# Software Product Line Analysis and Detection of Clones

Ritika Rani
*Dept. of Computer Science and Engineering*
*HGPI*
*Himachal Pradesh, India*

Er. Pooja Rani
*Dept. of Computer Science and Engineering*
*HGPI*
*Himachal Pradesh, India*

***Abstract*— In this thesis software product lines are an important aspect. We introduce the main concepts such as Software Product Line Engineering (SPLE), variability modeling or implementation approaches for SPLs. we Firstly explain code cloning in software product lines. In particular, we emphasize to what extent code clones occur in SPLs and whether differences exist regarding the implementation approach of SPLs. Furthermore, we provide some characteristics of these clones as a first step towards managing such clones proactively or even avoiding them in future. Second, we present a first approach of how to remove code clones in software product lines by applying refactoring. While this is a common and well-explored approach in standalone programs, refactoring is a non-trivial task in the presence of variability. In particular, we present how to find clone refactoring candidates and how to take variability into account during the refactoring process. Before all approaches the Software Clones and Detection, Analysis, and Management of software clones explained.**

***Keywords*—Software product line, software clones.**

## I. Introduction

To – Code clones have been recognized to be the most intrinsic and worst code smell in software systems. Indeed, a multitude of studies account for the existence of code clones in such systems. Generally, they are used in a copy, paste adapt fashion to reuse existing part of the source code. Recently, Software Product Lines (SPLs) have been proposed as a more structured approach for reusing source code artifacts (as well as non-code) amongst similar, variable software systems. To this end, different languages, paradigms, and implementation approaches have been proposed that partially overcome problems of current approaches for implementing highly variable and customized software systems. This thesis focuses on analysis of software product lines with respect to code clones. In particular, we investigate whether clones exist and how to characterize them in software product lines, depending on the respective implementation approach. Furthermore, we propose a first approach for code clone removal in SPLs by means of refactoring. Replicated code fragments, commonly referred to as code clones, have been subject to intensive research for over two decades. Since they play a pivotal role in the process of software maintenance, considerable effort has been expended to analyze when and how code clones negatively influence software quality and maintenance. Most commonly, researchers report about inconsistent changes and propagating and introducing errors as the main drawbacks of code clones for software quality Additionally, increased code size and multiple modifications for one change request impede maintenance of the software systems as well. In contrast, recent studies express doubt on the longstanding sentiments about the harmfulness of clones. In particular, they show that code cloning is used as kind of implementation concept such as templating or forking and that clones are relatively stable with respect to changes. However, while code clone research mainly focuses on general purpose (monolithic) software systems, software development changed from single programs to program families in recent past. To this end, software product line engineering provides means to develop a set of related systems from a common code base. The different programs (also called variants) that are part of the resulting SPL can be described by their commonalities and variability's in terms of features. Consequently, a particular variant of a software product line can be derived by selecting the respective features. Although it is still a quite new way of developing software systems, the product line approach has been adopted by industrial as well as open source systems and it is expected to increase in the future.

## II. RELATED WORK

### *Software Clones (Detection, Analysis, and Management)*

Software clones, that is, the replication of code fragments also known as code clones .we investigate how and why code clones occur. To get an idea of what is a clone, we introduce different types of clones, as detected by current tools, clone detection is the process of finding code fragments that are similar to each other. Within this thesis, we mainly focus on syntactical similarity. We give an overview of existing clone detection techniques and we analysis where the code occurred. Finally, the treatment of clones, called clone management, is an important aspect in code clone research.

### *Type-I Clones*

Code fragments that are identical are called Type-I clone. Only one difference is to be occurred related to formatting such as given comments and whitespaces are allowed in this type of cloning.

*Type-II Clones*

Type-I clones are easy to detect with simple tools, they are not very common. Instead, a common pattern of cloning is Copy Paste-and-Modification, which leads to Type-II clones.

*Type-III Clones*

Type-III clones go even one step further than Type-II clones in the way that they additionally allow to modify or deleted the statements. Then deleted statement from one code fragment is to be inserted into another code fragment. We treat both terms (deleting and adding statements) synonymously

*Type-IV Clones*

We introduce this category just for completeness, though this type of clones does not fall into the category of syntactical clones, Type-IV clones is to be syntactically different In this type of cloning the relation for this clones is to be semantically similar with more than one code fragment there for its also called called semantic clones.

*Beyond Code Clones*

Recently, clone researcher put their focus on other artifacts that are different from source code. Nevertheless, all of these non-code artifacts are related to source code or to the overall software development process.

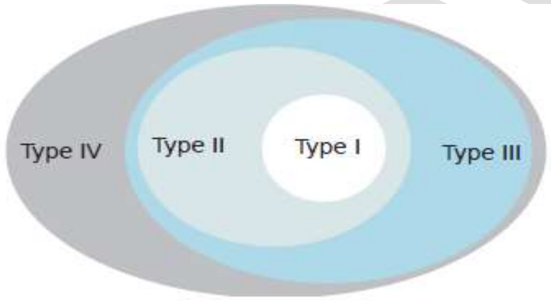


Figure: A Venn diagram, illustrating the relation between the different clone types

We also discuss Detection and Analysis of Clones and Clone Detection Techniques in our thesis .after all these basic notations we investigate code clones in SPLs and Software Product Line Engineering with Variability Modeling in this modeling Different approaches exist how to model the variability in SPLs such as Feature Models (FMs) , grammars , or propositional formulas .

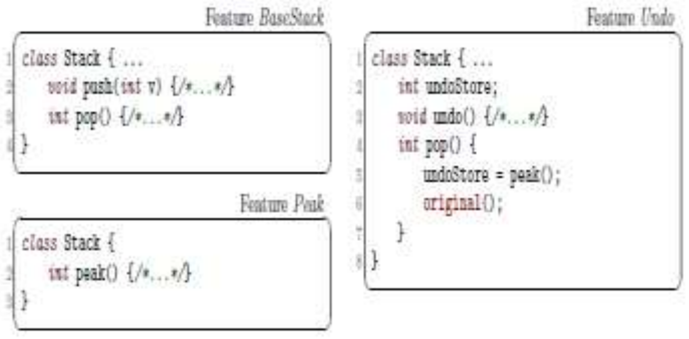
## III. IMPLEMENTATION APPROACHES FOR SPLs

While variability modeling defines the scope of an SPL during the domain analysis phase, the main concern of the domain implementation phase is the actual development of the reusable assets, defined in this phase. we define two categories: compositional and annotative implementation approaches.

Both approaches are used with the same goal in mind, they represent two opposite sides of the same idea. While annotative approaches, especially preprocessor-based ones, are mainly used in industry, compositional approaches gain momentum in academia.

## IV. REASONING ABOUT CODE CLONES IN SOFTWARE PRODUCT LINES

We discussed different reasons for the occurrence of code clones. Beside external reasons such as ad-hoc code reuse or time constraints, limitations of the programming paradigm itself may be a source of code clones. For instance, procedural programming languages may cause clones due to a lack of appropriate reuse mechanisms such as inheritance. Furthermore, in some languages such as COBOL, code replication is an accepted concept for templating. But even in object-oriented languages, existing mechanisms for abstraction such as inheritance or generics are not always sufficient for expressing variability in programs and thus contribute to code cloning. We define how expressing variability may cause code clones by means of our Stack product line



```
class Stack { ...
    void push(int v) {/*...*/}
    int pop() {/*...*/}
}
```

```
class Stack {
    int peak() {/*...*/}
}
```

```
class Stack { ...
    int undoStore;
    void undo() {/*...*/}
    int pop() {
        undoStore = peak();
        original();
    }
}
```

Figure Feature-oriented implementation of the Stack product line features Peak and Undo using Feature House.

## V. CONCLUSION

Software Product Lines provide facilities for efficiently managing thousand of (software) products at once by means of variability's and commonalities. Such an approach comes with different advantages such as fast time-to-market and reuse at large-scale. Hence, it plays a pivotal role for commercial success of software development. Consequently, SPLs gain momentum in both, academia as well as industry.

In research, major work on product lines encompasses implementing, testing, and verification. Furthermore, evolution of SPLs, especially of the problem space (e.g., variability models) is subject of research. In contrast, reengineering & maintenance (where clone detection and analysis belongs to) has not been subject of intensive research so far. However, we argue that software product lines evolve similar to single software systems or even more. As a result, maintenance and code quality become a problem. Due to the complexity of industrial SPLs, caused by different variability spaces, feature

Semantics etc., it is a challenging task to counter this evolutionary decays with common approaches. New approaches is to be specifically defined and mechanisms of SPLs have to be investigated to avoid some problems. With this thesis, we bridge this gap by tailoring clone analysis and removal to software product lines. In a broader sense, we aim at encouraging other researcher to put emphasis on this field of research. Our main contribution is to provide insights on code clones in SPLs (compositional and annotative) and how to remove them by the application of refactoring.

## VI. FUTURE WORK

For future work same as SPL we define Feature-Oriented Software Product Lines (FOSPL) To this end, we present an empirical analysis on different feature-oriented SPLs. In particular, we describe the setup, the methodology and results of analysis. Furthermore, we discuss the results and threats to validity.

For FOSPL we need basic concepts of SPL which are explained in our thesis.

## REFERENCES

[1]. L. Aversano, L. Cerulo, and M. Di Penta. How Clones are Maintained: An Empirical Study. In Proceedings of the European Conference on Software Maintenance and Reengineering (CSMR), pages 81{90. IEEE Computer Society, 2007.

[2]. T. Anderson and J. Finn. The New Statistical Analysis of Data. Springer-Verlag, 1996.

[3]. V. Alves, R. Gheyi, T. Massoni, U. Kulesza, P. Borba, and C. Lucena. Refactoring Product Lines. In Proceedings of the International Conference on Generative Programming and Component Engineering (GPCE), pages 201{210. ACM Press, 2006.

[4]. S. Apel and C. K□astner. An Overview of Feature-Oriented Software Development. Journal of Object Technology (JOT), 8(5):49{84, 2009.

[5]. S. Apel, C. K□astner, and C. Lengauer. FeatureHouse: Language-Independent, Automated Software Composition. In Proceedings of the International Conference on Software Engineering (ICSE), pages 221{231. IEEE Computer Society, 2009.

[6]. S. Apel, S. Kolesnikov, J. Liebig, C. K□astner, M. Kuhlemann, and T. Leich. Access Control in Feature-Oriented Programming. Science of Computer Programming, 77(3):174{187, 2012. Feature-Oriented Software Development(FOSD 2009).

[7]. S. Apel, T. Leich, M. Rosenm□uller, and G. Saake. FeatureC++: On the Symbiosis of Feature-Oriented and Aspect-Oriented Programming. In Pro-ceedings of the International Conference on Generative Programming and Component Engineering (GPCE), pages 125{140. Springer-Verlag, 2005.

[8]. S. Apel, T. Leich, and G. Saake. Aspectual Feature Modules. IEEE Transactions on Software Engineering (TSE), 34(2):162{180, 2008.

[9]. U. Dev A Study on the Nature of Code Clone Occurrence Predominantly in Feature Oriented Programming and the Prospects of Refactoring International Journal of Computer Applications, May 2016

[10]. http://www.ijcaonline.org/

[11]. http://dpt.kupin.de/

[12]. http://jrefactory.sourceforge.net/

[13]. http://www.eclipse.org/