Detecting Collaborative Patterns in Programs

Kuldeep Kumar
Advisor: Stan Jarzabek
School of Computing, National University of Singapore
kuldeep@comp.nus.edu.sg

I. PROBLEM DESCRIPTION

In the thesis, we aim at detecting software similarity patterns in the programs called collaborative patterns that have not been addressed in the research on software clones so far.

Definition: Collaborative pattern is defined as a recurring configuration of program entities (e.g., classes or methods) inter-related by means of calling relationships (method calls or message passing). In these configurations, the corresponding entities should be similar to each other based on some selected similarity metrics. The selected similarity metrics may be based on the textual, functional, structural, or logical similarities among program entities.

Example: Consider six program entities A, A', B, B', C, and C' as shown in Fig. 1 by rectangular boxes. Given that, boxes of the same shade are code clones (e.g. method clones, or class clones) of each other, and arrows indicate calling relations among the corresponding program entities. The recurring configurations of program entities as shown in the figure form a collaborative pattern.

II. PROPOSED METHODOLOGY

Fig. 2 gives brief overview of our approach for detecting collaborative patterns. We find code clones first and based on them detect large granular collaborative patterns in three steps: pre-detection analysis, collaborative pattern detection and post-detection analysis with user involvement. In Step 1, we detect code clones from the source code. Also, we instrument the source code with hooks to generate method execution traces. Code clones and method execution traces are then used for detecting collaborative patterns (Step 2). We find recurring patterns in method execution traces as candidates for collaborative patterns. We adapted a token-based string pattern matching algorithm for that. Having found collaborative method patterns, we gradually raise the level of detection from methods to classes, files and directories, using the same principle and detection techniques. Automated detection may result in many collaborative patterns in large software systems. Post-detection analysis of the detected collaborative patterns is helpful in focusing users’ attention to the areas of their interest (Step 3).

III. EXPECTED CONTRIBUTION

Proposed work extends the clone research from simple clones—similar code fragments—to the large-granular software patterns. Collaborative patterns are large-granular clones. They signify the use of standardized solutions and/or repetitions that arise naturally in the software analysis or design space. As such, collaborative patterns often embody important design information. Many simple clones are grouped around high-level repetitions [1]. Therefore, collaborative patterns form a convenient conceptual window for developers for understanding the overall cloning in the system. Hence, the proposed methodology for collaborative pattern detection will enhance the value of similarity analysis in software maintenance and in the process of re-engineering for reuse that involves finding the candidate modules for reuse in legacy code.

IV. APPLICATIONS AND CONCLUSION

All the traditional clone detection techniques, in general, help in program understanding, reuse, plagiarism detection, software evolution, maintenance, and others. In addition to all these traditional benefits, some of the other benefits that signify the impact of the proposed work are change impact analysis, finding library candidates, code compaction, and creating generic representations. The detected units are big enough to exhibit conceptual similarities that help in better understanding of the cloning in the software system. These units are large enough to be considered for incorporating them in the libraries. We also expect better code compaction due to bigger units of detected clones. Also, collaborative patterns can help in creating generic representation for the entire system using variant configuration languages [2][3].

Acknowledgements: I sincerely thank my advisor, Dr. Stan Jarzabek for his guidance during my PhD study.

REFERENCES