Empir Software Eng (2017) 22:235–258
DOI 10.1007/s10664-015-9419-z

Raters' reliability in clone benchmarks construction

Alan Charpentier[1] · Jean-Rémy Falleri[1] · Floréal Morandat[1] · Elyas Ben Hadj Yahia[1] · Laurent Réveillère[1]

Published online: 9 February 2016

Abstract Cloned code often complicates code maintenance and evolution and must therefore be effectively detected. One of the biggest challenges for clone detectors is to reduce the amount of irrelevant clones they found, called *false positives*. Several benchmarks of true and false positive clones have been introduced, enabling tool developers to compare, assess and fine-tune their tools. Manual inspection of clone candidates is performed by raters that do not have expertise on the underlying code. This way of building benchmarks might be unreliable when considering context-dependent clones i.e., clones valid for a specific purpose. Our goal is to investigate the reliability of rater judgments about context-dependent clones. We randomly select about 600 clones from two projects and ask several raters, including experts of the projects, to manually classify these clones. We observe that judgments of non expert raters are not always repeatable. We also observe that they seldomly agree with each others and with the expert. Finally, we find that the project and the fact that a clone is a true or false positive might have an influence on the agreement between the expert and non experts. Therefore, using non experts to produce clone benchmarks could be unreliable.

Communicated by: Audris Mockus

✉ Alan Charpentier
acharpen@labri.fr

Jean-Rémy Falleri
falleri@labri.fr

Floréal Morandat
fmoranda@labri.fr

Elyas Ben Hadj Yahia
elyas.bhy@labri.fr

Laurent Réveillère
reveillere@labri.fr

[1] LaBRI, UMR 5800, University of Bordeaux, F-33400 Talence, France

Keywords Duplication · Code clone · Empirical study · Software metrics

1 Introduction

Redundant code in software systems is referred as *code clones* in the literature. Two code fragments form a code clone if they are similar or identical (Bellon et al. 2007). Code clones can arise due to the use of so-called *"copy-paste"* development, in which a developer creates new code by copying existing code that is expected to have similar intent, and possibly modifying it slightly according to its new context. Duplicating code can reduce dependencies between modules and help new developers conform to the prevailing coding style. While cloning code is not always harmful to the system quality (Kapser and Godfrey 2006), it often complicates code maintenance and evolution, as fault fixes and changes must be propagated from one instance of a clone to the others. Therefore, software developers tend to keep the number of code clones as low as possible.

The desirability of identifying, and sometimes eliminating, clones has led to the development of *clone detectors*, which scan a code base for potential clones (Gode and Koschke 2009; Jiang et al. 2007; Kamiya et al. 2002; Li et al. 2006; Roy and Cordy 2008). Several studies have used such tools to estimate the number of code clones in a software project, reporting from 5 % to 23 % of cloned code (Baker 1995; Baxter et al. 1998; Lague et al. 1997). One of the biggest challenges for an effective application of clone detection in practice is detection precision. Indeed, some code clones reported by a clone detector may be viewed as *false positives* as they are irrelevant for the user of the detection tool. In addition, identifying false positives is made challenging simply by the amount of data that a clone analysis of a large project generates.

A traditional approach to reduce the number of false positives is to enhance the clone detection algorithm and fine-tune the detection tool. To support this tuning, several benchmarks of clones have been proposed (Bellon et al. 2007; Krutz and Le 2014; Svajlenko et al. 2014), enabling tool developers to assess the results of their tools. Recently, machine learning techniques have been introduced to automatically classify generated clones in true or false clones based on feedback from the user of the tool (Yang et al. 2014). These approaches rely on a set of clones that have already been rated as true or false positives. Producing such a set of clones always requires a human judgment. Manual inspection of clone candidates is usually performed by students or researchers who may have no or few expertise on the underlying software projects.

Clone candidates from existing clone benchmarks are usually classified without any specific purpose in mind. We call such benchmarks context-free as there is no relation between their usage and the way they are defined. As a consequence, one could ask to which extent the present way of building clone benchmarks is reliable when considering context-dependent clones i.e., clones classified as useful with respect to a specific purpose.

In this paper, we investigate the reliability of rater judgments about context-dependent clones. Our goal is to bring out several guidelines to ease the construction of context-dependent clones benchmarks. In this experiment, we address both refactoring and co-evolution activities. We consider two software projects and for each of them a set of four raters to judge the results produced by a clone detector. Among the raters, three are *external raters* with no or few preliminary knowledge about the project and one is an *expert*, one of the main developers of the project. Expert's judgment is very important for a context-dependent clone benchmark because the clones identified by a clone detector have to be useful for her. Hence, the two experts are oracles deciding whether a clone is a true or false

positive, according to a given context. Finally, we use the answers of the raters to discuss the reliability of rater judgments.

We show that using external raters to build context-dependent clone benchmarks may be unreliable. We also show that the project being studied may have a significant impact on the agreement between external raters and experts. Additionally, we find that true positives clones (as judged by the experts) seem to be significantly harder to judge by external raters. Based on our findings, we recommend to use several external raters to build context-dependent clone benchmarks and to involve experts to validate true positives.

The structure of this paper is as follows. Section 2 describes related work. Section 3 introduces the issue of detection precision. Next, we describe our empirical study methodology in Section 4. Section 5 presents the findings of our empirical study; it highlights a number of research questions and their answers. We discuss the threats to the validity of our study in Section 6. We conclude and mention future work in Section 7.

2 Related work

In this section, we provide some background about clone benchmarks and clone rater reliability.

2.1 Clone benchmarks

Bellon et al. provide a benchmark of clones to compare and evaluate clone detection tools (Bellon et al. 2007). Six researchers helped Bellon to construct the benchmark. Each of them applied his own clone detector on eight large C and Java programs and provided the found clones to Bellon. Bellon classified alone 2 % of the 325,935 submitted clones and built a reference corpus by retaining only clones he judged as true positives. Clone detectors' results are compared to the reference corpus, and assessed using the traditional information retrieval measures: precision and recall. This benchmark is largely used by the research community (Ducasse et al. 1999; Koschke et al. 2006; Nguyen et al. 2012; Selim et al. 2010; Wang et al. 2013).

Krutz and Le built a set of function-level clones to help the evaluation of clone detection tools (Krutz and Le 2014). They asked seven persons to judge 1,536 randomly drawn function pairs from three open source programs: Apache, Python and PostgreSQL. Three are experts who have research experience with code clones and four are students with programming expertise but no prior experience with clones. Students were asked to read related papers beforehand to familiarize with the notion of clones. Their process is more rigorous than the one used by Bellon et al. (2007) but the size of their benchmark is significantly smaller, which might impact its usefulness.

Svajlenko et al. provide a function-level benchmark of inter-project clones (Svajlenko et al. 2014), called BigCloneBench. This benchmark is built independently of clone detection tools, by using a search-based approach. Hence, it is not limited to the clones that tools are able to identify. This benchmark aims at containing clones related to ten functionalities (such as bubble sort or zip decompression). For a candidate clone, several raters manually check if it is a true or false positive clone by deciding whether or not related code fragments implement the same functionality. The judges are provided a clear specification of what implementing the functionality requires. This benchmark differs from the others on clone validation. It is the only one considering context-dependent clone. Context matches a functionality.

Roy and Cordy propose a mutation / injection based framework for empirically evaluating clone detection tools (Roy and Cordy 2009). The framework has two main phases. First, mutated code fragments are created from their proposed editing taxonomy for cloning. Second, these mutated code fragments are injected into original code base. Then, precision and recall values are computed and used to evaluate individual tool or to compare different tools. Their framework does not rely on clone validation since the clones are automatically created. Thus, it avoids the validation issues. However as mentioned in the paper, their editing taxonomy cannot guarantee to create clones for a particular task, such as refactoring or co-evolution.

In all the benchmarks we present above, no raters are experts on the programs used to build the clone benchmarks.[1] While this way of building benchmarks is valid for context-free clones, one can ask whether it is still correct when considering context-dependent clones. In our study, we investigate this potential threat to validity of the construction of context-dependent clone benchmarks and assess its impact.

2.2 Clone raters reliability

Walenstein et al. use the corpus provided by Bellon et al. (2007) to investigate the level of agreement among raters (Walenstein et al. 2003), who are researchers. They claim that past reports of relevance and precision for clone detectors have to be interpreted with caution because of the problem of inter-rater reliability. Additionally, they find that the agreement among raters strongly depends on the projects being studied and on the question asked to raters. In this article, we also observe that the choice of the project has a significant influence on the agreement.

Kapser et al. perform a study to assess agreement among researchers when classifying potential clones as true or false positives (Kapser et al. 2006). They use CCFinder (Kamiya et al. 2002) to select 20 candidate clones from the PostgreSQL source code. They observe that only 50 % of candidate clones are classified in the same way by more than 80 % of the raters. They highlight the importance of reporting the criterion used for clone judgment in order to make possible the comparison with previous works. In this article, we also observe that raters can have different judgments about clones.

Mende et al. propose an approach to support the grow-and-prune model (Faust and Verhoef 2003) in the evolution of software projects by using clone detection (Mende et al. 2009). They evaluate their technique by measuring recall and precision with respect to a benchmark. They consider multiple raters to increase the confidence of the judgments. The raters, five researchers and four graduate students in computer science, judge the same pairs of functions. A concrete scenario is given to the raters to help them decide whether or not two functions are similar. Authors find a reasonable agreement among the raters. This result is different from the observation we make in this article. However, the work of Mende et al. considers function clones, and our work is about all types of clones. This could have an influence on the agreement of raters. Additionally, the clones of Mende et al. were not validated by an expert of the analyzed code. Such an expert may have a different judgment about these clones.

In a previous work (Charpentier et al. 2015), we analyzed the reliability of the reference clones contained in Bellon's benchmark. We showed that there are debatable clones in this benchmark, and that it might alter the values of precision and recall computed using the

[1] These studies mentioned no information about any raters' expertise of the analyzed code.

benchmark. In this article we go beyond this previous work. First, raters are asked to judge context-dependent clones, either for refactoring or co-evolution activity. Second, we add experts of the underlying code in the experiment. We found that agreeing with an expert is hard, especially for true positive clones.

3 Research questions

Manual inspection of clone candidates in existing benchmarks is usually performed by students or researchers who are not experts of the underlying software projects. Additionally, these raters generally do not receive any training before judging the clones. Nevertheless, since these benchmarks are context-free, one can ask to which extent this way of building clone benchmarks is reliable when considering context-dependent clones?

In this study, we explore the reliability of rater judgments about context-dependent clones. We focus our research on two main points. First we investigate the repeatability of raters' answers. We wonder if, when a rater is presented several times the same clones, she will always judge the clone consistently. Second, we explore inter-rater reliability. More precisely, we evaluate if several raters judge the clones identically. We include an expert of the code in the set of raters. Reflecting the judgment of an expert is very important for a context-dependent clone benchmark because the clones reported by a clone detector must be useful for her. Therefore we evaluate if external raters can judge clones in the same way as an expert of the code. Finally we also evaluate if there are some clones for which it is easier to reach an agreement between external raters and experts. To sum-up, we investigate the following research questions.

3.1 Repeatability of rater answers

RQ1. *Are rater answers consistent over time?*

As previously explained, raters are usually not trained before judging clones. This could be a threat to the repeatability of their answers, since there is a chance that they can be subject to a learning effect when rating clones. In this research question, we want to assess if a rater would always judge the same clone in the same way. We also want to refine this question by distinguishing experts from external raters. If the judgments of the clones are not repeatable, it could be a sign of a learning effect.

3.2 Inter-rater reliability

RQ2. *Do external raters agree with each others, and also with the expert of a project?*

In this research question, we want to assess if several raters judge the clones identically. First of all, we want to evaluate to which extent several external raters agree with each others. Then we want to evaluate if the external raters agree with the experts. By answering this research question, we know if it is reliable to build context-dependent clone benchmarks by using external raters instead of experts.

RQ3. *What are the characteristics that influence the agreement between experts and external raters?*

In this research question, we want to evaluate if some characteristics make easier for external raters to have the same judgment as the expert. We investigate this question because

if some characteristics are shown to make the clone very easy to judge in the same way as the expert by the external raters, we can build reliable context-dependent clone benchmarks by including clones having these characteristics, without the need of an expert.

We investigate two types of characteristics: clone based characteristics and project and expert based characteristics.

The clone-based characteristics are as follows:

- *Type:* is it easier to judge Type-2 or Type-3 clones (as defined by Bellon et al. (2007))? We do not consider Type-1 clones as they are less debatable.
- *Size:* is it easier to judge clones having big fragments than clones having small fragments?
- *Distance:* is it easier to judge clones in completely different files or in the same file?

The project and expert based characteristics are as follows:

- *Project:* is it easier to judge clones from one project rather than another one?
- *False/true positives:* is it easier to judge true or false positive clones, as judged by the expert?

In the remainder of the paper, clones refer to context-dependent clones.

4 Experimental setup

We describe in this section the empirical study we conduct to answer our research questions and investigate the accuracy of user judgments about clones. All data used and collected during this study is available online.[2]

4.1 Overall approach

Our experiment consists in drawing randomly a set of clones from selected projects and present them to four raters, including an expert of each selected project. The raters are asked to judge the clones, through a web interface. The two code fragments of each clone are displayed to the rater and she is asked to rate *yes* in case of a true clone, *no* in case of a false clone, and *unknown* otherwise, i.e., if there is no obvious answer. Finally we use a statistical analysis of the produced data to answer our research questions. This analysis is presented in Section 5.

4.2 Subjects and objects

We describe in this section the subjects and objects of our experiment. First we describe the software projects on which we compute the clones. Then we explain which clone detector was used, and how it has been configured. Finally, we present the raters who participated in this experiment.

[2] http://www.labri.fr/perso/acharpen/ese15/materials.zip

Table 1 Software projects' description

Project	Java Files	Java LOC	# Clones
FastR	343	54,511	49,911
GumTree	77	4,750	216

4.2.1 Software projects

Our experiment requires an expert for each project under study. Thus, we select two projects developed by people from our research group: FastR[3] and GumTree.[4] Both projects use Java as the main language. FastR (Kalibera et al. 2014) is an open-source efficient implementation of the R statistical language (Ihaka and Gentleman 1996). GumTree (Falleri et al. 2014) is an open-source abstract syntax tree based code diff tool.

To prevent the clone detector to search clones in locations that are irrelevant for the user, the code base is pre-processed according to the advice of the expert of each project. First, files that are automatically generated and never modified manually are discarded. Second, files corresponding to tests and examples are also discarded. Indeed, such code is usually maintained in a different way than the code from the core. At the end, we stripped out blank lines and comments in all remaining files. Table 1 summarizes the number of files and the number of lines of code (LOC) in Java we obtain for each project. GumTree is a small-sized project and FastR is a medium-sized project. The small size of GumTree allows to manually investigate all clones a clone detector could report.

4.2.2 Clone detector

Various clone detectors have been introduced to identify clones from source code. All of them can be tuned using plenty of parameters. Therefore, the choice of a specific clone detector and its configuration parameters may have a huge impact on the list of computed clones. In this study, we rely on the work of Wang et al. (2013) to minimize the impact of this choice. They introduce an approach to find suitable configurations for empirical studies. We choose a configuration that maximizes recall because we want an overview of all the clones present in the selected projects.

We choose iClones (Gode and Koschke 2009) as the clone detector for two reasons. First, it is easily available for replication purposes. Second, it is the only reputed Type-3 clone detector in the literature that is compatible with Java Generics, and both selected projects use this feature a lot. iClones has two parameters: `minblock`, the minimum length of identical token sequences that are used to merge near-miss clones, and `minclone`, the minimum length of clones measured in tokens. As recommended by Wang et al. to maximize recall for Java projects, we set `minblock` to 6 and `minclone` to 26.

[3] https://github.com/allr/fastr/tree/v0.168

[4] https://github.com/jrfaller/gumtree/tree/v1.0.0

4.2.3 Clone raters

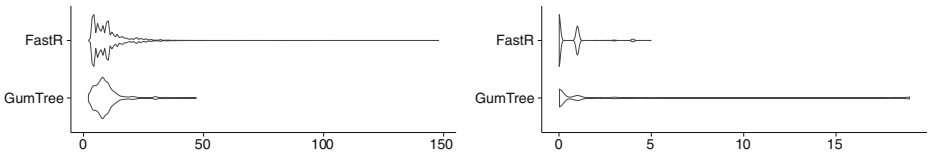
The four raters participating to this experiment are authors of the paper. The fifth author, namely Alan Charpentier, does not rate clones. He is responsible for running a fair experiment to answer predefined research questions. He was the only one to know the research questions during the survey.

All participants have an extensive Java programming experience and all are more than familiar with the notion of clones. For each project, we consider one expert and three external raters. For FastR, the expert is Floréal Morandat, one of the main developer of this project. For GumTree, the expert is Jean-Rémy Falleri, its official maintainer. Non expert raters are external raters and denoted by rater1, rater2 and rater3 in the remainder of the paper.

4.3 Clone selection

For each project, we run iClones to compute a list of clones. Table 1 reports the number of clones (only Type-2 and Type-3) identified by iClones in the two selected projects, using the configuration presented above. We remind that in **RQ3**, we want to explore if there are factors that have an effect on the judgment of raters. Some factors, namely *type*, *size* and *distance* are information computed from the clones. We use the same definition of type as Bellon et al. (2007). Clone size is defined as the number of lines of code of the two code fragments involved laid end to end. Clone distance represents how close are these two code fragments in the file system tree. If the two fragments are in the same file, the distance is 0. If they belong to different files within the same directory, the distance is 1. In any other cases, the distance is 1 plus the minimum number of hops to reach one file from the other.

To ensure having at least some representatives of each characteristic, clones are randomly drawn according to their characteristic. Concerning the *type* factor, once Type-1 removed (see **RQ3**), there are two natural clusters: *Type-2* and *Type-3*. However, there is no natural definitions of clusters for *size* and *distance*. By performing an exploratory analysis in our projects of both the clone sizes and distances distribution (Fig. 1), we noticed that these distributions are right skewed with a very long tail. Therefore we choose to separate the clones in two clusters, the ones from the head of the distribution, and the ones from the tail. To avoid the bias of selecting a threshold, we use an automatic technique to create the clusters for *size* and *distance*. Additionally, we want this technique to be deterministic so that researchers can reproduce our results. To fulfill these requirements we use the *neural gas* data-mining algorithm (Martinetz and Schulten 1991) which is a generalization of *k-means* that produces stable clusters. On our corpus, tens of runs of *neural gas* produce clusters without any significant difference. Finally we end up with four clusters. Two are from the *size* factor: Σ-*Big* and Σ-*Small*, and two from the *distance* factor: Δ-*Close* and



(a) Size characteristic (b) Distance characteristic

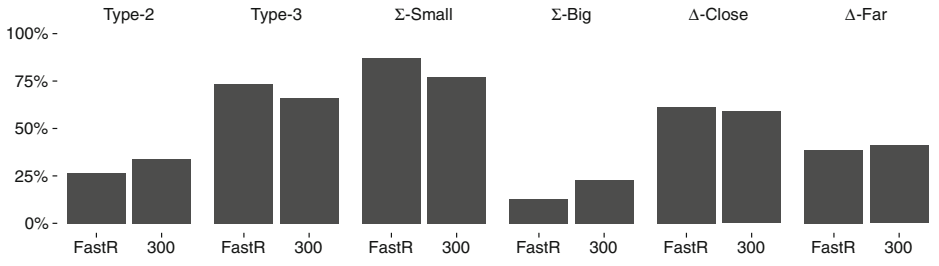
Fig. 1 Distribution of clone sizes and distances in FastR and GumTree

Δ-*Far*. The computed threshold to separate the two clusters of the *size* is 17 (resp. 18) for FastR (resp. GumTree). In other words, clones in FastR (resp. GumTree) having a size lower or equal to 17 lines (resp. 18) are in Σ-*Small*, while others are in Σ-*Big*. The thresholds for the *distance* are 0 for FastR and 5 for GumTree.

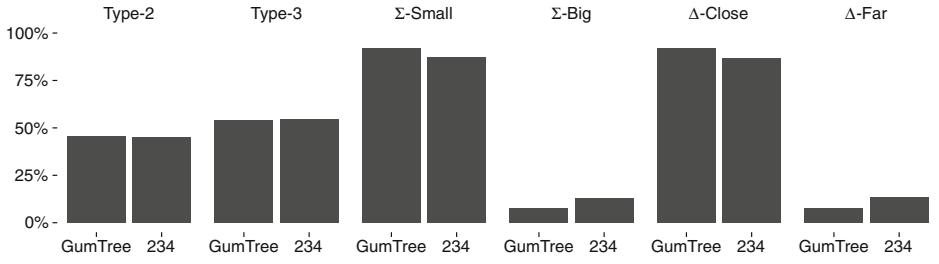
Based on our previous experience of rating clones, the number of clones to rate by each participant has been limited to 600. Since we have six clusters, we draw at random 50 clones in each cluster for both projects. However, if a cluster has less than 50 clones, we select all clones of this cluster. This case only happens for GumTree in two clusters that do not contain enough elements: Σ-*Big* and Δ-*Far* with both 17 clones. As a result, we obtain a set of 300 clones for FastR and another of 234 clones for GumTree, amounting to 534 clones to rate.

In addition to the *type*, *size* and *distance* factors, we also evaluate the influence of the expert opinion on the clones. Therefore we also compute the union of all clusters for each project. We call this union *general* and we use this sample to evaluate the effect of the two aforementioned factors. Since cluster sampling is likely to produce biased samples when clusters are joined, we plot the distribution of the clones of our samples against the clones of the whole project in Fig. 2. As reported in Table 1, FastR contains 49,911 clones and GumTree 216. For GumTree, we can see that clusters Σ-*Big* and Δ-*Far* have very few elements (less than 17 %, corresponding to 50/300). Additionally, one can see that each project and its associated sample have a similar distribution of clones according to the six clusters.

Since clones of each cluster are randomly selected from the whole set of clones identified by iClones for a project, duplicates may appear. In particular, since the number of clones drawn in GumTree was greater than the total number of clones, duplicate clones were mandatory. This was done on purpose to evaluate the repeatability of rater judgments



(a) FastR and our sample of 300 clones



(b) GumTree and our sample of 234 clones

Fig. 2 Distribution of clones in each cluster for each project and its associated sample

as stated in **RQ2**. One author of the paper, Alan Charpentier, has checked for duplicates in the set of 534 clones. As a result, there are 65 duplicates in the 234 clones from GumTree and 1 in the 300 clones from FastR. In other words, there are in fact 163 unique clones for GumTree and 299 for FastR. As previously explained, this author did not warn the other raters of the presence of duplicates in the set of clones to rate. Thus, only Alan Charpentier knew that there were duplicates. Figure 3 shows the distribution of the duplicates from GumTree in the clusters. One can notice that this distribution is similar to the general distribution shown in Fig. 2.

4.4 Data collection

In this section, we first describe the procedure we use to gather judgments from raters and then we present the raw results we obtained.

4.4.1 Procedure

Answers are collected through a web interface. This interface displays the two files involved in the clone, highlights in yellow the fragments belonging to the clone in each file, and shows a text diff in orange inside both fragments. To avoid rater fatigue, the web interface is able to save and restore the work session, and to go back to already processed clones to change the answer. It therefore enables the raters to process their clones taking as much time as they need. Raters were able to complete the survey in several days, splitting their work as they wished. The order of the clones has been randomized, and they are presented to each rater in the same order. The question we ask to the participants is the following: "*Is this clone useful for refactoring or co-evolution purpose*". To avoid to forcing the rating of clones, the external raters were allowed to answer *unknown* to the question, meaning that they do not have a judgment. The participants were forbidden to exchange about the clones and the experiment until it was completed by everybody. During the clone rating process, only Alan Charpentier had knowledge of the research questions and how the clones were selected. Other raters did not know the research questions.

4.4.2 Results

Every participant went through the 534 clones. For each project, additionally to the answers from the expert and the external raters, we also compute a majority vote that aggregates the votes from the external raters. To compute this majority vote we use the following rules. If

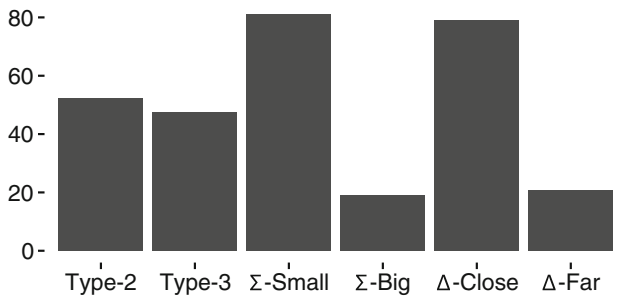


Fig. 3 Distribution in each cluster of the 65 duplicates in GumTree sample

Fig. 4 Ratios of *yes* and *no* answers given by the expert on the clones of each project

two external raters have answered *yes* (resp. *no*) the majority vote is *yes* (resp. *no*). In any other cases, the majority vote is *unknown*.

Results of this experiment for the experts of each project are shown in Fig. 4. One can see ratios of true and false positive clones found by the expert of each system. For each project, ratios are given for each of the six clusters. Overall, the FastR expert finds a slightly greater number of true clones than the GumTree expert, but the mean ratio of true clones is in the same range for both projects.

Both experts seem to agree that the Σ-*Big* cluster contains mostly true positive clones. Additionally, the Δ-*Far* cluster seems to contain also a higher ratio of true clones than the other clusters for both experts. All other clusters contain more false positives than true positives. In this study, we consider expert answers as references, that is to say when an expert answers *yes* (resp. *no*) we have a true positive or true clone (resp. a false negative or false clone).

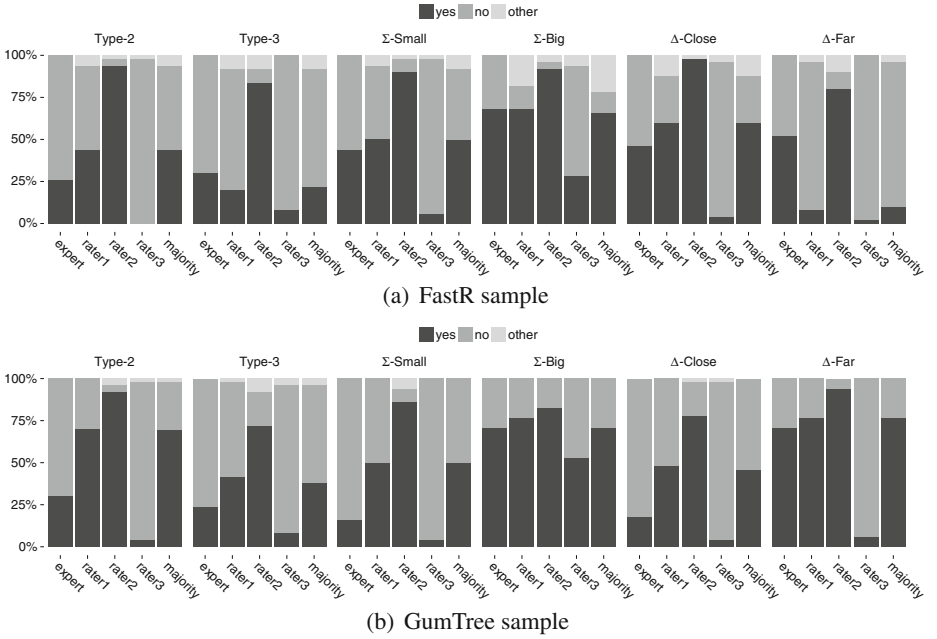
The external raters have different behaviors. Their answers are shown in Fig. 5. For the FastR project, rater 1 and rater 3 are pessimistic about the clones while rater 2 is optimistic. The Σ-*Big* and the Δ-*Close* clusters are the only ones in which a majority of external raters finds more true positives than false positives. For the GumTree project, raters 1 and 2 are much more optimistic than rater 3. The Σ-*Big* cluster is the only one in which all raters find a majority of true clones. In this project, the majority vote has a significant ratio of true positives in the *Type-2* and Δ-*Far* clusters.

5 Results and discussion

In this section, we first present the results of our study according to each research question and provide some guidelines to make the construction of context-dependent clone benchmarks more reliable. We provide the feedback from the raters collected in a discussion session organized after the experiment.

5.1 Repeatability of raters answers (RQ1)

The first research question is related to the consistency of rater answers. By consistent, we mean that for a given rater, the same clone always receive the same judgment. To evaluate this phenomenon we assess if there is a change of behavior of raters across time. We postulate that since the order in which clones are presented to the raters is randomized, the ratio



(a) FastR sample

(b) GumTree sample

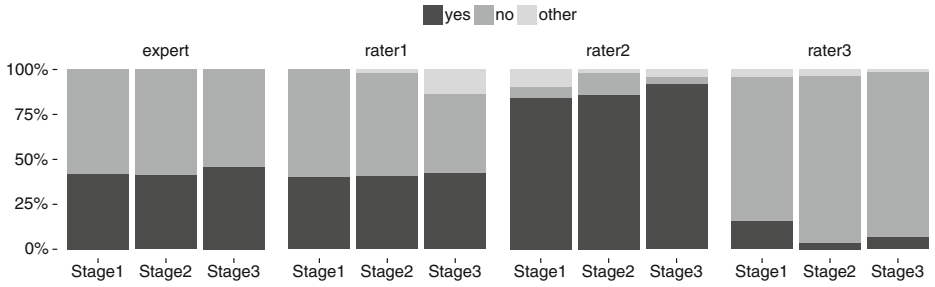
Fig. 5 Ratios of *yes*, *no* and *unknown* answers given by the experts, the external raters and the majority for the clones of each project

of *yes*, *no* and *unknown* answers should stay stable during all the experiment. If it is not the case, it means that a learning effect or a fatigue effect biased the answers of the raters. To quantify this phenomenon we split the clones of each project in three sets. Stage 1 is the first third of the clones rated clones, stage 2 (resp. 3) is the second (resp. third) third of clones. As shown in Fig. 6, there is no significant variation of the ratio between the stages which means that the behavior of raters remains stable across time. However, this does not indicate that a rater will judge consistently the same clone.

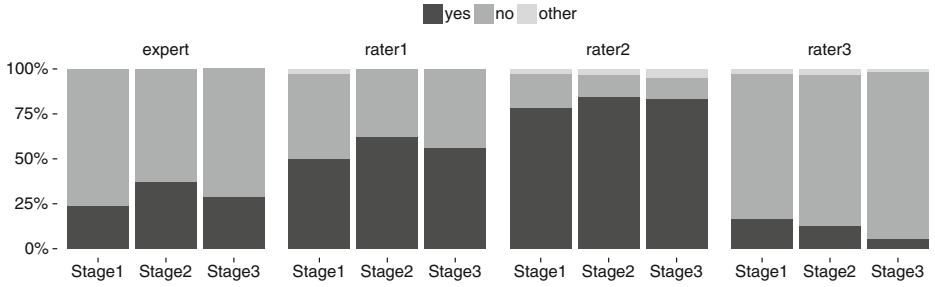
To evaluate the consistency of rater answers, we use the 59 duplicates contained in GumTree clones. First we investigate if raters can change between a *yes*, *no* or *unknown* answer for a same clone. For each rater, we compute the ratio of these clones for which the rater did not give the same judgment. Results depicted in Table 2 show that the behavior of the expert is different from the one of the external raters. The expert gives very consistent answers and only 1 duplicate received a different judgment, which is negligible. While for external raters, the number of inconsistently rated clones varies from about 5 % to about 20 % of the rated duplicates, which is significant. We investigated if there are some trends in the way inconsistent answers evolve across time, however there are not enough inconsistent duplicates to conclude. As a general conclusion on inconsistencies, they may be attributed to some kind of learning effect, hence a training session would be beneficial for external raters.

5.2 Inter-rater reliability

In this section, we investigate the inter-rater reliability of clone judgments. First, we evaluate if several external raters give or not the same judgments about the same clones. Then,



(a) FastR



(b) GumTree

Fig. 6 Rater answers over time (stage 1: first third of the clones, stage 2: second third and stage 3: last third)

we investigate if the external raters give the same judgments as an expert of the project. Finally, we investigate if there are factors that allow external raters to better approximate the judgment of experts.

5.2.1 Agreement among raters (RQ2)

Firstly, we evaluate the agreement between the external raters using Fleiss' Kappa. The result of this measure of an agreement in a group of person is shown in the first line of Table 3. This statistic is interpreted using the thresholds provided by Landis and Koch (1977) shown in Table 4. We can see that in both projects, there is no agreement among the external raters, meaning that they give divergent judgments about the same clones. It therefore seems that judging clones is a very subjective task. Secondly, we evaluate the agreement between each external rater and the expert of both projects. Results are shown in lines 2 to 4 of Table 3. Finally, we evaluate the agreement between the majority vote of external raters (as defined in Section 4.4) and the expert of both projects. Line 5 shows the values obtained

Table 2 Inconsistent answers for the 59 duplications in GumTree

Rater	#	%
expert	1	1.7
rater1	11	18.6
rater2	3	5.1
rater3	7	11.9

Table 3 Agreement among raters

	FastR		GumTree	
Raters	κ	Agreement	κ	Agreement
external raters	−0.12	none	−0.05	none
expert and rater1	0.24	fair	0.42	moderate
expert and rater2	0.09	slight	0.13	slight
expert and rater3	0.16	slight	0.28	fair
expert and the majority	**0.25**	fair	**0.44**	moderate

using Cohen's Kappa to measure the agreement. We observe a different behavior in the two projects. It seems easier for the external raters to agree with the expert in GumTree than in FastR. We also observe that the choice of a particular external rater has a high impact on the agreement with the expert. For instance in FastR the agreement is at worse slight and at best fair, and for GumTree the agreement is at worse slight and at best moderate. Therefore, using only one external rater to judge clones is unreliable. Finally, we can see that using the majority vote of external raters instead of using a particular external rater is a good strategy for both projects. Indeed the maximum value of the Kappa statistic (bolded in Table 3) is reached in this case for both projects which means that the majority always has a best agreement with the expert than any particular external rater.

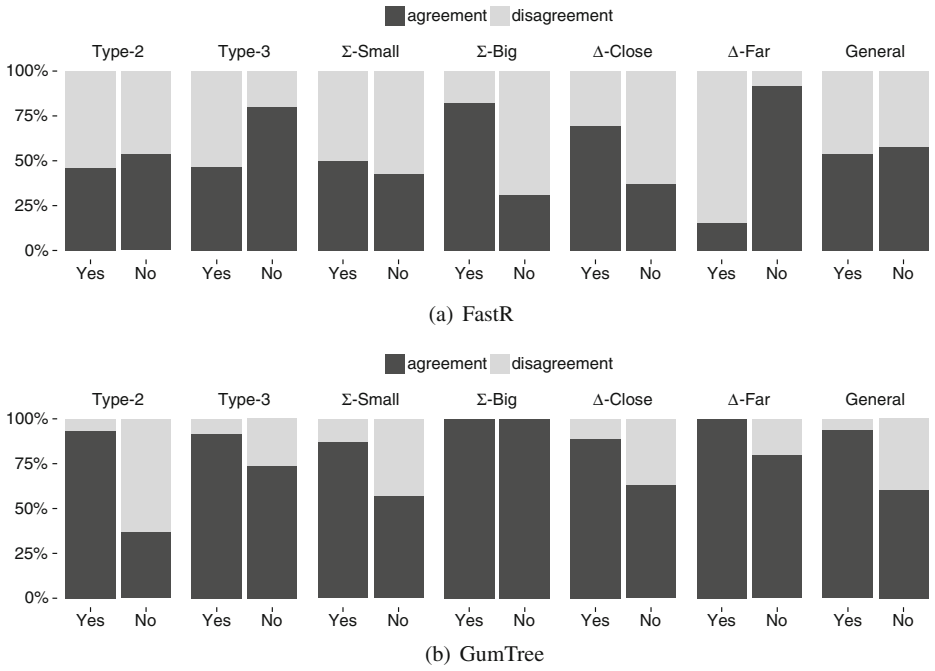
5.2.2 Factors influencing agreement between external raters and experts (RQ3)

In this research question, we investigate if there exists factors that make it easier for external raters to agree with the expert. Since the majority vote of external raters has proven to be the best strategy in the previous section, we only consider for this research question the judgment of the majority. Figure 7 reports raw agreement between the expert and the majority for each characteristic under study. First observations tend to indicate that some characteristics have an impact on the agreement between the majority and the expert (e.g. *size* or *project*). In the remainder of this section, we evaluate statistically the influence of characteristics on the agreement between the expert and the majority. First, we present the effect of clone characteristics. Then, we present the effect of the factors related to the expert judgment on the clones.

Clone characteristics To evaluate the effect of clone characteristics we use two clusters for each characteristic (as explained in Section 4.3). For each cluster we are interested in two numbers: the numbers of identical and different answers with the expert. Therefore, for

Table 4 Kappa statistic interpretation

κ	Interpretation
≤ 0	No agreement
]0, 0.2]	Slight agreement
]0.2, 0.4]	Fair agreement
]0.4, 0.6]	Moderate agreement
]0.6, 0.8]	Substantial agreement
]0.8, 1]	Almost perfect agreement



(a) FastR

(b) GumTree

Fig. 7 Agreement between the expert and the majority

each characteristic, we construct a 2×2 contingency table as shown in Table 5. Our null hypothesis is that the number of identical/different answers with the expert is independent of each characteristic. Our alternative is that this number is not independent. We use the chi-square test to assess our hypothesis. We apply Yates' continuity correction to chi-square computation when the value in a cell is too small i.e., lower than 5. Additionally, we compute the effect size of each characteristic using Cramer's V that is interpreted as shown in Table 6.

For the *type* characteristic, the p-value is 0.07 ($V = 0.19$) for FastR and 0.01 ($V = 0.25$) for GumTree. The p-value for FastR is not significant using a 0.05 ratio. However, the p-value for GumTree is significant, and the effect size moderate. It indicates that this characteristic could have an influence in the GumTree project. For the *size* characteristic,

Table 5 Number of identical and different answers between the expert and the majority

	FastR		GumTree	
Characteristic	Identical	Different	Identical	Different
Type-2	26	24	27	23
Type-3	35	15	39	11
Σ-*Small*	23	27	31	19
Σ-*Big*	33	17	17	0
Δ-*Close*	26	24	34	16
Δ-*Far*	26	24	16	1
General	169	131	164	70

Table 6 Cramer's V interpretation

Cramer' V	Effect size
≤ 0.10	Very weak
]0.10, 0.19]	Weak
]0.20, 0.29]	Moderate
≥ 0.30	Strong

the p-value is 0.04 ($V = 0.20$) for FastR and $7.1e^{-3}$ ($V = 0.37$) for GumTree. The p-value for FastR is significant, and the effect size moderate. The p-value for GumTree is also significant, and the effect size strong. It indicates that this characteristic could have an influence in both projects. One can note that Yate's continuity correction has been used because of the small number of clones in the Σ-*Big* cluster for GumTree. For the *distance* characteristic the p-value is 1 ($V = 0$) for FastR and 0.07 ($V = 0.26$) for GumTree. Both p-values are not significant. Finally, the *size* characteristic is the only one that has an influence on both projects. It is significantly harder for external raters to agree with the expert on small clones. For these clones, using external raters could be unreliable, and thus using an expert of the project seems mandatory.

Project and expert judgment To assess the influence of the project we construct a 2×2 contingency table with the projects as lines, and as rows the number of identical and different answers in the related project, as shown in Table 7. We use a chi-square test, that yields a p-value of $1.1e^{-3}$ ($V = 0.14$). This p-value is significant and the effect is weak. It means that it seems significantly harder to judge the clones of FastR than GumTree in the same way as the expert. Therefore, clones of some projects may not be judged by external raters.

To evaluate if the judgment of the expert about clones has an effect on the number of identical answers with the majority of raters, we use the union of clones of all clusters, denoted as *general*, as explained in Section 4.3. We start by investigating if it is easier to agree with a false or true positive as judged by the expert. To that extent we partition all clones into two sets: clones judged positively by the expert (the *yes* set) and clones judged negatively by the expert (the *no* set). For each set, we count the number of identical/different answers between the majority and the expert of each project. We use these numbers to build a 2×2 contingency table as shown in Table 8. Similarly to the previous section, we use a chi-square test that yields a p-value of 0.49 ($V = 0.04$) for FastR and a p-value of $2.8e^{-7}$ ($V = 0.34$) for GumTree. The p-value for GumTree is significant. It means that it is significantly harder for external raters to judge the true positives than the false positives in GumTree. Indeed, when judging true positives, external raters give more different answers than identical answers with the expert. On the contrary, when judging false positives, external raters give more identical answers than different ones. This indicates that using external raters to judge true positives could be unreliable. For these clones, using an expert of the

Table 7 Number of identical and different answers between the expert and the majority

Project	Identical	Different
FastR	199	101
GumTree	191	43

Table 8 Number of identical and different answers between the expert and the majority for true and false positives as judged by the expert

	FastR		GumTree	
Expert judgment	Identical	Different	Identical	Different
Yes	72	61	64	4
No	97	70	100	66

project seems mandatory. Some mitigations are required as we do not observe the same effect in FastR. We acknowledge the fact that more projects need to be examined to draw more general conclusions.

5.3 Participant feedback

After having conducted the experiment, all the participants have performed a meeting where the important issues concerning the difficulty of rating clones were discussed. These issues are categorized and discussed in the remainder of this section.

5.3.1 Clone rating

Every participant found that it was very difficult to rate the clones in a homogeneous way. Indeed, their judgment evolved as they rated clones (as confirmed in Table 2).

Every participant found that it would have been easier to rate the clones if they were ordered by clone classes. At least it would have improved the homogeneity of the answers.

Two participants found that it would have been easier to rate the clones if they were ordered by context (clones within the same function and the same Java class). They explained that it sometimes takes time to understand the surrounding code of a clone, and it would thus be better to make this effort only once.

Every participant noticed that it was easier to decide if a clone is of interest for refactoring purpose than co-evolution purpose. They motivated their answer by explaining that co-evolution requires a much deeper knowledge of the code. However, they do not always agree on refactoring opportunities. For instance, some of them would apply refactoring operations for a single line while others require a significant amount of code to perform refactoring.

5.3.2 Clone detection tool

Alignment Every participant had trouble with some clones detected by iClones that are not aligned with the structure of the code (for instance a clone that starts in a function and ends in another). This is caused by the fact that iClones is a token-based detection tool. All the participants discarded several clones that could have been useful if better aligned to the code structure.

Extension Every participant also remarked that iClones computed some clones that could have been extended easily. Experts and non-experts had trouble to rate these clones. Each participant judged these clones consistently according to its own strategy. Some decided to keep the clones, while several others decided to discard them.

The participants identified all together 43 clones that could have been extended, 32 for FastR and 11 for GumTree, amounting to 8 % of the rated clones. Results about inter-rater

reliability (Section 5.2 page 13) have been recalculated from the set of 491 clones (534 − 43) corresponding to the ones that do not require to be extended. No significative changes have been observed. Hence, our results are not affected by these clones. As a consequence, during the construction of a clone benchmark, raters should be able to modify the boundaries of a clone candidate before its classification. One can note that Bellon used this good practice in its benchmark construction (Bellon et al. 2007).

To conclude, it seems that a clone detector should both, increase the size of a clone to the maximum according to the code structure, then shorten clones to align their boundary on the boundary of the code. This however requires either a language dependent clone detector tool or at least a post-processing step.

5.3.3 Programming language

Two participants noticed that the features of the programming language had sometimes a big impact on the relevance of a clone. For instance, static typing of the Java language makes some clones irrelevant, but these clones would have been of great help in a dynamically typed language. These was particularly emphasized on FastR which requires type inlining due to Java boxing.

Two participants found that the visitor pattern induces the presence of a lot of clones which are hard to rate.

6 Threats to validity

We have identified the following threats to validity of our study.

6.1 Construct validity

The main threat to construct validity is related to the process of building the clusters of clones, from which we do sampling. There could be some influence between the clusters, i.e., a big clone can have more odds to be a Type-3 clones. This could bias the result of the influence of the *type*, *size* and *distance* characteristics. However, we did not observe any influence of these characteristics. Similarly, the effect of the expert judgments is based on the general set of clones as explained in Section 4.3. Since this set is the union of all the clusters, and not a random sampling of all clones, it may be biased. However, we did show that the distributions of the characteristics of these clones is similar to the ones of all clones.

6.2 Internal validity

Our empirical evaluation bears several threats to internal validity. First, raters may talk to each other during the evaluation process, and influence their judgments. Since participants are authors of the paper, they agreed beforehand not to talk about the survey until all have completed it. A second threat is that all the raters are authors of the paper, and this could influence their answers. To minimize this threat, we make all the data collected during the experiments publicly available for examination and replay (see footnote of Section 4). Another threat is related to the process of constructing our clusters. We minimize this threat

by using a well known clustering algorithm (*neural gas*) to extract the clusters for each characteristic. This clustering algorithm has been selected because it has the reputation to be very stable (it always finds the same clusters if run multiple times). The number of samples we draw within each cluster may also not be sufficient to be representative in some clusters (especially the one with a huge population such as Σ-*Small* or Δ-*Close*). Unfortunately, manual clone inspection is very expensive and we were limited by the total number of clones a rater can assess. One of the external raters of GumTree has already submitted several patches to this project, and this could have influenced his agreement with the expert. However he was not considering himself as an expert of the project, as his intervention in the code is very limited. Another threat is related to the removal of comments in the two projects under study. Non-experts might have a more difficult time judging the clones. This threat should be addressed in a replication study in order to complete our results. Thus, the impact of comments in such a study could be evaluated.

6.3 External validity

Threats to external validity refer to the generalization of our findings. First, we have focused this study on two projects which definitely may not represent all real-world projects. However, the number of projects that can be investigated is limited by the necessity of having one expert in the set of raters, and by the number of clones a rater can assess. Additionally, both projects use Java as the main language. Therefore, the results could be different in projects in other programming languages. Nevertheless, Java is recognized as one of the most popular programming languages used in software projects (Bissyandé et al. 2013). Moreover, Walenstein et al. also find that system specifics appear to have an impact on inter-rater reliability (Walenstein et al. 2003). Further validation on more projects written in other programming languages should be performed. In this way, we provide all necessary data to replicate this study and complete the findings (see footnote of Section 4). A second threat is that we only rely on four raters and the results may change with more raters. Although each selected project has a different expert, our findings may not be generalizable to other open source projects. However, all raters of our study have an extensive Java programming experience and none of them is an undergraduate student. Another threat is related to the use of only one clone detection tool in our study. Once again, the number of clones a rater can assess limits the number of clone detectors that can be considered. Nevertheless, to minimize this threat we rely on both a reputed Type-3 clone detector, iClones, and the work done by Wang et al. (2013) to define the configuration file. The configuration recommended by Wang et al. is such that it maximizes the set of clones that would have been discovered by all main clone detection tools, thus limiting this threat to validity. Additionally, we provide all necessary data to replicate this experiment with other clone detection tools that would be compatible with Java generics. Finally, we investigated only one question about the clones in our study. Our results could change with a different question. However, the question we asked is very representative of the use of code clones by software developers. A last threat is related to the Type-1 clones we ignored in this experiment. In a previous work (Charpentier et al. 2015), we found that developers have a better agreement on Type-1 clones than on other types. Based on this result, we focused on other characteristics, despite no experts were involved in this previous work. Finally, although Type-1 clones should be included to complete the study, our findings are not affected.

7 Conclusion and future work

In this article, we investigate reliability of rater judgments about context-dependent clones. We examine two kinds of raters: external raters and experts. We find that external rater judgment may not be reliable. Firstly, external rater answers repeatability seems not good. This indicates that a training session would be beneficial to increase the repeatability of answers. Secondly, the inter-rater reliability seems poor among external raters, and also between external raters and experts. This is a critical issue since it might lead to benchmarks that do not reflect the judgment of project experts, and thus conduct to unreliable context-dependent clone benchmarks. We show that using the majority vote is one way to mitigate this issue. Finally we have seen that there are several characteristics that appears to have a significant effect on the number of identical and different answers between external raters and experts. First of all, the chosen project and expert has a weak to moderate influence on this matter. It means that some projects yield clones that could not be judged by external raters. Second true positive clones seem harder to judge by external raters than false positive clones. Indeed, this characteristic has a strong influence on the number of identical and different answers. Third, it seems harder for external raters to agree with the expert on small clones. Therefore, it seems mandatory to rely on a project expert to validate either true positives or small clones in a benchmark. While the generalization of certain findings is limited, this study is a first and necessary step in the construction of reliable and high quality context-dependent clone benchmarks. The availability of all data used and collected during this study is heading in this direction.

As future work, we plan to investigate more clone characteristics and their effect on the agreement (for instance the textual similarity between two fragments of a clone). We also plan to ask other questions than the one we use in our study to assess the impact of the question in the agreement level. Finally we want to use our findings to build high quality benchmarks of clones and to design a machine learning based approach that better eliminates irrelevant clones.

References

Baker BS (1995) On finding duplication and near-duplication in large software systems. In: Proceedings of the Second Working Conference on Reverse Engineering, WCRE '95, pp 86–. IEEE Computer Society, Washington, DC, USA

Baxter I, Yahin A, Moura L, Sant'Anna M, Bier L (1998) Clone detection using abstract syntax trees. In: International conference on software maintenance, 1998. Proceedings, pp 368–377. doi:10.1109/ICSM.1998.738528

Bellon S, Koschke R, Antoniol G, Krinke J, Merlo E (2007) Comparison and evaluation of clone detection tools. IEEE Trans Softw Eng 33(9):577–591. doi:10.1109/TSE.2007.70725

Bissyandé TF, Thung F, Wang S, Lo D, Jiang L, Réveillère L (2013) Empirical Evaluation of Bug Linking. In: Proceedings of the 17th european conference on software maintenance and reengineering (CSMR 2013), pp 1–10, Genova, Italy

Charpentier A, Falleri JR, Lo D, Réveillère L (2015) An empirical assessment of bellon's clone benchmark. In: Proceedings of the 19th international conference on evaluation and assessment in software engineering, EASE '15, pp 20:1–20:10. ACM, New York, NY, USA. doi:10.1145/2745802.2745821

Ducasse S, Rieger M, Demeyer S (1999) A language independent approach for detecting duplicated code. In: IEEE international conference on software maintenance, 1999. (ICSM '99) Proceedings, pp 109–118. doi:10.1109/ICSM.1999.792593

Falleri JR, Morandat F, Blanc X, Martinez M, Monperrus M (2014) Fine-grained and accurate source code differencing. In: Proceedings of the international conference on automated software engineering, pp –. Sweden

Faust D, Verhoef C (2003) Software product line migration and deployment. Software Practice and Experience, vol 33. Wiley, pp 933–955

Gode N, Koschke R (2009) Incremental clone detection. In: 13th european conference on software maintenance and reengineering, 2009. CSMR '09, pp 219–228. doi:10.1109/CSMR.2009.20

Ihaka R, Gentleman R (1996) R: a language for data analysis and graphics. J Comput Graph Stat 5(3):299–314

Jiang L, Misherghi G, Su Z (2007) Deckard: Scalable and accurate tree-based detection of code clones. In: ICSE, pp 96–105

Kalibera T, Maj P, Morandat F, Vitek J (2014) A fast abstract syntax tree interpreter for R. In: 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '14, Salt Lake City, UT, USA, March 01 - 02, 2014, pp 89–102. doi:10.1145/2576195.2576205

Kamiya T, Kusumoto S, Inoue K (2002) Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. IEEE Trans Softw Eng 28(7):654–670. doi:10.1109/TSE.2002.1019480

Kapser C, Anderson P, Godfrey M, Koschke R, Rieger M, van Rysselberghe F, Weißgerber P (2006) Subjectivity in clone judgment: Can we ever agree? In: Duplication, Redundancy, and Similarity in Software, no. 06301 in Dagstuhl Seminar Proceedings. IBFI, Schloss Dagstuhl, Germany

Kapser C, Godfrey M (2006) cloning considered harmful considered harmful. In: 13th working conference on reverse engineering, 2006. WCRE '06, pp 19–28. doi:10.1109/WCRE.2006.1

Koschke R, Falke R, Frenzel P (2006) Clone detection using abstract syntax suffix trees. In: Proceedings of the 13th working conference on reverse engineering, WCRE '06, pp 253–262. IEEE computer society, Washington, DC, USA. doi:10.1109/WCRE.2006.18

Krutz DE, Le W (2014) A code clone oracle. In: Proceedings of the 11th working conference on mining software repositories, MSR 2014, pp 388–391. ACM, New York, NY, USA. doi:10.1145/2597073.2597127

Lague B, Proulx D, Mayrand J, Merlo EM, Hudepohl J (1997) Assessing the benefits of incorporating function clone detection in a development process. In: Proceedings of the international conference on software maintenance, ICSM '97, pp 314–. IEEE computer society, Washington, DC, USA. http://dl.acm.org/citation.cfm?id=645545.853273

Landis JR, Koch GG (1977) The measurement of observer agreement for categorical data. biometrics:159–174

Li Z, Lu S, Myagmar S, Zhou Y (2006) Cp-miner: finding copy-paste and related bugs in large-scale software code. IEEE Trans Softw Eng 32(3):176–192. doi:10.1109/TSE.2006.28

Martinetz T, Schulten K (1991) A Neural-Gas Network Learns Topologies. In: Artificial Neural Networks, vol. I, pp 397–402

Mende T, Koschke R, Beckwermert F (2009) An evaluation of code similarity identification for the grow-and-prune model. J Softw Maint Evol 21(2):143–169

Nguyen HA, Nguyen TT, Pham N, Al-Kofahi J, Nguyen T (2012) Clone management for evolving software. IEEE Trans Softw Eng 38(5):1008–1026. doi:10.1109/TSE.2011.90

Roy C, Cordy J (2008) Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In: The 16th IEEE international conference on program comprehension, 2008. ICPC 2008, pp 172–181. doi:10.1109/ICPC.2008.41

Roy C, Cordy J (2009) A mutation/injection-based automatic framework for evaluating code clone detection tools. In: International conference on software testing, verification and validation workshops, 2009. ICSTW '09, pp 157–166. doi:10.1109/ICSTW.2009.18

Selim G, Foo K, Zou Y (2010) Enhancing source-based clone detection using intermediate representation. In: 2010 17th Working Conference on Reverse Engineering (WCRE), pp 227–236. doi:10.1109/WCRE.2010.33

Svajlenko J, Islam JF, Keivanloo I, Roy CK, Mia MM (2014) Towards a big data curated benchmark of inter-project code clones. ICSME:5

Walenstein A, Jyoti N, Li J, Yang Y, Lakhotia A (2003) Problems creating task-relevant clone detection reference data. In: 10th working conference on reverse engineering, 2003. WCRE 2003. Proceedings, pp 285–294. doi:10.1109/WCRE.2003.1287259

Wang T, Harman M, Jia Y, Krinke J (2013) Searching for better configurations: A rigorous approach to clone evaluation. In: Proceedings of the 2013 9th joint meeting on foundations of software engineering, ESEC/FSE 2013, pp 455–465. ACM, New York, NY, USA. doi:10.1145/2491411.2491420

Yang J, Hotta K, Higo Y, Igaki H, Kusumoto S (2014) Classification model for code clones based on machine learning. Empir Softw Eng:1–31

Alan Charpentier is currently a Ph.D. student in the software engineering research group at the University of Bordeaux. He received his MSc. degree in software engineering from the University of Bordeaux in 2013. His research interests focus on software engineering, and especially code duplication.

Jean-Rémy Falleri is currently associate professor at the Bordeaux Institute of Technology and head of the software engineering research group of the LaBRI laboratory. He received his Ph.D. degree in 2009 from the University of Montpellier 2 and his habilitation in 2015 from the University of Bordeaux. His research interests focus on software engineering, and especially software evolution.

Floréal Morandat is a member of the software engineering group of Bordeaux and holds a position of associate professor at the Bordeaux Institute of Technology since 2012. He received his Ph.D. degree from the University of Montpellier 2 in 2010. His research interest is focused on programming languages, from specification to implementation.

Elyas Ben Hadj Yahia received his MSc. degree in software engineering from the University of Bordeaux in 2014. He is currently a Ph.D. candidate at the University of Bordeaux, and member of the software engineering research group of the LaBRI laboratory. His research interests span across software engineering, web services composition, web development and distributed systems.

Laurent Réveillère received the PhD degree from the University of Rennes I, in 2001 and the Habilitation a Diriger des Recherches degree from University of Bordeaux, in 2011. Currently, he holds a position of associate professor at Bordeaux INP, France. His research interests are in the area of improving the quality of distributed system code, using a variety of approaches including program analysis, program transformation, and the design of domain-specific languages.