Towards a Curated Collection of Code Clones

Ewan Tempero
Computer Science
The University of Auckland
Auckland, New Zealand
e.tempero@auckland.ac.nz

***Abstract*—In order to do research on code clones, it is necessary to have information about code clones. For example, if the research is to improve clone detection, this information would be used to validate the detectors or provide a benchmark to compare different detectors. Or if the research is on techniques for managing clones, then the information would be used as input to such techniques. Typically, researchers have to develop clone information themselves, even if doing so is not the main focus of their research. If such information could be made available, they would be able to use their time more efficiently. If such information was usually organised and its quality clearly identified, that is, the information is curated, then the quality of the research would be improved as well. In this paper, I describe the beginnings of a curated source of information about a collection of code clones from the Qualitas Corpus. I describe how this information is currently organise, discuss how it might be used, and proposed directions it might take in the future. The collection currently includes 1.3M method-level clone-pairs from 109 different open source Java Systems, applying to approximately 5.6M lines of code.**

***Index Terms*—Code Clones; Corpus; Code Analysis; Empirical Studies**

I. Introduction

Those doing empirical studies face a number of barriers to their research. Empirical research is fundamentally about studying the "real world", and so one barrier is accessing the relevant part of the real world. In the world of software clones, the relevant part is source code and the clones (or other forms of redundancy) that exists in it.

A crucial aspect of good science is replication. While there may be some disagreement as to exactly what this means (see e.g. [4]), fundamentally it is about doing studies in such a way as to allow some reasonable form of comparison of their results with other studies (those being replicated). A barrier to replication is getting access to enough of the infrastructure and materials of past studies to support their replication. Again, in the world of software clones, the materials are the source code and clone information.

The use of standard datasets to support research is common in other areas such as Linguistics ([10]), but also in Computer Science ([20] and Software Engineering ([7], [3]). In this paper I describe a first attempt at developing a similar dataset for clone research. This work is based on a curated corpus of open source Java software — the Qualitas Corpus [17], and inspired by the Bellon clone benchmark. [2]

Specifically, I describe the Qualitas Corpus Clone Collection (hereafter, "the Collection"), a set of datasets describing possible clone pairs in a number of open source Java systems. The current release includes 1.3M method-level clone-pairs from 109 different open source Java Systems, applying to approximately 5.6M lines of code.

The rest of the paper is organised as follows. In the next section, I briefly summarise current clone research to establish the need for the Collection and to identify likely uses for it, and discuss other efforts to develop datasets to support clone research. This provides the background to develop the requirements for a curated collection of code clones, which are described in Section III. Section IV describes the current organisation of the Collection and summarises its current contents. Section V discusses how well the current Collection meets the stated requirements, the short-term development plans, and potential future development. Finally, Section VI summarises the paper and presents conclusions.

II. Background and Related Work

A. Terminology

This section introduces the terminology used in the rest of the paper and gives an overview of related research. Roy et al. provide a good survey [16] and much of their terminology is used here.

A *code fragment* is any sequence of code lines that can be any granularity, such as a complete method definition or a block. A *clone pair* is defined by having two code fragments that are similar by some given definition of similarity. When more than two fragments are similar, they form a *clone cluster*.

There are two main types of similarity between code fragments: textual similarity and semantic similarity. Two fragments have textual similarity when the text they contain is matches to a large degree. This might be the consequence of copying and pasting one fragment to the other, perhaps with minor modifications. Two fragments are semantically similar if they have similar functionality, but may have completely different text. The Collection is (for now) intended to support research with respect to textual similarity.

Clone types can be categorised in different ways. A common classification is the following four types based on both textual and semantic similarities ([16]):

Type-1: Identical code fragments except for variations in whitespace, layout and comments.

Type-2: Syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout and comments.

Type-3: Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments.

Type-4: Two or more code fragments that perform the same computation but are implemented by different syntactic variants.

B. Clone Research

Of the large body of existing research on code clones, what is relevant to this work is the *variety* of research goals. Different goals may have different requirements for the data they need or even the data format. In this section I provide summaries of a sample from the research literature. This sample is necessarily selective, where the primary criteria is the kind of research being done rather than, for example, what the results are.

A major concern in clone research is clone detection. The primary goal for this kind of research is to improve detection characteristics such as the accuracy (e.g. [1]) or performance (e.g. [12]), or both (e.g. [21]). For this kind of research, the kind of input that is needed is, the source code to detect the clones in and, ideally, what clones exist. Just having the source code available without the clone information can represent a major savings of effort, as well as supporting replication (see below), but clearly having the clone information would be even better.

If the clone information does not exist, then the output by the detectors must be evaluated by some other means. Often, these means are not reported, but clearly must take time and effort. Currently, the only means of evaluation reported is that of human judgement. One documented example of this is the work by Bellon et al [2]. They compared 6 clone detectors that used different approaches over 4 C and 4 Java systems with respect to their accuracy and performance. The programs varied from 11K SLOC to 235K SLOC (a similar range for both languages).

The results were evaluated against a benchmark set developed by Bellon. It was developed by evaluating the candidates proposed by the tools being studied. An automatic process randomly chose 2% of the 325,935 candidate clones reported from all tools with the chosen candidates uniformly distributed across the tools. These were presented anonymously (that is, without identifying the tool the candidates came from) to Bellon, who made a decision as to whether or not the candidate is an actual clone pair. The authors note that this was an expensive process. Despite looking at only 2% of all candidates, it took nearly 80 hours to complete the classification. The benchmark also included some clone pairs injected by Bellon.

The Bellon benchmark has been used by other studies. For example, Falke et al [8], who also compared several tools using 6 of the 8 systems from the Bellon et al study. They created their own reference set of clones, but did so following the same process as Bellon.

Clearly Falke et al. benefited from the existence of the Bellon benchmark. However, by not reusing the original benchmark reference set, the benefit is not as much as it could have been. Had Falke et al. instead been able to put their efforts into re-evaluating the Bellon benchmark reference set, they would have then had a reference set that was of higher quality, since it would have been checked by two people independently. This is especially important as both groups raised the fact that their reference sets were dependent on the judgement of one person. Having even only two people perform the check would have significantly raised the quality of the reference set, and so increased the quality of the results of the studies.

Many studies pre-process the code being analysed in some way. Bellon et al. normalised all code by removing empty lines, moving lines containing only an open or close brace to a proceeding line, and "paying attention" to comments. Yuan and Guo did not include constructors and "small" methods in their CMCD clone detector [21]. In fact it seems quite common to have a minimum limit on clones that are reported (e.g number of lines [2] or token density [8]).

The construction of a reference set by human judgement is not only expensive, but also subjective. It is very rarely reported what criteria have been used. Having only one person involved introduces a threat to validity, as Bellon et al. and Falke et al. acknowledged. It is known that there is disagreement in the community regarding what constitutes a clone. Walenstein et al. carried out some studies (also using systems from the Bellon benchmark) to explore this issue [19]. They report that in some cases the participants agreed on only 1 in 4 candidates. They did note that some of the disagreement was due to the intended purpose of the classification.

More recently, Koschke has called for a "general notion of redundancy, similarity, and cloning" [14]. Furthermore, the issue of clone classification still exists 5 years later, as demonstrated by the study carried out by Chatterji et al. [5]. While it is not the goal of this work to resolve this issue, is does impact the development of a clone collection if clone classification is intended to be part of the data collected. Merely providing a "list of clones" is unlikely to be of use to everyone, or even the majority of potential users.

Chatterji et al. also identified a number of research questions regarding clone research. One of specific relevance to the development of a clone collection is that of investigating clone evolution, which suggests that for the Collection to be useful, it should include multiple versions of a system. Exactly what constitutes a "version" and what version information is necessary depends on what it is about clone evolution that is of interest. For example, some studies may consider it sufficient to use the public releases for a system, whereas others may need more fine-grained information such as individual line changes (see e.g. [13], [15].

An issue that arises in any empirical research is how to manage the data. As well as what should be included (such as clone classification), there are also questions about how to effectively share it. Harder and Göde discuss this issue and their solution to it [9]. Specifically, they introduce their Rich Clone Format (RCF) data format, and Cyclone, their clone inspection tool that is based on RCF. I will return to this issue

in section V.

Depending on the research goals, various attributes of clones are important. Clone pairs are usually reported as locations (e.g. file + line numbers). A number of studies classify the clones according to the classification given above. For some studies, the goal of the research is directly related to this classification (for example, such as that by Tiarks et a. [18]). Some research requires describing the relationship between clone pairs in more details, such as Juergens et al.'s concept of *inconsistent clones* [11]. Juergens et al. also related clones with fault information.

C. Corpus Development

Clone research depends, directly or directly, on having code (or something like it) available. Finding enough code can be a chore that contributes little to the fundamental research being done. The Qualitas Corpus (hereafter "the Corpus") is intended to reduce the cost of this kind of chore [17]. It is a curated collection of open-source Java software systems, consisting (currently) of the source and compiled code for 111 systems. There are multiple versions for most systems, with 14 having 10 or more versions. My experience with the Corpus provides some guidance in developing a clones collection, and the Corpus also provides a good starting point for the Collection. Further details of the Corpus makeup will be provided as needed.

III. Requirements

This section provides the an initial set of, and motivation for, requirements that a developer of a clone collection should take into consideration.

The first requirement is to agree upon the source code that the clones come from. This seems like a simple requirement to meet, but in fact it can be quite expensive in time and effort. Furthermore, it is not enough to just acquire an archive file with some source code. Decisions need to be made about which files in the archive will actually be analysed, and those decisions need to be *recorded* and *reported* if replication (or at least comparison between studies) is to be supported.

For example, many open source Java systems come with test suites. Presumably these are not going to be of interest to a clone researcher, and probably any clone researcher would make the same decision. But presumptions and probabilities lead to uncertainties, which lead to difficulties when trying to replicate a study or compare results. A number of open source Java systems come with example code showing their use. Probably this code also is not of interest to a clone research, except, it might be interesting if the examples contain clones of code in the system, so different researchers may make different decisions about example code. Some systems have multiple implementations of some code to support different infrastructure. Analysing all the code might lead to many clones being reported, but if the analysis is limited to just one implementation, that needs to be reported. A longer discussion of such issues can be found in the development of the Qualitas Corpus [17].

With so many possibilities, there is unlikely to be a "correct" choice, and in fact for many it probably does not matter which choice is made, so long as it is recorded and reported. This means there needs to be some agree-upon means to specify exactly what source code has been used in any study.

The next requirement might be a reference set of clone pairs. Since there is no agreement within the community as to what exactly a clone is, this requirement would be difficult to meet. Instead, I propose a reference set of clone pair *candidates*. The distinction is, instead of the data providing an absolute truth — clone or not clone — it provides a continuum of possibilities, together with information about the likelihood, or knowledge about, each possibility. The candidate set might also include definite non-clones, for example pairs of code fragments that are often reported by clone detection algorithms as clones even though everyone agrees they are not. Other candidates may only be considered clones under some circumstances, such as depending on whether or not constructors are analysed. The principle here is, whatever decision is made, its rationale be recorded and reported.

Specifically, a clone pair candidate consists of a specification of a code fragment, that includes its location within the source code being analysed. This most likely will be a file name and line or character positions for the beginning and end of the fragment. Associated with this candidate is metadata describing its various attributes. For every attribute with a value, there should be details of the *provenance* of that value — information that explains why the value has been assigned to that attribute. Examples of what this might look like are given below and in the next section. Values for which there is no provenance should be treated as little better than a guess (such a thing is explicitly allowed).

From a practical points of view, the candidate data needs to be such that it can be easily used. No one will want to have to puzzle through all the qualifications as to whether a given candidate is really a clone or not right at the beginning of their research. For many researchers, they will be happy with a data set that is good enough to get started with. So long as they have some assurance that the accuracy is reasonable (e.g., false positives and false negatives less than 10%) they can worry about the details once their research has progressed enough. For this reason, every candidate will have a value for the "is this a clone" attribute.

The provenance for the "is this a clone" attribute could consist of many things. As noted in section II, cloneness is often decided by human judgement. If this is the case for a given candidate, then that is what would be noted. It would be good to also include some means of identifying who provided the judgement and when. If the human was following some guidelines to make the judgement, then those should be provided. A candidate that has three different people providing the same judgement several months apart would give more confidence of the judgement than just one person. I am not necessarily advocating naming names. It is probably sufficient to just identify the research group or something similar.

A common source of candidates is from clone detectors.

The more clone detectors that agree on a given candidate, the more confidence we might have on both the assessment of the candidate and the performance of the detectors. Detectors often have parameters for tuning, and so the values used also need to be recorded. Detectors are software too, and so they change, perhaps changing the results they produce. This means that some form of version identification should also be provided. Just indicating when the detector was run may not be sufficient to identify the version, although any data set should include a timestamp indicating when it was created. Many detectors also give a "score", with "cloneness" determined by a threshold parameters. While the threshold would be probably a parameter to the detector, it is likely to be useful to also report the score. This would, for example, allow studies to be performed using a different threshold, without the expense of re-creating the candidate set.

There are many other attributes that might be included. Some examples, including some comments about possible provenance, give listed below.

- Granularity. This might be determined by the location information (from which number of lines might be determined), but other indications of granularity (e.g "method") may not be so clear.
- Code fragment size. This will actually be multiple attributes, as different researchers have found value is measuring clone size in different ways (e.g. number of tokens, different variations of lines of code).
- Code fragment description. There may be other information, other than granularity and size, that is relevant to a researcher's use of the data. For example, some researchers may want to include "methods" that are constructors, and some may not. Other examples include concepts such as initialiser blocks (static or not) in Java. There may be questions regarding code fragments belonging to nested classes. Another question is whether (in Java) interfaces are considered or not (complicated by the fact that interface definitions can — and do — have nested classes).
- Clone Type. Type 1 and 2 might be done via automatic analysis, but even so it would be useful to have some indication of the criteria used.
- Cluster. If the candidates have been identified as belonging to a cluster, then which cluster it is and how clusters were determined should be included. For example, is the cluster a clique (all pairs of code fragments in the cluster are also clone pairs) or a connected component (there is a path of clone pairs between each pair of code fragments).
- Clone proportion. The proportion of the code base that the clone pair (or, more typically, clone cluster) is. This will depend on how code fragment size is determined, and will require the measurement with same metric for the system being analysed. For example, if constructors and "small" methods are not considered (as was the case with Yuan and Guo's work [21]), should the proportion be computed against the complete code base, or just the code that was considered?
- Origin. Any information as to which fragments were copied from which, determined either from information provided by the developers, or from analysis of version control systems (such as was done by Krinke et al. [15]), or similar.

Generally, as more studies are done on the collection, they will validate or refute the data, or possibly provide new data to add.

Some data about the candidates is likely to be common across a set, for example, all the candidates from the same software system might have been determined by a single execution of a detector with the same parameters and (about) the same time. Such sets need to be managed as units, to ensure that the fact that this relationship is not lost. There will also be less duplication of common information is all candidates for a given system are

Finally, I believe there should be one or more attributes providing some assessment of the quality of the data. Exactly what this should look like will probably only be determined over time by the community, so I propose to start with a simple ordinal scale indicating the level of confidence for each candidate. The lowest value will be assigned to any candidate that has been identified as a clone (or a non-clone) by a clone detector. The confidence level can be increased only by providing more independent information about the assessment.

For example, a candidate whose classification is checked by a human may have its confidence level increased, but it will depend on the circumstances. If the candidate currently is on the lowest level, then a human check will almost certainly raise the level (assuming the human agrees with the tool). If the human disagrees with the current classification that is based on other humans' judgement, then the level may not be increased. As another example, if the detector providing the original classification is mature and its accuracy well-established, then a candidate's classification may also be increased.

IV. Contents of the Collection

A first attempt at a clone collection as described in the previous section exists. It currently includes 1.3M method-level clone-pairs from 109 different open source Java Systems (see Figure 1), applying to approximately 5.6M lines of code (excluding "small" methods and constructors). Details of the Collection are available at www.qualitascorpus.com/clones.[1]

The clone corpus uses the Qualitas Corpus as the source of the code[17, Release 20120401]. All location information for clone candidates is given with respect to a Qualitas Corpus installation. The full Qualitas Corpus consists of 111 open source Java systems, with multiple versions available for most systems. There are 14 systems that have 10 or more versions, which can provide the basis for some code evolution studies. Full details are provided on the Corpus website.

[1]**REVIEWERS**: A preliminary release is available from www.qualitascorpus.com/clones/reviewonly. It is made available to you for review purposes only. It is still under development but what is there should confirm the claims made about it.

```
ant antlr aoi argouml aspectj axion
azureus batik c_jdbc castor cayenne
checkstyle cobertura collections colt
columba compiere derby displaytag drawswf
drjava emma exoportal findbugs fitjava
fitlibraryforfitnesse freecol freecs
freemind galleon ganttproject gt2 hadoop
heritrix hibernate hsqldb htmlunit informa
ireport itext ivatagroupware jFin_DateMath
jag james jasml jasperreports javacc jboss
jchempaint jedit jena jext jfreechart
jgraph jgraphpad jgrapht jgroups jhotdraw
jmeter jmoney joggplayer jparse jpf jrat
jre jrefactory jruby jsXe jspwiki jtopen
jung junit log4j lucene marauroa maven
megamek mvnforum myfaces_core nakedobjects
nekohtml openjms oscache picocontainer
pmd poi pooka proguard quartz quickserver
quilt roller rssowl sablecc sandmark
springframework squirrel_sql struts sunflow
tapestry tomcat trove velocity wct webmail
weka xalan xerces xmojo
```

Fig. 1. The current release of Collection contains data for the latest version available from the Qualitas Corpus for these systems.

```
.../QCCC/
      Systems
        ant/
          ant-1.1/
            ant-1.1-clones.rcf
            provenance/
              ant-1.1-mete-cmcd.csv.gz
              ...
         ant-1.8.2/
            ant-1.8.2-clones.rcf
            provenance/
              ant-1.8.2-mete-cmcd.csv.gz
              ...
        /antlr
          ...
        /aoi
          ...
        ...
```

Fig. 2. Planned Collection Organisation

The Corpus comes with metadata providing information about the software it contains. Included in this information (in the `contents.csv` metadata file) is, for every Java type mentioned in a system's software source and binary (deployment) distribution, an assessment as to whether that type was developed for that system's deployment, whether the source code for that type is provided in the distribution, and whether the type is deployed (provided as byte code, that is, in compiled form). This information can be used to specify precisely what is being analysed, and so resolving issues such as: are classes for which there is source code but which are not deployed (e.g. test classes) included ; are types for which both source code and byte code is provided but not developed for the system (e.g. third-party library code) included. In the case of the Collection, the types that are analysed are those for which there is source code, byte code, and are considered developed for the system.

Figure 1 shows the systems for which there is currently clone data available. The data exists for the latest version of each system that is in release 20120401 of the Corpus (which versions those are can be found on the Corpus website). In fact, data exists for all systems except for `eclipse` and `netbeans`. These are the two largest systems in the corpus, and the current version of tool used to generate the data cannot handle systems their size to to memory limitations. The largest systems for which data is provided is `jre`, at about 0.9M non-comment non-blank lines of code. There is also data for 28 versions of one system (`freecol`).

The candidate attribute data comes from a clone detector based on the CMCD technique developed by Yuan and Guo [21]. This detector, called `mete-cmcd`, is described elsewhere [6]. The current version of it evaluates at the level of method granularity, and does not evaluate constructors or "small" methods (a parameter). This detector reports a "difference" score for each candidate, with the determination of whether the candidate represents a clone pair or not dependent on a "threshold" parameter. For performance reasons and to reduce false positives, the detector uses two heuristics: the ratio of the size of the methods (if too different then the methods cannot be clones) and text difference ratio (ditto). There are other parameters specific to its operation but do not affect the clones reported.

Figure 2 shows the planned organisation for the collection. The data is organised along the same lines as the Corpus; data in the Collection for multiple versions of systems (where it exists) is kept together and named using the same system and version identifier used by the Corpus. The figure shows 4 systems, with the first (`ant`) having at least two versions (`1.1` and `1.8.2`).

For each system version, there is the master file containing the data pertaining to that system version. It will be in RCF format (however this is not implemented at the time of writing). This data should include the specification of each clone pair and references to the relevant provenance information. Also associated each system version is any relevant provenance information. How this is organised and what format the data is in will depend on how it is gathered. The figure shows the output files from the `mete-cmcd` detector (which produces the data as a tab-separated file).

To give an idea of the kind of information that should be provided, the `mete-cmcd` output is described in more detail. Figure 3 shows an outline of such a file. Each file consists of some information common to all candidates listed, a list of candidates, and also a list of clone clusters and data about them.

The common information includes the tool version used to

generate the data file, the time the data file was created, and the parameters provided to the tool (mostly described above). The file is intended to be relatively self-contained, that is, anyone should be able to determine what the different values mean without having to refer to some other source. In fact, there is always likely to be some specialised information that would make such a goal difficult to achieve, but there should be enough hints to remind people already somewhat familiar with the content.

Next there is summary information about the data set, such as what exactly the data applies to, how many files were examined, how many clone pairs were found, and what proportion the fragments in clone pairs are of the code base ("Code clones").

In fact there are two proportions given. The other ("Cloned code") is based on the assumption that every cluster has an "original" code fragment and all other fragments have been cloned (directly or indirectly) from that. The "Cloned code" proportion is the proportion of code fragments in clone pairs other than this "original" fragment. Even if this assumption is true, which fragment is the original is not (currently) known, so `mete-cmcd` chooses the smallest fragment as the original. Because `mete-cmcd` uses the method granularity, and will only consider methods to be candidate clone pairs if they are roughly the same size, the "Cloned code" proportion is going to be close to what it should be had the original method (existed and) been used to compute the proportion. Choosing the smallest means the proportion will always be the same value for the same code base, and be conservative with respect to what the actual value might be.

While some information, in particular which system version a given data file applies to, can be inferred from context (such as which directory the file is stored in), such information is included keeping with the principle of self-containment.

The next section of the data file provides the clone data. Each line represents a clone pair, described in 13 fields. Figure 4 describes those fields in the order they appear (in fact the text in this figure appears in the data file). Some of this information is specific to the source of the information (that is, `mete-cmcd`), but the information describing the code clone pairs (cluster and location of code fragments) would appear in the master file.

The final section of the data file contains information for the clusters. This includes, for each cluster ID, the number of clone pairs in the cluster, the number of distinct code fragments (methods) in the cluster, the sum of the size (lines of code) of each code fragment in the cluster, and the size for the cluster not counting the smallest code fragment. All of this information can be inferred from the clone pair data, but it is useful to have it provided explicitly.

Cluster	A unique ID identifying the cluster the clone pair belongs to.
File1	The name of the file (sans foldpath prefix) containing the lexically first method in the clone pair.
Method1	The name of the lexically first method in the clone pair.
Location1	The beginning and ending line numbers in the file where the lexically first method can be found.
ELOC1	The number of lines of code in the lexically first method.
Nodes1	The number of nodes in the AST for the the lexically first method.
File2	The name of the file (foldpath common prefix) containing the lexically second method in the clone pair.
Method2	The name of the lexically second method in the clone pair.
Location2	The beginning and ending line numbers in the file where the lexically second method can be found.
ELOC2	The number of lines of code in the lexically second method.
Nodes2	The number of nodes in the AST for the the lexically second method.
Diff	The normalised difference score between the two methods.
RawDiff	The raw difference score between the two methods.

Fig. 4. Clone Pair fields from `mete-cmcd`

V. Status and Discussion

Most of the data in the Collection as at a low level of confidence, since it has been provided by a relatively untested clone detector. One of the systems (`freecol-0.8.0`) has been examined by the initial developers of `mete-cmcd`. That data has not yet been incorporated, but once that is done the confidence level for that data will be increased.

Another possible source of confidence information is the Bellon Benchmark. All of the Java systems in that are only sub-systems of systems in the Corpus (e.g. `swing` from the Bellon benchmark is part of `jre` in the Corpus). In fact all Bellon systems are subsystems of either `jre`, `eclipse`, or `netbeans` — that is, including two systems for which there is currently no data in the Collection. While it would be good to include the Bellon reference set, it will need to be matched to the appropriate code (version) in the Corpus first.

What the Collection does have is a good amount of data. Hopefully, even though it is only at a low level of confidence, it is still of good enough quality, and there is enough of it, to be useful to someone. Over time, as it is used, its quality will hopefully improve.

That said, there are some limitations in the Collection. One is that the candidates are only at the method granularity level. While the data will still be of use to those interested in clone pairs that are non-method fragments, some effort will be required to map their results to the Collection data. The main reason for including line number location information for candidates is to help with this mapping.

Another limitation is that the Corpus (and hence the Collection) is only a single language (Java) and is all open source. It would be much more useful if other languages were available (however including close-source is likely to be more difficult, given intellectual property concerns).

The limitations listed above apply to the data in the Collection, however there is also a limitation with regards how the Collection should be updated. The question here is, if someone does use the Collection and gathers data that could be used to

Tool:	mete-cmcd: 2013-01-20T1620	
Timestamp:	Mon Jan 21 01:10:53 NZDT 2013	
Parameters		
Fold Path:	...	A path prefix that is elided when displaying paths *not shown for presentation purposes*
Difference Threshold:	45	The largest value of the normalised CM difference that is considered a clone pair
Minimum AST nodes:	50	The definition of 'small' (in AST nodes) for omitting small methods.
Text difference threshold:	0.5	If the method texts differ by more than this ratio then not clones.
Size ratio threshold:	0.65	If the method size (measure as number of nodes in AST) ratio is more than this then not clones.
Comments ignored:	true	
Global Values		
Sysver:	checkstyle-5.1	Identification of what was analysed (typically a corpus System Version)
Files:	267	Number of files analysed
Methods:	680	Number of methods, not counting constructors or methods that are too small.
ELOC (Methods only):	9920	ELOC for methods, not counting constructors or methods that are too small. ELOC is lines of code, not counting lines that are blank, contain only comments, or only braces.
Clone pairs:	244	Number of clone pairs
Clusters:	57	Number of clusters
Code clones:	2097 (0.21)	ELOC of code that is in a clone pair (proportion of ELOC).
Cloned code:	1470 (0.15)	ELOC of code in clone pair minus size of smallest fragment of each cluster (proportion)

Fig. 3. Example data from `mete-cmcd`

improve confidence levels for candidates, what happens next? At the moment, there is no defined process to following other than to submit that data in whatever form it is available. Ideally the data might be provided in a standard format such as RCF, however any such data will need to extend the core RCF. If different researchers extend RCF in different ways, there will no longer be any benefit to using a standard. Resolving this issue, and generally advancing the state of the Collection, will have to wait until we all have more experience.

VI. Conclusions

In this paper, I have discussed the need for a standard dataset to support clone collection, and described an attempt to design and implement one — the Qualitas Corpus Clone Collection. How useful the Collection will be in its current form remains to be seen, but hopefully the data is of sufficient quality and quantity that some researchers will find it useful.

References

[1] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *ICSM*, pages 368–377, 1998.

[2] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. Comparison and Evaluation of Clone Detection Tools. *IEEE Transactions on Software Engineering*, 33(9):577–591, September 2007.

[3] S M Blackburn, R Garner, C Hoffmann, A M Khang, K S McKinley, R Bentzur, A Diwan, D Feinberg, D Frampton, S Z Guyer, M Hirzel, A Hosking, M Jump, H Lee, J B Moss, B Moss, A Phansalkar, D Stefanović, T VanDrunen, D von Dincklage, , and B Wiedermann. The {DaCapo} benchmarks: {Java} benchmarking development and analysis. In *OOPSLA*, pages 169–190, Portland, Oregan, October 2006.

[4] Nancy Cartwright. Replicability, reproducibility, and robustness: Comments on Harry Collins. *History of Political Economy*, 1991.

[5] Debarshi Chatterji, Jeffrey C. Carver, and Nicholas A. Kraft. Claims and beliefs about code clones: Do we agree as a community? A survey. In *IWSC*, pages 15–21, June 2012.

[6] Xiliang Chen, Alice Yuchen Wang, and Ewan Tempero. A replication and reproduction of code clone detection studies. Unpublished manuscript (Under Review), 2013.

[7] Hyunsook Do, Sebastian Elbaum, and Gregg Rothermel. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. *Empirical Softw. Engg.*, 10(4):405–435, 2005.

[8] Raimar Falke, Pierre Frenzel, and Rainer Koschke. Empirical evaluation of clone detection using syntax suffix trees. *Empirical Software Engineering*, 13(6):601–643, July 2008.

[9] Jan Harder and Nils Göde. Efficiently handling clone data. In *IWSC*, page 81, New York, New York, USA, May 2011.

[10] Susan Hunston, editor. *Corpora in Applied Linguistics*. Cambridge University Press, 2002.

[11] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. Do code clones matter? In *ICSE*, pages 485–495, 2009.

[12] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, July 2002.

[13] Miryung Kim, Vibha Sazawal, and David Notkin. An empirical study of code clone genealogies. In *ESEC/FSE-13*, page 187, New York, New York, USA, September 2005.

[14] Rainer Koschke. Survey of Research on Software Clones. In *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.

[15] Jens Krinke, Nicolas Gold, Yue Jia, and David Binkley. Distinguishing copies from originals in software clones. In *IWSC*, pages 41–48, May 2010.

[16] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, 74(7):470–495, May 2009.

[17] Ewan Tempero, Craig Anslow, Jens Dietrich, Ted Han, Jing Li, Markus Lumpe, Hayden Melton, and James Noble. Qualitas corpus: A curated collection of Java code for empirical studies. In *APSEC2010*, pages 336–345, December 2010.

[18] Rebecca Tiarks, Rainer Koschke, and Raimar Falke. An extended assessment of type-3 clones as detected by state-of-the-art tools. *Software Quality Journal*, 19(2):295–331, November 2011.

[19] A. Walenstein, N. Jyoti, and A. Lakhotia. Problems creating task-relevant clone detection reference data. In *WCRE 2003*, pages 285–294, 2003.

[20] I H Witten, S J Cunningham, and M D Apperley. The {New Zealand Digital Library} project. *New Zealand Libraries*, 48(8):146–152, 1996.

[21] Yang Yuan and Yao Guo. CMCD: Count Matrix Based Code Clone Detection. In *APSEC*, pages 250–257, December 2011.