Pattern Mining for Systematic Code Changes

Kim Mens, Siegfried Nijssen, Hoang-Son Pham
ICTEAM, UCLouvain
Belgium
{Kim.Mens, Siegfried.Nijssen, Hoang.S.Pham}@uclouvain.be

Johan Fabry
Raincode Labs
Belgium
johan@raincode.com

Vadim Zaytsev
Universiteit Twente
The Netherlands
vadim@grammarware.net

Software developers repeatedly perform similar but non-identical changes to a system's source code. Such groups of similar systematic code changes are performed for various reasons: adapting code to a changed API, migrating to a different library, refactoring to improve code quality, performing routine code maintenance tasks, fixing multiple manifestations of the same bug, implementing or modifying crosscutting concerns, managing code clones, making similar changes to multiple branches of a system, etc. While developers may have some notion of the systematic changes they performed in their own code, it requires significant effort to manually locate existing systematic code changes in projects they do not know. Consequently, an automated approach for locating such systematic code changes between versions is required.

To automatically discover unknown systematic code changes between different versions of a system, various approaches have been proposed, which are often based on data mining techniques: first creating a database of changes, then mining for patterns in this database. For instance, Č. Gerlec et al. [1] apply standard data mining approaches to source code and source code changes. H. A. Nguyen et al. [2] mine source code changes at the API usage level, extracting an API usage graph for two versions of a system, and using frequent itemset mining to look for migration patterns. T. Molderez et al. [3] also apply data mining techniques to discover source code changes, based on a closed itemset mining algorithm to find patterns in a database of changes.

In our work, to mine for systematic source code changes, we propose a new approach using a frequent tree mining algorithm. A significant difference of our approach compared to the approaches above is that it mines for source code changes directly from abstract syntax trees (ASTs) instead of using a database of changes. As a result, the discovered patterns can include any kind of structural source code changes and are not limited to the changes occurring in the database of changes. In addition, our approach is able to mine for patterns from two versions (or two commits) of a system that could have hundreds or thousands of commits in between.

To mine for source code changes, we extend our original FREQTALS pattern mining algorithm, which was specifically designed for analysing code repositories for frequently occurring code patterns [4]. The main difference is that now we apply it to mine for significant patterns of code changes *between* versions. FREQTALS is a constraint-based tree mining algorithm that combines two approaches: (i) *maximal frequent subtree mining* to ensure that a condensed representation of only large patterns is found, (ii) *constraint-based data mining*, in which additional constraints can be imposed on the patterns to be found. In the adapted version of our algorithm, the input data is considered to be supervised, where each class corresponds to one version of the code. The goal is to discover code fragments that occur more frequently in one collection than in the other. To evaluate the difference of occurrences of the pattern in the two classes, we use a χ^2 measure. Intuitively, the higher the χ^2 score of a pattern, the more interesting it is.

We evaluated our algorithm on various systems written in Java and Python. For Java we selected 4 medium size open source projects: ANTLR, Checkstyle, JGraphX and JHotDraw. For each project, we chose two versions having hundreds or thousands of commits in between them. The algorithm finds a high proportion of patterns that are interesting, representing source code fragments which changed over the two versions. To group similar patterns together, a variety of clustering algorithms is used.

For mining Python code, we adopted a slightly different experiment. We compare similar code of different groups of students, collected from an online exam system. The first group consists of submissions having obtained a score of 50% or more at the exam; all other submissions are put in the second group. Our algorithm discovered a large amount of patterns that occur more frequently in one group than in the other. Such patterns are representative of good solution strategies to an exam question, or of misconceptions in the solutions of students who failed the question.

These experimental results do show that our approach is able to automatically discover systematic code differences between two versions of a system. To further evaluate the quality of our change pattern mining algorithm we need to run the algorithm on more and larger systems and to compare the results to alternative mining approaches.

References

[1] Č. Gerlec, A. Krajnc, M. Heričko, and J. Božnik, "Mining Source Code Changes from Software Repositories," in *CEE-SECR*, 2011, pp. 1–5.

[2] H. A. Nguyen, T. T. Nguyen, G. Wilson, A. T. Nguyen, M. Kim, and T. N. Nguyen, "A Graph-Based Approach to API Usage Adaptation," in *OOPSLA*. ACM, 2010, p. 302–321.

[3] T. Molderez, R. Stevens, and C. De Roover, "Mining Change Histories for Unknown Systematic Edits," in *MSR*. IEEE, 2017, pp. 248–256.

[4] H. S. Pham, S. Nijssen, K. Mens, D. D. Nucci, T. Molderez, C. D. Roover, J. Fabry, and V. Zaytsev, "Mining Patterns in Source Code Using Tree Mining Algorithms," in *Discovery Science*. Springer, 2019, doi:10.1007/978-3-030-33778-0_35.