Electronic Communications of the EASST
Volume 63 (2014)

Proceedings of the Eighth International Workshop on Software Clones (IWSC 2014)

Clones and Macro co-changes

Angela Lozano, Fehmi Jaafar, Kim Mens, Yann Gaël Guéhéneuc
angela.lozano@uclouvain.be, fehmi.jaafar@polymtl.ca, kim.mens@uclouvain.be,
yann-gael.gueheneuc@polymtl.ca

14 pages

Guest Editors: Nils Göde, Yoshiki Higo, Rainer Koschke
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Clones and Macro co-changes

Angela Lozano[1]*, **Fehmi Jaafar**[2]†, **Kim Mens**[3], **Yann Gaël Guéhéneuc**[2]
angela.lozano@uclouvain.be, **fehmi.jaafar@polymtl.ca**, **kim.mens@uclouvain.be**, **yann-gael.gueheneuc@polymtl.ca**

[1]Vrije Universiteit Brussel.
Pleinlaan 2. Brussels, Belgium. B-1050
[2]École Polytechnique de Montréal
2700, chemin de la Tour. Montreal, Canada. H3T 1J4
[3]Université Catholique de Louvain
2 Place Sainte Barbe. Louvain La Neuve, Belgium. B-1348

Abstract: Ideally, any change that modifies the similar parts of a cloned code snippet should be propagated to all its duplicates. In practice however, consistent propagation of changes in clones does not always happen. Current evidence indicates that clone families have a 50% chance of having consistent changes. This paper measures cloning and co-changes at file level as a proxy to assess the frequency of consistent changes. Given that changes to a clone group are not necessarily propagated in the same commit transaction (i.e., late propagations), our analysis uses macro co-changes instead of the traditional definition of co-changes. Macro changes group bursts of changes that are closer among themselves than to other changes, regardless of author or message. Then, macro co-changes are sets of files that change in the same macro changes. Each cloned file is tagged depending on whether any of the files with which it macro co-changes is cloned with it (during the macro change) or not. Contrary to previous results, we discovered that most of the cloned files macro co-change *only* with files with which they share clones. Thus providing evidence that macro changes are appropriate to study the conjecture of clones requiring co-changes, and indicating that consistent changes might be the norm in cloned code.

Keywords: Cloning, Mining Software Repositories, Empirical Software Engineering, Macro co-changes, Maintenance, Impact, Stability.

1 Introduction

A fragment of code is cloned if there is at least another similar[1] fragment. Fragments that are similar among themselves form a clone group. Two source code entities are said to share a clone when they have cloned fragments belonging to the same clone group. This study measures the

* Funded by the *Cha-Q* SBO project sponsored by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen), Belgium.
† Partly funded by a FQRNT team grant, the Canada Research Chair in Software Patterns and Patterns of Software and the Tunisian Ministry of Higher Education and Scientific Research.
[1] Similarity usually equates to having the same structure.

proportion of cloned files that: (1) co-change with files with which they share clones and (2) co-change with files with which they do not share clones. Our hypothesis is that the proportion of cloned files that co-change with files with which they share clones (1) is higher than the proportion of cloned files that co-change with files with which they do not share clones (2).

1.1 Motivation: Consistent Changes

One of the most common criticisms against clones is that they require consistent changes. A consistent change is a change that alters in the same way and at the same time, all instances of a clone group. Consistent changes can also be interpreted as coupling due to clone relations [GFGP06].

Consistent changes have been analyzed in two ways. First, by classifying changes to clone groups into consistent and inconsistent changes [ACP07, Kri07, GÖ9, JDHW09, TCAD10, GH11b]. Second, by checking whether co-changes can be expressed as a function of cloning [GFGP06].

The former approach has shown that other negative consequences attributed to clone code are rare like instability [Kri07], unintentional incomplete changes [JDHW09], and bugs[JDHW09, GH11b]. It has also shown that clone removals are rare and due to general code restructuring rather than to clone refactoring [GH11a]. This may partially debunk the belief that all clones are harmful. However, the need for consistent changes has not been proved false or true yet. Current results indicate that just 50% of all changes to clone families are consistent [Kri07, JDHW09] , which can be interpreted as whenever there is a change to a clone there is a 50% chance of it being propagated to its group.

The latter approach has shown that a high number of files that share clones also co-change. However, the formula[2] proposed to express co-change relations in function of cloning relations was not statistically significant [GFGP06].

What is the problem with counting consistent and inconsistent changes?

Changes that affect cloned code can be divided into two sets depending on whether or not they modify the structure of the clone group. Examples of changes that modify the structure of the clone group are additions and eliminations of cloned fragments. This paper analyzes only changes that *do not* affect the structure of the clone group, i.e., it focuses on the consistency of changes in a clone group.

There are two problems when classifying clone changes as consistent or inconsistent: the definition of consistent changes, and the interpretation of the classification.

The definition of consistent changes is too generic, and therefore, too strict. A consistent change is *any* change to a clone instance *must* be propagated to the rest of the cloned fragments in the clone group. Nevertheless, it is easy to identify changes to cloned fragments that should not require propagation. For example, renaming identifiers, and in general, any modification within the cloned fragments to parts of code that are not shared with the clone group. In fact, inconsistent changes are rather common, accounting for half of the changes to clone groups, with two thirds of them being intentional [JDHW09].

[2] CouplingCoverage(A, B, I) = 1.038 * CloneCoverage(A, B) + 0.097 where A and B are files, and I is the interval analyzed

The interpretation of the classification can also pose problems. The percentage of consistent and inconsistent changes mixes information that can be interpreted better in isolation. In particular, the distribution of types of changes across clone groups. This information would allow to identify clone groups that do not change, and whether or not clone groups tend to have the same type of changes. In terms of consistent changes, this means that the percentages found may not represent the way in which clones cause changes *in the general case* and that (in case they exist) evolution patterns of different types of clone groups cannot be identified.

What is the problem with counting co-changes?

Commit habits tend to be inconsistent in different projects (due to project policies), for different developers (due to personal preferences), and at different moments in time (due to evolution in the development team, policies, and preferences). In fact, when analyzing commit messages of open source projects, it is evident that a commit may mix changes of several features, refactorings, documentation, addition of features, bug-fixes, etc. Therefore, it is important to take into account all these subtleties when interpreting co-changes. In particular, when analyzing *raw co-changes*. We say that a co-change is raw if it is defined as the set of entities modified in a same commit, without any further restriction. Raw co-changes reflect the nature of the commit habits, which may not have any defined trend. Therefore, raw co-changes could result into a set of entities coupled by random relations instead of the expected domain or functionality couplings.

Moreover, even if it is possible to assume that co-changes represent the expected domain or functionality couplings, their interpretation should consider diverse implementations that can result in such coupling relations. Knowing that two code entities implement the same domain concept or feature, even if they share clones, does not mean that their co-changes are due to their clone relation. For example, a previous attempt to link co-changes to clones [GFGP06] failed to find statistical significance possibly because alternative coupling causes were not removed from the data set or were not taken into account in the model built.

What are the problems of using commits to analyze co-changes?

When extracting data from a code repository, a commit (or a change) is usually identified as the set of files marked with the same commit message and author [ZW04]. The time elapsed between files consecutively modified in a change cannot exceed three minutes [ZW04]. Thus, the time limitation for the definition of changes make it impossible to identify delayed co-changes and co-changes managed by different authors. Using this traditional definition of changes will restrict the co-changes that can be analyzed.

Furthermore, the time constraint reduces the amount of consistent changes that can be identified. Manual [ACP07] and automatic analyses [BMG06, Kri07, LW10, TCAD10] of consistent changes have shown that some clone groups change in the same way but not necessarily at the same time (i.e., late propagations). Therefore, the traditional definition of a change might be inadequate to analyze co-changes in cloned fragments.

1.2 How does this paper contribute to the consistent changes conjecture?

This paper takes a strict definition of co-changes to reduce the chance of fluke co-changes, and a loose definition of changes to allow for delayed co-changes made by different authors (which can be the case for cloned code [BMG06]). In consequence, we analyze less changes and co-changes but we argue that they are more likely to convey whether or not clones require consistent changes or not. Results have shown that whenever cloned files co-change, they tend to do it with files with which they share cloned fragments, and that that files that co-change with their clones (i.e., that require consistent changes) are likely to create bugs.

1.3 Paper Organization

The rest of the paper is organized as follows. Section 2 explains in detail the data collected and its appropriateness to assess the relation between clone coupling and change coupling. The results are described in Section 3 and discussed in Section 4. Section 5 explores threats to the validity of this study. Related work is presented in Section 6. Finally, section 7 summarizes the conclusions and future work.

2 An Alternative Take on Measuring Consistent Changes

This paper analyzes cloned files to check if they tend to co-change with those files with which they share clones or not.

2.1 Macro changes

Instead of the traditional definition of change, this paper uses macro changes. A macro change [JGHA11] is composed of all files marked with time-stamps belonging to the same cluster of time-stamps, regardless of author or commit message (see Fig.1). To find changes occurring close to one another, i.e., belonging to a same macro change, we use the KNN algorithm. KNN was chosen because it does not make any assumptions on the underlying data distribution. There are two limitations in the definition of a macro change (based on empirical findings on the average and longest duration of change requests [Hat07], and validated with the co-changes extracted from bug repositories [JGHA11]). First, the span between the first and the last time-stamp of a macro change is limited to 40 hours. Second, the delay between two consecutive time-stamps in a macro change cannot exceed 5 hours. Given that macro changes group bursts of traditional changes, they can be interpreted as changes at a higher level of abstraction (like changes to a feature).

2.2 Macro co-changes (MCC)

We also use an alternative definition of co-changes called macro co-changes. Macro co-changes cluster files that have the same macro change profiles. A macro change profile is a bit vector that summarizes in which macro changes a file was modified (see Fig.2). Macro co-changes are particularly useful to detect coupling relations documented in bug reports [JGHA11], which is the type of coupling that cloning is supposed to cause.

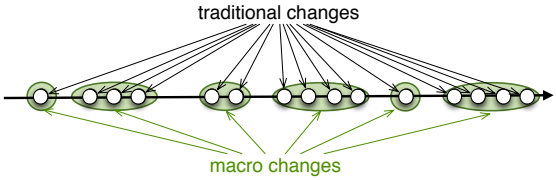

Figure 1: Macro changes group commits (or traditional changes) into time clusters.

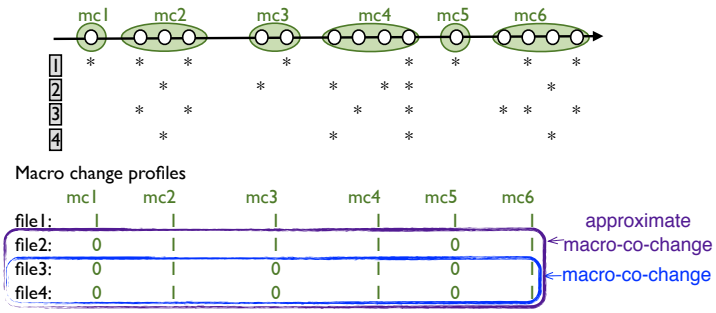

Figure 2: Macro co-changes. Each file is depicted as a small rectangle aligned to the left, below the time line. Each modification is shown with an asterisk that crosses the horizontal line that corresponds to the file modified, and the vertical line that corresponds to a commit transaction. The table shows the macro change profiles. Macro co-changes are shown in blue (i.e., for files with the same macro change profile), and approximate macro co-changes are shown in purple (i.e., for files with similar macro change profiles)

Notice that some files will have no macro co-changes in spite of having co-changes. Nevertheless, this also means that there are no macro co-changes by chance. However, if the only coupling relation between the two files are due to their shared clones, only a fraction of their changes will be shared. Therefore, macro co-changes might be too strict to analyze change coupling due to clones. For that reason, we also calculate approximate macro co-changes, which group files with *similar* macro change profiles (based on their Hamming distance). After analyzing several values of hamming distance between two profiles in different programs, we found that that a Hamming distance of 2 gives the best trade-off between precision and recall. Based on this finding, we consider that two profiles are similar if the Hamming distance between them is equal to 1 or 2.

2.3 Cloned files per macro change

The clones were identified using the standard configuration of CCFinder version 10.2.7 (i.e., minimum 50 consecutive tokens and at least 12 different tokens per cloned fragment). Clones were detected after the end of each macro change. These results were filtered to obtain the set of cloned files, and the set of pairs of files that shared a cloned fragment at that macro change.

2.4 Deciding if cloned files tend to macro co-change

At this point, we have the macro co-change profiles per file, and the macro changes in which each file has been cloned. This information is used to label cloned files. If a file macro co-changes with *at least one file* with which it shares cloned fragments it is labeled '*macro co-changed with its clones*'. If a file macro co-changes with *at least one file* with which it does not share cloned fragments it is labeled '*macro co-changed with others*'. Therefore, a file can have both labels. `The percentage of cloned files labeled only with the tag 'macro co-changed with its clones' is an indicator of their need for consistent changes.`

3 Data collected

We have analyzed six open source systems of different domains and programming languages stored in SVN repositories[3]. For each application we identified: its commits, the files that were cloned per commit, the pairs of files that shared a clone per commit, the cloned files that modified per commit, and finally which commits were linked to bug-fixes documented in their issue-tracking systems (e.g., Bugzilla). This information allowed us to calculate the macro changes of each application. For each file that was cloned at least once in the period analyzed, we identified its macro change profile. Using the profiles we could identify the cloned files that have identical and similar profiles (i.e., the macro co-changes, and approximate macro co-changes respectively). Finally, these results are filtered to locate the percentage of macro co-changes, and approximate macro co-changes in cloned files that can be related to bug-fixes.

Table 1 summarizes the data collected showing the case studies, programming language, revisions analyzed, number of macro changes found in those revisions, the number of files (and cloned files) in that time interval, and the number of different pairs of files sharing cloned fragments in that time interval.

Case study	Programming language	Revisions: from-to	Macro changes	Files: cloned/alive	Pairs of files cloned
ArgoUML	java	16826-19111	190	847/1928	2591
JFreeChart	java	741-2275	130	692/951	46728
XercesC	c++	171415-173224	203	79/236	181
XalanC	c++	338282-340526	218	38/209	57
OpenSER	c	40-2343	246	170/360	334
FreeBSD	c	173512- 197419	84	4/33	2

Table 1: Case studies.

[3] ArgoUML: modeling tool that supports all standard UML. http://guest:@argouml.tigris.org/svn/argouml/trunk/src/
JFreeChart: a chart library for the Java platform. https://jfreechart.svn.sourceforge.net/svnroot/jfreechart/trunk/
XercesC: a validating XML parser. https://svn.apache.org/repos/asf/xerces/c/trunk/
XalanC: an XSLT processor for transforming XML documents. https://svn.apache.org/repos/asf/xalan/c/trunk/
OpenSER: a robust, secure and scalable server implementation. http://openser.svn.sourceforge.net/svnroot/OpenSER/trunk/
FreeBSD: a lite based operating system. svn://svn.freebsd.org/base/head/sys/ufs

Given that it is not clear how long it takes for an inconsistent change to become a late propagation we decided to show the time span of the macro changes analyzed. Table 2 shows the frequency of hours a macro change lasts. For instance, the majority of macro changes in FreeBSD last less than 5 hours, there are no macro changes that last more than 10 hours but less than 20, but around 20 macro-changes last more than 20 hours. Notice that the majority of macro changes lasts less than an hour, and that there is no clear pattern on the duration of longer macro changes across applications.

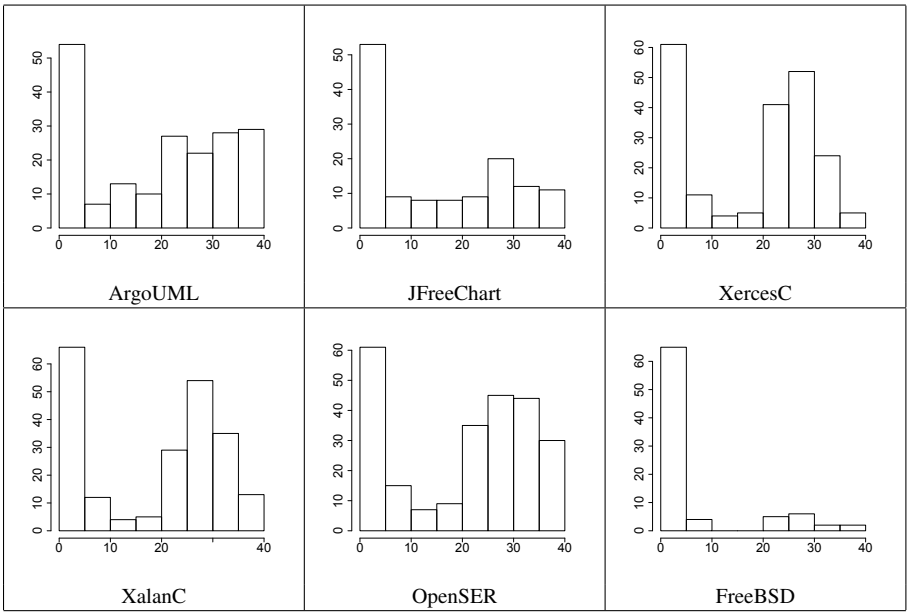

Table 2: Duration of macro changes in hours

4 Results

Tables 3 and 4 summarize the results. Table 3 indicate how many of the cloned files co-changed: with files with which they share clones, and with files which they do not, for different definitions of co-change. Table 4 show how many of the cloned files were modified due to a bug-fix, and to what extent that bug-fix could be correlated to its clone (i.e., when it co-changed with files with which they share clones).

4.1 Co-changes (CCs)

All cloned files co-changed with files with which they do not share cloned fragments. The percentage of cloned files that macro co-changed with files with which it is cloned is: 71% in ArgoUML, 77% in JFreeChart, 69% in XercesC, 92% in XalanC, 76% in OpenSER, and 25% in

		ArgoUML	JFreeChart	XercesC	XalanC	OpenSER	FreeBSD
Files cloned		847	692	79	38	170	4
Co-changed	with its clones	609	534	55	35	48	1
	with others	847	692	79	38	130	4
	with both	609	534	55	35	48	1
Macro Co-changed	with its clones	107	185	3	3	0	0
	with others	23	35	0	0	1	0
	with both	0	0	0	0	0	0
Approximate Macro Co-changed	with its clones	136	239	4	4	4	0
	with others	44	64	0	0	5	0
	with both	0	0	0	0	0	0

Table 3: Co-changes of *cloned files*

FreeBSD. This means that, except for FreeBSD[4] , at least 70% of cloned files also co-changed with files with which they share cloned fragments, and that 30% of cloned files *only* co-change with which they do *not* share cloned fragments. Finally, when considering traditional co-changes, it seems that cloned files co-change indistinctly with files with which they are coupled through clones and with those with which they are not coupled through clones.

4.2 Macro co-changes (MCCs)

Only a small proportion of cloned files macro co-changed. The percentage of cloned files that macro co-changed was: 15% in ArgoUML, 31% in JFreeChart, 3% in XercesC, 7% in XalanC, 0.5% in OpenSER, and 0% in FreeBSD.

For several of the applications analyzed (XalanC, XercesC, and OpenSER), there were very few examples of macro co-changes for cloned files[4]. This might have been caused by the fact that those applications were smaller, and some of them (XalanC and XercesC) had significantly fewer files cloned. We do not think that these differences in the amount of macro co-changes is related to the nature of the clones because previous studies have shown that cloned code tends to have similar properties regardless of the programming language in which it is written [RC10].

The proportion of cloned files that macro co-change with files with which they do not share clones is significantly lower from those that share with files with which they share clones. The percentage of cloned files that macro co-changed with clone-coupled files is: 82% in ArgoUML, 84% in JFreeChart, 100% in XercesC, 100% in XalanC, 0% in OpenSER, and 0% in FreeBSD. OpenSER is the only application for which there are more files that macro co-change with files other than those with which they share clones. However, it is not clear whether this is due to the small sample size or due to the nature of the application. There were no files that macro co-change with files with which they do not share clones *and* with files with which they share clones. In fact, more than 80% of the macro co-changes of cloned files occurred *only* with those files with which they shared cloned fragments.

[4] Results for FreeBSD might not be comparable with those found for other applications due to the portion analyzed was too small (just 33 files from which only 4 were cloned).

4.3 Approximate macro co-changes (AMCCs)

As expected, there are more approximate macro co-changes than macro co-changes. The percentage of cloned files that approximate macro co-changed was: 21% in ArgoUML, 43% in JFreeChart, 5% in XercesC, 10% in XalanC, 5% in OpenSER, and 0% in FreeBSD.

Again, FreeBSD had no AMCCs[4], while XalanC, XercesC, and OpenSER had very few examples of AMCCs for cloned files.

As with macro co-changes, cloned files tend to approximate macro co-change with files with which they share clones. 75% in ArgoUML, 78% in JFreeChart, 100% in XercesC, 100% in XalanC, 44% in OpenSER, and 0% in FreeBSD of cloned files approximate-macro co-changed with files with which they share clones. Again, the results for OpenSER seem to contradict the results for the rest of the applications. Also, none of the cloned files approximate macro co-changed with files with which they do not share clones *and* with files with which they share clones. Furthermore, the majority of approximate macro co-changes of cloned files occurred *only* with those files with clone couplings.

4.4 Co-changes, Clones, and Bugs

Although a high percentage of cloned files change during bug-fixes, a small percentage of these files also macro co-change during bug-fixes, 25% being the highest value for approximate macro co-changes in JFreeChart and 1% the lowest value for approximate macro co-changes in OpenSER. In any case, all the cloned files that (approximate) macro co-changed during bug-fixes co-changed with their clones, while none of them (approximate) macro co-changed with other files. This indicates that files that (approximate) macro co-change with others are unlikely of be modified during bug-fixes. Furthermore, files requiring consistent changes (those that co-change with their clones) are indeed likely to create bugs. Finally, from those files that require consistent changes, the percentage of cloned files that macro co-changed during bug-fixes varies significantly from 69% of the files in the worst case of ArgoUML, 12% in JFreeChart, 25% in XalanC* and XercesC*, and 50% in OpenSER*[5].

	ArgoUML	JFreeChart	XercesC	XalanC	OpenSER	FreeBSD
Files cloned	847	692	79	38	170	4
Changed in bug-fixes	726	116	33	33	129	3
MCC in bug-fixes w. its clones	74	23	0	0	0	0
MCC in bug-fixes w. others	0	0	0	0	0	0
AMCC in bug-fixes w. its clones	93	29	1	1	2	0
AMCC in bug-fixes w. others	0	0	0	0	0	0

Table 4: *Cloned files* and bugs

[5] * for approximate macro co-changes.

5 Discussion

The analysis of the data collected point to several conjectures that would require individual validation, that fall out of the scope of this paper:

- The lack of a common threshold to define the time span between the first and the last change to cloned fragments in a late propagation might be one of the reasons for the low proportions of co-changes in cloned code reported so far. It is necessary to check to what extent macro changes reflect late propagations, that is, manually validating to what extent the changes to cloned files that tend to co-change with their clones indeed affect their cloned fragments.

- The percentage of cloned files that co-change with the files with which they share cloned fragments is lower for approximate-macro co-changes than for macro co-changes. This indicates that a strict definition of co-changes is required to compensate for the leeway allowed when detecting macro changes. Therefore, an analysis of the precision and recall of detection of changes to clone groups using a range of values to configure the definitions of macro changes and co-changes is needed.

- For all the definitions of co-change, there were cloned files that never co-changed with files with which they shared clones. This indicates that a portion of cloned files are supposed to evolve independently from their clones. Further analysis is necessary to identify whether or not these files (or their clones) are significantly different from those that require consistent changes.

- Cloned files in OpenSER seemed to co-change more with files with which they did not have clone relations. Thus, this experiment should be replicated with other systems to define the limitations of its results.

6 Threats to validity

The following considerations must be taken into account when analyzing to what extent our conclusions reflect reality.

6.1 Internal Validity

This experiment analyzes the relation between clones and co-changes. Given that this study does not manipulate the independent variable (being cloned) to see its effect on the dependent variable (having co-changes), there is no bias due to manipulations. However, there is ambiguity on the direction of the causal relation. This paper assumes that clone relations cause additional co-changes, that is, co-changes in files that otherwise would not be related. Nevertheless, it might be the case that files that are semantically close to each other (and as a result that co-change), are the files that become cloned.

6.2 External Validity

Another issue regarding this study is the limited sample size (six applications analyzed). Although it is necessary to replicate the results with more case studies, the fact that these are real programs of different domains and with different sizes, histories, programming languages, indicate that the results might be generalizable to at least open source applications, implemented in imperative/object oriented languages, and whose size is small to medium.

6.3 Construct Validity

The macro co-changes might not reflect semantic links, like the ones one expects to find in source code entities that implement the same feature. This threat is mitigated by the fact that the definition of macro changes is based on the empirical analysis of change requests and validated with information extracted from bug repositories. However, our findings are limited to our definition of clones and co-changes.

6.4 Conclusion Validity

Other possible causes for co-changes are minimized by a non-exclusive tagging of the files. In this way, it is unmistakable which percentage of cloned files probably co-changed due to their clone[6], and which percentage of cloned files probably co-changed due to other reasons[7]. Notice that we do not claim that the observed co-changes or a (approximate) macro co-changes are due to clones because we do not verify that the modifications made affected the fragments cloned of these files. Also, the low percentage of files that macro co-change indicate that a large percentage of co-changes happen by chance rather than by semantic links among the files. In any case, the recurrent (approximate) macro co-changes among files that have cloning relations in comparison with those that do not have such relations indicates that macro changes might be an appropriate alternative to study the conjecture of clones requiring co-changes.

7 Related work

This section describes previous studies on correlating co-changes and clones, counting consistent changes, and reducing noise of co-changes

7.1 Correlating co-changes and clones

The first attempt to correlate co-changes and clones at file level showed a high number of pairs of cloned files that co-change among them, but the results were not statistically significant [GFGP06].

Previously, we had shown that from the pairs of methods that shared a clone 12% co-changed, while less than 0.1% of the method pairs that did not shared cloned fragments co-changed [LWN07].

[6] Those changed with files with which they shared cloned fragments but not with other files

[7] Those changed with files with which they shared cloned fragments and with other files, plus those changed with files with which they did not share cloned fragments

7.2 Counting consistent changes

Aversano et al. analyzed manually to what extent inconsistent changes became late propagations. They found that late propagations rare, and that the time span from inconsistent changes to late propagations is approximately 24 hours [ACP07].

Krinke classified consistent and inconsistent changes (neglecting changes to comments and indentation) to the exact clone families detected at a baseline version. He found that around half of the changes to clone families are inconsistent [Kri07].

Gode replicated Krinke's study and found that for the majority of applications analyzed the most common pattern was inconsistent changes (i.e., between 50 to 80% of changes), while the rest of applications had only between 30 to 50% of inconsistent changes [GÖ9].

Juergens et al. classified changes to clones into consistent and inconsistent and found that around half of the clone families have inconsistent changes. They also subdivided inconsistent changes into intentional and unintentional finding that unintentional changes are the minority (accounting for just one third of inconsistent changes)[JDHW09].

Thummalapenta et al. showed that only 11 to 38% of clone families had consistent changed, and that less than 16% of changes were late propagations [TCAD10].

Finally, Gode and Harder analyzed to what extent a type of change tends to be maintained along the lifetime of a clone group by counting different patterns of consecutive changes to clone families. They found that the most common pattern is consistent changes followed by consistent changes [GH11b].

7.3 Relating clones to bugs

Several of the papers mentioned before also explored the relation between clones and bugs, in particular, between late propagations and bug-fixes. Aversano et al. found that several late propagations were linked to bug-fixes [ACP07].

Juergens et al. showed that only half of the unintentionally inconsistent changes were related to faults [JDHW09].

Gode and Harder confirmed that very few of the inconsistent changes are unintentional [GH11b].

7.4 Reducing noise of co-changes

Mondal et al. also argue that there are co-changes that do not reflect any semantic relations, they just group source code entities by chance. For instance, updates to documentation or indentation. They propose to improve the accuracy of co-changes to express semantic relations by taking into account the significance of the commits. They define the significance of a commit as inversely proportional to the amount of source code entities that it modifies. The results indicate that source code entities that co-changed once or twice might nonetheless express semantic links [MRS13].

8 Conclusions and Future work

The analysis of previous work indicates that the relation between cloning and co-changes requires a relaxed definition of changes, balanced by a stricter definition of co-changes. Given that

clone groups may change in the same way but not necessarily at the same time, macro changes might be more appropriate to analyze consistent changes in clones than commits. Moreover, raw co-changes add noise to the analysis by adding relations that are the product of commit habits instead of the expected domain or functionality links; therefore, it is necessary to filter fluke co-changes. Although previous attempts at providing evidence that cloned code requires to be co-changed have been unsuccessful, this paper shows that cloned files tend to macro co-change only with files with which they share cloned fragments. Moreover, given that the cloned files analyzed either changed with files that shared cloned fragments with them or with files that did not shared cloned fragments with them (but not with both), allows us to believe that we could predict which set of files could co-change with their clones.
To confirm the conjectures established with this study, we are working in several extensions of this work. Among others: (1) the validation that the lines changed in the macro co-changes indeed affect the fragments cloned, (2) the replication of the study with (a) several configurations for macro changes, macro co-changes, and approximate macro co-changes, (b) different case studies, (c) several clone detection tools, and (d) several definitions of co-change, (3) a more detailed analysis that includes differentiation of results (a) per types of clones, (b) granularity of the clone, and finally (4) a comparison of the (a) clone characteristics, and (b) file characteristics of the cloned files that co-change with their clones vs. those that co-change with other files.

Bibliography

[ACP07] L. Aversano, L. Cerulo, M. D. Penta. How Clones are Maintained: An Empirical Study. In *Proc. of the European Conf. on Software Maintenance and Reengineering (CSMR'07)*. Pp. 81–90. 2007.

[BMG06] M. Balint, R. Marinescu, T. Girba. How Developers Copy. In *Proc. of the Int'l Conf. on Program Comprehension (ICPC)*. Pp. 56–68. IEEE Computer Society, 2006. thesis.

[GÖ9] N. Göde. Evolution of Type-1 Clones. In *Proceedings of the 2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*. SCAM '09, pp. 77–86. IEEE Computer Society, Washington, DC, USA, 2009.

[GFGP06] R. Geiger, B. Fluri, H. C. Gall, M. Pinzger. Relation of Code Clones and Change Couplings. In *Proc. of the Int'l Conf. of Fundamental Approaches to Software Engineering (FASE'06)*. Pp. 411–425. 2006.

[GH11a] N. Göde, J. Harder. Clone Stability. In *Proc. of the European Conf. on Software Maintenance and Reengineering (CSMR)*. Pp. 65–74. IEEE Computer Society, 2011.

[GH11b] N. Göde, J. Harder. Oops! . . . I changed it again. In *Proc. of the 5th Int'l Workshop on Software Clones (IWSC)*. Pp. 14–20. ACM, 2011.

[Hat07] L. Hatton. How Accurately Do Engineers Predict Software Maintenance Tasks? *Computer* 40(2):64–69, Feb. 2007.

[JDHW09] E. Juergens, F. Deissenboeck, B. Hummel, S. Wagner. Do code clones matter? In *Proc. of the Int'l Conference on Software Engineering (ICSE'09)*. Pp. 485–495. 2009.

[JGHA11] F. Jaafar, Y.-G. Gueheneuc, S. Hamel, G. Antoniol. An Exploratory Study of Macro Co-changes. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*. WCRE '11, pp. 325–334. IEEE Computer Society, Washington, DC, USA, 2011.

[Kri07] J. Krinke. A Study of Consistent and Inconsistent Changes to Code Clones. In *Proc. Working Conf. on Reverse Engineering (WCRE'07)*. 2007.

[LW10] A. Lozano, M. Wermelinger. Tracking clones' imprint. In *Proc. Int'l Workshop on Software Clones*. 2010.

[LWN07] A. Lozano, M. Wermelinger, B. Nuseibeh. Evaluating the harmfulness of cloning: a change based experiment. In *Proc. of the int'l workshop on Mining Software Repositories (MSR'07)*. Pp. 18–21. 2007.

[MRS13] M. Mondal, C. K. Roy, K. A. Schneider. Improving the detection accuracy of evolutionary coupling. *21st Int'l Conf. on Program Comprehension (ICPC)* 0:223–226, 2013.

[RC10] C. K. Roy, J. R. Cordy. Are Scripting Languages Really Different? In *Proc. of the 4th Int'l Workshop on Software Clones*. IWSC '10, pp. 17–24. ACM, New York, NY, USA, 2010.

[TCAD10] S. Thummalapenta, L. Cerulo, L. Aversano, M. Di Penta. An empirical study on the maintenance of source code clones. *Empirical Softw. Engg.* 15(1):1–34, Feb. 2010.

[ZW04] T. Zimmermann, P. Weibgerber. Preprocessing CVS data for fine-grained analysis. In *Proc. of the int'l workshop on Mining Software Repositories (MSR)*. Pp. 2–6. 2004.