An Empirical Assessment of Bellon's Clone Benchmark

Alan Charpentier
University of Bordeaux
LaBRI, UMR 5800
F-33400, Talence, France
acharpen@labri.fr

Jean-Rémy Falleri
University of Bordeaux
LaBRI, UMR 5800
F-33400, Talence, France
falleri@labri.fr

David Lo
School of Information Systems
Singapore Management University
davidlo@smu.edu.sg

Laurent Réveillère
University of Bordeaux
LaBRI, UMR 5800
F-33400, Talence, France
reveillere@labri.fr

ABSTRACT

Context: Clone benchmarks are essential to the assessment and improvement of clone detection tools and algorithms. Among existing benchmarks, Bellon's benchmark is widely used by the research community. However, a serious threat to the validity of this benchmark is that *reference* clones it contains have been manually validated by Bellon alone. Other persons may disagree with Bellon's judgment. **Objective:** In this paper, we perform an empirical assessment of Bellon's benchmark. **Method:** We seek the opinion of eighteen participants on a subset of Bellon's benchmark to determine if researchers should trust the reference clones it contains. **Results:** Our experiment shows that a significant amount of the reference clones are debatable, and this phenomenon can introduce noise in results obtained using this benchmark.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; D.2.8 [**Software Engineering**]: Metrics

General Terms

Experimentation, Measurement

Keywords

Code clone, Empirical study, Software metrics

1. INTRODUCTION

A *clone* is a fragment of code that is similar or identical to another fragment of code in the same piece of software. Clones 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 a similar intent, and possibly modifying it slightly according to its new context. Cloning is a common practice in software systems: several studies report 5 to 23 percent code duplication in such systems [1, 2, 13]. Cloning code can complicate code maintenance and evolution, as fault fixes and evolutions must be propagated from one instance of a clone to the others.

The desirability of identifying, and sometimes eliminating, clones has led to the development of *clone detectors*, which scan a code base for potential clones [7, 8, 9, 14, 20]. One of the biggest challenges for an effective application of clone detection in practice is to identify relevant clones for the user of the tool. However, since clone definition is open to interpretation, some clones reported by a clone detector may be viewed as *false positives*. Identifying false positives is challenging considering the large amount of data that a clone analysis of a large system generates.

A traditional approach to reduce the number of false positives is to enhance the clone detection algorithm and finetune the detection tool. Several benchmarks of true clones have been proposed, enabling tool developers to compare and assess results of their tools. The construction of these benchmarks is usually performed by a single person, deciding herself which clone is or not a true positive. As a consequence, one could ask to which extent this way of building benchmarks is effective?

Bellon's benchmark [3] has emerged as the main benchmark to compare and evaluate clone detectors [5, 11, 18, 25]. Bellon built a *reference corpus* of 4,319 reference clones, by looking at 2% of all 325,935 clones identified by six clone detectors on eight large C and Java programs. Clone detectors' results are compared to the reference corpus, and assessed using the traditional information retrieval measures: precision and recall.

However, a serious threat to the validity of this reference corpus is that it has been constructed by only one person, namely Bellon. Since Bellon is not at all an expert of the software systems containing the clones, and since clones are very subjective, it is possible that other persons would have a different opinion about some of the clones of the reference corpus. This phenomenon could therefore bias the precision and recall measures computed using this benchmark.

In this paper, we perform an empirical assessment of the reference corpus of Bellon. We seek the opinion of eighteen additional persons on a subset of Bellon's benchmark. We

EASE '15, April 27 - 29, 2015, Nanjing, China

http://dx.doi.org/10.1145/2745802.2745821.

investigate the following research questions:

1. *Can researchers trust the clones from Bellon's reference corpus?*
2. *Are the effectiveness measures computed using Bellon's benchmark reliable?*
3. *Are there some characteristics of reference clones that make them more trustable?*

Our contributions are as follows:

1. We perform a controlled experiment with nine groups of two participants. For each group, we randomly select 120 clones from Bellon's reference corpus and ask the two participants if the clones are true clones or not.
2. We analyze if there are clones from the reference corpus for which participants and Bellon have different opinions.
3. We examine if these reference clones that do not yield consensus among new raters and Bellon can have an impact on recall and precision values computed using the benchmark.
4. We evaluate if some clone characteristics can be associated to a higher trust level.

The remainder of this paper is organized as follows. Section 2 presents Bellon's benchmark and our research problem. Next, we describe our empirical study methodology in Section 3. Section 4 presents the findings of our controlled experiment. We describe related work in section 5. We discuss the threats to the validity of our study in Section 6. Finally, we conclude and mention future work in Section 7.

2. BACKGROUND

In this section, we present Bellon's benchmark and the research problem we explore in this study.

2.1 Bellon Benchmark

Bellon's benchmark has been constructed by Bellon *et al.* [3] as a tool to compare and evaluate clone detectors. Several researchers have participated to the benchmark construction by applying their own clone detector on the same programs and providing results to Bellon. We report below the clone definition used in the article of Bellon *et al.*.

Definition 1. A clone (pair) is a triple (f_1, f_2, t) where f_1 and f_2 are two similar code fragments and t is the associated type of similarity (type 1, 2 or 3).

They used the following definition of a code fragment.

Definition 2. A code fragment is a tuple (f, s, e) which consists of the name of the source file f, the start line s, and the end line e of the fragment. Both line numbers are inclusive.

They distinguished three types of similarity:

- Type 1 is an exact copy without modifications (except for white space and comments).
- Type 2 is a syntactically identical copy; only variable, type, or function identifiers were changed.
- Type 3 is a copy with further modifications; statements were changed, added, or removed.

Additionally, Bellon *et al.* require that clones are pairs of code fragments that could be replaced by function calls, meaning that they must be syntactically complete (for instance a code fragment starting in the middle of a method and finishing in the middle of another one cannot be part of a clone). Also, code fragments of a clone must contain at least six lines of code.

Table 1: Participants of the Bellon study.

Participant	Tool	Comparison
Brenda S. Baker	Dup	Token
Ira D. Baxter	CloneDR	AST
Toshihiro Kamiya	CCFinder	Token
Jens Krinke	Duplix	PDG
Ettore Merlo	CLAN	Function Metrics
Matthias Rieger	Duploc	Text

2.1.1 Benchmark Construction

Six researchers helped Bellon to construct the benchmark. Each of them applied its own clone detector on eight large C and Java programs. Table 1 shows the clone detector used by each researcher and table 2 reports the number of submitted clones in the eight programs.

Bellon examined 2 percent of all 325,935 submitted clones and built a *reference corpus* by retaining only clones he judged as true positives. Sometimes he modified clones before inserting them, by extending their code fragments. He did not know which tool provided the clones he was examining.

It took 77 hours to classify the clones. Table 2 reports for each program the number of clones he retained in the reference corpus. According to Bellon judgment, the reference corpus contains 4,319 true clones. In the remainder, each clone identified by a clone detector is called *candidate* and each clone of the reference corpus is called *reference*.

2.1.2 Benchmark Use

Bellon's benchmark provides a set of true clones — the reference corpus — to evaluate and compare the results of clone detectors. The evaluation is a two-step process. First a mapping between candidate clones, found by a clone detector, and reference clones is produced. Then the recall and precision effectiveness measures are computed.

Bellon *et al.* define a methodology, based on the amount of overlap between a candidate and a reference clone, to establish mappings from the candidate to the reference clones. Then, Bellon *et al.* compute the set $DetectedRefs$ of candidate clones having a matching reference clone. Similarly, they compute the set $RejectedCands$ of candidate clones having no matching references.

We report below definitions of recall and precision used by Bellon *et al.* to evaluate clone detectors. Recall is the number of clones of the reference corpus of type τ detected by tool T in program P relative to all clones of the reference corpus for program P with type τ.

Definition 3.

$$Recall(P, T, \tau) = \frac{|DetectedRefs(P, T, \tau)|}{|Refs(P, \tau)|}$$

Precision is the number of clones of the reference corpus of type τ detected by tool T in program P relative to all clones identified for program P by tool T with type τ.

Definition 4.

$$Precision(P, T, \tau) = \frac{|DetectedRefs(P, T, \tau)|}{|Cands(P, T, \tau)|}$$

Table 2: Number of Submitted, examined and reference clones.

Program	Submitted clones	Examined clones	Reference clones	Yield (in %)
weltab (C)	13,901	280	252	90.00
cook (C)	27,122	544	402	73.90
snns (C)	66,331	1,329	903	67.95
postgresql (C)	59,114	1,182	555	46.95
netbeans-javadoc (Java)	7,860	159	55	34.59
eclipse-ant (Java)	2,440	51	30	58.82
eclipse-jdtcore (Java)	92,905	1,856	1,345	72.47
j2sdk1.4.0-javax-swing (Java)	56,262	1,127	777	68.94
Total	325,935	6,528	4,319	66.16

Since the number of clones inserted in the reference corpus for a program is only a small percentage of all submitted candidates for this program[1], the precision calculated with this definition is a lower bound of the real precision.

Hence Bellon *et al.* measure also the ratio of rejected clones as shown in the definition below. In this definition $OracledCands$ corresponds to the 2 percent of all submitted clones that Bellon manually examined. Table 2 reports the number of examined clones in the eight programs.

Definition 5.

$$Rejected(P,T,\tau) = \frac{|RejectedCands(P,T,\tau)|}{|OracledCands(P,T,\tau)|}$$

2.2 Research Questions

As explained in the previous section, recall and precision measures computed using Bellon's benchmark strongly depend on the reference corpus. Indeed, in the two formulae we report above, $DetectedRefs$ is computed from the clones of the reference corpus. Therefore, the reference corpus is a critical part in the comparison and evaluation of clone detectors using Bellon's benchmark.

However, a threat to the validity of this reference corpus is that it has been constructed by only one person, namely Bellon. Other persons could construct a different reference corpus using the same clone definition. In this respect, Bellon *et al.* report rightly in their article that their results rely on the judgment of Bellon [3], hence we wonder whether we can trust these reference clones. To what extent other persons would agree that the reference clones are indeed true clones?

In this study, we explore the following research questions to assess Bellon's reference corpus:

RQ 1. *Can researchers trust the clones from Bellon's reference corpus?* To answer this research question, we seek the opinion of eighteen participants on a subset of Bellon's reference corpus and evaluate if they all judge the clones as true clones.

RQ 2. *Are the effectiveness measures computed using Bellon's benchmark reliable?* To answer this question, we distinguish the clones that yielded a consensus among Bellon and the new raters from the others, and evaluate the effect it could have on precision and recall measures.

RQ 3. *Are there some characteristics of reference clones that make them more trustable?* To answer this question, we further investigate if there exists certain kinds of clones which are always evaluated as true clones by Bellon and the new raters.

[1]One can see this difference for each program in the last column of table 2.

3. EXPERIMENTAL SETUP

We describe in this section the experiment we conduct to assess Bellon's benchmark. We assume that a reference clone can be trusted if anyone that is presented the clone won't doubt that it is a true clone. Therefore we present a subset of the reference clones from Bellon's benchmark to additional persons and gather their opinions on the clones. In the remainder of the section, we describe the participants of this study, how we selected a subset of Bellon's reference clones, how we gather the opinion of the participants and how we affect a trust level to the clones. All data used and collected during this study is available online[2].

3.1 Participants

We ask 18 students to examine the reference clones. Four are graduate students and fourteen are undergraduate students. Three graduate students come from the Singapore Management University, and 1 from the University of Bordeaux. The fourteen undergraduate students come from the Telecommunication's department of the Polytechnic Institute of Bordeaux. They are in the last year of their degree. All students have an IT background and have been trained in Java and C programming languages.

3.2 Clone Selection and Examination

We perform an informal experiment using a member of our research group to estimate how many clones provide to each participant. We find that 120 clones can be examined in two hours, without having to hurry and without being to tired or bored of the experiment.

In this experiment, we want to assess if multiple persons have the same opinion on the reference clones, therefore we need to have at least two opinions on each reference clone. Hence we randomly selected $1,080$ clones ($18 \times 120 \div 2$) from the $4,319$ clones of Bellon's reference corpus. Thus, the participants evaluate approximately 25% of the reference clones. The $1,080$ clones are then randomly split in nine groups of 120 clones, and each group is examined by a pair of two randomly drawn students.

Undergraduate students make the experiment in a room of the Polytechnic Institute of Bordeaux. We consecrate a time slot of 3 hours to ensure they can rate all the clones. Two authors of the paper supervise the session and guarantee

[2]www.labri.fr/perso/acharpen/ease15/study.zip

students do not communicate with each other. Conversely graduate students make the experiment on the web. We inform them through mails to perform the experiment in 3 hours and to not communicate with each other during this time.

3.3 Opinion Collection

In this study, we ask participants to rate a group of reference clones and then to answer anonymously a questionnaire about the clones they examined. A key step of our experiment consists of asking participants to rate the clones. For that purpose, we design a web site to perform an online survey. For each clone, the participant is asked to rate *yes* for clones she deems as true clones and *no* for false clones. The answers are collected through the web interface shown in Figure 1.

For each clone, the web interface displays the two files involved in the clone, and highlights in yellow the code fragments belonging to the clone in each file. Differences between both fragments are shown in orange. The web interface is able to save and restore the work session, and to go back and forward in the set of rated clones so as to update the answer.

We conduct a controlled experiment with the students. At the beginning of the clones classification session, we describe the clone definition used by Bellon *et al.* and indicate to the students that they can refer to it during the rating session. However we do not tell the students that the clones they are rating were judged as true clones by Bellon. Instead we tell them that they are random clones detected using a clone detector. Then we explain in details how to use the web interface. At the end, we provide a questionnaire to all students to retrieve their feelings on the experiment. Every student went through the 120 clones and answered the questionnaire. As a result, we obtain a set of $1,080$ clones with three judgments.

3.4 Trust Level of Clones

After having gathered all the answers of the participants, we have a set of $1,080$ clones with 3 opinions each. The first opinion is a positive opinion from Bellon, and the two others have been given by the participants of the experiment and can be positive or negative. Using these three opinions, we affect a trust level to each clone. A clone with three positive opinions has a *good* trust level. A clone with two positive opinions has a *fair* trust level, because it has more positive than negative opinions. Finally, a clone with only one positive opinion has a *small* confidence level as it has more negative than positive opinions. These three values define the following ordinal scale: $small < fair < good$.

4. RESULTS AND DISCUSSION

In this section, we first present the results answering each of our research questions. Secondly, we provide a discussion about the feelings of participants of the experiment.

4.1 RQ1: Trust Level of Reference Clones

Our first objective is to investigate the trust level of clones from Bellon's reference corpus. Figure 2 shows the number of clones having a *small*, *fair* and *good* trust level as computed for each of the nine groups of participants. As we can see, apart from group 1, 4 and 8, the groups seem to have a fairly similar ratio of clones with a given trust level.

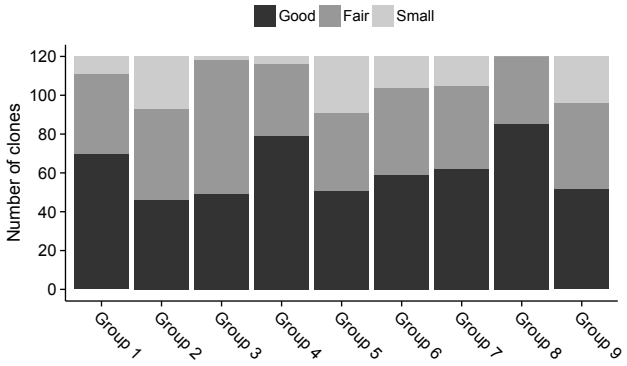

Figure 2: Trust level of clones according to Bellon and students of each group.

Table 3: 95% confidence intervals of the proportions of clones with *small*, *fair* and *good* trust level.

	Confidence interval	
Trust level	**Lower bound**	**Higher bound**
Small	0.10	0.14
Fair	0.34	0.40
Good	0.48	0.54

The first observation is that most of the groups have more than 50% of clones with a trust level lesser than *good*, which is a significant proportion. Concretely, it means that about only half of the reference clones of Bellon have a *good* trust level. The second observation is that, apart from groups 3, 4 and 5, there are between 10 and 20 % of clones with only a *small* trust level, which is also a significant number.

As an example, we provide in Figure 3 a clone having a *good* trust level, that is a clone judge as true positive by two participants and in Figure 4 a clone having a *small* trust level, that is a clone judge as true negative by two participants.

To estimate the proportion of clones with *small*, *fair* and *good* trust level in Bellon's reference corpus, we use bootstrapping [6] on our sample of $1,080$ clones. We compute the 95% confidence intervals for these ratios using $5,000$ replicates. The results are shown in Table 3. As we can see, there is between 48 and 54% of the reference clones with a *good* trust level (about the half). There is between 34 and 40% of the reference clones with a *fair* trust level, and between 10 and 14% of the reference clones with a *small* trust level.

> There is a significant number of reference clones that are debatable. About half of the clones have less than a *good* trust level. Between ten to fifteen percent of the clones have only a *small* trust level.

4.2 RQ2: Trust Level and Effectiveness Measures

Since we have a set of clones rated by three persons, we can examine two scenarios. Firstly, a clone is considered as a reference if it has at least a *fair* trust level. Secondly, a clone is considered as a reference only if it has a *good* trust

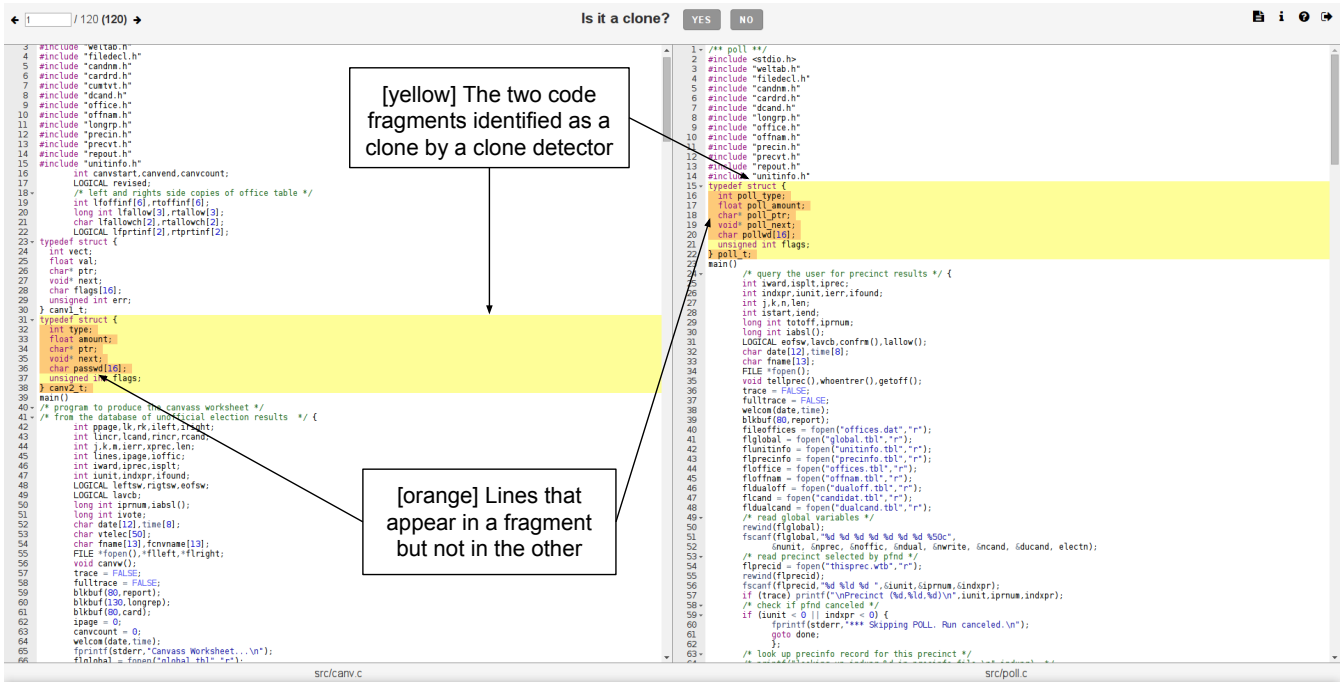

Figure 1: Web interface to gather answers about the clones.

Listing (1) src/ant/taskdefs/compilers/Jikes.java

```
119  if (debug) {
120    cmd.createArgument().setValue("-g"); }
121  if (optimize) {
122    cmd.createArgument().setValue("-O"); }
123  if (verbose) {
124    cmd.createArgument().setValue("-verbose"); }
```

Listing (2) src/ant/taskdefs/compilers/Jvc.java

```
105  if (debug) {
106    cmd.createArgument().setValue("/g"); }
107  if (optimize) {
108    cmd.createArgument().setValue("/O"); }
109  if (verbose) {
110    cmd.createArgument().setValue("/verbose"); }
```

Figure 3: A clone detected in eclipse-ant having a *good* trust level.

Listing (3) src/backend/nodes/copyfuncs.c

```
1170  static ResTarget *
1171  _copyResTarget(ResTarget *from) {
1172    ResTarget  *newnode = makeNode(ResTarget);
1173    if (from->name)
1174      newnode->name = pstrdup(from->name);
1175    Node_Copy(from, newnode, indirection);
1176    Node_Copy(from, newnode, val);
1177    return newnode; }
```

Listing (4) src/backend/nodes/copyfuncs.c

```
1707  static AlterGroupStmt *
1708  _copyAlterGroupStmt(AlterGroupStmt *from) {
1709    AlterGroupStmt *newnode = makeNode(AlterGroupStmt);
1710    if (from->name)
1711      newnode->name = pstrdup(from->name);
1712    newnode->action = from->action;
1713    Node_Copy(from, newnode, listUsers);
1714    return newnode; }
```

Figure 4: A clone detected in postgresql having a *small* trust level.

Table 4: Impact on recall and precision of trust level of Bellon's reference clones.

t detects exactly clones of	Precision and recall derived from		
	B	F	G
B		$p = 0.88$	$p = 0.51$
F	$r = 0.88$		
G	$r = 0.51$		

level. In any case, we consider that clones with only a *small* trust level are not likely to be reference clones.

As a consequence, we build two new reference corpora: the reference corpus F of clones having at least a *fair* trust level, and the reference corpus G of clones having a *good* trust level. According to the students' answers, F contains 954 clones and G contains 553 clones. Additionally, B represents the set of all clones from Bellon's reference corpus examined by the participants. By construction, we have $G \subset F \subset B$.

To assess if the trust level of Bellon's reference clones can modify the precision and recall values, we use a *worst-case approach*. The principle is simple, we assume that a fictive tool t finds exactly the clones of Bellon's reference corpus. It has therefore a precision and recall of 1. Then we assume that in fact the reference corpus should be F or G (because we want to take the trust level into account), and we compute the decrease of precision in both scenarios (since F and G are subsets of B, recall will not change). Conversely, we assume that t finds exactly the clones of F or G. Therefore it has a precision and recall of 1. Then we assume that in fact the reference corpus should be B, and we compute the decrease of recall (since F and G are subsets of B, precision will not change).

Table 4 reports the results of our approach and can be interpreted as follows. If a clone detection tool identifies exactly the clones of Bellon's reference corpus, its recall and precision are no longer equals to 1 but either 0.88 or 0.51, depending if we use respectively clones with at least *fair* or *good* trust level. The precision and recall decrease are therefore 0.12 or 0.49 respectively.

> Precision and recall can be significantly modified by the trust level of reference clones. When requiring a *good* trust level, a precision and recall decrease of up to 0.49 is possible. When requiring only a *fair* trust level, a precision and recall decrease of up to 0.12 is possible. Therefore, results from tools comparison within this range of these differences have to be interpreted with caution.

4.3 RQ3: Trust Level and Clone Characteristics

In research question 3, we investigate if there exists certain kinds of clones for which it is more likely to have a *good* trust level. We study three characteristics of the clones: *Type*, *Size* and *Language*. Clone types are defined in section 2. The size of a clone is the sum of the number of lines of its two code fragments. Finally, *Language* is the programming language of the program in which a clone has been identified. Programs from Bellon's benchmark are programmed using C or Java, as reported in table 2.

The percentages of clones having a given *Type*, *Size* and *Language* are shown in figures 5, 6 and 7 respectively. In these figures, the first column represents the 4,319 clones of Bellon's reference corpus. Then, each group represents a set of 120 clones selected from this reference corpus. The ratio of clones for each characteristic in the nine groups is fairly similar to the ratios in Bellon's reference corpus. One can see that most of the clones have a size lower than 50 lines. Only a few clones have a very big size, that is several hundreds of lines. Moreover there is a vast majority of type 2 clones in Bellon's reference corpus.

To answer research question 3, we assess the impact of these clone characteristics on the trust level as defined in the previous section.

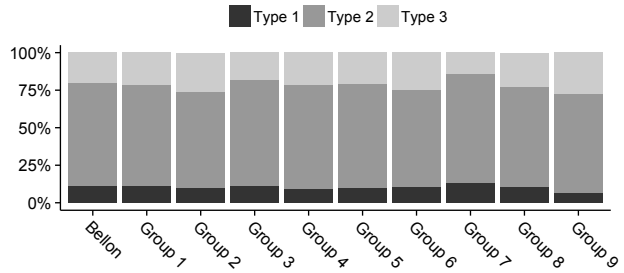

Figure 5: Ratios of clones with a given type.

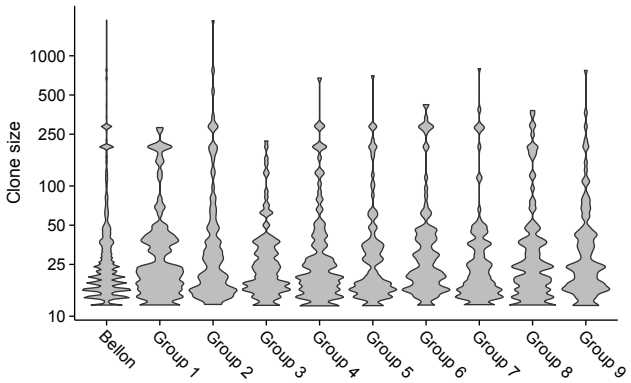

Figure 6: Distribution of clones for size characteristic.

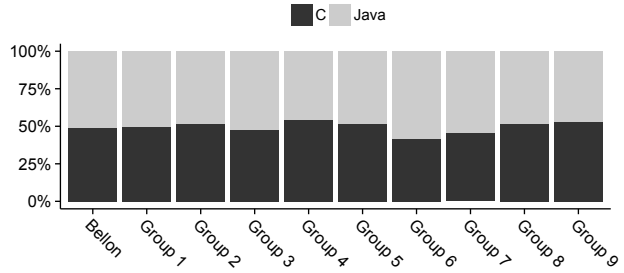

Figure 7: Ratios of clones with a given programming language.

Clone size.

Regarding the association between clone size and trust level, we formulate the following null and alternative hypotheses.

H_0^1: *there is no correlation between a clone size and its trust level.*

H_a^1: *the larger a clone is, the greater its trust level is.*

The alternative hypothesis follows the intuition that very big clones are very likely to be true clones. The trust level according to the clone size is shown in Figure 8. In this figure, it seems that clones with a bigger size have a slightly better trust level.

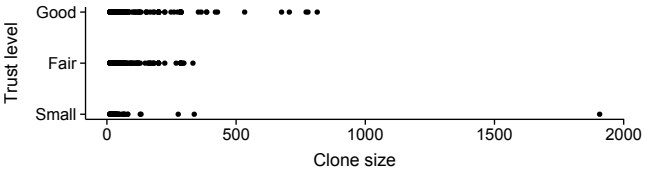

Figure 8: Trust level according to the clone size.

As clone size is measured on an interval scale and the trust level on an ordinal scale, we use a Spearman correlation test. It results in a significant effect: $\rho = 0.06$, $p = 0.03$ (one-tailed). We use bootstrapping to compute a 95% confidence interval for Spearman's ρ, with the following result: $0 \leq \rho \leq 0.12$. Therefore the effect size is very weak. It means that the clone size and trust level are only very loosely associated.

Clone type.

Regarding the association between clone type and trust level, we formulate the following null and alternative hypotheses.

H_0^2: *there is no correlation between a clone type and its trust level.*

H_a^2: *the bigger the type of a clone is, the lesser its trust level is.*

The trust level according to the clone type is shown in the three left bars of figure 9. In this figure, its seems that the type of a clone is associated to its trust level.

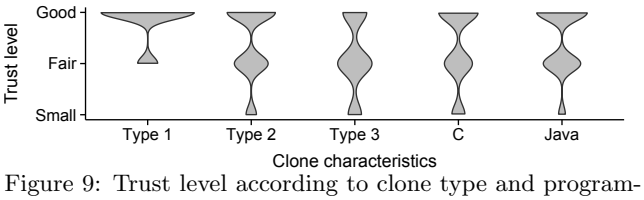

Figure 9: Trust level according to clone type and programming language.

We consider that a clone type is measured on the following interval scale type 1 $\leq$ type 2 $\leq$ type 3. This ordinal scale makes sense because when the type of a clone increase, its definition becomes less restrictive, as explained in Section 2. The alternative hypothesis follows the intuition that clones of identical code fragments have a higher trust level than other clones.

As both variables (type and trust level) are measured on an ordinal scale, we use a Spearman correlation test. It results in a significant effect: $\rho = -0.26$, $p = 0$ (one-tailed). We use bootstrapping to compute a 95% confidence interval for Spearman's ρ, with the following result: $-0.31 \leq \rho \leq -0.21$. Therefore the effect size is moderate and negative. It means that, as we postulated, the more the two code fragments of a clone are similar, the bigger its trust level is. On figure 9 we can see that clones with type 1 have at worse a *fair* trust level, and have very often a *good* trust level.

Programming language.

Regarding the association between programming language of a clone and trust level, we formulate the following null and alternative hypotheses.

H_0^3: *the programming language has no impact on the trust level.*

H_a^3: *the programming language has an impact on the trust level.*

We have no particular intuition about the direction of the effect of the association between programming language and trust level since our participants are trained in both languages. The two right bars of figure 9 show the trust level according to the language. Java clones seem to have a slightly better trust level than C clones.

As we have two groups of clones (C and Java) and a variable measured on an ordinal scale, we use the Mann-Whitney U test. It results in a significant difference $W = 133672.5$, $p = 0.008709$ (two-tailed). We calculate Cliff's delta [4] to evaluate the effect size. Cliff's delta, namely d, measures the amount of difference between two variables and ranges between 1 and -1. Effect size is interpreted using the thresholds provided by Romano *et al.* [19], that is if $d < 0.147$ effect is "negligible", $d < 0.33$ "small", $d < 0.474$ "medium" and otherwise "large". We find $d = -0.08$. The confidence interval is $-0.14 \leq d \leq -0.02$. The effect is therefore negligible.

> The only characteristic that has a significant effect size on the association to the trust level is the clone type. Moreover, only type 1 clones can be considered as having a *good* trust level as soon as they have a positive opinion from a rater. Other clones require more opinions.

4.4 Questionnaire

In this section, we summarize and discuss results of the questionnaire answered anonymously by the participants.

1. *Did you know code clones before this survey?* Half of students knew the concept of clones before the experiment.
2. *Did you know this clone definition?* 17% of students knew the clone definition used by Bellon *et al.* [3].
3. *Was the clone definition precise enough?* 72% of students found the definition precise enough.

The first two questions provide information about the background of students participating in our study. Question 3 reveals that a majority of students did not have trouble with the clone definition.

5. RELATED WORK

In this section, we present existing work related to our study. First we report several articles using or assessing Bellon's benchmark. Then we present some studies providing clone benchmarks and others dealing with the problem of raters subjectivity on clones.

5.1 Bellon Benchmark in the Literature

Bellon's benchmark has been widely used by the research community. Some studies have used it, some studies have used a subset of it and some studies have extended it. In this section, we present several papers corresponding to these three use cases of Bellon's benchmark.

Ducasse *et al.* [5] present a lightweight string-based approach to identify duplicated source code. They compare their approach to those of Baker [1] and Kamiya[9], which are present in Bellon's benchmark. They use the clones of Bellon's reference corpus to measure recall values of tools they evaluate. They find that their clone detection technique generally achieves high recall and acceptable precision.

Murakami *et al.* [16] propose a token-based clone detection method. They develop a tool, namely FRISC, implementing this approach. They use Bellon's benchmark as a set of real clones to evaluate FRISC. They find that FRISC detects more real clones than any other tools used in Bellon's benchmark.

Nguyen *et al.* [18] propose JSync, a clone management tool. They conduct empirical experiments on Bellon's benchmark. Because the reference corpus contains only 2% of the 325,935 submitted candidates, they use it only for comparing the recall among tools but not measuring the absolute recall of each individual tool.

Wang *et al.* [25] investigate the problem of the choice of clone detectors configuration. They introduce an approach to automatically find configurations for clone detectors maximizing a given criterion. They evaluate their approach with a large-scale empirical study based on Bellon's benchmark.

Selim *et al.* [21] present a hybrid clone detection technique based on source code transformation. They study the performance of their technique using Bellon's benchmark. They manually classify all clones produced by tools that are not present in Bellon's benchmark. It takes the authors over 8 days to perform the manual evaluation.

Koschke *et al.* [11] introduce a suffix tree based approach. They compare their technique to other techniques using Bellon's benchmark. Because some tools of their study are not present in Bellon's benchmark, they extend Bellon's reference corpus.

Murakami *et al.* [17] study clones where there exists a gap between the original code fragment and pasted fragments. They propose a method to detect these gapped clones, implement it as a software tool named CDSW, and evaluate it with Bellon's benchmark. They add locational information of gapped lines to each clone of Bellon's reference corpus. They calculate recall, precision and f-measure by using clones with and without gaps information. They find that information about where gaps are improve the accuracy of clone detection results. This extension of Bellon's benchmark is further detailed in another paper [15].

5.2 Assessment of Bellon's Benchmark

Svajlenko and Roy [23] conduct a study on the performance of modern clone detection tools. They evaluate and compare the recall of eleven modern clone detection tools (CCFinderX, ConQat, CPD, CtCompare, Deckard, Duplo, IClones, NiCad, Scorpio, SimCad and Simian) using four benchmark frameworks, including different variants of Bellon's benchmark. They formulate expectations for each tool and then check for agreement among these expectations and results of each benchmark. They suggest that Bellon's benchmark may not be accurate for modern tools. They conclude that is important to update Bellon's benchmark with clones detected by modern clone detection tools.

5.3 Clone Benchmarks

Krutz and Le build a set of method level clones to help the evaluation of clone detection tools [12]. They ask seven persons including four students and three experts in clone research to oracle 1,536 function pairs randomly selected from three open source programs: Apache, Python and PostgreSQL. Since they use several judges to decide whether or not a function pair is a clone, their validation process is more rigorous than the one used by Bellon *et al.* [3].

Svajlenko *et al.* propose a benchmark of clones [22] in a big data inter-project Java repository, called IJaDataset. This benchmark is built independently of clone detection tools. Hence it is not limited to the clones that tools are able to identify. First they use a search heuristic to find snippets that might implement a given functionality. Then they ask judges to manually check if these snippets are true or false positives. Their benchmark contains clone pairs of ten specific functionalities.

5.4 Developers Judgments on Clones

Kapser *et al.* [10] perform a study to assess agreement among clone experts when classifying potential clones as true or false positives. Authors use CCFinder [9] to select 20 candidate clones from the Postgresql source code. They find that only 50% of candidate clones are classified in the same way by more than 80% of the experts.

Yang *et al.* propose an automatic classification of code clones [26]. In their study, they ask four students to rate a sample of clone candidates and use the answers as input of a machine learning algorithm to perform an automatic classification on the original set of clones, thus reducing the number of clone candidates to manually investigate. However, raters must always have an opinion about a clone (*yes* or *no*) which is not obvious.

Walenstein *et al.* use the corpus provided by Bellon to investigate the level of agreement among raters [24]. They conclude that agreement among raters strongly depends on the projects being studied and on the question asked to raters.

6. THREATS TO VALIDITY

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

6.1 Construct Validity

The main threat to construct validity is related to the process of selecting clones from each group of participants, from which we do random sampling. Providing other clones to participants could have produced different results. However, as seen in section 4, the clone characteristics in each group are distributed similarly. Therefore we think that this threat has a low impact.

6.2 Internal Validity

We randomly created groups of students without considering their expertise on clones. It is possible the groups were not balanced. Trust level as computed from the answers of the two students of a group would have been different if we had created other groups. However, as seen in section 4, the trust level of clones is distributed similarly in each group. Therefore we think that this threat has a low impact.

6.3 External Validity

Our study bears two main threats to external validity. First, the participants, as Bellon, are students and thus their judgments could not match developers' judgments. However, they all have an IT background and have been trained in Java and C programming languages. Furthermore, their participation to the experiment was not mandatory.

Second, the participants only rate a subset of Bellon's benchmark. Therefore results on the whole benchmark could be different. Nevertheless the clones seen by the participants have been randomly selected.

7. CONCLUSION AND FUTURE WORK

In this article, we perform an empirical assessment of Bellon's benchmark. We seek the opinion of eighteen persons on a subset of Bellon's reference corpus, and explore three research questions. First we show that there is a significant number of reference clones that are debatable: a majority of the clones rated by the participants has less than a *good* trust level. Second we find out that precision and recall can be significantly modified by the trust level of reference clones. This phenomenon can introduce noise in results obtained using Bellon's benchmark. Results derived from this benchmark have to be interpreted with caution. Last we show that among the three clone characteristics we evaluate, only the type is associated to the trust level. Only type 1 clones can be considered as having a *good* trust level as soon as they have a positive opinion from a rater.

The lesson we can draw from this experiment is that a clone benchmark must not be built by only one person. In future work, we will investigate how to make easier the construction of clone benchmarks. First we will assess whether program expertise helps to build clone benchmarks. In other words, we will assess whether clone benchmarks have to be done in collaboration with experts of programs in which clones are identified. Second we will investigate whether providing a more precise clone definition to raters helps to categorize clones. One way to clarify this definition is to specify clone purpose. There are mainly two clone purposes to consider: refactoring and co-evolution. We will assess whether providing this information to raters makes the construction of clone benchmarks easier, and whether reference clones selected in such way have higher trust. Finally we plan to use our findings to build high quality clone benchmarks.

8. ACKNOWLEDGMENTS

The authors would like to thanks all the students participating in this experiment. They also would like to thanks Matthieu Foucault for his comments on the paper.

9. REFERENCES

[1] B. S. Baker. On finding duplication and near-duplication in large software systems. In *Proceedings of the Second Working Conference on Reverse Engineering*, WCRE '95, pages 86–, Washington, DC, USA, 1995. IEEE Computer Society.

[2] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 368–, Washington, DC, USA, 1998. IEEE Computer Society.

[3] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. *Software Engineering, IEEE Transactions on*, 33(9):577–591, Sept 2007.

[4] N. Cliff. *Ordinal methods for behavioral data analysis*. Psychology Press, New-York, USA, Sept. 1996.

[5] S. Ducasse, O. Nierstrasz, and M. Rieger. On the effectiveness of clone detection by string matching: Research articles. *J. Softw. Maint. Evol.*, 18(1):37–58, Jan. 2006.

[6] B. Efron. Bootstrap methods: another look at the jackknife. *The annals of Statistics*, pages 1–26, 1979.

[7] N. Göde and R. Koschke. Incremental clone detection. In *Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*, CSMR '09, pages 219–228, Washington, DC, USA, 2009. IEEE Computer Society.

[8] L. Jiang, G. Misherghi, Z. Su, and S. Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 96–105, Washington, DC, USA, 2007. IEEE Computer Society.

[9] T. Kamiya, S. Kusumoto, and K. Inoue. Ccfinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Trans. Softw. Eng.*, 28(7):654–670, July 2002.

[10] C. Kapser, P. Anderson, M. Godfrey, R. Koschke, M. Rieger, F. van Rysselberghe, and P. Weißgerber. Subjectivity in clone judgment: Can we ever agree? In R. Koschke, E. Merlo, and A. Walenstein, editors, *Duplication, Redundancy, and Similarity in Software*, number 06301 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[11] R. Koschke, R. Falke, and P. Frenzel. Clone detection using abstract syntax suffix trees. In *Proceedings of the 13th Working Conference on Reverse Engineering*, WCRE '06, pages 253–262, Washington, DC, USA, 2006. IEEE Computer Society.

[12] D. E. Krutz and W. Le. A code clone oracle. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 388–391, New York, NY, USA, 2014. ACM.

[13] B. Lague, D. Proulx, J. Mayrand, E. M. Merlo, and J. Hudepohl. Assessing the benefits of incorporating function clone detection in a development process. In *Proceedings of the International Conference on Software Maintenance*, ICSM '97, pages 314–, Washington, DC, USA, 1997. IEEE Computer Society.

[14] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. Cp-miner: Finding copy-paste and related bugs in large-scale

software code. *IEEE Trans. Softw. Eng.*, 32(3):176–192, Mar. 2006.

[15] H. Murakami, Y. Higo, and S. Kusumoto. A dataset of clone references with gaps. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 412–415, New York, NY, USA, 2014. ACM.

[16] H. Murakami, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto. Folding repeated instructions for improving token-based code clone detection. In *Proceedings of the 2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*, SCAM '12, pages 64–73, Washington, DC, USA, 2012. IEEE Computer Society.

[17] H. Murakami, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto. Gapped code clone detection with lightweight source code analysis. In *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*, pages 93–102, May 2013.

[18] H. A. Nguyen, T. T. Nguyen, N. H. Pham, J. Al-Kofahi, and T. N. Nguyen. Clone management for evolving software. *IEEE Transactions on Software Engineering*, 38(5):1008–1026, 2012.

[19] J. Romano, J. Kromrey, J. Coraggio, and J. Skowronek. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys? In *annual meeting of the Florida Association of Institutional Research*, pages 1–3, 2006.

[20] C. K. Roy and J. R. Cordy. Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension*, ICPC '08, pages 172–181, Washington, DC, USA, 2008. IEEE Computer Society.

[21] G. Selim, K. Foo, and Y. Zou. Enhancing source-based clone detection using intermediate representation. In *Reverse Engineering (WCRE), 2010 17th Working Conference on*, pages 227–236, Oct 2010.

[22] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. M. Mia. Towards a big data curated benchmark of inter-project code clones. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*, pages 476–480, 2014.

[23] J. Svajlenko and C. K. Roy. Evaluating modern clone detection tools. *Proc. ICSME*, 2014.

[24] A. Walenstein, N. Jyoti, J. Li, Y. Yang, and A. Lakhotia. Problems creating task-relevant clone detection reference data. In *Proceedings of the 10th Working Conference on Reverse Engineering*, WCRE '03, pages 285–, Washington, DC, USA, 2003. IEEE Computer Society.

[25] T. Wang, M. Harman, Y. Jia, and J. Krinke. 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, pages 455–465, New York, NY, USA, 2013. ACM.

[26] J. Yang, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto. Classification model for code clones based on machine learning. *Empirical Software Engineering*, pages 1–31, 2014.