Detection and Description of Variability Smells

Gustavo Vale, Eduardo Figueiredo (Orientador)

Programa de Pós-Graduação em Ciência da Computação (PPGCC),
Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG), Belo Horizonte – MG - Brasil

{gustavovale, figueiredo}@dcc.ufmg.br

Nível: Mestrado
Ano de ingresso no programa: Fevereiro de 2014
Época prevista de conclusão: Janeiro de 2016
Data de aprovação da proposta de dissertação: Março de 2015
Eventos relacionados: SBES, SBCARS

Abstract. *Software Product Line (SPL) is a set of software systems that share a common, managed set of features satisfying the specific needs of a particular market segment. The systematic and large reuse adopted in SPLs aim to reduce time-to-market and improve software quality. In spite of the benefits of SPLs and like any software, undesired properties may be present in all related artifacts, such as, source code and feature models. Undesired properties, in a general context, are called bad smells. Bad smells are symptoms that something may be wrong in system design or code. On the context of SPLs, bad smells are called variability smells. Variability smells is a relative new topic of research and need to be further explored. Additionally, variability smells are more useful when it is known to answer the three following questions: What are they? How to detect each of them (detection strategies)? How to solve each of them (refactoring methods)? We performed a Systematic Literature Review (SLR) to answer these three and other questions. As results of the SLR, we found 91 variability smells, detection strategies for 63 variability smells, 106 refactoring methods, and, 23 different SPLs in the context of bad smells and SPLs. The majority of detection strategies are based on logical combination of metrics. For this, it is necessary to know a method to derive software metric thresholds and, what metrics are proposed in the context of SPLs. Therefore, we are performing another SLR related to software metrics in the context of SPLs. We performed a comparison and proposed a method to derive thresholds after this comparison. As future plans, we want to propose variability smell detection strategies more effective than traditional ones and propose some variability smells to fill gaps found in the literature.*

Keywords: *Software Product Lines, Variability Smells, Detection Strategies, Refactoring Methods, Software Metrics, Thresholds*

1. Introduction

Software Product Line (SPL) is a set of software systems that share a common, managed set of features satisfying the specific needs of a particular market segment [Pohl et al., 2005]. The systematic and large-reuse adopted in SPLs aim to reduce time-to-market and improve software quality [Pohl et al., 2005]. The software products derived from an SPL share common features and differ themselves by their specific features [Pohl et al., 2005]. A feature represents an increment in functionality or a system property relevant to some stakeholders [Kastner et al., 2007]. The possible combinations of features to build a product, is called, SPL variability [Weiss and Lai, 1999] and, it can be represented in a feature model [Kang et al., 1990]. A feature model is a formalism to capture and to represent the commonalities and variabilities among the products in an SPL [Asikainen et al., 2006].

To develop an SPL, we can use different approaches, such as annotative and compositional [Apel et al., 2013]. For these approaches, we have several techniques, such as preprocessors, virtual separation of concerns, aspect-oriented programming, and feature-oriented programming. Those approaches and techniques aim to support configuration management at source code level and improve the software quality. In spite of that, undesired properties may be present in all related artifacts, such as source code and feature models [Apel et al., 2013]. Undesired properties, in a general context, are called bad smells. Bad smells are symptoms that something may be wrong in system design or code [Fowler et al., 1999]. On the context of SPLs, bad smells are called variability smells. A variability smell is a perceivable property of a product line that is an indicator of an undesired code property. It may be related to all kinds of artifacts in a product line, including feature models, domain artifacts, feature selections, and derived products [Apel et al., 2013]. Bad smells are extensively studied in single systems context, nevertheless, variability smells is still a young topic [Apel et al., 2013].

Variability smells are more useful when it is known: what they are, how to detect and how to solving them. Metric-based detection strategies and refactoring methods can be used to detect and solving variability smells, respectively [Fowler et al., 1999]. In addition, to use metric-based detection strategies it is necessary to know about metrics and define thresholds for each metric in which composes the detection strategies. This work aims to answer many questions related to these topics. And, for this we defined a strategy based on GQM (goal-question-metric) method [Basili et al., 1994]. In other words, we performed a task-oriented strategy. Figure 1 presents the nine tasks that we done, ongoing, or have to do in the Master degree period.

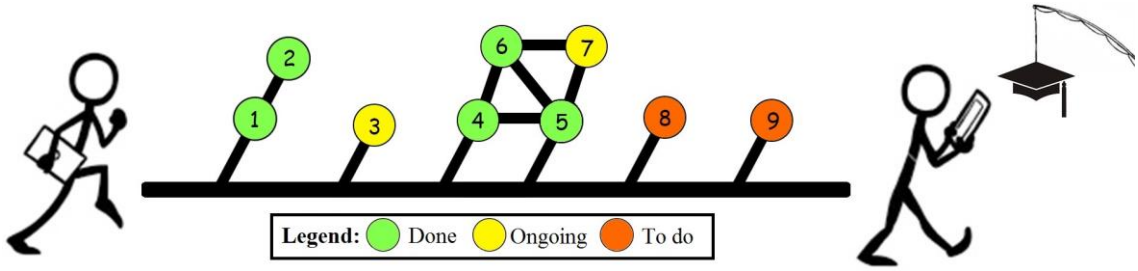

Figure 1. Tasks in Master degree Period

Tasks 1, 2, 3, and 4 are related to literature review and for this reason they appear first. The other tasks are more related to comparisons, evaluations, and proposals of something new. Tasks 1 and 2 provide a Systematic Literature Review (SLR) and an extension to know what are the variability smells, what are the variability smell detection strategies and what are the refactoring methods used to solve or minimize variability smells presented on literature. Based on the literature review, we noted that metric-based detection strategies are more common to detect variability smells. Nevertheless, the effectiveness of metric-based detection strategies is directly dependent on the definition of appropriate thresholds. Hence, we performed a search to find methods to derive thresholds. More than 10 methods were found, but we observed that the methods proposed recently are: (i) systematic, (ii) to consider the *skewed distribution* of the software metric, and, (iii) to receive as input data metrics from a benchmark of software systems. Therefore, we built a benchmark of feature-oriented SPLs (task 4).

Posteriorly, we performed a comparison of three methods to derive thresholds that match the three criteria cited above (task 5). With this comparison, we noted some desirable points in methods to derive thresholds, and as none of the found methods were completely fitted for our purpose, we got the desirable points listed and propose a method to derive thresholds (task 6). As three out of four methods, in spite to be systematic, they do not provide tool support we are developing a tool able to run these four methods to derive thresholds (task 7). At the moment, with the state of art of variability smells, we have plans to explore two points more. These points are related to propose better variability smell detection strategies in terms of effectiveness (task 8) and propose variability smells to fill gaps perceived in the literature (task 9). To achieve the tasks 8 and 9, we believe that it is necessary to know what metrics can be used to express variability mechanisms present in SPLs. For this reason, we are performing a SLR related to metrics proposed in the context of SPLs (task 3).

The rest of this paper is organized as follows. Section 2 presents the methodology and results of tasks that we done or ongoing. Section 3 describes how we pretend to evaluate the two tasks that we have to do. Section 4 presents a brief description of related work. Section 5 highlights the excepted contributions.

2. Literature Review and Thresholds Derivation

We performed a Systematic Literature Review (SLR) – 4 questions – [Vale et al., 2014] and after we extended this SLR – 2 questions more – [Vale et al., 2015a] to find and classify published work about bad smells and refactoring methods in SPLs. Our SLR aimed to answer the following questions: (RQ1) Is the definition of a bad smell the same in the context of SPL and single software systems? (RQ2) What were the SPLs used in studies of bad smells? (RQ3) What are the variability smells already defined and investigated? (RQ4) What are the strategies to identify variability smells? (RQ5) What are the refactoring methods applied to remove variability smells? (RQ6) What refactoring methods were used to minimize or solve an investigated variability smell?

The answers of these questions (task 1 and 2) were based on 22 relevant work found. We could see that bad smells is the general concept. Code and architectural smells are divisions of bad smells (types). Hybrid smells combine architectural and code

smells. Variability smells are bad smells specific to SPLs and can be divided in parts, such as architectural and code smells. Hence, the concept of bad smell is the same for single systems and SPLs (RQ1). We found 23 SPLs (RQ2), we identified 91 variability smells (RQ3), 7 types of variability smell detection strategies for 63 variability smells and 46 of the detection strategies are metric-based detection strategies (RQ4), and 106 refactoring methods (RQ5). Additionally, we matched 31 refactoring methods with 32 variability smells (RQ6).

Related to task 3, we are performing a SLR aiming to provide a catalogue of software metrics proposed in the SPL context. Related to task 4, we searched for feature-oriented SPLs in papers and well-known repositories. In total, we found 64 SPLs to compose our benchmark, in which 38 are different. The step-to-step filtering is further explained on the project website[1]. The benchmarks were built to task 5 in which we performed an evaluation of methods to derive thresholds. In the past few years, thresholds were calculated by software engineers experience and/or using a single system as a reference [Chidamber and Kemerer, 1994; Spinellis et al., 2008]. Recently, this concept has been changing and thresholds have been calculated considering three points [Alves et al., 2010; Ferreira et al., 2012; Oliveira et al., 2014]: (i) well-defined methods (systematics), (ii) methods that consider the skewed distribution of software metrics, and, (iii) derived from benchmarks.

We performed a comparison to find a method to define metric thresholds able to be used in detection strategies (task 5) [Vale et al., 2015b]. For this, we derived the thresholds using three methods [Alves et al., 2010; Ferreira et al., 2012; and Oliveira et al., 2014] found that follows the three points cited above. The thresholds were evaluated individually and by applying the thresholds in a detection strategy proposed to identify God Class in SPLs in terms of recall and precision. God Class is defined as a class that knows or does too much in the software system [Fowler et al. 1999]. As results, none of the methods excelled in the evaluations and we listed eight desirable points in methods to derive thresholds. The desirable points are: (i) to be systematic; (ii) to derive thresholds in a step-wise format; (iii) to be weak dependent with the number of systems; (iv) to be strong dependent with the number of entities; (v) not correlate metrics; (vi) to calculate upper and lower thresholds; (vii) to provide representative thresholds independent of metric distribution and (viii) to provide tool support. These desirable points were used as motivation for propose of a new method (task 6).

The proposed method tries to get the best of each method from the comparison and fits all desirable points, except the tool support which we are developing (task 7). The tool is being designed to run the three methods compared and the proposed method. To evaluate the derived thresholds from the proposed method, we choose a method used (not proposed) to derive thresholds in the context of SPL [Lanza and Marinescu, 2006] (baseline), though it does not consider the skewed distribution of software metrics. For the evaluation we followed similar steps to the previous evaluation, although we added an additional detection strategy to Lazy Class. Lazy Class is defined as a class that knows or does too little in the software system [Fowler et al. 1999]. The results show the proposed method is more effective to detect smells when compared with baseline.

---

[1] http://labsoft.dcc.ufmg.br/doku.php?id=%20about:spl_list

3. Detecting and Proposing Variability Smells

We aim to propose variability smell detection strategies more efficient than traditional ones (task 8) and new variability smells not yet described in the literature (task 9). For tasks 8 and 9, we think that it is better first to know what are the metrics proposed in the context of SPL and what is the mechanisms those metrics quantify (finish task 3).

Then, for task 8, we want to propose variability smell detection strategies and compare the effectiveness with traditional ones. For the evaluation, we are going to use the built benchmarks of feature-oriented SPLs and the proposed method to derive the thresholds for the metrics that we are going to use. The evaluation is expected to be in terms of effectiveness using recall and precision in target SPLs.

For task 9, we found some gaps in the literature and we use a strategy based on the other authors [Lanza and Marinescu, 2006; Abilio et al., 2015] that proposed bad smells that present at least (i) the description of the proposed variability smells; (ii) which artifact they are applied on (such class and method granularity); (iii) the impact of these variability smells in the SPL; and, (iv) how to detect these smells. Additionally, we want to analyze the source code of some SPLs to see if the undesired properties that we think that are a problem really occur and whether they are problems.

4. Related Work

Several studies can be cited as related with this work. Alves et al. (2010), Ferreira et al. (2012), and, Oliveira et al. (2014), proposed a method to derive thresholds in the context of object-oriented single systems. Differently, our method is designed in the context of SPL. Lanza and Marinescu (2006), Apel et al. (2013), and, Abilio et al. (2015) proposed bad smells. The first one presents 23 bad smells designed to Object-Oriented single system that are reference for the following work. The other studies proposed bad smells for SPLs and are part of our catalogue. Therefore, our work can be considered a continuation of the literature.

5. Intended contributions

As contributions, we expect to have 3 catalogues of bad smells, refactoring methods, and detection strategies for SPLs [Vale et al., 2014; Vale et al., 2015a]. In addition, we have already created a benchmark composed by 64 SPLs, a comparison of methods to derive thresholds [Vale et al., 2015b], and a method proposed for the same purpose [Vale et al., 2015c]. We are currently developing a tool to support four methods to derive threshold, including our own method and a Systematic Mapping related to metrics for SPLs. As future plans up to the end of the dissertation, we want to propose detection strategies more efficient than traditional ones and new variability smells for fill the gap found in feature-oriented software product line literature. Therefore, the dissertation aims to present a technically sound contribution in the three main steps of variability smells solving (description, detection, and refactoring), although the dissertation focusses on the first two steps.

Acknowledgments

This work was partially supported by CAPES, CNPq (grant 485907/2013-5), and FAPEMIG (grant PPM-00382-14).

References

Abilio, R. et al. (2015) “Detecting Code Smells in Software Product Lines - An Exploratory Study”. In: Int'l Conf. on Information Tech.: New Generations (ITNG).

Alves, T.L., Ypma, C., and Visser, J. (2010) “Deriving Metric Thresholds From Benchmark Data”. In: Proc. of 26th Int. Conf. on Soft. Maint. (ICSM), pp. 1–10.

Apel, S., Batory, D., Kastner C., and Saake, G. (2013) “Feature-Oriented Software Product Lines: Concepts and Implementation”. Springer, p.315.

Asikainen, T., Mannisto, T., and Soininen, T. (2006) “A Unified Conceptual Foundation for Feature Modelling”. In: 10th Int'l Soft. Product Line Conf. (SPLC), pp.31-40.

Basili, V., Caldiera, G., Rombach, H. (1994) “Goal Question Metrics Paradigm”. Encyclopedia of Software Engineering, pp. 528-532.

Chidamber, S.R., and Kemerer, C.F. (1994) “A Metrics Suite for Object Oriented Design”. IEEE Transactions on Soft. Eng., vol. 20, Issue 6, pp. 476–493.

Ferreira, K. Bigonha, M., Bigonha, R., Mendes L., and Almeida, H.. (2012) “Identifying Thresholds for Object-Oriented Software Metrics”. J. of Syst. and Soft., pp. 244–257.

Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999) “Refactoring: Improving the Design of Existing Code”. Addison-Wesley Professional.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990) “Feature-Oriented Domain Analysis (FODA) - Feasibility Study”. SEE Tech. Report.

Kastner, C., Apel, S. and Batory, D. (2007) “A Case Study Implementing Features Using AspectJ”. In: 11th Int'l Soft. Product Line Conf. (SPLC), pp.223-232, 2007.

Lanza, M., Marinescu, R. (2006) “Object-Oriented Metrics Practice”, Springer, p.205.

Oliveira, P., Valente, M., and Lima, F. (2014) “Extracting Relative Thresholds for Source Code Metrics”. In proc. of CSMR, pp.254-263.

Pohl, K., Bockle, G., and Linden, F. J. V. (2005) “Software Product Line Engineering: Foundations, Principles, and Techniques”. Berlin: Springer, p.490.

Spinellis, D. (2008) “A Tale of Four Kernels”. In Proc. of ICSE, pp. 381–390.

Vale, G. et al. (2014) “Bad Smells in Software Product Lines: A Systematic Review”. In: Brazilian Symp. on Soft. Components, Arch. and Reuse (SBCARS), pp. 84-93.

Vale, G., Abilio, R., Santos, I., Figueiredo, E., Costa, H., and Almeida, E. (2015a) “Bad Smells in Software Product Line: A Systematic Review”. Submitted to: Journal of Systems and Software (JSS).

Vale, G., Albuquerque, D., Figueiredo, E., Garcia, A. (2015b) “Defining Metric Thresholds for Software Product Lines: A Comparative Study”. In: 19th Software Product Line Conference (SPLC).

Vale, G., and Figueiredo, E. (2015c) “A Method to Derive Metric Thresholds for Software Product Lines”. In: 29th Brazilian Symp. on Soft. Eng. (SBES).

Weiss, D. M. and Lai, C. T. R. (1999) “Software Product-Line Engineering: A Family-Based Software Development Process”. Addison-Wesley.