An Analysis of cpp Preprocessor-Based Software Product Lines

Jörg Liebig

Department of Informatics and Mathematics
University of Passau, Germany
joerg.liebig@uni-passau.de

1 Introduction, Motivation, and Goal

A *software product line* (*SPL*) is is a set of software intensive systems that are tailored to a specic domain or market segment and that share a common set of features [3]. A *feature* represents a requirement according to the given domain, that is of interest to end-users. Although the research on SPL is still quite new, techniques and tools have been proposed for a long time. One tool that is available for more than 20 years is the cpp (The C Preprocessor).

The cpp is an extension to the programming language C, adding additional capabilities for meta programming. It has been observed that the cpp is quite common and used heavy when implementing software systems in C [2]. Nevertheless to our knowledge very little is known about cpp-based feature implementations and software product lines. Hence, we explore foundations for implementing SPLs with the cpp and give an overview of the cpp usage in open source software systems. To this end, we present metrics for analyzing 40 open source software projects regarding SPL implementation techniques. Using these metrics we are able to measure the kind, granularity, and complexity of cpp-based feature implementations.

Based on the statistics collected in our analysis, we discuss the possibility to refactor cpp-based feature implementations using alternative feature implementation mechanisms like aspects [4] or feature modules [1]. The statistics of our analysis are promising to give a complete overview about possible refactorings and are of interest to language designers and tool writers.

2 Implementing SPL with cpp

The cpp is a text-processing tool, which works on the basis of directives. These directives are part of the source code and provide capabilities to include the content of other files, to define macros for text-based substitution and to express conditional inclusion of source code fragments. We illustrate how, with a subset of these directives, programmers are able to implement features and we highlight connections to the fundamental concept of software product line engineering. Figure 1 shows an excerpt of a list SPL. The #define SORTALGO (Line 3) for

```
#define INSERTIONSORT 0
#define BUBBLESORT 1
#define SORTALGO INSERTIONSORT

void insert(node* elem) {
#if SORTALGO == BUBBLESORT || SORTALGO == INSERTIONSORT
  node *a = NULL;
  node *b = NULL;
#endif ...
```

Fig. 1. List example with sorting feature

instance marks a feature with the possibility to select between different sorting algorithms.

With the knowledge about feature implementations on the basis of the cpp, we setup our analysis to derive the information about the cpp usage.

3 Analysis, Discussion, and Conclusion

To get an overview of the cpp usage and to discuss alternative implementation techniques, we define a set of code metrics, which represent the kind, granularity, and complexity of cpp-based feature implementations. The metrics comprises the number of features, the lines of code attached to a feature, the behavior of an extension, and the granularity of an extension (where does the extension appear in source code and which extensions are made at this point).

We have developed a tool called cppstats, measuring the metrics listed above for 40 software systems from different domains including operating systems, database management systems, libraries, and application software. Early results reveal that most features add a new variable or function at global scope of a file. Additionally, features extend the body of functions, conditions, or loops. In contrast we rarely found the extension of statements, conditional expressions, or method signatures. We believe that the results of our analysis also apply to other software systems using the preprocessor, as the analyzed software systems are influenced both from academic and industrial background.

References

1. D. Batory et al. Scaling Step-Wise Refinement. *IEEE Transactions on Software Engineering (TSE)*, 30(6):355–371, 2004.
2. M. Ernst et al. An Empirical Analysis of C Preprocessor Use. *IEEE Transactions on Software Engineering (TSE)*, 28(12):1146–1170, 2002.
3. K. Kang et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Carnegie-Mellon University Software Engineering Institute, Pittsburgh, PA, USA, November 1990.
4. G. Kiczales et al. Aspect-Oriented Programming. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP)*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Verlag, 1997.