Detection Strategies for Modularity Anomalies: An Evaluation with Software Product Lines

Eduardo Fernandes*, Priscila Souza
Federal University of Minas Gerais
Belo Horizonte, Brazil
{eduardofernandes, priscilasouza}@dcc.ufmg.br

Kecia Ferreira
Federal Center for Technological Education of Minas Gerais
Belo Horizonte, Brazil
kecia@decom.cefetmg.br

Mariza Bigonha, Eduardo Figueiredo
Federal University of Minas Gerais
Belo Horizonte, Brazil
{mariza, figueiredo}@dcc.ufmg.br

***Abstract*—A Software Product Line (SPL) is a configurable set of systems that share common and varying features. SPL requires a satisfactory code modularity for effective use. Therefore, modularity anomalies make software reuse difficult. By detecting and solving an anomaly, we may increase the software quality and ease reuse. Different detection strategies support the identification of modularity anomalies. However, we lack an investigation of their effectiveness in the SPL context. In this paper, after an evaluation of existing strategies, we compared four strategies from the literature for two modularity anomalies that affect SPLs: God Class and God Method. In addition, we proposed two novel detection strategies and compared them with the existing ones, using three SPLs. As a result, existing strategies showed high recall but low precision. In addition, when compared to detection strategies from the literature, our strategies presented comparable or higher recall and precision rates for some SPLs.**

Keywords—Detection Strategies; Modularity Anomalies; Software Product Lines

I. Introduction

Software reuse consists of using existing code to develop new systems [20]. Reuse requires a satisfactory code modularity for its effective application [4][20]. Since existing software components may contain problems that should not be propagated to new systems, modularity anomalies may make reuse difficult. The literature also references modularity anomalies as bad smells [13]. Therefore, by detecting and solving an anomaly before reusing a component, we may increase the component quality and decrease time and efforts spent on maintenance, for instance [12][18].

In this context, we need effective methods to support the detection of modularity anomalies and, consequently, reuse [3]. Different detection strategies have been proposed to support the identification of anomalies [1][8][14]. Besides that, although several anomalies may affect reuse [8], we lack an investigation on the effectiveness of the existing strategies for detecting anomalies in Software Product Lines (SPL). An SPL is a set of systems that share common and varying features [4]. By combining features, we generate different SPL products [4]. SPL aims to support reuse with decreasing maintenance efforts to developers [20].

In this paper, we investigate modularity anomalies in SPL, since anomalies affect negatively the SPL modularity and makes reuse difficult. After an *ad hoc* literature review, we compared four detection strategies from the literature for two well-known modularity anomalies that may affect SPLs: *God Class* [13] and *God Method* [14]. In addition, we proposed novel detection strategies to support the identification of both anomalies. We designed the novel strategies given the small amount of strategies from the literature that are based on traditional, well known, and ease to compute software metrics. We analyzed three SPLs.

As a result, we presented a novel detection strategy for each anomaly, i.e., *God Class* and *God Method*. Through the comparison of our strategies with the ones from the literature, we observed positive results with respect to our strategies. For MobileMedia, our strategies obtained comparable recall and the highest precision results for both anomalies. In the case of Berkeley DB, we obtained the best and second best recall when compared to the existing strategies. Finally, our strategy for *God Class* obtained the best recall and precision rates, in comparison with the others, in the case of TankWar.

II. Background

Modularity anomalies are symptoms of deeper problems in the modularity of systems [13]. Several types of anomalies may affect the modularity of a system, such as *Lazy Class*, *Feature Envy*, and *Long Parameter List* [13][14]. In this study, we investigated two types of anomaly: *God Class* [13] and *God Method* [14]. We chose these anomalies because (i) they have different detection strategies in the literature for comparison and (ii) although both are general-purpose anomalies, they can affect negatively the SPL design. *God Class* is a class that contains excessive knowledge of the system and responsibilities [13]. *God Method* is a large method with high complexity and many responsibilities [14].

Two approaches may support the detection of modularity anomalies [18]. Manual detection relies on code inspection. Automated detection counts on the support of detection strategies, i.e., compositions of metric-based rules that define when a specific software component, e.g., class, method, or package, is prone to contain a modularity anomaly [14]. In turn, tools aim to support the automated detection of anomalies. These tools apply some type of detection strategy or equivalent techniques [6][18][22].

A Software Product Line (SPL) is a configurable set of systems that share common and varying features [20]. There are four types of features in a product line: mandatory, optional, alternative inclusive (OR), and alternative exclusive (XOR) [20]. Each product from an SPL is composed by general features that define the SPL basis (mandatory features) and specific features that differ a product from others (optional, OR, or XOR features) [4]. Artifacts of an SPL may contain modularity anomalies like in other types of software systems. However, there are few studies to investigate anomalies in this specific context [4].

III. Study Settings

Sections III-A, III-B, and III-C present the study goal and research questions, steps, and artifacts.

A. *Goal and Research Questions*

In this study, we were specifically concerned with detection strategies to identify modularity anomalies that hinder reuse in SPL. We then designed new strategies for these anomalies. We also conducted a comparative study of detection strategies in the SPL context. To guide our study, we designed two research questions as follows.

RQ1. *Are the existing detection strategies for modularity anomalies effective in the SPL context?*

RQ2. *Are the novel detection strategies more effective than the existing ones in the SPL context?*

B. *Study Steps*

We designed seven study steps discussed as follows. Steps 1-5 composed the study phase called *Selection of Artifacts*. In Step 1, we selected the SPLs for analysis. Step 2 consisted of the selection of modularity anomalies, based on anomalies that we were able to detect in the chosen systems from Step 1. Step 3 encompassed the selection of strategies from the literature for comparison. Step 4 was dedicated to the creation of new strategies for the anomalies chosen from Step 2. Our strategies relied on well-known anomaly definitions [13][14] and the SPL characteristics [4]. In the same step, we compared such strategies with strategies from the literature provided by Step 3. Step 5 consisted of selecting detection tools for modularity anomalies. This step was essential to support the definition of reference lists of anomalies for SPLs, collected from Step 1, without a previously computed reference list of anomalies. A reference list of anomalies is an itemization of anomalies that occur in a given system. Experts in a system can generate reference lists [19]. Otherwise, such lists may rely on the detection results provided by a detection tool.

The remaining steps, Steps 6 and 7, composed the last phase called *Comparative Study*. Step 6 comprised the comparison of existing detection strategies from the literature to answer RQ1. Finally, Step 7 targeted on RQ2 through the comparison of novel detection strategies with the existing ones.

C. *Selected Artifacts*

In Steps 1 and 2, we chose three SPLs extracted from a repository [21]: MobileMedia [10], Berkeley DB, and TankWar. These systems are implemented in AHEAD or FeatureHouse and have from 2 K to 42 K number of lines of code (LOC) [15]. We selected MobileMedia based on the availability of reference lists for *God Class* and *God Method*. We selected Berkeley DB and TankWar because of the three following reasons. First, Berkeley DB is one of the largest systems in the SPL catalog. Second, we are able to import the code of these systems in FeatureIDE for automated anomaly detection and generation of reference lists. Third, there are at least two occurrences of *God Class* and *God Method* in each system. Details in the discussion of Step 5.

A set of 11 pre-computed software metrics are provided by the SPL repository. Coupling between Objects (CBO) [5], Lines of Code (LOC) [15], Number of Attributes (NOA) [15], Number of Constant Refinements (NCR) [1], Number of Methods (NOM) [15], and Weighted Methods per Class (WMC) [5] are class-level metrics. McCabe's Cyclomatic Complexity (Cyclo) [16], Method Lines of Code (MLOC) [15], Number of Method Refinements (NMR) [2], Number of Operations Overrides (NOOr) [17], and Number of Parameters (NP) [15] are method-level metrics.

In Step 3, we selected strategies for each anomaly. After an *ad hoc* literature review, we selected two strategies for *God Class* and two for *God Method*. Although we have found other strategies, our selection relied on the metrics provided by the SPL catalog. To adapt each strategy, we discarded clauses with metrics that are unavailable in the SPL catalog. We derived thresholds for the metrics – i.e., values that support the characterization of a given metric [21] – based on the selected SPL repository and Vale's Method [21]. We chose Vale's method because it provided us sufficient flexibility to define detection strategies. To derive thresholds, we used the R statistical environment[1] and the entire SPL repository [21], with 33 SPLs, as target systems. Table I presents each strategy adapted from the literature. In Step 4, we proposed a new strategy for each modularity anomaly (details in Section V).

In Step 5, we extracted the reference lists of *God Class* and *God Method* for MobileMedia from a previous work [19], with 7 instances per anomaly. Experts from the MobileMedia development team composed these lists. For Berkeley DB and TankWar, we did not find reference lists. Therefore, we generated them based on the results of JSpIRIT [22], a plug-in tool for Eclipse IDE. The automated detection in BerkeleyDB and TankWar supported the composition of reference lists. Based on a SLR [9], we chose JSpIRIT because of its availability for download and sufficient recall and precision. The tool found 35 *God Class* and 37 *God Method* instances for Berkeley DB, and 2 and 3 instances, respectively, for TankWar. After the validation of the results by the paper's authors, we obtained (i) 17 and 12 instances for *God Class* and *God Method*, respectively, for Berkeley DB and (ii) 1 and 2 instances for the same anomalies, for TankWar. The study artifacts are available in the research website[2].

TABLE I. DETECTION STRATEGIES FROM THE LITERATURE

God Class
GC1 [21]: [(LOC > 77) AND (WMC > 17) AND (CBO > 0)] OR (NCR > 1)
GC2 [11]: (WMC > 34) AND (NOM > 14) AND (NOA > 8)
God Method
GM1 [6]: (MLOC > 50)
GM2 [14]: (MLOC > 13) AND (Cyclo > 2)

IV. COMPARISON OF EXISTING STRATEGIES

This section aims to answer RQ1. Table II presents recall and precision, per SPL, for both *God Class* and *God Method*. TP is the number of true positives (correct identification of real anomalies). FP is the number of false positives (incorrect identification of real anomalies). TN is the number of true negatives (correct non-identification of anomalies). Finally, FN is the number of false negatives (incorrect non-identification of anomalies) [7]. The used formula are *Precision = TP / (TP + FP)* and *Recall = TP / (TP + FN)* [7].

In general, we observed a mean and median recall of 66% and 90%, respectively, i.e., a significant result. However, regarding precision, we observed mean and median of 16% and 9%, respectively, a low result. Therefore, our data suggests that the existing detection strategies were not sufficiently effective in the SPL context. GC1 presented the highest recall rate for all SPLs and a slight highest precision for two of them. Therefore, our data suggests that GC1 was more effective than

[1] https://www.r-project.org

[2] http://labsoft.dcc.ufmg.br/doku.php?id=about:itng17-detection-strategies

GC2. In turn, both GM1 and GM2 performed similarly in terms of recall for two of the three SPLs, although precision was slightly higher for GM1. We provide a discussion per SPL as follows.

TABLE II. RESULTS FOR EXISTING STRATEGIES

SPL	Strat.	TP	FP	TN	FN	Recall	Precision
MobileMedia	GC1	5	10	126	2	**71%**	**33%**
	GC2	0	0	7	136	**0%**	**0%**
	GM1	0	1	365	7	**0%**	**0%**
	GM2	3	30	336	4	**43%**	**9%**
Berkeley DB	GC1	17	80	524	0	**100%**	**18%**
	GC2	15	21	583	2	**88%**	**88%**
	GM1	11	53	5618	1	**92%**	**17%**
	GM2	11	441	5230	1	**92%**	**2%**
TankWar	GC1	1	11	76	0	**100%**	**8%**
	GC2	0	2	85	1	**0%**	**0%**
	GM1	2	10	297	0	**100%**	**17%**
	GM2	2	66	241	0	**100%**	**3%**

MobileMedia. Regarding *God Class*, GC1 was the only strategy able to detect code anomaly instances, with recall and precision rates of 71% and 33%, respectively, against 0% for both measures in the case of GC2. The low percentages for GC2 may relate to the high threshold for WMC, computed based on MLOC that is generally low for the analyzed SPLs. Regarding *God Method,* we observed similar behavior, in which GM1 was unable to detect anomalies, although GM2 provided 43% and 9% of recall and precision, respectively. Again, the justification for the non-effectivity of GM1 relies on the high threshold for MLOC.

Berkeley DB. Regarding *God Class* detection strategies, GC1 presented 100% of recall (a result 12% higher than GC2), but a low precision of 18% (70% lower than for GC2). Considering the significant difference between precision rates, we observed that GC2 was more effective than GC1. Although such results differ from the previous ones, they reinforce our assumptions that MLOC for the SPLs affected the results, since the methods from classes of Berkeley DB are significantly larger than MobileMedia. With respect to *God Method*, both GM1 and GM2 presented the same recall, but GM1 provided 15% more precision than the other strategy, probably for the same reason discussed in the case of MobileMedia. Therefore, we assume that GM1 was slightly more effective than GM2.

TankWar. Regarding *God Class,* GC2 was unable to detect anomalies, while GC1 presented 100% and 8% of recall and precision, respectively. Besides the considerations valid for MobileMedia and Berkeley DB, NCR contributed to the high recall observed for GC1. On the other hand, NRC contributed to low precision, by causing an increase in the number of FP, mainly because the threshold is very low. Note that, in SPL, several methods tend to have more than 1 refinement, due to the modularization of features. We conclude that GC1 was more effective than the other strategy. However, regarding *God Method*, both GM1 and GM2 obtained the same recall. In turn, GM1 presented a precision only 14% higher than the other strategy, due to the higher, stricter threshold for MLOC. We conclude that GM1 was slightly more effective than GM2.

V. COMPARISON WITH NOVEL STRATEGIES

This section aims to answer RQ2. Table III presents the novel strategies for *God Class* (GC3) and *God Method* (GM3). We took into account the findings from Section IV to design each strategy aiming better recall and precision. For CG3, we used LOC because a high number of code lines may indicate excessive responsibilities of the class. We also used NOA and NOM because a high number of attributes (NOA) and methods (NOM) suggests excessive knowledge of the class (attributes) and responsibilities (methods). Finally, we used WMC because a high weight of the class indicates that the class is doing more than it should do. For GM3, we used MLOC because a high number of code lines is a symptom of complex method. We also used NP because a large list of parameters may point that the method requires too much knowledge of the current or external classes. Finally, we used Cyclo because it indicates too many responsibilities of the method.

TABLE III. NOVEL STRATEGIES PROPOSED IN THIS STUDY

God Class
GC3: (LOC > 77) AND (NOA > 4) AND (NOM > 10) AND (WMC > 17)
God Method
GM3: (MLOC > 13) AND (NP > 2) AND (Cyclo > 3)

Table IV presents recall and precision for the novel strategies, per SPL. We observed moderate rates of recall, but low precision – we obtained moderate-to-high rates only for MobileMedia. For *God Class*, GC1 presented the highest rates of recall for Berkeley DB and TankWar, and the highest precision for MobileMedia and TankWar. However, with respect to *God Method*, we observed a significant rate of precision for MobileMedia.

TABLE IV. RESULTS FOR THE NOVEL STRATEGIES

SPL	Strat.	TP	FP	TN	FN	Recall	Precision
MobileMedia	GC3	2	0	136	5	**29%**	**100%**
	GM3	1	1	365	6	**14%**	**50%**
Berkeley DB	GC3	17	51	553	0	**100%**	**25%**
	GM3	6	139	5532	6	**50%**	**4%**
TankWar	GC3	1	4	83	0	**100%**	**20%**
	GM3	0	10	296	3	**0%**	**0%**

MobileMedia. Regarding *God Class*, our strategy GC3 presented the highest precision (100%), 67% higher than for the second highest (GC1). This positive result relates to the discard of metrics that generated high FP in the other strategies (e.g. NCR). However, GC1 presented recall of 71%, a result 42% higher than for GC3, probably because GC3 is stricter since it has more metrics to compare. Regarding *God Method*, GM3 presented the highest precision rate (50%), a result 41% higher than for GM2. This positive result relates to lower threshold for MLOC combined with two metrics that support the identification of *Long Method* even in SPL (i.e. NP and Cyclo). However, the precision for GM2 was 29% higher than for GM3. This result may relate to the very low thresholds for NP and Cyclo, although such metrics tend to be low in SPL. We conclude that GC3 and GM3 were more precise.

Berkeley DB. Regarding *God Class*, our strategy GC3 presented the highest recall (100%), similarly to GC1. Note that GC2 obtained 88% of recall, a result only 12% lower than our strategy. Regarding precision, GC3 presented the second highest rate, although it is 63% lower than for GC2. Therefore, our data suggests that GC2 was the most effective strategy, in general. For *God Method*, GM3 obtained the second highest recall, against 92% for both GM1 and GM2. In addition, GM3 had low rates of recall (50%) and precision (4%) when compared to the highest rates obtained by GM1 (92% and 17%, respectively). Therefore, GM2 was the most effective strategy. Overall, our findings for Berkeley DB have similar justification than in the case of MobileMedia, for both GC3 and GM3.

TankWar analysis. Regarding *God Class*, we observed the highest recall and precision (100% and 20%, respectively) for our strategy GC3. Although GC1 and GC3 presented the same recall of 100%, our strategy presented a 12% higher precision. The justification is similar to the MobileMedia case. We conclude that GC3 was more effective than the GC1. For *God Method*, GM3 did not find anomalies, and the highest results were obtained by GM1 (100% of recall and 17% of precision). In this case, an anomalously low NP (even for SPLs) for most of the methods affected significantly GM3.

VI. THREATS TO VALIDITY

We discuss threats to validity as follows.

Construct and Internal Validity. We designed our study with steps for replication. The study settings relied on the literature. We also provided the study artifacts in the research website. These treatments aim to minimize problems with study replication and reliability. Moreover, we conducted a careful data collection to prevent missing data, incorrect selection of metrics, and inappropriate use of threshold derivation methods. We double-checked the collected data. Therefore, we expect that our data collection is reliable for analysis. Regarding the automatically generated reference lists of anomalies, we chose an effective tool from a previous work [9].

Conclusion and External Validity. We carefully performed the data analysis to minimized problems with data interpretation. We based the choice of mathematical computation (recall and precision) on previous studies. Finally, some factors may prevent the generalization of our research findings, since we proposed strategies based on the authors' background and subjective perceptions of anomalies. To minimize this problem, we carefully chose metrics from the available metric set, and we defined each strategy in details.

VII. RELATED WORK

Fontana et al. (2012) [12] conducted a literature review and a comparison of detection tools for modularity anomalies. They concluded that the tools provide significantly different results for a same anomaly, some results are redundant, and the tools' agreement is high for anomalies as *Large Class*. Moha et al. (2010) [18] also evaluated tools. The authors compared a set of tools with a new one proposed by them. By relying on reference lists of anomalies built after manual code inspection, they computed recall and precision of the tools. However, their study did not compare an extensive set of tools, and there was no agreement computation.

In a previous work [9], we conducted a systematic literature review on detection tools for modularity anomalies. We found 84 tools, 29 of them available for download. We also compared tools by computing recall, precision, and agreement. We observed that most of the tools rely on detection strategies based on metrics. We also observed redundant results of the tools, and conclude that new detection strategies may be explored to improve the detection effectiveness. In the present study, we aimed to contribute by filling the gap observed in our previous work [9] focused on the SPL context.

VIII. Conclusion and Future Work

In this paper, we evaluated detection strategies for modularity anomalies in SPL. We compared detection strategies from the literature, for two well-known anomalies that may affect the SPL modularity: *God Class* and *God Method.* This comparison aimed to assess if they are effective in the SPL context. We then proposed novel strategies, one for each anomaly, and compared them with the existing strategies. Our study analyzed three SPLs.

As a result, when comparing existing detection strategies, we have observed high recall, with respective mean and median of 66% and 90%. However, we observed low precision with respective mean and median of 16% and 9%. We concluded that the existing strategies were not effective for SPLs. In turn, when comparing our novel strategies with the existing ones, we observed higher recall for our strategies, but low precision as observed for the other strategies. As future work, we suggest the investigation a larger amount of anomalies, new detection strategies, and the analysis of other SPLs.

Acknowledgements

This work was partially supported by CAPES, CNPq (grant 424340/2016-0), and FAPEMIG (grant PPM-00382-14).

References

[1] R. Abilio, J. Padilha, E. Figueiredo, and H. Costa, “Detecting Code Smells in Software Product Lines,” in Proc. of the 12th ITNG, 2015, pp. 433–438.

[2] R. Abilio, G. Vale, E. Figueiredo, and H. Costa, “Metrics for Feature-Oriented Programming,” in Proc. of the 7th WETSoM, 2016, pp. 36–42.

[3] E. Almeida, A. Alvaro, D. Lucrédio, V. Garcia, S. Meira, “RiSE Project,” in Proc. of the 5th IRI, 2004, pp. 48–53.

[4] S. Apel, D. Batory, C. Kästner, and G. Saake, Feature-Oriented Software Product Lines. Springer Science & Business Media, 2013.

[5] S. Chidamber and C. Kemerer, “A Metrics Suite for Object Oriented Design,” Transactions on Software Engineering (TSE), vol. 20, no. 6, pp. 476–493, 1994.

[6] A. Fard and A. Mesbah, “JSNose,” in Proc. of the 13th SCAM, 2013, pp. 116–125.

[7] T. Fawcett, “An Introduction to ROC Analysis,” Pattern Recognition Letters, vol. 27, no. 8, pp. 861–874, 2006.

[8] W. Fenske and S. Schulze, “Code Smells Revisited,” in Proc. of the 9th VaMoS, pp. 3-10, 2015.

[9] E. Fernandes, J. Oliveira, G. Vale, T. Paiva, and E. Figueiredo, “A Review-based Comparative Study of Bad Smell Detection Tools,” in Proc. of the 20th EASE, 2016.

[10] E. Figueiredo, N. Cacho, C. Sant’Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. Castor Filho, and F. Dantas, “Evolving Software Product Lines with Aspects,” in Proc. of the 30th ICSE, pp. 261–270, 2008.

[11] T. Filo, M. Bigonha, and K. Ferreira, “A Catalogue of Thresholds for Object-Oriented Software Metrics,” Proc. of the 1st SOFTENG, pp. 48–55, 2015.

[12] F. Fontana, P. Braione, and M. Zanoni, “Automatic Detection of Bad Smells in Code,” Journal of Object Technology (JOT), vol. 11, no. 2, pp. 5–1, 2012.

[13] M. Fowler, Refactoring: Improving the Design of Existing Programs. Addison-Wesley Publishing, 1999.

[14] M. Lanza and R. Marinescu, Object-Oriented Metrics in Practice. Springer Science & Business Media, 2007.

[15] M. Lorenz and J. Kidd, Object-Oriented Software Metrics: A Practical Guide. Prentice-Hall, 1994.

[16] T. McCabe, “A Complexity Measure,” Transactions on Software Engineering (TSE), no. 4, pp. 308–320, 1976.

[17] B. Miller, P. Hsia, and C. Kung, “Object-Oriented Architecture Measures,” in Proc. of the 32nd HICSS, pp. 8069–8086, 1999.

[18] N. Moha, Y.-G. Gueheneuc, L. Duchien, and A.-F. Le Meur, “DECOR,” Transactions on Software Engineering (TSE), vol. 36, no. 1, pp. 20–36, 2010.

[19] T. Paiva, A. Damasceno, J. Padilha, E. Figueiredo, and C. Sant’Anna, “Experimental Evaluation of Code Smell Detection Tools,” Proc. of the 3rd VEM, pp. 17–24, 2015.

[20] K. Pohl, G. Böckle, and F. van der Linden, Software Product Line Engineering. Springer Science & Business Media, 2005.

[21] G. Vale and E. Figueiredo, “A Method to Derive Metric Thresholds for Software Product Lines,” in Proc. of the 29th SBES, 2015, pp. 110–119.

[22] S. Vidal, C. Marcos, and J. Díaz-Pace, “An Approach to Prioritize Code Smells for Refactoring,” Automated Software Engineering (ASE), pp. 1–32, 2014.