Leveraging Clone Detection for Internet-Scale Source Code Search

Iman Keivanloo
Ambient Software Evolution Group (ASEG)
Concordia University
Montreal, Canada
i_keiv@cse.concordia.ca

***Abstract*—Different approaches to search for source code on the Internet exist. In this paper, we propose techniques that support a special type of Internet-scale code search using two novel syntactical and semantic clone search and detection methods. The syntactical search focuses on providing a scalable real-time engine, while the semantic clone detection is being used to enrich our knowledge base during the offline processing step.**

***Index Terms*—code search, clone detection, Semantic Web.**

I. Introduction

Internet-scale code search is the process of searching over source code available on the Internet to find pieces of working code fragments [1]. The intention of such a search is typically to reuse a code fragment as reference sample or to complete an implementation task. Similar to the Web search, for this type of code search, (1) *scalability* and (2) *response time* are major requirements while recall and precision are less important.

Structural Internet-scale code search engines (e.g., Sourcerer [2]) are designed to provide fine-grained precise structural queries (e.g., SQL) over large amounts of source code found on the Internet. These engines are implemented on top of a relational or graph database. However, due to the sheer data volume and query complexity, these code search engines sometimes fail to answer at real-time (e.g., a few seconds). Furthermore, developers are also required to provide when formulating their queries, detailed search criteria (e.g., class and method names).

To address the two enumerated requirements and overcome the shortcoming of pure structural code search approaches, we propose in this research an alternative search strategy (Fig. 1). The presented approach is based on a two-step querying process, which uses instead of fully specified complex structural queries (e.g., fully qualified class names have to be provided as part of the query), a much simpler structural query to derive an initial result set. As part of the second step, a user browses the preliminary result set and selects a relevant record (code fragment). The selected record (code fragment) constitutes then the input for the second step, the real-time clone search. In other words, we engage real-time clone search within the Internet-scale code search process to address the performance and query complexity challenges. This two-step approach has the following advantages, (1) it eliminates the need for the user, to know the complete search criteria in advance (2) and queries can be performed in real-time since they are simple lookup queries rather than complex queries that require functions such as SQL Joins.

II. Research Objectives, Approach, and Constraints

Figure 1 illustrates the proposed search approach. Our research objective is to develop search algorithms that are required to support the search approach presented in Fig. 1. First, a user will trigger a simple structural code search query with some of the search criteria (not all of them). The structural code search engine is responsible to provide a preliminary result set. Second, due to (1) query simplification and (2) potentially large number of matches, further processing steps are necessary. The next step in our proposed approach involves therefore the utilization of clone detection techniques. The input is (1) a code fragment (see Fig. 1 step 1) and (2) a target line that matches the relevant functionality in the result set from the structural query. In this case the selected item (i.e. the input) is a code fragment that the user finds a close match to the expected ideal answer. The user then selects the item which is closest to the expected answer, which becomes the input to the second step of the search process (see Fig. 1 step 2), with the objective to find similar fragments. We have selected this search strategy since we believe that providing a "real-time *semantic-aware* clone search" will allow the user to obtain the expected answer after the second search step (i.e., activity) addressing all enumerated requirements.

The objective of this research therefore is to investigate if a *semantic-aware* clone search platform can be derived that provides results consistently in real-time (i.e., fractions of a second) while being scalable to large corpora (i.e., billions of LOC). For our approach presented in Fig. 1, a clone search algorithm is required that meets four major requirements: scalability, real-time response-time, scalable incremental repository updates, and the support of several clone types (i.e., type-1, 2, 3, and semantic clones [3]).

Moreover, in order to take advantage of clone search during the source code search, first, the clone search (i.e., syntactical component) granularity should be ideally at the line level to be able to match directly the target (input) line. Second, it (i.e., semantic clone detection component) must be optimized for small size cloned fragments such as method level.

978-1-4673-1216-5/12/$31.00 © 2012 IEEE

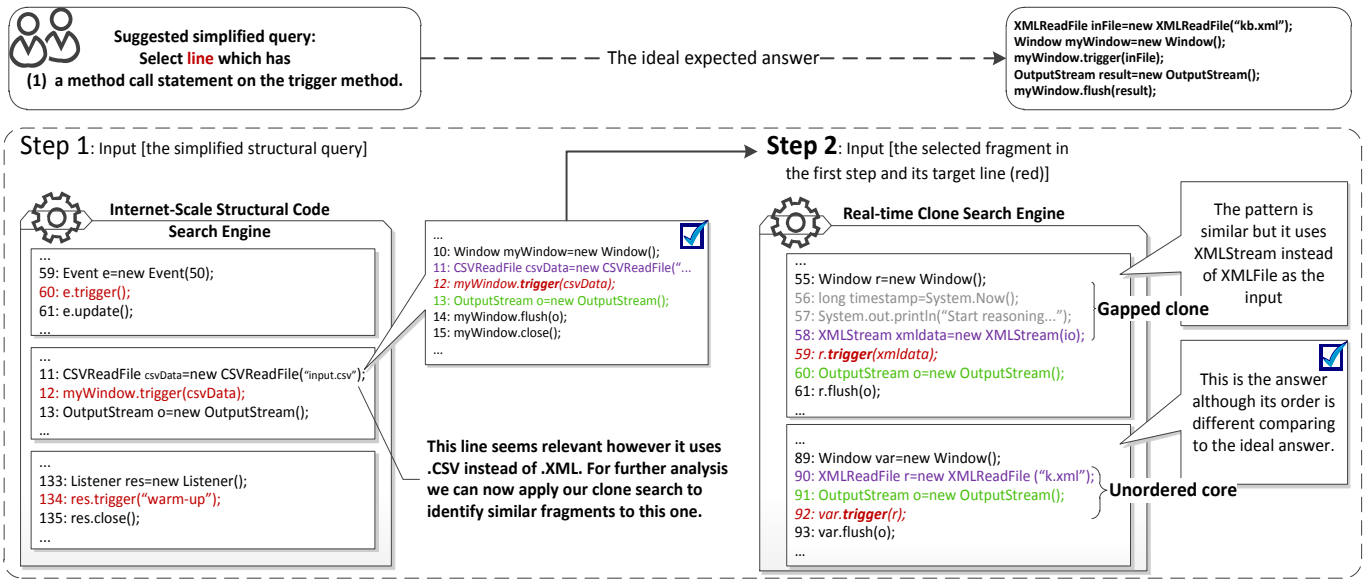

Fig. 1. Leveraging real-time and semantic clone search/detection to support Internet-scale code search

A. Research Approach

The presented research approach involves four major steps. First, identify and establish a logic-based source code sharing and search platform that can support our search requirements. Second, a large data set containing 18,000 open source Java projects was identified and various characteristics of source code patterns within the data set were analyzed to determine appropriate *granularity levels* for real-time Internet-scale clone search. Third, a syntactical clone search approach was designed and implemented to evaluate its applicability in the context of real life Internet-scale data. In addition an experimental performance evaluation of our clone search approach has been conducted, to validate if real-time clone search can support Internet-scale code search. The fourth step involved a further improvement to the quality of the search approach, by taking advantage of Semantic Web technologies to model and analyze source code semantics.

B. External Research Constraints and Preferences

We provided a more detailed discussion [4] on how the adoption of Semantic Web in source code analysis and the openness of the modeling approach can address current drawbacks in the source code analysis domain [5]. Hence, we consider taking advantage of the Semantic Web and its supporting technologies as an external constraint of our research.

III. Our Open Source Code Knowledge Base - SeCold

As stated earlier, addressing openness in the form of providing both data and analysis knowledge in a standard machine understandable format, is a key objective of our research. We addressed this issue from the beginning by creating the SeCold knowledge repository [6, 7, 8], a large data set containing 18,000 open source Java projects.

SeCold takes advantage of the very first layers of the Semantic Web platform, utilizing the provided formalism and basic reasoning functionality. Our major contribution in this part of the research are the idea of Reproducible Identifiers and the SeCold Ontology family. The repository is available online at http://secold.org and it is an official member of Linking Open Data Cloud [9].

Structural Code Search. Figure 1 illustrates how the structural query approach contributes to the early steps of our search scenario. This structural query approach was implemented as SE-CodeSearch [10, 11], a Semantic Web-based structural code search engine. The engine covers the very first search process step in Fig. 1 using graph-based querying languages (i.e. SPARQL) and Description Logic (i.e., OWL DL) formalism which are parts of Semantic Web platform.

IV. Scalable Real-time Syntactical Clone Search

For designing and implementing the real-time clone search engine (which constitutes the core approach of the second step in Fig. 1), we applied a three-step research process which resulted in SeClone [12, 13]. SeClone is one of the fastest and most scalable clone search engines available. Thus it addresses two of our major research objectives, to provide a (1) scalable clone search engine with (2) short latency time.

Since data characteristics affect the real-time performance, we conducted first a set of empirical studies [12, 13] to determine typical code patterns distribution across a large Internet scale source code corpus, containing approximately 1.5 million Java class files. Our analysis included (1) unique patterns distribution (2) pattern frequencies and (3) 32-bit hashing strength for code indexing using 1.5 million Java files. Second, we design our clone search approach based on the first step result. In the final part of our evaluation, we investigated the actual performance (response times) of our *syntactical* real-time clone search [12, 13].

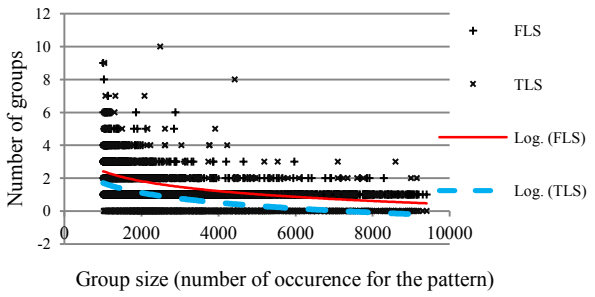

Fig. 2. TLS and FLS [13] outlier groups distribution

For example, Fig. 2 shows part of our statistical analysis. According to this analysis, if one considers the FLS distribution (the solid red line), she notices that the number of FLS's groups with more than 2,000 occurrences (outliers) is significantly larger compared to the ones in the TLS's. While the TLS has only a few in the 1-level group, the FLS has a significant number of outliers. In general, these analyses gave us insight about characteristics (e.g., pattern frequency) of source code available on the Internet which helped us to design and implement SeClone successfully.

V. Semantic Clone Detection at Binary Level

As discussed earlier, we are not only interested in similar code fragment from a pattern but also from a semantic point of view (i.e., Fig. 1 step 2 - finding semantically similar/dissimilar fragments). As a result, we proposed a novel method [14, 15] for semantic clone detection at Java binary (i.e., bytecode) level. The motivation for investigating clone detection at binary level is based on some of earlier studies [16], where they showed that it is possible to detect clones at the binary levels which are not detectable at the source code level or missed by other clone detection techniques [14]. Consequently, the binary approach is a vital complementary module in our code search framework. In the following, we review the three key features of binary clone detection approach called SeByte [14, 15].

A. Relaxation on Code Fingerprint

The state of the art in clone detection is to compare the full source code representation (e.g., abstract syntax tree). Our first heuristic is to filter out the input data based on its type of token. For example, in the current status, we only fingerprinted method call and Java type tokens. We enumerated the motivations to include this heuristic comprehensively earlier [14] such as increasing scalability.

B. Multi-Dimensional Comparison

The second heuristic is to compare each type of information separately. Therefore, SeByte compares first method call token sequence (i.e., the first dimension) for each two code fragments. Independently, it also compares Java type token sequence (i.e., the second dimension). We believe this heuristic helps SeByte to survive in case of high pattern dissimilarity which is important for semantic clones.

C. Metric-Based Clone Detection

The first two heuristics direct us to adopt the third heuristic which is the *metric-based clone detection*. It means that we (1) calculate similarity value for each dimension independently and (2) consider it as a contributing metric to the final decision making formula. The final decision making formula sums up all metrics and if the result is higher than a specific threshold, it detect the candidate pairing as a clone pair.

The interesting part of SeByte is its comparison functions for input dimensions. Interestingly, it has two comparison functions (instead of one which is the state of the art), according to our requirements (Fig. 3 in [14]). Moreover, we selected two diverse similarity functions to increase our chance to detect semantic clones, which are discussed in the following.

1) Pattern Similarity - Transitivity via Semantic Web Querying and Reasoning

Although we are interested to detect *semantic clones*, we do care about pattern similarity as well to some extent [14]. Considering our *research constraints and preferences* (Section 2), we adopted Semantic Web for pattern similarity. Mostly, we rely on transitive closure computation using Semantic Web inference engines. For example, SeByte is able to detect fragments with large gaps (no limitation or threshold) as similar (e.g., $\langle t_1, t_2, t_3, \dots, t_{500} \rangle$ and $\langle t_1, t_2, t_{500} \rangle$) when we enable the inference engine within SeByte query engine which accepts SPARQL queries.

2) Content Similarity - via Jaccard Coefficient

Since semantic similarity between two fragments is important to us, we adopted *Jaccard Coefficient* (1) as well. *Jaccard Coefficient* compares two sets (regardless of the ordering and repetition) and returns their similarity degree. We found this function also useful to detect semantic clones during our preliminary observations [14].

$$J(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|} \qquad (1)$$

VI. Shuffling and Randomization is Useful to Achieve Scalability

Scalability was another major requirement in our research project. Earlier, we have observed [12, 13] that it is possible to provide scalable syntactical clone detection and search. Semantic clone detection techniques (e.g., our SeByte [14, 15]) are not scalable in most cases due to high computational complexity (e.g., $O(n^2)$).

Although distributed recursive algorithmic approaches such as MapReduce is popular to address scalability issue in clone detection, we proposed a diverse method based on shuffling, randomization, and repetition. Our approach [17] uses a simplified divide and conquer paradigm with no feedback or recursion loop which can be applied on any clone detection technique. According to our preliminary studies [17] this approach is able to recover up to 85% recall (no precession loss) with 10 times repetition for large datasets.

VII. Open Questions

There are still several open research questions, which we have not yet addressed as part of our current research. In what follows we review two of them.

A. Usefulness

In our research we focused on providing a specific type of Internet-scale code search activity (Fig. 1). Moreover, our proposed approach have been designed considering some external research constraints and preferences such as being openness, knowledge-based (i.e., Semantic Web) code search. It is unclear to us if such novel type of search (including similar research projects e.g., [18]) will be adopted by software development and maintenance community considering the numerous internal and external factors.

B. Open Source License

Since the major source of information for us is open source code available on the Internet, incompatibility between open source licenses [19, 20] hinders the future of such approach. We discussed [21] the importance and challenges of this issue specifically how it affects us by several fold since our infrastructure is based on Linked Data.

VIII. Future Work

As future work, we plan to, first include inheritance tree semantics for Object Oriented code into our semantic clone detection approach according to our Semantic Web-based code analysis approach [4] and our formal solution [22]. Second, enrich our knowledge repository (i.e., SeCold [6, 7, 8]) with clone facts extracted from other clone detection tools. Obviously, since some of them are not scalable to our code repository, we use our shuffling idea [17]. Third, we study the search engine performance in practice using a controlled user study in terms of speed and accuracy (e.g., considering only the top 10 result sets are displayed) after completing our search plug-in [23].

IX. Conclusion

In this research, we designed two clone search and detection approaches for Internet-scale code search that provides results consistently in real-time (fractions of second) using our Semantic Web-based platform while being scalable to large corpora (millions of files). Moreover, we provided (1) an open knowledge base for source code information called SeCold [6, 7, 8] (http://secold.org), and (2) a novel approach to increase the scalability of clone detection tools for offline processing which helps us to enrich our SeCold knowledge base in long term.

Acknowledgment

I would like to thank my advisor, Dr. Juergen Rilling for his invaluable support. I also thank Dr. Chanchal K. Roy for his kind collaboration.

References

[1] R. E. Gallardo-Valencia and S. Elliott Sim, "Internet-scale code search," ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation (SUITE), 2009, pp. 49-52.

[2] S. Bajracharya, J. Ossher, and C. Lopes, "Sourcerer: An internet-scale software repository," ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation (SUITE), 2009, pp. 1-4.

[3] 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, no. 7, May 2009, pp. 470-495.

[4] I. Keivanloo, J. Rilling, and P. Charland, "Semantic Web - The missing link in global source code analysis?," 36th IEEE International Computer Software and Applications Conference (COMPSAC), 2012.

[5] D. Binkley, "Source code analysis: A road map," Future of Software Engineering (FOSE), 2007, pp. 104-119.

[6] I. Keivanloo, C. Forbes, and J. Rilling, "Towards sharing source code facts using linked data," 3rd International Workshop on Search-driven Development: Users, Infrastructure, Tools, and Evaluation (SUITE), 2011, pp. 25-28.

[7] I. Keivanloo, "Online sharing and integration of results from mining software repositories," 34th International Conference on Software Engineering (ICSE), ACM–SRC Track, 2012.

[8] I. Keivanloo, C. Forbes, A. Hmood, M. Erfani, C. Neal, G. Peristerakis, and J. Rilling, "A linked data platform for mining software repositories," 9th Working Conference on Mining Software Repositories (MSR), Short paper track, 2012.

[9] R. Cyganiak and A. Jentzsch, "Linking Open Data cloud diagram," http://lod-cloud.net/. Jan. 2012.

[10] I. Keivanloo, L. Roostapour, P. Schugerl, and J. Rilling, "SE-CodeSearch: A scalable semantic web-based source code search infrastructure," 26th IEEE International Conference on Software Maintenance (ICSM), 2010.

[11] I. Keivanloo, L. Roostapour, P. Schugerl, and J. Rilling, "Semantic web-based source code search," 6th International Workshop on Semantic Web Enabled Software Engineering (SWESE), 2010.

[12] I. Keivanloo, J. Rilling, and P. Charland, "SeClone - A hybrid approach to Internet-Scale real-time code clone search," 19th IEEE International Conference on Program Comprehension (ICPC), 2011, pp. 223-224.

[13] I. Keivanloo, J. Rilling, and P. Charland, "Internet-scale real-time code clone search via multi-level indexing," 18th Working Conference on Reverse Engineering (WCRE), 2011, pp. 23-27.

[14] I. Keivanloo, C. K. Roy, and J. Rilling, "Java bytecode clone detection via relaxation on code fingerprint and semantic web reasoning," 6th International Workshop on Software Clones (IWSC), 2012.

[15] I. Keivanloo, C. K. Roy, and J. Rilling, "SeByte - A semantic clone detection tool for intermediate languages," 19th IEEE International Conference on Program Comprehension (ICPC), 2012.

[16] G. M. K. Selim, K. C. Foo, and Y. Zou, "Enhancing source-based clone detection using intermediate representation," 17th Working Conference on Reverse Engineering (WCRE), 2010, pp. 227-236.

[17] I. Keivanloo, C. K. Roy, J. Rilling, and P. Charland, "Shuffling and randomization for scalable source code clone detection," 6th International Workshop on Software Clones (IWSC), Short paper, 2012.

[18] C. McMillan, "Searching, selecting, and synthesizing source code," 33th International Conference on Software Engineering (ICSE), 2011, pp. 1124-1125.

[19] D. M. Germán, M. Di Penta, and J. Davies, "Understanding and auditing the licensing of open source software distributions," 17th IEEE International Conference on Program Comprehension (ICPC), 2010, pp. 84-93.

[20] J. Krinke, N. Gold, Y. Jia, and D. Binkley, "Distinguishing copies from originals in software clones," 4th International Workshop on Software Clones (IWSC), 2010, pp. 41-48.

[21] C. Forbes, I. Keivanloo, and J. Rilling, "When open source turns cold on innovation – the Challenges of navigating licensing complexities in new research domains," 34th International Conference on Software Engineering (ICSE), Poster Track, 2012.

[22] I. Keivanloo and J. Rilling, "Clone detection meets semantic web-based transitive closure computation," First International Workshop on Realizing AI Synergies in Software Engineering (RAISE), 2012.

[23] I. Keivanloo, C. Forbes, and J. Rilling, "Similarity search plug-in: clone detection meets Internet-scale code search," 4th ICSE Workshop on Search-Driven Development: Users, Infrastructure, Tools and Evaluation (SUITE), Short paper track, 2012.