increasing evolution.

4.3 Research Questions

- **RQ4.1: Do clone mutation and clone migration occur frequently in software systems?**

  We examine if clone mutation and clone migration exist in clone genealogies and their frequency. We observe that clone mutation and clone migration affect respectively 31% and 48% of clone genealogies in JBoss, 61% and 56% of clone genealogies in Apache-Ant, and 40% and 68% of clone genealogies in ArgoUML. Overall, the two evolution phenomenons affect an important number of clones and are therefore worth investigating further.

- **RQ4.2: Are some clone mutation more fault-prone than others?**

  We analyze whether clone groups that are mutated to certain types of clones are more fault-prone than others. Results show that clone genealogies predominated by Type-2 or Type-3 clones are more prone to faults than clone genealogies predominated by Type-1 clones. The clone mutation between Type-1 clones and near-miss (Type-2 and Type-3) clones increases the risk for faults.

- **RQ4.3: Are some clone migration more fault-prone than others?**

  We use the metric proposed by Kamiya *et al.* [7] to measure the distance between every two code segments (contained in a clone group) and identify different clone migration patterns followed by clone groups. We analyze whether clone groups following certain clone migration patterns are more fault-prone. Results show that in general, clone groups involved in migration patterns characterized by an increase of the distance between cloned code segments, are more fault-prone than others. Globally, a modification of the location of cloned code segments during the evolution of a software system increases the risk for faults in the system.

4.4 RQ4.1: Do clone mutation and clone migration occur frequently in software systems?

Motivation.

Since clones are proved more fault-prone than non-cloned code for software system as Kamiya *et al.* [7] conclude, clone mutation and clone migration may make the clones more fault-prone due to more changes on clones. This question is preliminary to **RQ4.2** and **RQ4.3**. It aims at providing quantitative evidences of the occurrence of clone mutation and clone migration in our studied systems. If these two phenomenons are very frequent, then they are worth studying in more details in order to advise developers and managers about potential side effects resulting from those phenomenons.

Approach.

Following the method introduced in Section 3.1.3, we build clone genealogies from

Table 4.4: Number of Clone Genealogies that Follow Clone Mutation

Patterns	JBoss		Apache-Ant		ArgoUML	
	number	%	number	%	number	%
G<1>	71	4.27	2565	11.17	251	1.61
G<2>	587	35.34	1994	8.68	5706	36.54
G<3>	492	29.62	4497	19.59	3348	21.44
G<1,2>	195	**11.74**	7189	**31.31**	5193	**33.25**
G<1, 3>	184	11.08	3632	15.82	471	3.02
G<2, 3>	120	7.22	1999	8.71	459	2.94
G<1, 2, 3>	12	0.72	1085	4.73	188	1.20
Total	**1661**	**100**	**22961**	**100**	**15616**	**100**
Total Mutation %	-	**30.77**	-	**60.56**	-	**40.41**

three subject systems to address this research question. We classify the clone genealogies using the clone mutation patterns described in Table 4.1 and clone migration patterns described in Table 4.2 respectively. For each pattern of clone genealogy, we report the number of occurrences in each system and the proportion of clone genealogies in each pattern. Moreover, to compare the existences of clone genealogies for clone mutation and clone migration patterns, we compute the total proportion of clone genealogies that belong to the patterns that contain clone mutation and clone migration respectively.

Findings.

Table 4.4 lists the seven clone mutation patterns of clone genealogies defined in Section 4.1 and the number and proportion of their occurrences for each of three subject systems. For each pattern of clone genealogies for each subject system, we present the number and the percentage of clone genealogies that belong to that pattern. Similar, Table 4.5 presents the number and the proportion of clone genealogies that follow each one of ten clone migration patterns identified previously in Section 4.2.

Table 4.5: Number of Clone Genealogies that Follow Clone Migration Patterns

Migration patterns	JBoss		Apache-Ant		ArgoUML	
	number	%	number	%	number	%
Constant	865	52.08	10107	44.02	4921	31.51
Wave Stable	317	**19.08**	5401	**23.52**	5543	**35.50**
High Density Strong Up	44	2.65	220	0.96	28	0.18
Low Density Strong Up	182	10.96	66	0.29	57	0.37
High Density Wave Up	40	2.41	1751	7.63	2118	13.56
Low Density Wave Up	173	10.42	506	2.20	216	1.38
High Density Strong Down	4	0.24	1682	7.33	558	3.57
Low Density Strong Down	23	1.38	515	2.24	48	0.31
High Density Wave Down	0	0.00	1617	7.04	298	1.91
Low Density Wave Down	13	0.78	1096	4.77	1829	11.71
Total	**1661**	**100**	**22961**	**100**	**15616**	**100**
Total Migration %	-	**47.92**	-	**55.98**	-	**68.49**

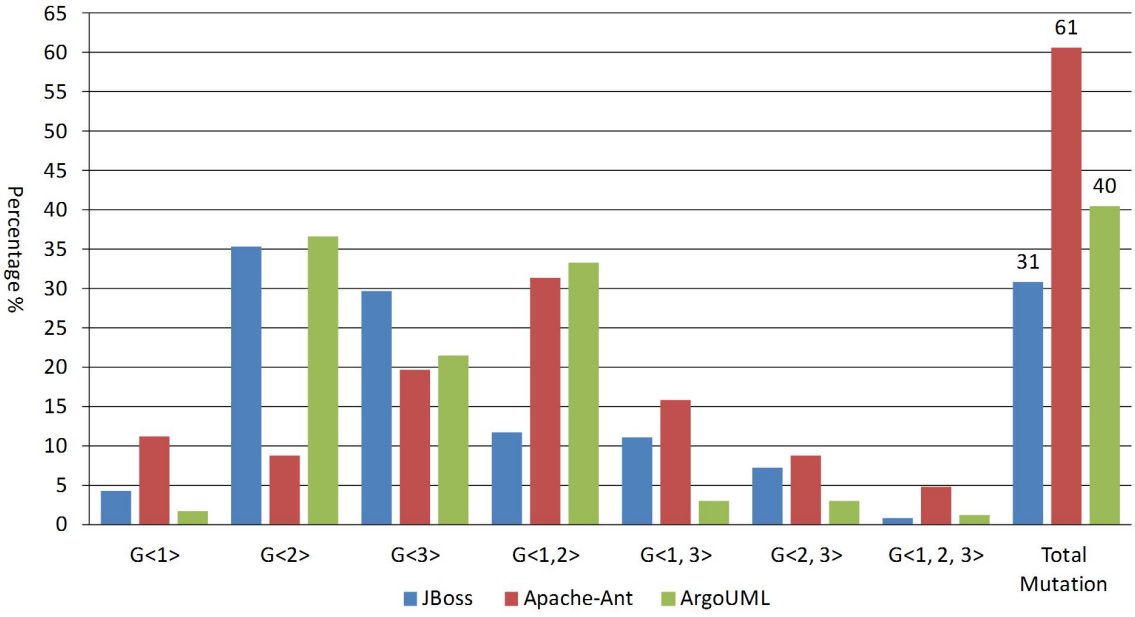


Figure 4.1: Result of Clone Genealogies that Follow Clone Mutation Patterns

Figure 4.1 and Figure 4.2 show the same results in Table 4.4 and Table 4.5 respectively. From Figure 4.1, we can find out the percentage of each clone mutation

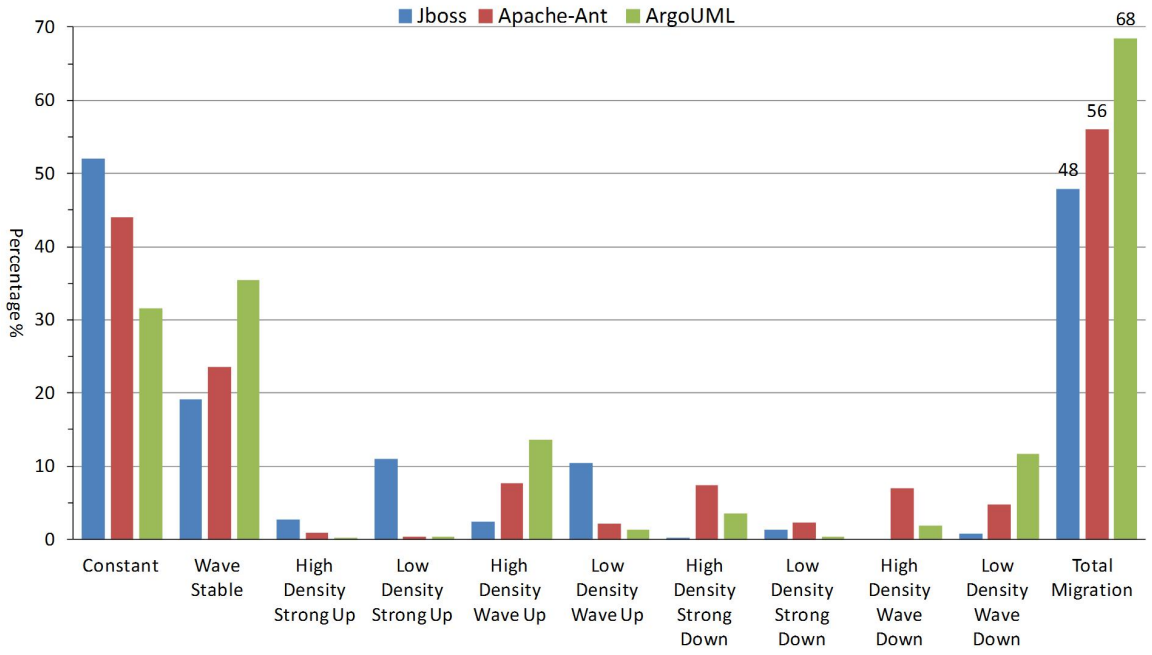


Figure 4.2: Result of Clone Genealogies that Follow Clone Migration Patterns

pattern in all patterns and the total percentage for clone mutation patterns. Similarly, Figure 4.2 shows the total percentage for clone migration patterns. More discussions are made by following paragraphs.

As summarized in Table 4.4, there are 30.77% clone genealogies belong to clone mutation patterns in JBoss, *i.e.,* G<1,2>, G<1,3>, G<2,3>, or G<1,2,3>. This means about one third of clone genealogies experience clone mutation. For Apache-Ant, 60.56% of clone genealogies experience clone mutation. And for ArgoUML, 40.41% of clone genealogies belong to the clone mutation patterns. As shown in Table 4.4, the most frequent pattern of clone mutation in clone genealogies is between Type-1 clone groups and Type-2 clone groups (*i.e.,* G<1,2>) for all three subject systems.

Similar, Table 4.5 shows the number and proportion of clone genealogies in different clone migration patterns. For clone migration, at least one clone segment has been changed on location in software systems during the evolution of about 48% of the clone groups. For Apache-Ant and ArgoUML two systems, there are respectively nearly 56% and about 68% of clone genealogies experience clone migration. The most frequent clone migration pattern is the "Wave Stable" pattern for all three subject systems.

Overall, we conclude that both clone mutation and clone migration two phenomenons affect a significant number of clones of clone genealogies for all three subject systems.

In the following two research questions, we investigate these two phenomenons throughly to identify whether some clone mutation patterns and clone migration patterns are more risky than other patterns in clone genealogies.

4.5 RQ4.2: Are some clone mutation more fault-prone than others?

Motivation.

Developers are interested in identifying areas in their software systems that are more likely to contain faults. Cloned code have been reported to contain more faults than non-cloned code [20, 29, 28]. Because a software system can contain up to 20% of cloned code [5], it can be very expensive to monitor all cloned code in a software system. Therefore, it will be interesting for developers to identify clones that are most at risk of faults, in order to allocate their limited testing and review resources towards these clones. A change that modifies the type of a clone group (*i.e.,* a mutation) could affect the ability of developers' to keep track of all related

clone segments in the clone group. Developers may have trouble propagating changes to all clone segments in the group consistently; resulting in an increased risk for faults. In this research question we examine the mutation of clones during software evolution. More precisely, we analyze the fault-proneness of the seven patterns of clone genealogies identified in Section 4.1. We aim to identify risky types of clone mutation that should be highlighted for monitoring.

Approach.

For each subject system, we build the clone genealogies for clone groups using the approach introduced in Section 3.1.3. Then we categorize the clone genealogies by identifying different clone mutation patterns shown in Table 4.4. We compute the number of fault-containing and fault-free clone genealogies in each clone mutation pattern for all three subject systems. We perform the Chi-square test introduced in Section 3.3.2 and calculate the odds ratio (OR) introduced in Section 3.3.1 to test this null hypothesis (there is no H_1 because **RQ4.1** is exploratory): *H_2: Each clone mutation pattern has the same proportion of clone genealogies that experience fault fixes.*

For the computation of OR, we select the first clone mutation pattern of clone genealogies that containing only Type-1 clone groups (*i.e.,* G<1>) as the control group. We consider each of the remaining patterns as experimental group for the control group. We perform the Chi-square test using the 5% level (*i.e., p*-value < 0.05) as the threshold for significant results.

Because our detection of Type-3 clones is done with a selected similarity threshold of 80% (shown in Table 3.1), we perform a sensitivity analysis to assess the impact of this chosen threshold on the results. Precisely, we detect clones for Type-3 clones

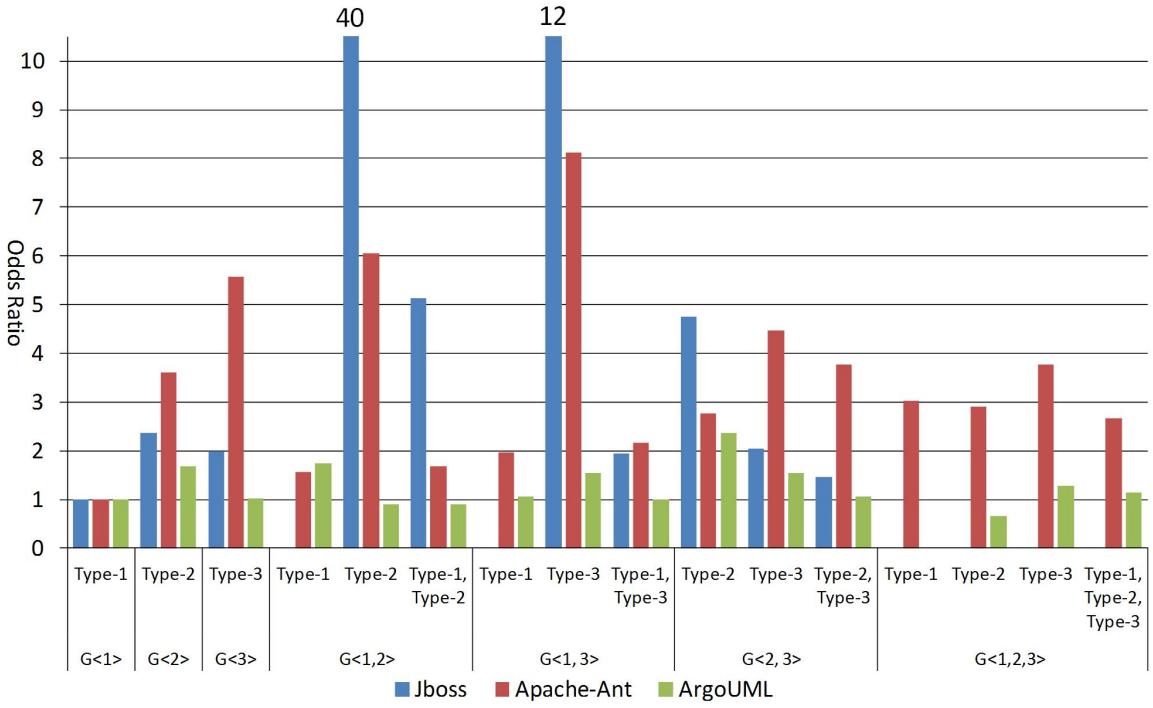


Figure 4.3: Odds Ratio Result of the Clone Mutation Patterns of Clone Genealogies

using other five similarity thresholds, *i.e.,* 70%, 75%, 80%, 85% and 90%. For each of these similarity thresholds, we build clone genealogies, classify them following the categorization described in Table 4.4, and repeat the testing of H_2 using the Chi-square test and OR.

Findings.

Table 4.6 presents the results of OR and Chi-square test for the seven clone mutation patterns of clone genealogies shown in Table 4.4. In the computation of OR, the pattern with only Type-1 clone (G<1>) is the control group. As summarized in the last row of Table 4.4, the results for Chi-square test of three subject systems are all statistically significant. We analyze the results of all the six similarity thresholds to detect clones of Type-3 clones. The results of the sensitivity analysis are presented

in Figure 4.4 and are all statistically significant. The different lines connecting some points in Figure 4.4 show the results for different similarities. Each line of points presents the results for one similarity.

Figure 4.3 shows the same results in Table 4.6. From Figure 4.3, we find that some clone mutation patterns are more fault-prone than G<1> pattern. More discussions are made by following paragraphs.

To make conclusion, we remove the exception on results of one or two similarities for some patterns in JBoss and ArgoUML. The exception on results come from the small number of clone genealogies. In summary, we can reject H_2. We discuss the results by each pattern shown in Table 4.6 in the following paragraphs.

Genealogies G<1>, G<2>, or G<3>:

In the results of JBoss and ArgoUML, we can conclude that clone genealogies containing only Type-2 clone groups (*i.e.,* G<2>) are more fault-prone than clone genealogies containing only either Type-1 or Type-3 clone groups. While for Apache-Ant, clone genealogies containing only Type-3 clone groups (*i.e.,* G<3>) have higher risk for faults. All these conclusions are consistent with the results of the sensitivity study. But the only exception is clone genealogies containing Type-3 clones (G<3>) in ArgoUML detected with similarity threshold of 70% are more fault-prone than G<2> clone genealogies.

Genealogies G<1,2>:

In the results, the ORs for clone genealogies containing Type-1 and Type-2 clone groups (G<1,2>) are generally higher than the ORs for clone genealogies containing only either Type-1 or Type-2 clone groups (G<1> or G<2>). This means the clone mutation between Type-1 and Type-2 has higher fault-proneness. In JBoss and

Apache-Ant, clone genealogies in G<1,2> pattern predominated by Type-2 clone groups are more fault-prone. For ArgoUML, clone genealogies of G<1,2> pattern with a dominant of Type-1 clones are more risky. Moreover, the equal occurrence of Type-1 and Type-2 clone groups in G<1,2> pattern reduces the risk for faults. This finding is also confirmed by results from sensitivity analysis.

Genealogies G<1,3>:

Similar to G<1,2>, ORs for genealogies containing both Type-1 and Type-3 clone groups (G<1,3>) are in general higher than ORs for genealogies containing either Type-1 or Type-3 clones only; implying that the mutation of a clone from Type-1 to Type-3 or conversely increases the risk for faults. G<1,3> genealogies predominated by Type-3 clones are more risky. For JBoss and ArgoUML, when Type-1 and Type-3 clones are equally frequent in a G<1,3> genealogy, this risk for fault is reduced. These findings are confirmed by the sensitivity analysis for all cases, except Type-3 clones are detected using either 70% or 75% similarity threshold in ArgoUML.

Genealogies G<2,3>:

For JBoss and ArgoUML, ORs for clone genealogies containing Type-2 and Type-3 clone groups (G<2,3>) are generally higher than ORs for clone genealogies containing only either Type-2 or Type-3 clone groups (G<2>, G<3>). This finding claims that the clone mutation between Type-2 and Type-3 increase the fault-proneness for those two systems. While for JBoss and ArgoUML, the G<2,3> pattern predominated by Type-2 clones are the most risky clone mutation pattern. In Apache-Ant, the clone mutation between Type-2 and Type-3 decreases the fault-proneness of clone genealogies. G<2,3> clone genealogies dominated by Type-2 are less fault-prone for Apache-Ant. These results are consistent with the sensitivity

Table 4.6: Odds Ratio Result of the Clone Mutation Patterns of Clone Genealogies

Systems		JBoss			Apache-Ant			ArgoUML		
Patterns	Dominant Clone Type	# Genealogies		ORs	# Genealogies		ORs	# Genealogies		ORs
		Faults	No Faults		Faults	No Faults		Faults	No Faults	
G<1>	Type-1	21	50	1	364	2201	1	115	136	1
G<2>	Type-2	293	294	2.37	745	1249	3.61	3355	2351	1.69
G<3>	Type-3	224	268	1.99	2159	2338	5.58	1550	1798	1.02
G<1,2>	Type-1	3	0	-	186	723	1.56	34	23	1.75
	Type-2	50	3	39.68	1091	1091	6.05	1649	2157	0.90
	Type-1, Type-2	95	44	5.14	890	3208	1.68	573	757	0.90
G<1,3>	Type-1	6	0	-	28	86	1.97	9	10	1.06
	Type-3	73	14	12.41	1447	1078	8.12	177	135	1.55
	Type-1, Type-3	41	50	1.95	261	732	2.16	64	76	1
G<2,3>	Type-2	2	1	4.76	55	120	2.77	38	19	2.37
	Type-3	25	29	2.05	480	648	4.48	117	89	1.55
	Type-2, Type-3	24	39	1.46	267	429	3.76	93	103	1.07
G<1,2,3>	Type-1	0	0	-	15	30	3.02	4	0	-
	Type-2	0	0	-	24	50	2.90	21	37	0.67
	Type-3	8	0	-	283	454	3.77	39	36	1.28
	Type-1, Type-2, Type-3	4	0	-	70	159	2.66	25	26	1.14
p-values		**<0.05**			**<0.05**			**<0.05**		

analysis study. The exceptions are for the results of Type-3 clones detected by the 95% similarity threshold in JBoss and ArgoUML.

Genealogies G<1,2,3>:

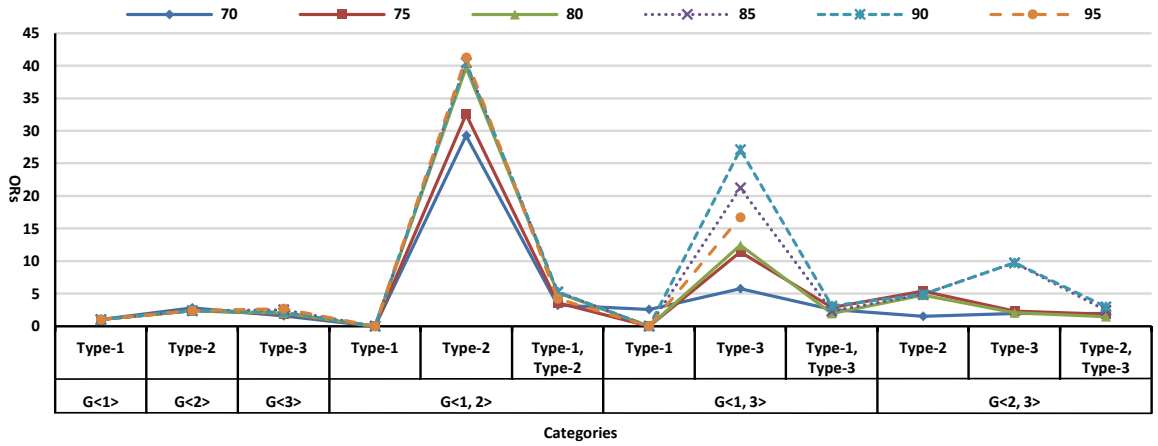
For the clone genealogies containing all three types of clones with same frequency, the riskiness is reduced. Clone genealogies of G<1,2,3> pattern that are dominated by Type-3 clones have the highest fault-proneness. The sensitivity analysis for other similarities verify these conclusions for Apache-Ant. While in ArgoUML, when Type-3 clones are detected for all other five similarity thresholds (*i.e.,* 70%, 75%, 85%, 90% or 95%), clone genealogies for the G<1,2,3> pattern with equally frequency for all three clone types are more fault-prone than others.

These results show that the predominance of Type-3 and Type-2 clones in a clone genealogy increases the risk for faults significantly. Developers should be cautious when modifying cloned code which genealogies are predominated by Type-2 or Type-3 clones. Overall, we conclude that the existence of clone mutation of clone groups with clone types of Type-2 or Type-3 makes the clone genealogies more fault-prone. Between Type-2 and Type-3 clone groups, Type-3 clone groups are more dangerous for developers. When all the clone types in a clone genealogy are equally frequent, this risk for fault is reduced.

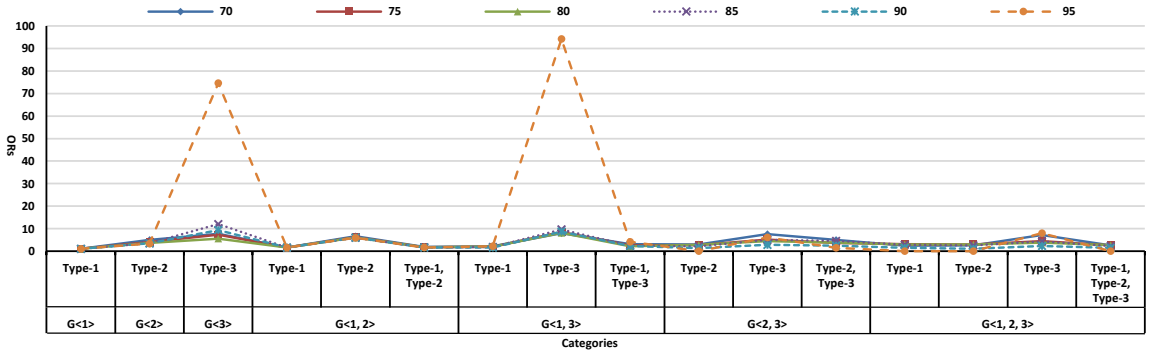
4.6 RQ4.3: Are some clone migration more fault-prone than others?

Motivation.

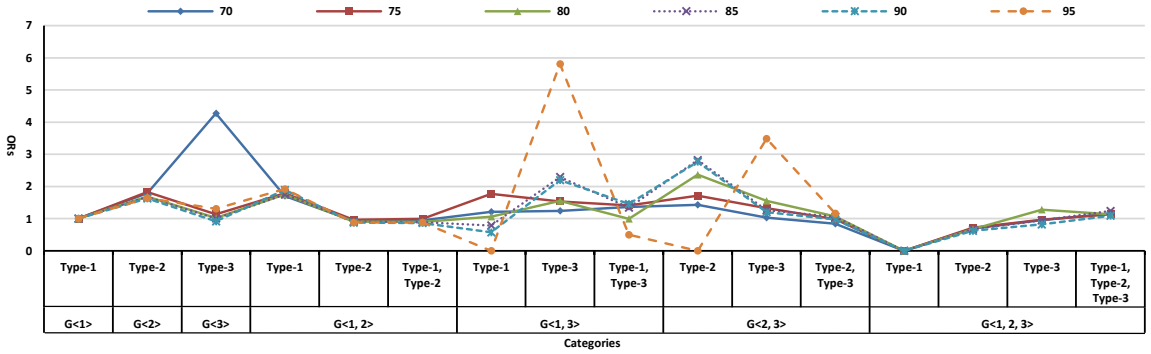
When performing modifications on cloned parts of a software system, developers should be aware of all the code segments involved in the clone groups. A developer overlooking a cloned code segment is at risk of introducing a fault in the software



(a) JBoss



(b) Apache Ant



(c) ArgoUML

Figure 4.4: Sensitivity Analysis Results for Clone Mutation in JBoss, Apache-Ant, and ArgoUML

system. The purpose of this research question is to investigate the potential impact on fault-proneness of the migration of cloned code segments across the directories of a software system. The clones in a group can be changed, so the directories of those clones in that group can also be changed. More specifically, we want to verify using the ten migration patterns described in Table 4.5, if some displacements of cloned code segments during maintenance and evolution activities are likely to increase the risk for faults in a system.

Approach.

For each clone group, we compute the distance ($d_{dir}(A, B)$) between A and B as the method defined in Section 4.2. More specifically, we calculate the distance between each two code segments in clone groups of the clone genealogies extracted in Section 4.4. We compute the size of clone group. We identify the clone migration patterns based on combinations of the evolution trend of the size of clone group, and the evolution trend of the median distance among the code segments in clone groups. We perform this computation for all clone genealogies using the standard described in Table 4.2. Then we categorize the clone genealogies using the clone migration patterns we defined.

For each clone migration pattern shown in Table 4.2, we calculate the numbers of both fault-containing and fault-free clone genealogies. We define the following null hypothesis: H_3*: The proportion of clone genealogies containing fault fixes is the same for ten clone migration patterns.*

We compute the Chi-square test and OR to test H_3. We choose the "Constant" clone migration pattern as the control group for OR. We consider each of the other nine patterns as an experimental group for the control group. We use 5% threshold to

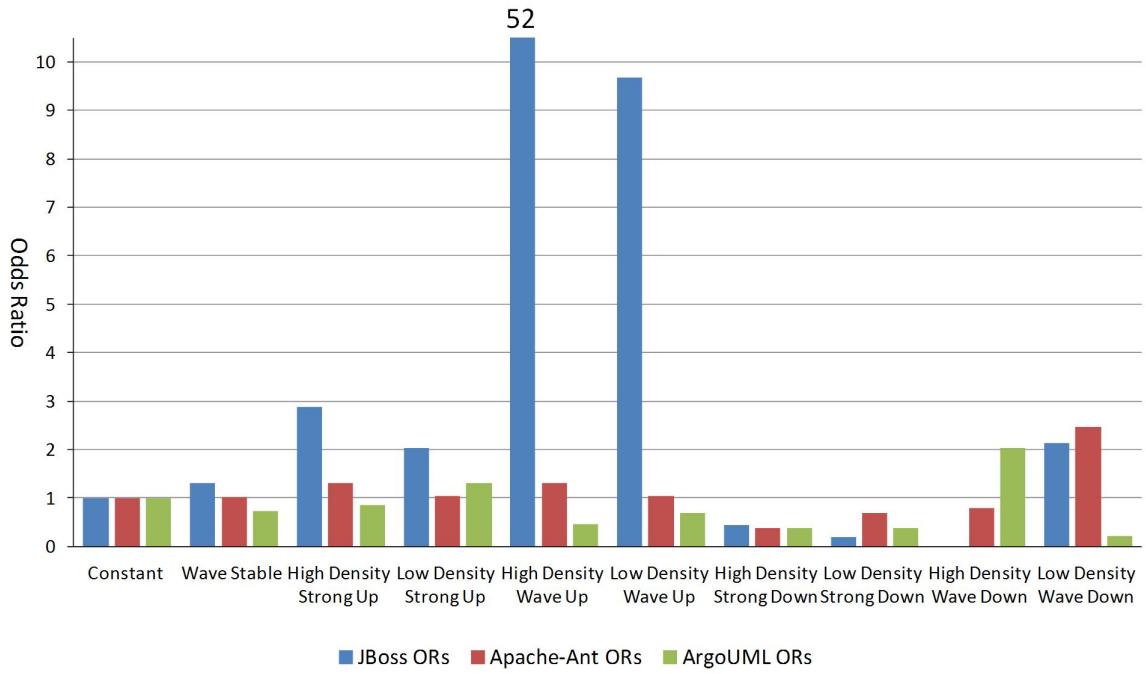


Figure 4.5: Odds Ratio Result of the Clone Migration Patterns of Clone Genealogies

identify the significance result from the Chi-square test. Similar with 4.5, we conduct the sensitivity analysis to evaluate the potential impact on clone detection results for Type-3 clones from the selection of similarity threshold. We make our conclusion from the clone detection results of all similarity thresholds.

Findings.

As shown in Table 4.7, the results describe the Chi-square test and odd ratios of fault-proneness for ten clone migration patterns described in Table 4.5. By checking the results of Chi-square test, all results are statistically significant for three systems, *i.e.,* JBoss, Apache-Ant, and ArgoUML. In summary, we can reject H_3.

Figure 4.5 shows the same results in Table 4.7. From Figure 4.5, we can find that the clone migration in some clone migration patterns are more fault-prone than the

one in "Constant" pattern. A clear discussion is made by following paragraphs.

In JBoss and Apache-Ant, most clone migration cross the different locations of a software system increase the risk for fault. The results in Table 4.7 shows "High Density Strong Up", "Low Density Strong Up", "High Density Wave Up", and "Low Density Wave Up" patterns are more risky than the "Constant" pattern in JBoss and Apache-Ant. While for ArgoUML, the "Low Density Strong Up" pattern has higher risk for faults than the "Constant" pattern of clone genealogies. All these findings claim that the risk for faults of clone genealogies will be increased if the distances among the clones in a clone group is increased.

We find that removing or adding a clone from a clone group (*i.e.,* Wave) is dangerous. This is proved by the OR of the "High Density Wave Down" pattern for ArgoUML (*i.e.,* 2.03), the OR of the "Low Density Wave Down" pattern for JBoss (*i.e.,* 2.14) and Apache-Ant (*i.e.,* 2.47). These findings are generally consistent with the results from sensitivity analysis for three subject systems presented on Figure 4.6. Similar with Figure 4.4, the different lines connecting several points in Figure 4.6 present the results of different similarities. Therefore, we suggest that developers avoid moving cloned code during the maintenance and updating activities.

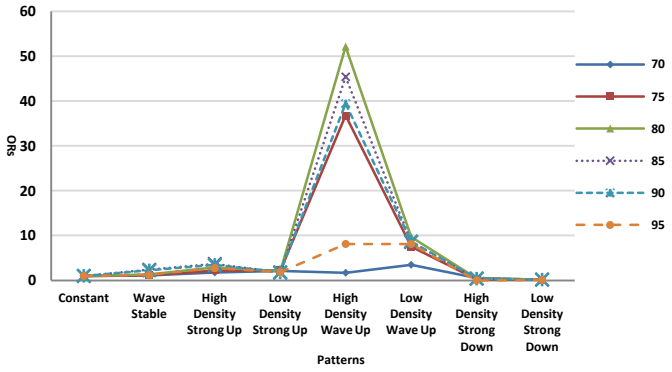
In conclusion, we conclude that clone migration makes the clone genealogies more fault-prone. Increasing the distances among cloned codes in a clone group is a fault-prone activity.

4.7 Summary

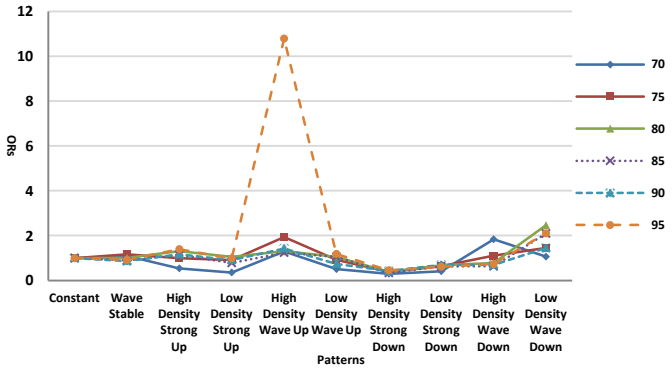
To examine the potential effect of LOC (line of code) on our results, we computed and compared the LOC of all clones (*i.e.,* Type-1, Type-2, and Type-3 clones) contained

Table 4.7: Odds Ratio Result of the Clone Migration Patterns of Clone Genealogies

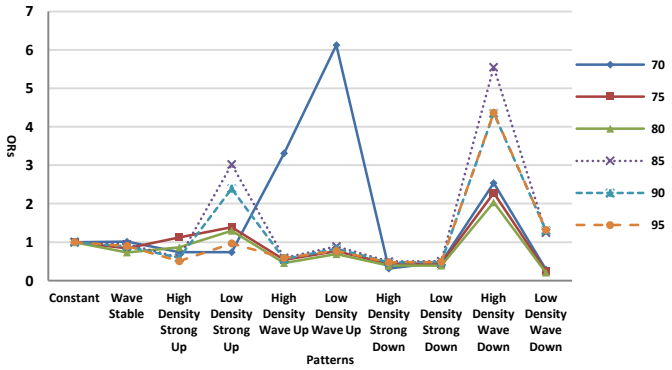
Clone Migration Patterns	JBoss			Apache-Ant			ArgoUML		
	# Faults	# No Faults	ORs	# Faults	# No Faults	ORs	# Faults	# No Faults	ORs
Constant	370	495	1.00	3709	6398	1.00	2986	1935	1.00
Wave Stable	156	161	1.30	1991	3410	1.01	2939	2604	0.73
High Density Strong Up	30	14	2.87	95	125	1.31	16	12	0.86
Low Density Strong Up	110	72	2.04	25	41	1.05	38	19	1.30
High Density Wave Up	39	1	52.18	755	996	1.31	876	1242	0.46
Low Density Wave Up	152	21	9.68	192	314	1.05	111	105	0.69
High Density Strong Down	1	3	0.45	301	1381	0.38	209	349	0.39
Low Density Strong Down	3	20	0.20	146	369	0.68	18	30	0.39
High Density Wave Down	0	0	-	506	1111	0.79	226	72	2.03
Low Density Wave Down	8	5	2.14	645	451	2.47	444	1385	0.21
p-values	**<0.05**			**<0.05**			**<0.05**		



(a) JBoss



(b) Apache Ant



(c) ArgoUML

Figure 4.6: Sensitivity Analysis Results for Clone Migration in JBoss, Apache-Ant, and ArgoUML

in the genealogies extracted from three subject systems. We observed that, except in the case of JBoss, where the LOC of Type-1 clones are smaller than Type-2 and Type-3 clones, in general, there is no significant difference between the sizes of the clones. Therefore, we can conclude that the size of clones cannot affect the results in this empirical study.

In this chapter, we introduce our first study about clone mutation and clone migration. We investigate clone mutation and clone migration for clone groups in clone genealogies of software systems. We define seven clone mutation patterns and ten clone migration patterns in clone genealogies. In these patterns of clone genealogies, we identify the clone mutation and clone migration patterns that have the highest frequencies. Then we examine the fault-proneness of the different clone mutation patterns and clone migration patterns of clone genealogies to investigate the impacts of these two phenomenons on the fault-proneness of clone genealogies.

Our study results show that clone genealogies predominated by Type-2 or Type-3 clone groups have higher risk for faults than clone genealogies predominated by Type-1 clone groups. The clone mutation between Type-1 and Type-2 clone groups or between Type-1 and Type-3 clone groups increase the fault-proneness of clone genealogies.

We conclude that clone genealogies containing clone migration with the increasing distance among clones in clone groups are more fault-prone. Specifically, changing the location of cloned code during the clone genealogies increases the risk for faults for a software system. It is risky to move a clone code for developers.

Overall, we conclude that clone mutation and clone migration are frequent and fault-prone. Some clone mutation and clone migration patterns are required to be

monitored carefully.

Chapter 5

An Empirical Study on the Fault-Proneness of Clone Migration in Clone Genealogies

In this chapter, we discuss the second empirical study that is extended from our first empirical study. We firstly investigate whether clone migration will affect the risk for faults of three clone contexts, *i.e.,* clones, clone groups, and clone genealogies. Moreover, we examine the impact of clone migration associated with clone mutation in clone genealogies on the fault-proneness of clone genealogies. Finally, we introduce whether the length of time interval between clone migration and last change of the cloned code affects the risk for faults of clone genealogies.

We propose three research questions about further studies on clone mutation and clone migration in clone genealogies. Compared with the first study in Chapter 4, this study uses same study design process, but different approaches to identify the patterns of clone mutation and clone migration in clone genealogies. In the following two sections, we introduce the two approaches used in this study in details.

Since in our first empirical study, we proved the considerate existence of clone migration and clone mutation in clone genealogies, we want to find more details how

Table 5.1: Migration Proportion of Clone Groups and Migration Density of Clone Genealogies

Levels	0	1	2	3	4
Names	No Migration	Very Low Migration	Low Migration	High Migration	Very High Migration
Migration Proportion of Clone Groups	P = 0	$0<P \leqslant 0.25$	$0.25<P \leqslant 0.5$	$0.5<P \leqslant 0.75$	$0.75<P \leqslant 1$
Migration Density of Clone Genealogies	D = 0	$0<D \leqslant 0.25$	$0.25<D \leqslant 0.5$	$0.5<D \leqslant 0.75$	$0.75<D \leqslant 1$

different patterns of clone migration affect the risk for faults. In this empirical study, we aim to classify clone migration from three different methods to identify how clone migration affects fault-proneness of clones, clone groups, and clone genealogies.

5.1 Migration Proportion and Migration Density

Since there are different numbers of migrated clones in clone groups and clone genealogies, we aim to examine the fault-proneness of clone groups and clone genealogies that have different frequencies of migrated clones. We categorize the clone groups by the proportions of migrated clones in a clone group. While for clone genealogies, we define the migration density to measure the frequency of clone migration. To compute the migration density (D), we use the number of clone migration and the number of clone groups of clone genealogies and follow the Equation (5.1), where N_m presents the number of clone migration in the entire clone genealogy, and N_g presents the number of clone groups in the clone genealogy.

$$D = \frac{N_m}{N_g - 1} \tag{5.1}$$

After computing the proportion of migrated clones in clone groups and the migration density in clone genealogies, we divide the clone groups and clone genealogies into different levels respectively based on the frequency of clone migration. We define five levels (*i.e.,* level 0 to 4) for different proportions of migrated clones in clone groups. For clone genealogies, we also define five levels (*i.e.,* level 0 to 4). Each level presents a different proportion of the maximum migration density for all clone genealogies. As shown in Table 5.1, the first row shows level 0 to level 4, each of which presents a different migration frequency range for clone groups and clone genealogies.

In details, level 0 presents clone group or clone genealogies without clone migration, level 1 presents very low migration frequency, level 2 presents low migration frequency, level 3 presents high migration frequency, and level 4 presents very high migration frequency. Thus level 4 refers to the largest migration density for all clone genealogies and the largest proportion of migrated clones for clone groups. Shown in the third row of Table 5.1, P refers to the proportion of migrated clones in a clone group. For example, the clone groups belong to level 0 have no clone migration, which means they have 0 percent of migrated clones (P=0). Clone groups belong to level 4 (very high migration) have the largest frequency of migrated clones. Similar to clone groups, D in the fourth row presents the migration density of clone genealogies. For example, the clone genealogy with a migration density (D) locating in a range from 0.75 to 1 has the largest frequency of clone migration.

5.2 Patterns of Clone Migration Associated with Clone Mutation

If the clone type is changed in clone migration, the migrated clone will have a higher or lower similarity with another clone. This phenomenon for changing clone types along

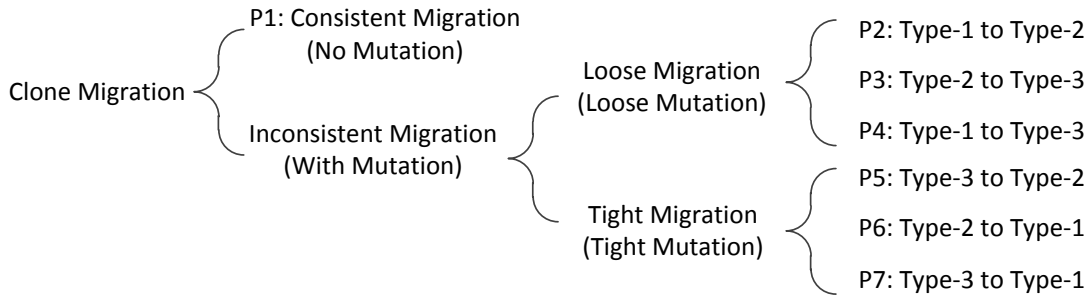


Figure 5.1: Clone Migration Patterns of Clone Genealogies

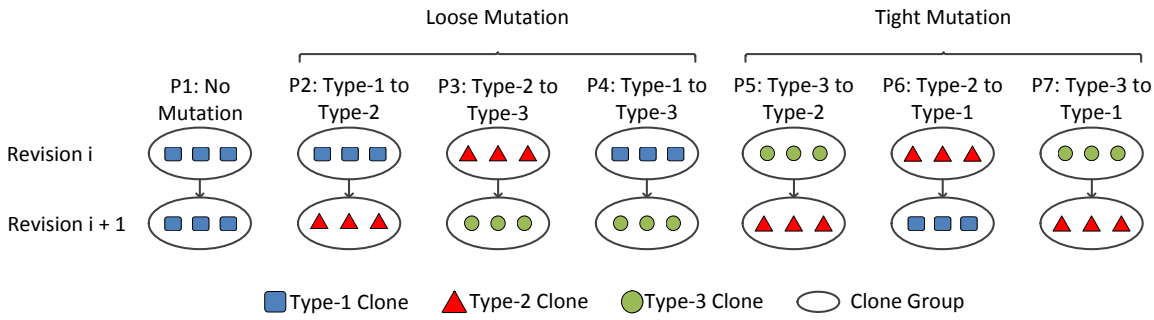


Figure 5.2: Example of Clone Mutation Patterns

with clone migration is defined as clone migration associated with clone mutation. For the modifications of clone types along with clone migration, we define the one increasing the similarity as the tight mutation and the one decreasing the similarity as loose mutation. More specifically, loose mutation stands for changing clone type from a lower type with a higher similarity to a higher type with a lower similarity. While the tight mutation changing the clone type in a opposite direction.

We define seven clone migration patterns, *i.e.,* P1 to P7, to identify the clone migration associated with clone mutation. In these seven patterns shown in Figure 5.1, P1 is the consistent migration pattern without clone mutation. We define three

patterns for both tight mutation and loose mutation. For the clone migration associated with loose mutation, as shown in the right part of Figure 5.1, we define three patterns with all change possibilities among three types of clones, *i.e.,* P2: change from Type-1 clone to Type-2 clone, P3: change from Type-2 clone to Type-3 clone, and P4: change from Type-1 clone to Type-3 clone. Similar as loose migration, tight migration has three patterns, *i.e.,* P5: change from Type-3 clone to Type-2 clone, P6: change from Type-2 clone to Type-1 clone, and P7: change from Type-3 clone to Type-1 clone.

We present the examples of seven clone migration patterns along with clone mutation in Figure 5.2. For example, the type of a clone is changed from Type-2 to Type-3 for P3: Type-2 to Type-3. In summary, we define three loose mutation patterns (*i.e.,* P2, P3, and P4) and other three tight mutation patterns (*i.e.,* P5, P6, and P7), which present all the possible clone mutation scenarios among the three types of clones.

5.3 Research Questions

- **RQ5.1: Are clone genealogies that experienced clone migration more fault-prone than other clone genealogies?**

  We categorize clone migration in clone genealogy in terms of clone types at three contexts: clones, clone groups, and clone genealogies. We also categorize the clone groups and clone genealogies by the frequency of clone migration. All migrated clones are identified at the clone context. At clone group (respectively clone genealogy) context, we examine the impact of the frequency of clone migration on the fault-proneness of clone groups (respectively clone genealogies). We observe that the clone migration occur in clones, clone groups, and clone

genealogies increase the fault-proneness of migrated clones. The clone groups with a large proportion of migrated clones are more fault-prone than the clone groups with a smaller proportion of migrated clones. Overall, the clone migration affects the fault-proneness of a significant number of code segments, thus clone migration is worth investigating further.

- **RQ5.2: Is clone migration associated with clone mutation more fault-prone than other clone migration?**

  We analyze whether the clone migration associated with clone mutation is more fault-prone the clone migration without clone mutation. Specifically, we investigate the impacts of different patterns of clone mutation (*i.e.,* from Type-1 to Type-2) during clone migration on the fault-proneness of migrated clones. Results show that clones experiencing both clone migration and clone mutation are more fault-prone than clones that experienced only clone migration. Moreover, loose migration where the similarity of clones is decreased (*i.e.,* the clone mutation from Type-1 to Type-2) are found to be more fault-prone in two of our three subject systems. The risk for faults is the highest when clone mutation contains Type-3.

- **RQ5.3: Does the time interval between the migrating change and the last change of the cloned code before migration affect the fault-proneness?**

  We investigate the impact of the length of time interval between the migrating change (clone migration) and the last change of the cloned code on the risk for faults of migrated clones. We divide the time interval into different period

levels and compare the fault-proneness in those period levels. We observe that a longer time interval between clone migration and the last change of the cloned code increases the risk for faults in the migrated clones.

5.4 RQ5.1: Are clone genealogies that experienced clone migration more fault-prone than other clone genealogies?

Motivation.

The first study in Chapter 4 has demonstrated the high frequency of clone migration in clone genealogies for all three subject systems. However, clone migration can be observed in different contexts: clones, clone groups, and clone genealogies. In this question, we want to study the impact of clone migration on these three clone contexts.

In particular, we are interested in understanding if migrated clones are more faulty than non-migrated clones in the context of a clone itself. We want to understand if the proportion of migrated clones in a clone group would affect the risk for faults when modifying the clones in the clone groups. Moreover, we examine the effect of clone migration over the evolution of clone groups (*i.e.,* clone genealogies). The results of this research question will enable developers to better estimate the efforts and the risks related to clone migration (*i.e.,* a change modifying the location a clone). This question is preliminary to **RQ5.2** and **RQ5.3**, which identify more migration patterns from two different aspects.

Approach.

To identify clone migration in three clone contexts, we classify clones genealogies into four categories as shown in Table 5.2. The four categories of clone genealogies

contain all the three clone types. For example, G<1> represents the clone genealogies containing only Type-1 clones. While G<1,2,3> category stands for the clone genealogies containing all three clones types.

Then we extract the fault-proneness of clone migration for clones and clone groups in each category of clone genealogies. We check the fault-proneness of clone migration in clone genealogies. Using four categories of clone genealogies, we want to identify the effect of clone migration on the fault-proneness for different clone contexts, *i.e.,* clones, clone groups, and clone genealogies. Using the approach described in Section 5.1, we compare the impact from clone migration in different frequencies on the risk for faults for clone groups and clone genealogies.

In particular, we detect each migrated clone to identify the impact of clone migration on the fault-proneness of code clones in the subsequent revisions. Since the frequencies of migrated clones in clone groups are different, we examine the proportions of migrated clones in a clone group to identify if they affect the fault-proneness of the clone group. While for the context of clone genealogies, because there are also different numbers of migrated clones in clone genealogies, we use the migration density defined in Section 5.1 to measure the frequency of clone migration. Then we investigate if a higher frequency of clone migration will make clone groups and clone genealogies more fault-prone. We conduct these computations for three clone contexts in all four categories of clone genealogies.

We calculate the p-value in Chi-square test and odds ratio (OR) to verify if clone migration affect the code clone in different clone context. For computing OR, we define the control group as non-migrated clones, clone groups without clone mutation (level 0 defined in Section 5.1), and clone genealogies without clone mutation (level

Table 5.2: Categories of Clone Genealogies

Categories	Clone types in the genealogy
G<1>	Type-1
G<2>	Type-2
G<3>	Type-3
G<1,2,3>	Type-1, Type-2, Type-3

0) respectively for the context of clones, clone groups, and clone genealogies. We use results for other four levels defined in Section 5.1 (level 1 to level 4) as experimental groups for both clone group and clone genealogy contexts.

To prevent the potential impact of choosing different similarity thresholds to detect clones of Type-3 clones on the results, we conduct the computation using six different similarity thresholds, *i.e.,* 70%, 75%, 85%, 90% and 95%, for two categories (G<3> and G<1,2,3> introduced in Section 5.2) that involved with Type-3 clones.

To compute the results, we build clone genealogies and identify clone migration for three subject systems based on the approaches described in Chapter 3. We examine the number of fault fixing changes for all migrated clones, clone groups and clone genealogies that contain migrated clones in four categories of clone genealogies. To test this research question, we define the following null hypothesis: H_1*: Clone migration does not affect the fault-proneness of clones, clone groups, and clone genealogies.*

Findings.

We summarize the OR results for migrated and non-migrated clones, different levels of clone groups, and different levels clone genealogies in Table 5.3. We analyze the OR results for for clone, clone group, and clone genealogy contexts in following three paragraphs.

Figure 5.3 and Figure 5.4 show the same results with 80% similarity in Table 5.3.

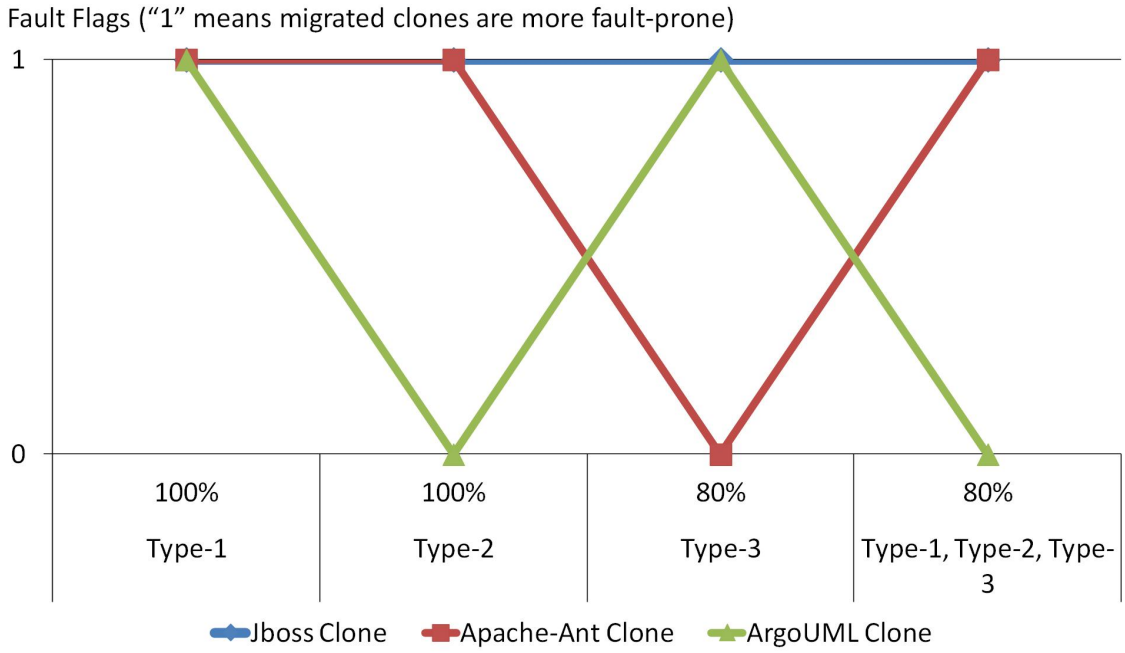


Figure 5.3: Fault-Proneness Result of Clone Migration in Clones

From Figure 5.3, we can find out if the migrated clones are more fault-prone than non-migrated clones. While Figure 5.4 shows the which proportion level of clone migration has the most fault-proneness for clone groups and which density level of clone migration has the most fault-proneness for clone genealogies. More discussions are made by following paragraphs for different clone contexts.

Clone Context:

To compare the fault-proneness between migrated clones and non-migrated clones, we compute the ORs for them. In the columns for clone context in Table 5.3, we define the flag as "1" when migrated clones have larger ORs than non-migrated clones. In this case, migrated clones are more fault-prone than non-migrated clones. The flag "0" represents the opposite result. Though "p-value" column of clone context shows

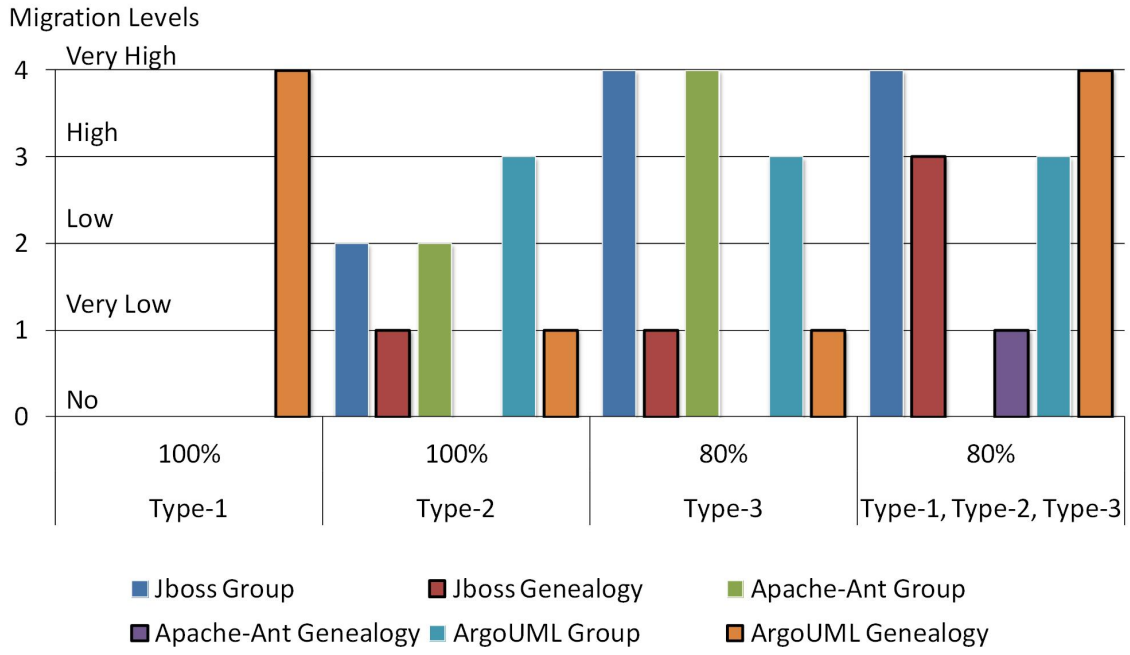


Figure 5.4: Fault-Proneness Result of Clone Migration in Clone Groups, and Clone Genealogies with 80% Similarity

that one value in Apache-Ant and three values in ArgoUML are larger than the threshold (0.05) we have defined for Chi-square test.

Overall, the results of the context of clones are statistically significant. Therefore, we can conclude from the results that migrated clones have higher risk for faults than non-migrated clones. The only two exceptions are Type-3 clones in G<3> clone genealogies of Apache-Ant and clones in G<1,2,3> clone genealogies of ArgoUML.

Clone Group Context:

In Table 5.3, the "proportion level" column shows the proportion level with the largest odd ratios, thus it is the most fault-prone proportion level. Moreover, the clone groups containing clone migration are more fault-prone than the clone groups containing no clone migration if the proportion level shown is not 0. And a higher

proportion level represents that a clone group with the larger proportion of migrated clones is more fault-prone than other clone groups. Since there are only two results for Chi-square test in JBoss are larger than our predefined threshold (0.05), we believe the overall result for the clone group context is statistically significant.

In the results in Table 5.3, most of the proportion levels are level 3 and level 4. This suggests that clone groups with the larger proportions of migrated clones are more fault-prone. In conclusion, for three subject systems, clone groups are more fault-prone when they contain clone migration and a higher frequency of migrated clones increases their risk for faults. While in all three subject systems, Type-1 clone groups (in G<1>) containing only non-migrated clones are more fault-prone. This exception may due to there are few clone migration in Type-1 clone groups.

Clone Genealogy Context:

As shown in Table 5.3, the "proportion level" column of clone genealogy context presents the most risky proportion levels for four categories of clone genealogies. The proportion level is computed by migration density and represents the frequency of clone migration in a clone genealogy. A higher proportion level refers to a higher migration density hence a higher frequency of clone migration in clone genealogies.

In the results of Table 5.3, there are only one value in Apache-Ant and four values in ArgoUML for Chi-square test that are larger than the predefined threshold (0.05). Therefore, overall the results for clone genealogies are statistically significant.

Because most of the proportion levels are larger than 0, we can conclude that clone migration make clone genealogies more fault-prone. The only exception is Type-1 clone genealogies (G<1>) without clone migration, which are more fault-prone than migration-containing clone genealogies in JBoss and Apache-Ant.

Table 5.3: Odds Ratio Result of Clone Migration in Clones, Clone Groups, and Clone Genealogies

Contexts		Clones						Clone Groups						Clone Genealogies					
System		JBoss		Apache-Ant		ArgoUML		JBoss		Apache-Ant		ArgoUML		JBoss		Apache-Ant		ArgoUML	
Clone Type	Similarity	Fault Flag	P-Value	Fault Flag	P-Value	Fault Flag	P-Value	Proportion Level	P-Value	Proportion Level	P-Value	Proportion Level	P-Value	Proportion Level	P-Value	Proportion Level	P-Value	Proportion Level	P-Value
G<1>	100%	1	<0.05	1	0.831	1	0.205	0	0.152	0	<0.05	0	<0.05	0	<0.05	0	<0.05	4	<0.05
G<2>	100%	1	<0.05	1	<0.05	0	<0.05	2	<0.05	2	<0.05	3	<0.05	1	<0.05	0	<0.05	1	<0.05
G<3>	70%	1	<0.05	1	<0.05	0	<0.05	4	<0.05	4	<0.05	2	<0.05	4	0.370	1	<0.05	0	<0.05
	75%	1	<0.05	1	<0.05	1	<0.05	4	<0.05	4	<0.05	3	<0.05	1	<0.05	2	0.156	1	<0.05
	80%	1	<0.05	0	<0.05	1	<0.05	4	<0.05	4	<0.05	3	<0.05	1	<0.05	0	<0.05	1	<0.05
	85%	1	<0.05	0	<0.05	1	0.082	2	<0.05	0	<0.05	2	<0.05	1	<0.05	3	0.164	1	<0.05
	90%	1	<0.05	0	<0.05	1	<0.05	2	<0.05	4	<0.05	2	<0.05	2	<0.05	3	<0.05	1	<0.05
	95%	1	<0.05	0	<0.05	1	<0.05	2	<0.05	4	<0.05	2	<0.05	0	<0.05	1	0.109	1	<0.05
G<1, 2, 3>	70%	1	<0.05	1	<0.05	0	<0.05	4	0.601	0	<0.05	0	<0.05	1	<0.05	2	<0.05	0	<0.05
	75%	1	<0.05	1	<0.05	0	<0.05	4	<0.05	0	<0.05	3	<0.05	1	<0.05	2	<0.05	3	<0.05
	80%	1	<0.05	1	<0.05	0	<0.05	4	<0.05	0	<0.05	3	<0.05	3	<0.05	1	<0.05	4	<0.05
	85%	1	<0.05	1	<0.05	0	<0.05	4	<0.05	0	<0.05	3	<0.05	1	<0.05	0	<0.05	2	<0.05
	90%	1	<0.05	0	<0.05	0	<0.05	4	<0.05	2	<0.05	3	<0.05	4	<0.05	1	<0.05	3	<0.05
	95%	1	<0.05	0	<0.05	0	0.459	2	<0.05	0	<0.05	3	<0.05	1	<0.05	2	0.132	2	<0.05

We observe that the results from the other five similarity thresholds are statistically significant, which suggests the difference between fault fixes in migration-containing and in non-migration-containing clones, clone groups, and clone genealogies receptively. Overall, we reject H_1.

Overall, we conclude that clone migration increase the risks for faults of clones, clone groups, and clone genealogies in software systems. In the following two research questions, we investigate clone migration in depth.

5.5 RQ5.2: Is clone migration associated with clone mutation more fault-prone than other clone migration?

Motivation.

When making clone migration for clone segments, the clone type could be changed due to the code changes on the cloned code segment. We refer this change on the clone type as clone mutation. While some cloned codes may have no clone mutation. We aim to identify the effect of the clone migration associated with clone mutation on the fault-proneness of cloned codes. To study the impact on risk for faults from the clone migration associated with clone mutation, we define seven clone migration patterns containing different clone mutation. More specifically, we try to identify if the existence of clone mutation in clone migration will leads to more fault fixes in migrated clones during the subsequent revisions. This study will help developers to evaluate the risk for changing cloned code and movement activities.

Approach.

After building the clone genealogies and identifying clone migration using the approaches introduced in Chapter 3, we detect clone mutation in these migrated

clones. Based on the seven clone migration patterns with different clone mutation defined in Figure 5.1, we categorize the cloned codes and compute the number of fault-containing and fault-free cloned codes with different clone mutation. We form the following null hypothesis: *H_2: The clone migration with and without clone mutation have the same opportunity to introduce defects.*

We detect Type-3 clones using six similarity thresholds, *i.e.,* 70%, 75%, 80%, 85%, 90% and 95%. We detect clones for Type-3 clones using different similarity thresholds to avoid the impact on our results from selecting the specific similarity threshold. For all these similarity thresholds, we locate the migrated clones in clone genealogies, classify these migrated clones into seven clone migration patterns shown in Figure 5.1 based clone mutation involved with those migrated clones. We test the H_2 using the Chi-square test and ORs for all these migrated clones.

We compute OR using all seven clone migration patterns (*i.e.,* P1 to P7) in Figure 5.1. We select the clone migration pattern (P1) without clone mutation as the control group. We use P2 to P7 clone migration patterns as six experimental groups for the control group. Same as before, we use the 5% threshold for Chi-square test.

Findings.

We summarize the results of Chi-square test and OR in Table 5.4. The clone migration patterns, *i.e.,* P1 to P7, represent the migrated clones without clone mutation, with different tight mutation and loose mutation patterns. The results shown in Table 5.4 come from six similarities for G<1,2,3> category. Since the migration pattern P1 containing no clone mutation is the control group for computing OR, the OR for P1 is 1.

Figure 5.5 shows the same results with 80% similarity in Table 5.4. In Figure 5.5,

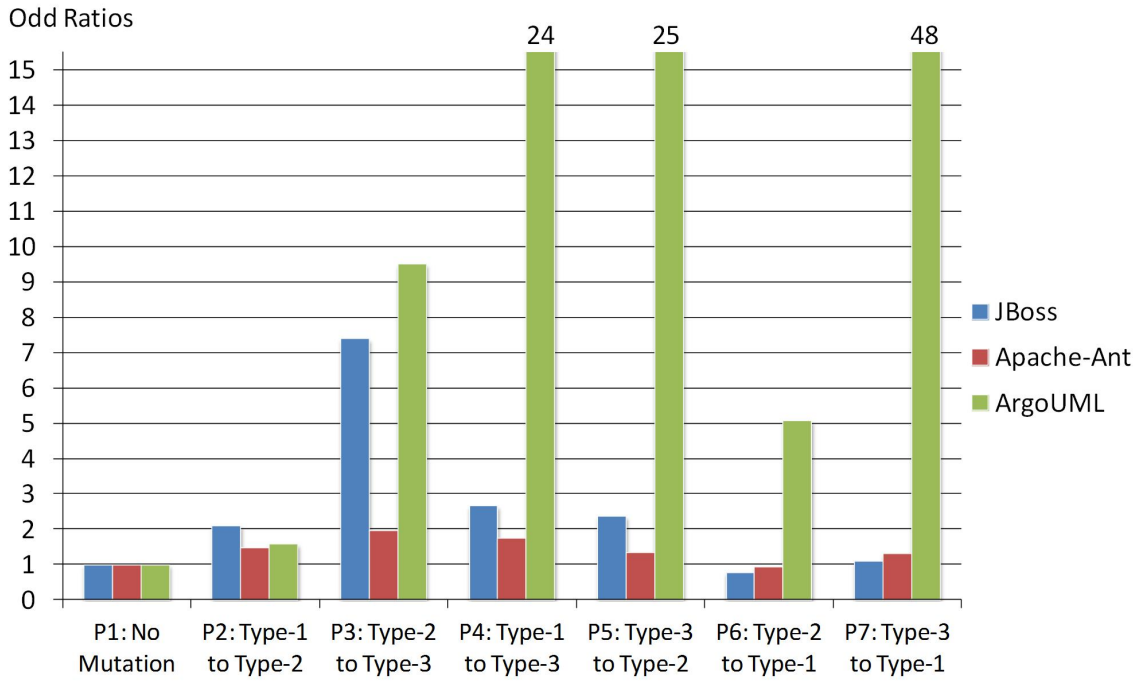


Figure 5.5: Result of Clone Migration Patterns along with Clone Mutation with 80% Similarity

we can find that loose mutation patterns (P2, P3, P4) are fault-prone than tight mutation (P5, P6, P7) in clone migration except for ArgoUML. Detailed discussions are made by following paragraphs.

Since all values for P-Value in Table 5.4 are larger than the threshold of 0.05, all results are statistically significant. Therefore, we can reject H_2 based on the results.

We find that except for the result of 90% similarity in JBoss, 90% and 95% similarity in Apache-Ant, the one with highest odd ratio in loose mutation migration patterns (*i.e.,* P2, P3, or P4) has higher OR than all non-mutation clone migration and tight migration patterns (*i.e.,* P5, P6, or P7) for JBoss and Apache-Ant. Thus for JBoss and Apache-Ant, there is always at least one of three loose migration

patterns (*i.e.,* P2, P3, or P4) has higher risk for faults than other patterns. While for ArgoUML, except the results for 95% similarity, cloned codes of at least one in three tight migration patterns (*i.e.,* P5, P6, or P7) has higher fault-proneness than other patterns.

Moreover, in Table 5.4, the migrated clones with clone mutation between Type-2 and Type-3 (*i.e.,* Type-2 to Type-3 (P3) and Type-3 to Type-2 (P5)) are more fault-prone for about more than half the similarity thresholds in both JBoss and Apache-Ant systems. Clone migration containing clone mutation from Type-3 to Type-1 (P7) are more fault-prone in Apache-Ant. While for ArgoUML, the clone migration pattern with changing clone type from Type-3 to Type-1 (P7) are the most fault-prone pattern for five of six similarity thresholds.

Overall, we conclude that the clone migration associated with clone mutation have higher risk for faults. Specifically, loose migration patterns are more fault-prone in two of the three systems and clone mutation involved with Type-3 clone type result in higher risk for faults for clone migration.

5.6 RQ5.3: Does the time interval between the migrating change and the last change of the cloned code before migration affect the fault-proneness?

Motivation.

Clone migration can be carried out immediately after the last change made to the cloned code segment or in a longer time period after the last change. A longer time interval between the migrating change (clone migration) and the last change of the cloned code may lead to a higher chance to introduce defects when making

Table 5.4: Odds Ratio Result of Clone Migration Patterns along with Clone Mutation

System	Type-3 Similarity	P1: No Mutation	P2: Type-1 to Type-2	P3: Type-2 to Type-3	P4: Type-1 to Type-3	P5: Type-3 to Type-2	P6: Type-2 to Type-1	P7: Type-3 to Type-1	P-Value
JBoss	70%	1	**2.67**	1.19	0.71	0.93	**1.38**	0.69	<0.05
	75%	1	**2.09**	1.07	0.66	0.82	**1.08**	0.67	<0.05
	80%	1	2.11	**7.39**	2.66	**2.37**	0.78	1.1	<0.05
	85%	1	1.87	**9.36**	3.06	**2.72**	0.69	1.36	<0.05
	90%	1	1.5	**7.5**	4.14	**10.91**	0.55	4.64	<0.05
	95%	1	1.12	0	**4.73**	-	0.44	-	<0.05
Apache -Ant	70%	1	1.66	**1.99**	1.53	**1.81**	1.2	1.62	<0.05
	75%	1	1.37	**1.71**	1.63	1.08	0.92	**1.38**	<0.05
	80%	1	1.48	**1.97**	1.74	**1.35**	0.94	1.32	<0.05
	85%	1	1.37	**2.16**	1.94	**1.5**	0.89	1.35	<0.05
	90%	1	**1.77**	1.12	1.42	1.19	1.22	**1.86**	<0.05
	95%	1	1.29	1.93	**2.77**	1.18	0.91	**2.97**	<0.05
Argo -UML	70%	1	0.35	3.11	**5.43**	5.2	1.12	**8.43**	<0.05
	75%	1	1.1	9.2	**18.01**	13.55	3.53	**25.89**	<0.05
	80%	1	1.58	9.5	**24.2**	24.84	5.07	**48.49**	<0.05
	85%	1	1.77	16.51	**37.76**	31.12	5.69	**83.4**	<0.05
	90%	1	2.19	14.41	**27.97**	37.83	7.02	**122.48**	<0.05
	95%	1	2.62	0	**122.29**	0	8.42	**88.05**	<0.05

clone migration. We examine this question to help developers learn about the risk of introducing defects when changing or migrating clones by considering the length of time interval after the last code change before clone migration.

Approach.

To measure the different lengths of interval time between clone migration and the last change of the cloned code, we use 200 days as a unit to divide the time period into different period levels. We define the following equation to compute the period level: $\frac{N_p}{200}$, where N_p represents the number of days between clone migration and last change of the cloned code. Our choice for the unit, which is 200 days, will not affect our results since different unit values will lead to the same conclusion. Because there are different lengths of history for three systems, they are divided into different period levels. For the cloned codes in G<1,2,3> category of clone genealogies, we perform this process on all six similarity thresholds, which are 70%, 75%, 80%, 85%, 90% and 95%.

Moreover, we examine the cloned codes in other four categories of clone genealogies. We compute the numbers of fault-containing and fault-free of clone migration made in different period levels after the last change of the cloned code. We perform the Chi-square test and compute the OR to test the following hypothesis: *H_3: The length of time interval between clone migration and the last change of the cloned code will not affect the fault-proneness of the migrated clones.*

Finding.

Table 5.5 and Table 5.6 present the OR results for different period levels we defined. Level 0 represents clone migration is made during the time interval after the last change of the cloned code. While the highest period level refers to the longest

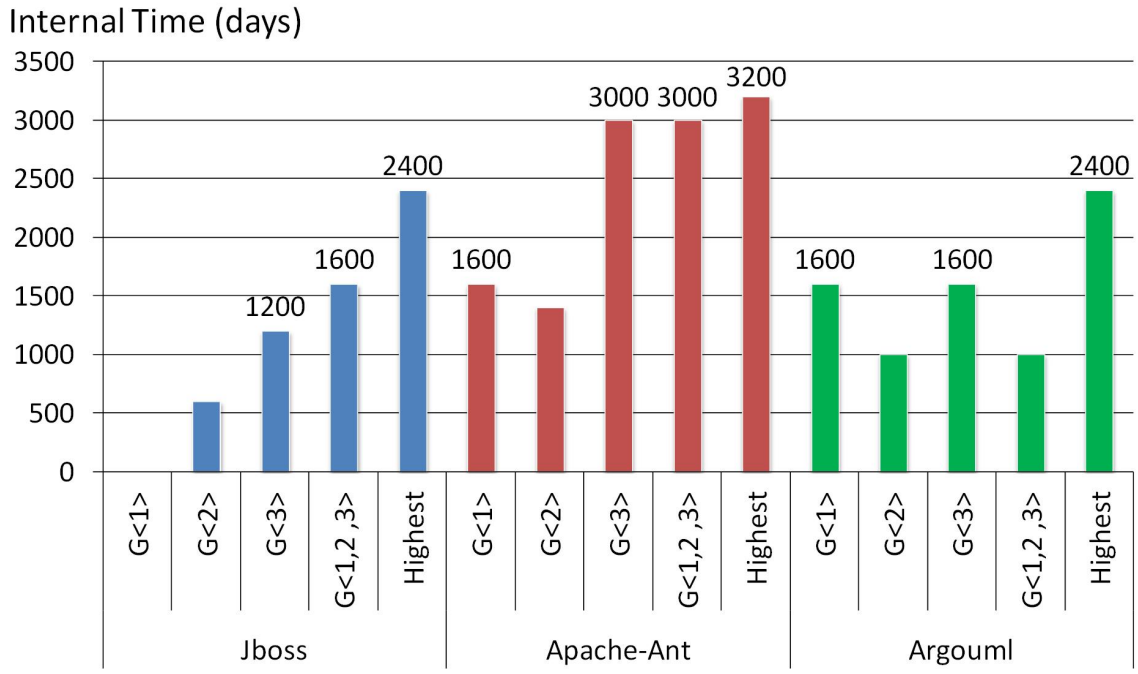


Figure 5.6: Result of Clone Migration for Different Period Levels in Four Categories of Clone Genealogies with 80% Similarity

time interval after last change of the cloned code. We use the level 0 as the control group and other period levels as experimental groups.

Figure 5.6 shows the same results with 80% similarity in Table 5.6. For each system in Figure 5.6, we find that the most fault-proneness time interval levels (2 or 3 of 4 categories) are longer than the half of the length for the highest time interval levels. Following are detailed discussions.

The results for cloned code in clone genealogies containing all three types of clones are presented in Table 5.5. In JBoss system, the highest ORs for all similarity thresholds are presented in period level 8 (from 1601 to 1800 days), which is the ninth period level of thirteen period levels. For Apache-Ant system, the most

fault-prone period levels are 9 (from 1801 to 2000 days), 14 (from 2801 to 3000 days), 15 (from 3001 to 3200 days), and 16 (3201 to 3400 days) out of 17 period levels for different similarity thresholds. While in ArgoUML, the period level 5 (from 1001 to 1200 days) out of 12 period levels is most fault-prone for five of six similarity thresholds. The only exception is period level 7 (from 1401 to 1600 days) has the most fault-prone result in 70% similarity. Overall, for cloned codes in three single types of clone genealogies, we can conclude that the clone migration made after the last change of the cloned code with a time interval for more than half of the history for software systems has the higher risk for faults.

Moreover, Table 5.6 shows the results for four different categories of clone genealogies defined in Section 4.4. There is no valid OR for cloned codes in Type-1 clone genealogy of JBoss. This is due to the small number of clone migration. In these results, we find that in Type-3 clone genealogies (G<3>) and clone genealogies with all three clone types (G<1,2,3>), migrated clones made in a longer interval time than the middle period level are more fault-prone for three systems, except in G<1,2,3> of ArgoUML. For clone migration in Type-1 clone genealogies (G<1>), the higher period level (*i.e.,* 8 out in 17) has the highest risk for faults in two of three systems.

In general, we conclude that the clone migration made in a longer time interval after the last change of the cloned code than the time interval in middle period level, are more fault-prone in JBoss and Apache-Ant. While clone migration in Type-1 and Type-3 clone genealogies of ArgoUML have higher risk for faults when clone migration is made in a longer time interval after the last change of the cloned code. In the clone genealogies involved with Type-3 clones of all three systems, the clone

migration occurring in a longer time interval since the last change are more fault-prone than in a shorter time interval, compared with the length of time interval for middle period levels.

All the P-value results shown in Table 5.5 and Table 5.6 are larger than 0.05, therefore the results are statistically significant and we can reject H_3. Overall, we conclude that the clone migration made after a longer time interval since the last change of the cloned code has higher risk for faults.

5.7 Summary

At last, we identify the effect of LOC (lines of code) on our results to prove our conclusions. We compare the average LOC of all migrated clones of the clone genealogies in each of the three systems. Our results are the average LOC for faulty and non-faulty migrated clones are respectively, 32 and 33 in JBoss, 48 and 43 in Apache-Ant, and 25 and 24 in ArgoUML. Therefore, there is no obvious difference exists for the average LOC between faulty and non-faulty migrated clones. We believe that the LOC of the migrated clone does not affect our conclusions.

After we identifying the frequency for clone migration and clone mutation, in this study, we continue to identify different clone migration patterns to investigate the influence from clone migration and clone mutation on the fault-proneness of clones, clone groups, and clone genealogies.

Firstly, we identify the impact on fault-proneness from clone migration in clones and clone groups in four categories of clone genealogies. Examining three clone contexts, *i.e.,* clones, clone groups, and clone genealogies, we find that clone migration will increase the fault-proneness of the clones in all three contexts, which are clones,

Table 5.5: Odds Ratio Result of Clone Migration for Different Period Levels in Type-3 Clone Genealogies

Days Ranges		0-200	201-400	401-600	601-800	801-1000	1001-1200	1201-1400	1401-1600	1601-1800	1801-2000	2001-2200	2201-2400	2401-2600	2601-2800	2801-3000	3001-3200	3201-3400	P-Value
System	Similarity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
Jboss	70%	1	0.95	0.93	0.57	0.89	1.87	0.9	0	**107.13**	1.28	0	-	1.28					<0.05
	75%	1	1.06	1.44	1.8	0.92	0.69	1.54	0	**116.08**	1.66	0	-	2.76					<0.05
	80%	1	0.64	1.22	1.46	0.44	0	0.92	0	**137.13**	0.65	-	-	0					<0.05
	85%	1	0.59	1.18	1.83	0.42	0	0.91	-	**118.87**	0	-	-	-					<0.05
	90%	1	0.66	1.02	1.33	0.56	0	0.73	-	**90.55**	0	-	-	-					<0.05
	95%	1	0.7	0.73	1.15	0.18	0	0	-	**1.74**	0	-	-	-					<0.05
Apache-Ant	70%	1	1.09	1.32	0.77	0.39	0.69	1.11	3.03	2.15	1.96	0.71	5.59	0.83	1	20.89	5.99	**39.52**	<0.05
	75%	1	1	0.84	0.89	0.33	0.62	0.83	0.67	0.71	2.84	0.18	6.81	6.82	0.29	23.07	**86.75**	28.92	<0.05
	80%	1	0.85	1.11	0.51	0.25	0.42	0.85	0.56	0.4	161.89	0.26	2.98	4.6	0.87	52.22	**167.11**	0	<0.05
	85%	1	0.93	1.22	0.6	0.22	0.48	0.29	0.58	0.42	143.61	0.3	4.2	0.17	-	**51.29**	-	0	<0.05
	90%	1	1.54	0.23	0.13	0.06	0.27	0.24	0.12	0.14	**147.84**	0.52	0	-					<0.05
	95%	1	1.8	0.37	0.25	0.11	0.32	0.3	0.22	0.13	**121.53**	0.4	0	-					<0.05
ArgoUML	70%	1	8.65	2.1	1.02	2.96	33.11	57.5	**186.49**	21.97	11.11	8.37	0.85	0					<0.05
	75%	1	11.46	1.64	1.83	7.03	**241.88**	68.63	145.04	69.35	37.68	9.23	0	0					<0.05
	80%	1	11.13	1.5	1.24	5.48	**374.38**	80.69	204.92	67.73	57.2	2.19	0	0					<0.05
	85%	1	11.69	1.64	1.3	6.71	**614.82**	118.05	281.71	67.92	118.25	2.86	0	0					<0.05
	90%	1	11.72	1.2	1.3	7.97	**777.94**	147.66	327.93	86.44	149.18	3.68	0	0					<0.05
	95%	1	13.3	1.27	1.06	7.52	**964.93**	161.65	322.71	36.34	168.85	4.5	0	0					<0.05

Table 5.6: Odds Ratio Result of Clone Migration for Different Period Levels in Four Categories of Clone Genealogies

Days Ranges			0-200	201-400	401-600	601-800	801-1000	1001-1200	1201-1400	1401-1600	1601-1800	1801-2000	2001-2200	2201-2400	2401-2600	2601-2800	2801-3000	3001-3200	3201-3400	P-Value
System	Category	Similarity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
JBoss	G<1>	100%	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	G<2>	100%	1	1.48	0.48	**3.41**	0	-	0	-	1.56	-	-	-	-	-	-	-	-	<0.05
	G<3>	80%	1	0.51	1.02	1.81	1.2	0	**3.61**	0	-	1.81	-	-	0	-	-	-	-	<0.05
	G<1,2,3>	80%	1	0.64	1.22	1.46	0.44	0	0.92	0	**137.13**	0.65	-	-	0	-	-	-	-	<0.05
Apache-Ant	G<1>	100%	1	0.96	0.57	0	0	1.6	2.86	0	**6.67**	-	3.33	-	-	-	-	-	-	<0.05
	G<2>	100%	1	2.52	0.4	0.17	0	0	0	**18.75**	0	-	3.13	0	-	-	-	-	-	<0.05
	G<3>	80%	1	0.44	1.15	0.57	0.21	0.34	0.57	0.71	0.29	71.54	0.04	2.34	4.87	1.04	22.59	**203.31**	0	<0.05
	G<1,2,3>	80%	1	0.85	1.11	0.51	0.25	0.42	0.85	0.56	0.4	161.89	0.26	2.98	4.6	0.87	52.22	**167.11**	0	<0.05
ArgoUML	G<1>	100%	1	0.21	0.08	0	0	0.76	1.21	0	**1.52**	1.18	0	0	-	-	-	-	-	<0.05
	G<2>	100%	1	12.86	1.58	0.96	3.54	**3117.74**	249.42	683.5	88.97	383.6	11.27	0	-	-	-	-	-	<0.05
	G<3>	80%	1	1.31	0.37	0.34	0.36	4.12	0.72	4.68	**9.73**	0.15	0.1	0	0	-	-	-	-	<0.05
	G<1,2,3>	80%	1	11.13	1.5	1.24	5.48	**374.38**	80.69	204.92	67.73	57.2	2.19	0	0	-	-	-	-	<0.05

clone groups, and clone genealogies. More migrated clones increase the risk for faults of clone groups.

Moreover, we investigate the impact of the clone migration associated with clone mutation on the risk for faults of cloned codes in subsequent revisions. We define seven clone migration patterns with or without clone mutation and demonstrate that loose migration is more fault-prone than both tight migration and the clone migration without clone mutation in two of three subject systems. When the clone migration associated with clone mutation involved with Type-3 clones (*i.e.,* clone mutation between Type-1 and Type-3 or between Type-2 and Type-3), the migrated clones are more risky in all three subject systems.

Finally, we examine the impact of the length of time interval between clone migration and the last change of the cloned code on the risk for faults of migrated clones. Comparing with the length of time interval of the highest period level, a longer time interval between clone migration and the last change of the cloned code has a higher risk for faults in clone genealogies containing Type-1 or Type-3 clones.

Chapter 6

Conclusion

In this Chapter, we describe the contributions of this thesis and our recommendations for developers based on our conclusions. We also discuss the threats to validity for our empirical studies and explore the future work.

6.1 Contribution

This thesis makes the following contributions:

- *Building clone genealogies based on clone groups.*

  We build clone genealogies based on clone groups. Since the evolution of the clone pairs only capture two clones, they cannot measure the evolution of multiple code segments cloned with each other. Tracking clone evolution by clone groups can identify more clone genealogies. Whenever two clone groups share a common cloned code, there will be a new evolution for the first clone group. Using clone genealogies based on clone groups, we can identify all clone migration and clone mutation since all clones are captured in the clone genealogies of clone groups. Therefore, we consider it is very essential to build clone genealogies for

clone groups.

To build the clone genealogy of a clone group, we look for each two clone groups that share a common cloned code. Then we connect these clone groups by the time sequence. Since a clone group can have many connections with other clone groups, there are many possibilities for a clone group to evolve. Building the clone genealogies is the foundation of our empirical studies.

- *Identifying clone migration.*

  Clone migration is defined as a movement of cloned code to a different location of software structure for clones in clone genealogies. It is important for us to identify the impact of the movements of clones on the fault-proneness of migrated clones. Therefore we can provide suggestions to developers for those movement activities.

  We identify clone migration through looking for clones with changed location in software system without changing filenames. We locate each clone migration in the changes of clones. A migrated clone is the clone moved into a new location between two revisions or two clone groups. Then we can define different patterns for clone migration to investigate their impacts on fault-proneness of migrated clones.

- *Identifying clone mutation.*

  Since we use NiCad to detect near-miss (*i.e.,* Type-2 and Type-3) clones, we obtain the clone type for each clone group. Thus we can identify the change of clone types in a change between two revisions. When the changes of the clone types result from the modifications on code statements, there will be a

variation of the similarity between clones. Thus the clone mutation with changes like adding, deleting and modifying statement is valuable for us to study their risk for faults. We identify the clone migration that occurs at the same time with clone mutation. This helps us study the impact of the clone migration associated with clone mutation on the risk for faults of migrated clones.

6.2 Recommendation

We conduct two empirical studies about clone migration and clone mutation in clone genealogies. We prove the existences of clone mutation and clone migration in clone genealogies and conclude the effects on fault-proneness of clones from different patterns or categories of those two phenomenons. This thesis provides following conclusions and recommendations:

- *Studies on Clone Mutation.*

  Our results show that clone genealogies predominated by Type-2 or Type-3 clone groups present higher risk for faults. Moreover, we find that the clone mutation between Type-1 and Type-2, between Type-1 and Type-3 have higher risk for faults in clone genealogies for clone groups.

  For developers, we suggest them pay more attention to modifications that lead to the change of clones types. Especially when they make changes on identifiers and statement in source code, they need to be cautious. Since there more differences between the clones with higher clone types and lower similarities, developers should consider all the differences in source code when co-change these clones.

Moreover, we suggest developers do not make too many changes on a duplicated code segment, which will be more likely to become a near-miss clone. Too many changes for copied code makes the further evolution and maintenance more difficult.

- *Studies on Clone Migration.*

  We conclude that overall clone migration is fault-prone in clone genealogies and migrated clones have higher risk for faults than non-migrated clones. The clone migration associated with the loose mutation containing the decrease of similarity between clones has higher risk for faults. Moreover, we find that the clone migration made in a longer time interval after the last change of migrated code is more fault-prone.

  Based on our findings, we suggest developers be cautious when changing the location in software structures of near-miss (*i.e.,* Type-2 and Type-3) clone groups in clone genealogies. When a cloned code is moved to a new location, too many associated changes make the code segment become more fault-prone because these further evolution is difficult. Developers have to work together to change all clones distributed in different locations of the software system. Moreover, more attentions should be made from developers to the clone migration with the changes of clone types in a revision of the system, especially the changes from making the clone types mutated from lower to higher ones (*i.e.,* Type-1 to Type-3).

6.3 Threats to Validity

In this section, we discuss the threats to validity for the two studies of this thesis. We follow the common guidelines [40] for our discussion.

Construct validity threats are about the relation between theory and observation. Our study results are basd on the results from two tools, *i.e.,* NiCad and J-Rex. In this thesis, the first construct validity threat is due to the usage of the tool to detect clones. We use NiCad to detect clones since it can detect not only exact (Type-1) but also near-miss (Type-2 and Type-3) clones [6]. The change of the tool to detect clones in our empirical studies will not affect the final conclusion if we can remove the false positive values from the different results of other tools to detect clones.

Another *construct validity* threat is about the usage of J-Rex, which is the tool to identify change and fault fixes from software repositories. J-Rex implements the algorithm in Hassan *et al.* [41] and Mockus *et al.* [42]. A experiment has been done by Barbour [30] for recognizing fault fixes in commit messages using the J-Rex. They conclude that the precision is larger than 0.85 after the manual evaluation on the commit messages from the results of J-Rex for ArgoUML.

The last *construct validity* threat is when we identify clone migration, we do not consider the clones in the file that has been changed on its file name as the migrated clones. While the file name can be changed when the file is moved. This is the limitation for observations in our studies since it is difficult to track the change of the filenames.

Threats to *internal validity* are not applied in the two studies in this thesis. We conduct two exploratory studies [40]. We cannot claim causation, but we categorize the clone genealogies using different clone migration and clone mutation patterns. We

report findings from the results, where we compare the fault-proneness of different patterns of clone genealogies.

Reliability validity threats concern the replication for our two empirical studies. All details about empirical studies can be provided and the software repositories of three subject systems in our empirical studies are available in public for other researchers.

Conclusion validity threats are about if the treatment affects the result. We do not make assumption, we use non-parametric tests to test the assumptions for the statistical test. Since we use Chi-Square test and 0.05 threshold to determine if there are non-random associations among different variables, there is a possibility that one out of 20 Chi-Square test computation results can be wrong. This means one of 20 groups may not have a significant conclusion. We take 171 Chi-Square test computations in our two empirical studies.

External validity threats are about the process that we generalize the conclusion. All three subject systems used in both studies in this thesis are large-scale open source software systems with plug-in architectures. They are all written in JAVA but within different domains and with different sizes (line of code). ARGOUML has some different results compared with other two subject systems. This may because the size of the clone is smaller in ARGOUML. Since some of our conclusions are system dependent, we need to analyze more systems to prove these conclusions.

6.4 Future Work

In the future, we aim to expand the studies in this thesis by performing our empirical study on more subject systems written in other programming languages. We want to apply more tools to detect clones that use different techniques to evaluate our

conclusion and compare the results of different tools. Moreover, more patterns and approaches would be used to classify clone mutation, clone migration, and clone genealogies. We are going to identify the risk for faults for different patterns for clones, clone groups, and clone genealogies.

Bibliography

[1] S. Xie, F. Khomh, and Y. Zou, "An empirical study of the fault-proneness of clone mutation and clone migration," in *Proc. 10th Working Conference on Mining Software Repositories (MSR)*, (San Francisco, California, USA), May 2013.

[2] CVS, "http://cvs.nongnu.org," 2013.

[3] SVN, "http://subversion.apache.org," 2013.

[4] GIT, "http://git-scm.com," 2013.

[5] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," *SCHOOL OF COMPUTING TR 2007-541, QUEENS UNIVERSITY*, vol. 115, 2007.

[6] C. K. Roy and J. R. Cordy, "Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *Proc. 16th IEEE International Conference on Program Comprehension (ICPC)*, pp. 172–181, 2008.

[7] T. Kamiya, S. Kusumoto, and K. Inoue, "Ccfinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, pp. 654–670, 2002.

[8] S. Ducasse, O. Nierstrasz, and M. Rieger, "On the effectiveness of clone detection by string matching: Research articles," *J. Softw. Maint. Evol.*, vol. 18, pp. 37–58, Jan. 2006.

[9] J. H. Johnson, "Substring matching for clone detection and change tracking," in *Proc. International Conference on Software Maintenance (ICSM)*, pp. 120–126, 1994.

[10] A. Marcus and J. I. Maletic, "Identification of high-level concept clones in source code," in *Proc. 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 107–114, 2001.

[11] Simian, "http://www.harukizaemon.com/simian," 2013.

[12] W. Evans, C. Fraser, and F. Ma, "Clone detection via structural abstraction," in *Rroc. 14th Working Conference on Reverse Engineering (WCRE)*, pp. 150–159, 2007.

[13] V. Wahler, D. Seipel, J. W. von Gudenberg, and G. Fischer, "Clone detection in source code by frequent itemset techniques," in *Proc. 4th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)*, pp. 128–135, 2004.

[14] W. Yang, "Identifying syntactic differences between two programs," *Software - Practice and Experience*, vol. 21, pp. 739–755, 1991.

[15] K. Kontogiannis, R. de Mori, E. Merlo, M. Galler, and M. Bernstein, "Pattern matching for clone and concept detection," *Automated Software Engineering*, vol. 3, pp. 77–108, 1996.

[16] F. Lanubile and T. Mallardo, "Finding function clones in web applications," in *Proc. 7th European Conference on Software Maintenance and Reengineering (CSMR)*, p. 379, 2003.

[17] G. A. D. Lucca, M. D. Penta, and A. R. Fasolino, "An approach to identify duplicated web pages," in *Proc. 26th Annual International Computer Software and Applications Conference (COMPSAC)*, pp. 481–486, 2002.

[18] J. Mayrand, C. Leblanc, and E. Merlo, "Experiment on the automatic detection of function clones in a software system using metrics," in *Proc. International Conference on Software Maintenance (ICSM)*, p. 244, 1996.

[19] I. Keivanloo, C. K., and J. Rilling, "Sebyte: A semantic clone detection tool for intermediate languages," in *Proc. 20th IEEE International Conference on Program Comprehension (ICPC)*, pp. 247–249, 2012.

[20] L. Aversano, L. Cerulo, and M. Di Penta, "How clones are maintained: An empirical study," in *Rroc. 11th European Conference on Software Maintenance and Reengineering (CSMR)*, pp. 81 –90, 2007.

[21] H. C. G. Reto Geiger, Beat Fluri and M. Pinzger, "Relation of code clones and change couplings," in *Proc. 9th International Conference of Funtamental Approaches to Software Engineering (FASE), number 3922 in LNCS*, pp. 411–425, Springer, 2006.

[22] A. Lozano, M. Wermelinger, and B. Nuseibeh, "Evaluating the harmfulness of cloning: A change based experiment," in *Rroc. International Workshop on Mining Software Repositories (MSR)*, p. 18, 2007.

[23] J. Krinke, "Is cloned code more stable than non-cloned code?," in *Proc. IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)*, (Los Alamitos, CA, USA), pp. 57–66, IEEE Computer Society, 2008.

[24] C. Kapser and M. W. Godfrey, ""cloning considered harmful" considered harmful," in *Proc. 13th Working Conference on Reverse Engineering (WCRE)*, pp. 19–28, 2006.

[25] M. Kim, L. Bergman, T. Lau, and D. Notkin, "An ethnographic study of copy and paste programming practices in oopl," in *Proc. International Symposium on Empirical Software Engineering (ISESE)*, pp. 83–92, 2004.

[26] W. Shang, Z. M. Jiang, B. Adams, and A. Hassan, "Mapreduce as a general framework to support research in mining software repositories," in *Proc. 6th IEEE International Working Conference on Mining Software Repositories (MSR)*, pp. 21 –30, May 2009.

[27] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, "An empirical study of code clone genealogies," in *Proc. 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC/FSE-13, (New York, NY, USA), pp. 187–196, ACM, 2005.

[28] S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta, "An empirical study on the maintenance of source code clones," *Empirical Software Engineering*, vol. 15, pp. 1–34, 2010.

[29] L. J. Barbour, F. Khomh, and Y. Zou, "Late propagation in software clones," in *Proc. 27th IEEE International Conference on Software Maintenance (ICSM)*, pp. 273 – 282, sept. 2011.

[30] L. J. Barbour, "Empirical studies of code clone genealogies," Master's thesis, Depart of Electrical and Computer Engineering, Queen's University, Kingston, Ontraio, Canada, 2012.

[31] E. Duala-Ekoko and M. P. Robillard, "Tracking code clones in evolving software," in *Proc. 1st Annual India Software Engineering Conference (ISEC)*, pp. 19–20, 2008.

[32] N. Göde, "Evolution of type-1 clones," in *Proc. 9th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pp. 77–86, 2009.

[33] N. Göde and R. Koschke, "Studying clone evolution using incremental clone detection," *Journal of Software: Evolution and Process*, vol. 25, no. 2, pp. 165–192, 2013.

[34] A. Alali, B. Bartman, C. D. Newman, and J. I. Maletic, "A preliminary investigation of using age and distance measures in the detection of evolutionary couplings," in *Proc. 10th Working Conference on Mining Software Repositories*, MSR '13, (Piscataway, NJ, USA), pp. 169–172, IEEE Press, 2013.

[35] R. K. Saha, C. K. Roy, K. A. Schneider, and D. E. Perry, "Understanding the evolution of type-3 clones: an exploratory study," in *Proc. 10th International Workshop on Mining Software Repositories (MSR)*, pp. 139–148, IEEE Press, 2013.

[36] A. Mockus and L. Votta, "Identifying reasons for software changes using historic databases," in *Proc. International Conference on Software Maintenance (ICSM)*, 2000.

[37] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proc. 31st International Conference on Software Engineering (ICSE)*, (Washington, DC, USA), pp. 78–88, IEEE Computer Society, 2009.

[38] M. F. Zibran, R. K. Saha, M. Asaduzzaman, and C. K. Roy, "Analyzing and forecasting near-miss clones in evolving software: An empirical study," in *Proc. 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 295–304, 2011.

[39] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 4 ed., 2007.

[40] R. K. Yin, *Case Study Research: Design and Methods - Third Edition*. London: SAGE Publications, 2002.

[41] A. E. Hassan and R. C. Holt, "Studying the evolution of software systems using evolutionary code extractors," in *Proc. 7th International Workshop on Principles of Software Evolution (IWPSE)*, pp. 76–81, 2004.

[42] A. Mockus and L. G. Votta, "Identifying reasons for software changes using historic databases," in *Proc. International Conference on Software Maintenance (ICSM)*, pp. 120–130, 2000.