SeClone - A Hybrid Approach to Internet-scale Real-time Code Clone Search

Iman Keivanloo, Juergen Rilling
Dept. of Computer Science and Software Engineering
Concordia University
Montreal, Canada
{i_keiv, rilling@cse.concordia.ca}

Philippe Charland
System of Systems Section
Defence R&D Canada – Valcartier
Quebec, Canada
philippe.charland@drdc-rddc.gc.ca

***Abstract*— Real-time code clone search is an emerging family of clone detection research that aims at finding clone pairs matching an input code fragment in fractions of a second. For these techniques to meet actual real world requirements, they have to be scalable and provide a short response time. Our research presents a hybrid clone search approach using source code pattern indexing, information retrieval clustering, and Semantic Web reasoning to respectively achieve short response time, handle false positives, and support automated grouping/querying.**

Keywords- Code clone; real-time search; clustering; ontology

I. INTRODUCTION

Source code clone (i.e., similar fragments) detection has been a major focus of the software research [1] and has resulted in various clone detection techniques. Common to all of these traditional detection applications (e.g., plagiarism detection) is that they have a complete off-line search step to find *all* possible clone pairs within a static source code repository. In contrast to these traditional clone detection applications, an emerging family of code clone research has been introduced known as instant, real-time, or just-in-time clone search (e.g., [2]), providing scalable clone search across very large repositories (e.g., SourceForge). These clone search approaches can be considered as specialized search engines designed to find code clone pairs within very large corpora of data. They index source code repositories as part of their off-line processing and then accept at run-time input in the form of a code fragment (i.e., query criteria). As part of their run-time analysis, the clone search engines use their indices to find matching clone pairs for the specified query criteria.

Addressing the tradeoffs between scalability, response time, precision, and recall of real-time code clone search is the major focus of our research. We introduce our SeClone hybrid clone search, which uses a combination of source code pattern indexing and searching (e.g., [2]) on transformed source code (e.g., [3]) to achieve fast response time and scalability. Since this approach alone tends to produce false positives, we also apply information retrieval clustering to classify the detected clone pairs based on their approximate usage type. For the grouping of the clone pairs, we take advantage of Semantic Web reasoning and its native support for this form of clone pair grouping. While clone pair grouping is a common practice in clone detection research [1], clone pair clustering as used in our research is both a novel and complementary step. For clone pair grouping, querying, integration, and information sharing, we designed an ontology called Clone Ontology (*CLON*) which models the *vocabulary set* discussed in [1, 3] and is available online at http://aseg.cs.concordia.ca/seclone. Our approach is motivated by Würsch et al. [4] on addressing integration challenges in source code analysis research. Using such a *vocabulary set*, results can be populated into online source code fact repositories such as SECOLD [5] (http://www.secold.org) to provide structural querying facility for supporting on-the-fly integrated multi-disciplinary data (e.g., querying about clone pairs and their project licenses).

II. A HYBRID APPROACH TO CODE CLONE SEARCH

Our SeClone implementation involves four major processing steps (Fig. 1), which are (1) *preprocessing*, (2) *indexing*, (3) *searching* (find pairs module), and (4) *post-processing* (grouping and clustering modules).

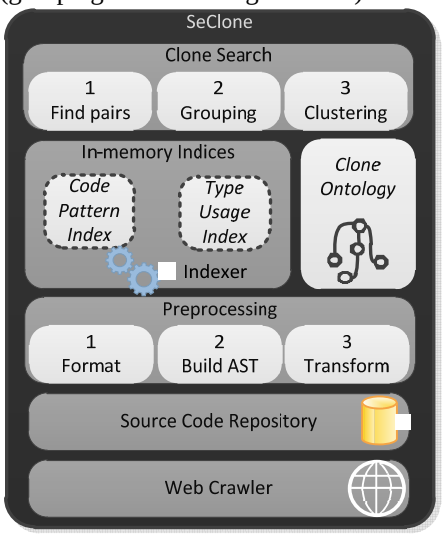

Figure 1. SeClone internal architecture overview

Preprocessing. Source code format/style unification is an essential preprocessing step to preserve an acceptable recall rate, since SeClone is a line-based tool. It creates Java ASTs (Abstract Syntax Tree) for every formatted file. Using the ASTs, it transforms the explored facts into tokens, similar to CCFinder [3]. For example, as part of this step, `for(AttributeEntity var : t.getAttributes())`will be transformed to `for(# #:#.getAttributes())`.

Indexing. SeClone creates two types of indices, a *Code Pattern Index* and a *Type Usage Index*. The *Code Pattern Index* uses transformed code as its information source, whereas the *Type Usage Index* covers file usage patterns (i.e., imported/included types). The *Code Pattern Index* is essential for achieving fast response times, while the *Type Usage Index* is used to reduce the effect of false positives during real-time clone search.

Searching. Our SeClone's search algorithm accepts a code fragment and a target line number within this fragment as input. Using the *Code Pattern Index*, the approach finds all files containing lines similar to the input criteria. The search approach itself is similar to the one discussed in [2].

Post-processing. As part of the hybrid search approach, SeClone performs both clone pair clustering and grouping during its post-processing analysis. In this context, we define *clone pair clustering* as an approach that categorizes a set of clone pairs based on a specified criterion other than pattern similarity. Clone pair clustering differs from *clone pair grouping* [1], a more commonly used approach, by addressing the classification of corresponding clone pairs into larger groups.

Code pattern index-based clone search tends to produce false positives, as it uses transformed code which has all Java types replaced by a general symbol to increase the recall rate. To overcome this deficiency, SeClone uses its *Type Usage Index* (a complementary information source to the pattern index) and an *information retrieval clustering* technique. The goal is to create clusters that contain either true positives or false positives.

We also use a Semantic Web reasoner to perform a novel *grouping* task of corresponding clone pairs. Finally, the output clone pairs will be reported to the user using our Clone Ontology (*CLON*). CLON models the common vocabulary in the clone detection domain, such as clone types, location, similarity properties etc.

III. SeClone's Clone pair Clustering

Applying clone detection on an Internet-scale repository will more likely create scenarios where one has to deal with a large number of returned results. As discussed earlier, one approach to deal with large result sets is to categorize them based on some common attributes (e.g., time) to facilitate result browsing for end users. Our tool suggests an approximate result clustering based on file-level type usage maintained as part of the SeClone's indices. It is implemented using the Suffix Tree Clustering (STC) algorithm [6]. A sample query and the grouping of all highly similar patterns into clusters (e.g., Cluster#1) are shown in Fig. 2. This approach can improve precision by reducing the number of reported false positives.

In Java, *import* statements are used as a formal means to express type usage. Based on this observation, one can now approximate accessed types within a clone pair through the corresponding import statements used. Increasing the granularity of type access at the method level allows for a decrease of our index size to one-eighth (based on our statistical analysis). This reduction allows us to keep our indices in main memory to improve response time. We addressed ambiguities in import statements (e.g., on demand imports) or inheritance trees by using a technique known as *loose unqualified name resolution* [7].

Input Fragment (query)

```
for (Attribute
attribute:exampleSet.getAttributes())
```

Output

Cluster #1

```
1. for (Attribute
   attribute:exampleSet.getAttributes())
2. for (Attribute attribute:es1.getAttributes())
```

Cluster #2

```
3. for (AttributeEntity
   theAttributeEntity:aTableEntity.ge…
4. for (JAttribute
   attribute:formType.getAttributes()) {
5. for (IAttribute att:source.getAttributes()) {
```

Figure 2. A sample query and the matching code clones which are clustered based on accessed Java types.

IV. Conclusion

In this research, we introduced our SeClone clone search tool, a novel hybrid clone detection approach, which is based on multi-layer indexing. Our approach takes advantage of information retrieval clustering and Semantic Web reasoning to cluster and group clone pairs. A clone ontology (*CLON*) is introduced to model the code clone detection *vocabulary* to support the use of reasoning services and to provide a formal result sharing and integration approach. The ontology is available online at http://aseg.cs.concordia.ca/seclone. As part of future work, we plan to enrich our indexing system to further optimize precision and response times.

Acknowledgment

This research was partially funded by DRDC Valcartier (contract no. W7701-081745/001/QCV).

References

[1] C.K. Roy, J.R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of Computer Programming*, vol. 74, May. 2009, pp. 470-495.

[2] B. Hummel, E. Juergens, L. Heinemann, and M. Conradt, "Index-based code clone detection: incremental, distributed, scalable," *Proc. 32th IEEE International Conference on Software Engineering*, 2010.

[3] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder : a multilinguistic token-Based code clone detection system for large scale source code," *IEEE Trans. Software Engineering*, vol. 28, 2002, pp. 654-670.

[4] M. Würsch, G. Reif, S. Demeyer, and H.C. Gall, "Fostering synergies: how semantic web technology could influence software repositories," *Proc. ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, 2010.

[5] I. Keivanloo, C. Forbes, J. Rilling, and P. Charland, "Towards Sharing Source Code Facts Using Linked Data," *Proc. ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, 2011.

[6] O.E. Zamir. Clustering Web Documents: A Phrase-Based Method for Grouping Search Engine Results. PhD thesis, University of Washington,1999.

[7] I. Keivanloo, L. Roostapour, P. Schugerl, and J. Rilling, "SE-CodeSearch: A Scalable Semantic Web-based Source Code Search Infrastructure," *Proc. 26th IEEE International Conference on Software Maintenance*, 2010.