Is Cloned Code older than Non-Cloned Code?

Jens Krinke
University College London
Centre for Research on Evolution, Search and Testing (CREST)
j.krinke@ucl.ac.uk

ABSTRACT

It is still a debated question whether cloned code causes increased maintenance efforts. If cloned code is more stable than non-cloned code, i.e. it is changed less often, it will require less maintenance efforts. The more stable cloned code is, the longer it will not have been changed, so the stability can be estimated through the code's age. This paper presents a study on the average age of cloned code. For three large open source systems, the age of every line of source code is computed as the date of the last change in that line. In addition, every line is categorized whether it belongs to cloned code as detected by a clone detector. The study shows that on average, cloned code is older than non-cloned code. Moreover, if a file has cloned code, the average age of the cloned code of the file is lower than the average age of the non-cloned code in the same file. The results support the previous findings that cloned code is more stable than non-cloned code.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Software configuration management*; D.2.13 [**Software Engineering**]: Reusable Software—*Reusable libraries*

General Terms

Algorithms

Keywords

Clone detection, mining software archives, software evolution

1. INTRODUCTION

The duplication of code is common practice to make software development faster, to enable "experimental" development without impacting the original code, or to enable independent evolution [3]. Since these practices involve both duplication and modification, they are collectively called *code cloning* and the duplicated code is called a *code clone*. During the software development cycle, code cloning is easy and inexpensive (in both effort and money). However, this cloning practice can complicate software maintenance and it has been suggested that too much cloned code is a risk, albeit the practice itself is not generally considered harmful [11].

An important question for software maintenance is if cloned code is more stable than non-cloned code during the evolution of a system, i.e. if non-cloned code is changed more often than cloned code. If cloned code is generally less stable than non-cloned code, it can be assumed that cloned code requires more attention and is indeed more expensive to maintain. If cloned code is generally more stable than non-cloned code, its maintenance costs will be lower.

To answer the above question, there have been some empirical studies [6, 8, 14, 17, 18] that explicitly looked at the stability of cloned code vs. non-cloned code. Most of them extracted the changes to a system from a version repository and mapped the changes on the clones as reported by a clone detector. The studies usually measured the amount of changes or the frequency of changes to answer research questions about the stability of cloned code. The extraction and mapping of the changes is a considerable effort.

Another approach to answer research questions on the stability of cloned code is to look at the *age* of the cloned and the non-cloned code—the more stable cloned code is, the longer it will not have been changed. The age of source code can be estimated by the date of the last change applied to it. Most current version control systems can track changes to a file line-by-line to show for each line the version when the line was last changed. *CVS* has an "annotate" command and *subversion* names the command "blame" because it shows the version and the author ('to be blamed'). These commands give crude information about the origins of the code based on when it was last changed and who made that change. However, for a study on the age of source code, this crude information is more than sufficient.

The rest of the paper will present an initial study on three large open source systems with a long history of evolution. The study compares the average age (as the date of the last change) of cloned code to the average age of non-cloned code. For the three Java systems studied, the following results were found:

- Cloned code is usually older than non-cloned code.
- Cloned code in a file is usually older than the non-cloned code in the same file.

Both results suggest that cloned code is more stable than non-cloned code. Although the setup of the study is simple and the age information from version control systems is crude, the results appear to be stronger than previous studies on the stability of cloned code. In particular, it will show that cloned code is on average 85 days older

IWSC 2011 May 23, 2011, Waikiki, Hawaii, USA

than non-cloned code and that for 81% of files with cloned code, the cloned code is on average older than the non-cloned code in the same file.

The next section presents the setup of the empirical study. The study itself and its results is presented in Section 3. After Section 4 discusses related work, the last section will conclude.

2. STUDY SETUP

This section will present the framework in which code clones and the age of source code are defined. It will also show how the age can be measured and compared between cloned and non-cloned code. It will also state the research questions to be answered by the study.

2.1 Code Clones

Code clones are usually described as source code ranges (or fragments) that are identical or very similar. They are grouped into *clone groups* (sometime called *clone classes*) which are sets of identical or very similar code clones. A code clone $c = (s, l, f)$ is the source code range starting at line s with the following l lines of code in file f, thus the last line of the code clone is $s + l - 1$. A clone group $G = \{c_1, \ldots, c_n\}$ is a set of n code clones $c_1, \ldots, c_n$, where each of the code clones is a clone of the others. Most of the available tools for code clone detection generate a list of such clone groups. For simplicity, we assume the absence of gapped clones, i.e. there are no clones that consist of multiple non-contiguous code fragments.

A source code line l' in a file f' is called to be *cloned*, if the line is part of some code clone, i.e. if a code clone $c = (s, l, f')$ exists such that $s \leq l' < s + l$. Otherwise the line is called to be *non-cloned*.

2.2 Age Information

Most current version control systems can track changes to a file line-by-line to show for each line the version when the line was last changed. *CVS* has an "annotate" command and *subversion* names the command "blame" because it shows the version and the author ('to be blamed'). These commands give crude information about the origins of the code based on when it was last changed and who made that change.

Usually, the blame command retrieves the version information for the current version or for a specific version for one file or a list of files. In the following, the existence of an age function $\mathrm{A}(f, l)$, which retrieves the age (date of the last change) of source code line l from source file f of the current version of the program is assumed.

2.3 Measuring and Comparing the Age

For the purpose of the study, a software system $S = \{f_1, \ldots, f_n\}$ consists of a set of n source code files $f_i, 1 \leq i \leq n$. Two functions exist that return the cloned ($C(f_i)$) and non-cloned ($N(f_i)$) lines of a source code file $f_i \in S$.

The average age $A_C(S)$ of cloned code of a system S is then the average of all $\mathrm{A}(f_i, l_j)$ for all $f_i \in S$ and $l_j \in C(f_i)$. The average age $A_N(S)$ of non-cloned code of a system S is the average of all $\mathrm{A}(f_i, l_j)$ for all $f_i \in S$ and $l_j \in N(f_i)$.

Systems evolve differently in their components, so measuring the overall average age may give too unspecific results. Thus, the average ages for single files can be computed, too. However, this only make sense for files that actually contain cloned code. So for files f with $C(f) \neq \emptyset$, the average age $A_C(f)$ of cloned code is the average of all $\mathrm{A}(f, l_j)$ for all $l_j \in C(f)$. The average age $A_N(f)$ of non-cloned code is the average of all $\mathrm{A}(f, l_j)$ for all $l_j \in N(f)$.

2.4 Research Questions

Based on the above framework, it is now possible to state the two important research questions and define how they can be answered by measuring the age of source code lines.

RQ1: *Is cloned code usually older or newer than non-cloned code?*

For a given system S, the question can be answered by computing and comparing the average ages $A_C(S)$ and $A_N(S)$.

RQ2: *Is the cloned code in a file usually older or newer than the non-cloned code in the same file?*

For a given file f, the question can be answered by computing and comparing the average ages $A_C(f)$ and $A_N(f)$.

These two research questions are similar to the research questions of empirical studies on the stability of cloned code. If the questions can be answered clearly with that cloned code is usually older than non-cloned code, then it will strongly support the previous findings that cloned code is (often) more stable than non-cloned code.

3. EXPERIMENT

To answer the above research question, an initial study has been performed on three open source systems with a sufficiently large evolution history. The setup and the results of the study will be presented in the following, followed by a short discussion on threats to validity.

3.1 Study Setup

For the study the version histories of three open source systems have been retrieved. All three systems have to have a sufficiently long development history of multiple years. To enable a comparison of the results to previous studies on clone evolution, the systems should have been used in previous empirical studies. The three selected systems are:

1. ArgoUML[1] is a UML modeling tool that includes support for standard UML diagrams. It is written in Java and its version archive is available via subversion at `http://argouml.tigris.org/svn/argouml/trunk`. This study used revision 18995.

   ArgoUML has been used in many previous studies on the evolution of code clones [1, 12–15]. Besides, other empirical studies have looked at ArgoUML and it can be considered a well-studied system.

2. JBoss[2] is a J2EE compliant application server written in Java. This system has been used in previous studies, too [17, 23]. The subversion archive is available at `http://anonsvn.jboss.org/repos/jbossas/trunk` and the revision 110455 has been used in this study.

3. jEdit[3] is a programmer's text editor written in Java. It has also been used in previous studies [2, 17].

   The subversion repository is available at `https://jedit.svn.sourceforge.net/svnroot/jedit/jEdit`; revision 19285 has been used.

[1] `http://argouml.tigris.org/`
[2] `http://www.jboss.org/`
[3] `http://www.jedit.org/`

System	Source Files	Source LOC	Cloned LOC	
ArgoUML	2202	398844	20554	5%
JBoss	6406	853653	63518	7%
jEdit	552	175191	5520	3%

Table 1: Analyzed systems

All three systems are large enough (>100KLOC) and cover different applications and different platforms, although they are all written in Java. The sources have been analyzed as-is and no modifications have been applied, although it is known that test code and generated source code will have an effect on the detected code clones and that differences in layout will reduce the recall of clone detectors. Future studies should identify end eliminate test code and generated code.

The sources of all three systems have been retrieved from their subversion archives on the same day. For each system the clone groups $G(v)$ have been identified by the use of the clone detection tool *Simian*[4] from RedHill Consulting Pty. Ltd., version 2.2.24. It is a text-based clone detector that detects almost identical clones. *Simian* has been instructed to identify clones with the default settings (clones must be at least six lines long).

Table 1 shows some properties of the analyzed systems: The second column shows the number of Java files, the third column contains the size of the analyzed source base (in LOC). The next two columns contain the size of cloned source code (in LOC and as an percentage of the source code). For example, JBoss is the largest system with 854 KLOC, from which 7% (66 KLOC) is cloned code.

3.2 Results

This section presents the results of the study as described in the previous section. Figure 1 shows two box plots for the (average) last change date for cloned code on the left and for non-cloned code on the right. The y-axis gives the age as the last change date. The top is 25 January 2011 and the bottom is 22 April 2000. The average last change date for cloned code is lower (earlier) than for non-cloned code—the difference is 85 days (not shown in Figure 1). This means that on average, cloned code is older than non-cloned code. This suggests that cloned code is more stable than non-cloned code, if stability is measured as the duration without a change. Interestingly, the box plots show identical (or at least similar) top and bottom quartiles—only the average and the median differs. This overall result seems to answer the first research question with *cloned code is usually older than non-cloned code*.

A very interesting result is also the average age itself, which is 1719 days—over three years. Previous studies on clone evolution estimated much lower lifetime of clones. For example, Göde [5] reported an average lifetime of 483 days and Bettenburg et al. [2] reported an average lifetime of 6.8 releases (although no time is given for the distance between releases, it is clear that 6.8 releases is much lower than 1719 days). This difference can be explained by the observation from previous studies that many clones only appear for a short time. With the setup of the presented study it is clear that the results are dominated by long living clones. Moreover, the numbers are clearly influenced by the long development history of around 10 years for the analyzed system. An analysis of systems with shorter development histories would result in lower ages.

[4]Available at `http://www.redhillconsulting.com.au/products/simian/index.html`

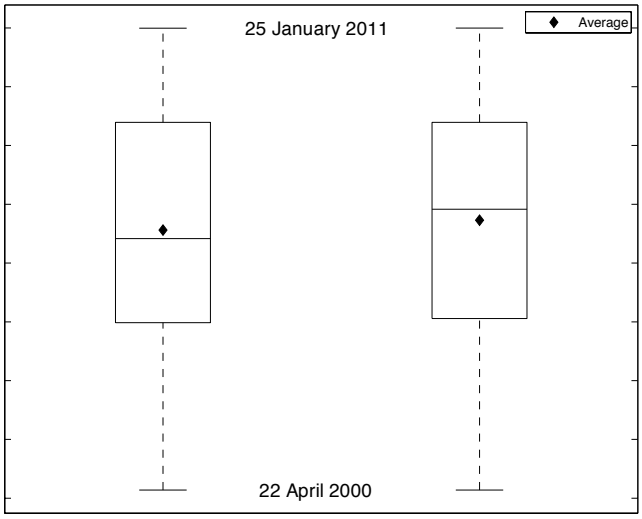

Figure 1: Last change date for cloned (left) and non-cloned code (right).

To focus on the second research question, only the Java source files that actually contain some cloned code are looked at. From the 9158 Java source files that have been analyzed, only 2828 files (31%) contain some cloned code. For 2277 files of the 2828 files (81%), the average last change date of the cloned code in the file is lower than the average date for the non-cloned code in the same file, meaning that cloned code is usually older than the non-cloned code. The second research question can thus be answered with *the cloned code in a file is usually older than the non-cloned code in the same file*.

In the following, the individual results for the three systems will be discussed. Figure 2 shows the box plots for the three systems. For each system, the are two box plots for the cloned and the non-cloned code.

3.2.1 ArgoUML

The last (youngest) change observed for the ArgoUML system was on 18 January 2011 and the first (oldest) change was on 4 September 2000. On the two dates, changes to cloned and non-cloned code exist. It can be seen that the average last change date of cloned code in ArgoUML is higher (112 days) than for the non-cloned code, meaning that cloned code is usually *younger* than non cloned code. (The actual average dates of 19 November 2006 for non-cloned code and 11 March 2007 for cloned code are shown in Figure 2.) This result differs from the more general result above which means that for the specific system ArgoUML, the first research question has a different answer. On a side note, it is interesting to see that the bottom quartile is the same for cloned and non-cloned code.

For the second research question, the individual source files of ArgoUML are analyzed: From the 2202 Java source files that have been analyzed, 840 contain some cloned code. For 552 files of the 840 files (66%), the average last change date of the cloned code in the file is lower, meaning that cloned code is usually older than the non-cloned code. This contradicts the the first finding for ArgoUML but supports the general results for the second research question, although the percentage is lower.

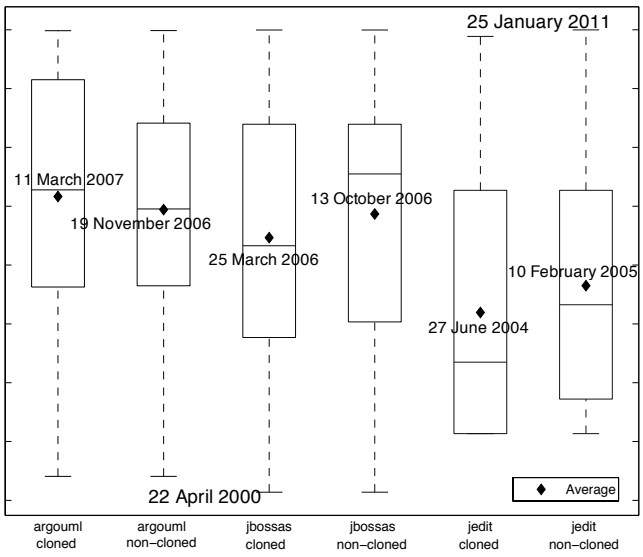

Figure 2: Last change date for cloned and non-cloned code for the three analyzed systems.

3.2.2 JBoss

The results for JBoss are slightly different to ArgoUML's results. The last observed change was on 24 January 2011 and the first observed change on 22 April 2000. The average last changed date for cloned code is 25 March 2006 and for non-cloned code it is 13 October 2006. Here, the average last changed date for cloned code is much lower (a difference of 202 days), meaning that cloned code is usually a lot older than the non-cloned code. Moreover, half of the non-cloned lines (the first two quartiles) are much younger than half of the cloned lines. This supports the general result for the first research question.

From the 6405 Java source files in JBoss that have been analyzed, 1845 contain some cloned code. For 1624 files of the 1845 files (88%), the average last change date of the cloned code in the file is lower, meaning that cloned code is usually older than the non-cloned code. This strongly supports the results for the second research question.

3.2.3 jEdit

The first observed change date for jEdit is 2 September 2001 and it is interesting to see that at least a quarter of the cloned source code lines have not been changed since. The last observed change date for cloned code is 30 November 2010 and for non-cloned code it is 25 January 2011. Overall it seems that the average age of the code in jEdit is higher than in the two other systems, independent if it is cloned or not. The average last changed date for cloned code is 27 June 2004 and for non-cloned code it is 10 February 2005. Again, the average last changed date for cloned code is much lower (228 days), supporting the general result for the first research question.

It is also interesting to see that at least a quarter (the bottom quartile) of the cloned code in jEdit dates back to the beginning of the development of jEdit and has never been changed since then. However, the date is misleading as it is not the day the development started but the day the project was imported into the subversion repository (it was in a CVS repository before—jEdit has a much longer development history which is not available via the subversion repository).

From the 551 Java source files that have been analyzed in jEdit, 143 contain some cloned code. For 101 files of the 143 files (71%), the average last change date of the cloned code in the file is lower than the average date for non-cloned code, strongly supporting the results for the second research question again.

3.3 Threats to Validity

There are some potential threats to validity in the presented study. First of all, there is no clear definition of a clone. Moreover, a clone detected by a clone detector may not be a clone in reality (false positive) or a clone in a system may be missed by a clone detector (false negative). In addition, the choice of the clone detector may influence the results. A study with different clone detectors is in preparation.

The second threat is the low number of systems (three) and the differences in size. JBoss is much larger than the other systems and the results for JBoss may dominate. This risk can only be eliminated by analyzing a large set of systems.

The third threat is the way how changes are detected in version control system. Usually, even a change of layout causes a change in the repository. Such a change does not influence the stability in practice. Other studies [6,8,13,14] have thus applied some canonic format to the sources to improve the results and Krinke [13] has shown that this has a large impact.

Moreover, it is known that automatically generated code or testing code will increase the number of detected code clones. A future study should eliminate such code beforehand.

The experiment is also influenced by the type of the analyzed systems. To be able to draw more general conclusions, more systems of different application types and written in different programming languages should be analyzed and this is planned for future work.

The study is based on the assumption that less changes to cloned code indicate that cloned code does not increase the maintenance cost. However, it might be the case that developers fear to change cloned code and try to circumvent changes to cloned code by changing non-cloned code. This would actually cause the non-cloned code to be less stable than the cloned code.

A threat to validity that will be addressed by a follow-up study is that inconsistent changes to cloned code will cause the clones to split into gapped clones where the variation between the clones is actually in the gap. The changes in the gap may cause the gap code to be younger than the surrounding cloned code, however the gap is considered to be non-cloned code. The follow-up study will therefore identify gap code in addition to cloned and non-cloned code, e.g. by using a clone detector that is able to detect gapped clones [9].

4. RELATED WORK

There are a few empirical studies that analyze the effect of changes on the code clones of a system. The first set of studies discussed in the following analyze clone genealogies, the evolution of code clones during the evolution of a system.

Kim et al. [12] investigated the evolution of code clones and provided a classification for evolving code clones. Their work already showed that during the evolution of the code clones, consistent changes are fewer than anticipated. However, the study only analyzed the evolution of two very small systems, DNSJava and CAROL, both written in Java, and both are a similar type of application.

Aversano et al. [1] followed up with a similar empirical study and a slightly refined framework. Similar to Kim et al., they analyze so called co-changes that are changes committed by the same

author, with the same notes, and within 200 seconds. They used a Java-only clone detector that compares subtrees in the abstract syntax tree. The analyzed systems were DNSJava and ArgoUML. Although Aversano et al. state "that the majority of clone classes is always maintained consistently", the numbers they present contradict this statement: For ArgoUML (which is much larger than DNSJava), they found that only 45% of the clone groups underwent consistent changes. This study has been extended later by Thummalapenta et al. [23] where they analyzed four systems, ArgoUML, JBoss, OpenSSH, and PostgreSQL. In that study, they tracked the evolution of clones and analyzed the evolution patterns of them. They only found in two systems that the majority of the clones evolved consistently. They also looked at (and found) late propagation of changes, i.e. the situation where a change has been missed to apply to all clones of a clone group and which is therefore applied to the missed clones of the group later.

Another study on clone genealogies from Saha et al. [21] on 17 open source systems written in C, Java, C++, and C# showed that the majority of the clone groups of clone genealogies either propagate without any syntactic changes or change consistently in the subsequent releases. Moreover, they found that only 11% to 38% of the clone genealogies changed consistently. However, they explain the low rate of consistent changes with the finding that volatile clones disappear very fast.

Göde [5] presented a model for clone evolution where he tracked the evolution of individual clones throughout the history of a program. He found that whether consistent or inconsistent changes to clone classes were more frequent depended on the analyzed system. He analyzed nine systems (C, C++, Java) during the evolution of 200 versions. Moreover, he discovered a high ratio of late propagation. Göde and Koschke [7] analyzed a large system (Bauhaus) and found that clones tend to be changed inconsistently and that most inconsistent changes stem from independent evolution because of variant algorithms and types.

All of the above studies focussed on the genealogies or lifetime of code clones created from all or a subset of the systems' versions while the presented study only takes a single snapshot of a system and generates the necessary information by 'blaming'. Moreover, it is interesting to see that the lifetime of cloned fragments as reported by the above related studies is much shorter than the average age of a clone as computed in the presented study. For example, Göde [5] reports an average clone lifetime of over a year (483 days) while the presented study shown an average last change date of cloned code of 1719 days (over four years)!

Krinke [13] studied the evolution of code clones in respect to consistent and inconsistent changes of five open source systems. Instead of analyzing clone genealogies, he mapped the changes directly on the detected clones for 200 versions where the versions are one week apart. He found that clone groups are consistently changed in roughly half of the time. With a similar study, Krinke [14] studied the number and amount of changes applied to cloned and non-cloned code during the evolution of five systems and found that non-cloned code is more often changed than cloned code and therefore cloned is more stable than non-cloned-code. Göde and Harder [6] replicated this study and confirmed the findings. Hotta et al. [8] studied the number of changes applied to cloned and non-cloned code, measuring modification frequency. They analyzed 15 systems using four clone detection tools and found that cloned code tend to be modified less frequently than non-cloned code.

Lozano and Wermelinger [17] analyzed the evolution of five systems in terms of their cloning. They found that lines of cloned code change less than those not cloned, but cloned code that changes is highly concentrated in certain methods. Earlier, Lozano et al. [18] found that the majority of methods indeed changed more, and more frequently, when they contain cloned code. Lozano [19] concluded that cloned methods have a poorer changeability than methods not cloned and that cloned fragments cause an increase of changeability decay on cloned methods.

The above studies all rely on mapping the changes to clones in source code by some automated process. Some of the studies even map clones between subsequent versions on each other to track clones. These mappings are sensitive to the used techniques, in particular the mapping of clones between versions usually use clone detection which is know to have low precision and recall. The presented technique to use last change information from version repositories does not suffer the same problems.

Geiger et al. [4] studied the relation of code clones and change couplings (files which are committed at the same time, by the same author, and with the same modification description), but could not find a (strong) relation.

Bettenburg et al. [2] studied the effect on inconsistent changes to code clones on software quality. The analyzed two open source systems and observed that only 1% to 3% of inconsistent changes introduce software defects. Selim et al. [22] also analyzed the impact of cloned code on software defects. They found that the defect-proneness of cloned methods is specific to the system under study. A study of Juergens et al. [10] analyzed commercial and open source systems and found that inconsistent changes to clones are very frequent and that a significant number of faults are induced by such changes.

The relation of code clones to the reliability and maintainability of a system has been examined by Monden et al. [20].

The last change date information from version control systems has been used by Krinke et al. [15, 16] to compare the history of clone pairs and distinguish the original from the copied code. Their approach is able to do the distinction for the majority of clone pairs even between different projects (they analyzed the subprojects of the GNOME Desktop Suite).

5. CONCLUSIONS AND FUTURE WORK

This work studied the question if cloned code is usually older than non-cloned code. The study analyzed three open source software systems und two research questions. The study has shown:

- Cloned code is usually older than non-cloned code. On average, cloned code is 85 days older for the observed systems.
- Cloned code in a file is usually older than the non-cloned code in the same file. For the observed systems, this holds for 81% of the files.

The above results confirm previous results that cloned code is more stable than non-cloned code. Therefore, it cannot be generally assumed that the maintenance of cloned code is more expensive than the maintenance of non-cloned code.

The above study can only be seen as an initial study as it only analyzed three systems, all programmed in Java with a similar length of development history. However, even the initial results are strong and it is expected that a larger follow-up study with many more and diverse systems will confirm the results.

Because the study setup is much simpler than previous studies as it exploits available last change date information from version control systems, the setup can be easily repeated without the need to extract and map the actual changes from the version control systems.

Currently, the study is expanded with the analysis of more and larger systems. It is also planned to use other clone detection tools than *Simian* to achieve more general results. Moreover, code in clone gaps will be analyzed to estimate the influence of variation in clones.

6. ACKNOWLEDGEMENTS

This work is funded in part by Hewlett Packard and the FOSSology Project. Jian Ren produced the box plots.

7. REFERENCES

[1] L. Aversano, L. Cerulo, and M. D. Penta. How clones are maintained: An empirical study. In *11th European Conference on Software Maintenance and Reengineering (CSMR)*, 2007.

[2] N. Bettenburg, W. Shang, W. Ibrahim, B. Adams, Y. Zou, and A. E. Hassan. An empirical study on inconsistent changes to code clones at release level. In *Reverse Engineering, Working Conference on*, pages 85–94, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[3] J. Cordy. Comprehending reality – practical barriers to industrial adoption of software maintenance automation. In *11th IEEE International Workshop on Program Comprehension*, pages 196–205, 2003.

[4] R. Geiger, B. Fluri, H. C. Gall, and M. Pinzger. Relation of code clones and change couplings. In *9th International Conference of Funtamental Approaches to Software Engineering (FASE)*, number 3922 in LNCS, pages 411–425. Springer, Mar. 2006.

[5] N. Göde. Evolution of type-1 clones. In *Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 77–86. IEEE Computer Society, 2009.

[6] N. Göde and J. Harder. Clone stability. In *Proceedings of the 15th European Conference on Software Maintenance and Reengineering*, 2011.

[7] N. Göde and R. Koschke. Studying clone evolution using incremental clone detection. *Journal of Software Maintenance and Evolution: Research and Practice*, 2010.

[8] K. Hotta, Y. Sano, Y. Higo, and S. Kusumoto. Is duplicate code more frequently modified than non-duplicate code in software evolution?: an empirical study on open source software. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, IWPSE-EVOL '10, pages 73–82, New York, NY, USA, 2010. ACM.

[9] Y. Jia, D. Binkley, M. Harman, J. Krinke, and M. Matsushita. KClone: a proposed approach to fast precise code clone detection. In *Third International Workshop on Detection of Software Clones (IWSC)*, 2009.

[10] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 485–495, Washington, DC, USA, 2009. IEEE Computer Society.

[11] C. Kapser and M. W. Godfrey. "Cloning considered harmful" considered harmful. In *13th Working Conference on Reverse Engineering (WCRE)*, pages 19–28, 2006.

[12] M. Kim, V. Sazawal, and D. Notkin. An empirical study of code clone genealogies. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE)*, pages 187–196, 2005.

[13] J. Krinke. A study of consistent and inconsistent changes to code clones. In *14th Working Conference on Reverse Engineering (WCRE)*, Oct. 2007.

[14] J. Krinke. Is cloned code more stable than non-cloned code? In *Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 57–66. IEEE Computer Society, September 2008.

[15] J. Krinke, N. Gold, Y. Jia, and D. Binkley. Cloning and copying between gnome projects. In *7th IEEE Working Conference on Mining Software Repositories*, may 2010.

[16] J. Krinke, N. Gold, Y. Jia, and D. Binkley. Distinguishing copies from originals in software clones. In *International Workshop on Software Clones*, May 2010.

[17] A. Lozano and M. Wermelinger. Tracking clones' imprint. In *Proceedings of the 4th International Workshop on Software Clones*, IWSC '10, pages 65–72, New York, NY, USA, 2010. ACM.

[18] A. Lozano, M. Wermelinger, and B. Nuseibeh. Evaluating the harmfulness of cloning: A change based experiment. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 18–, Washington, DC, USA, 2007. IEEE Computer Society.

[19] A. Lozano Rodriguez. *Assessing the effect of source code characteristics on changeability*. PhD thesis, The Open University, 2009.

[20] A. Monden, D. Nakae, T. Kamiya, S. ichi Sato, and K. ichi Matsumoto. Software quality analysis by code clones in industrial legacy software. In *Eighth IEEE International Symposium on Software Metrics (METRICS'02)*, 2002.

[21] R. K. Saha, M. Asaduzzaman, M. F. Zibran, C. K. Roy, and K. A. Schneider. Evaluating code clone genealogies at release level: An empirical study. In *Source Code Analysis and Manipulation, IEEE International Workshop on*, pages 87–96, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[22] G. M. Selim, L. Barbour, W. Shang, B. Adams, A. E. Hassan, and Y. Zou. Studying the impact of clones on software defects. In *Reverse Engineering, Working Conference on*, pages 13–21, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[23] S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta. An empirical study on the maintenance of source code clones. *Empirical Software Engineering*, March 2009.