Impact of Code Smells on the Rate of Defects in Software: A Literature Review

MITJA GRADIŠNIK and MARJAN HERIČKO, University of Maribor

A consequence of the rush in software development is that there are often suboptimal design decisions that, in some aspects, deviate from best practice in software development. In the software development community, there is a general belief that such suboptimal design decisions, which can be identified in source code as code smells, negatively impact software quality, e.g. the rate of identified bugs in post-release software. Consequently, the presence of code smells in a software project could be a predictor of upcoming issues related to low quality software. The methods and approaches applied to control and eventually remedy source code affected by code smells, constitute an important research field in software engineering.

The literature review presented in this paper aims to answer the question of whether the correlation between the presence of code smells in source code and a higher incidence of defects in software has been sufficiently analyzed in previous studies. The goal of the literature review is also to determine, which defined code smells were analyzed in studies and where the impact of studied code smells on low quality was confirmed. Furthermore, with the literature review we want to find out how effective the refactoring of smelly classes from the perspective of quality of software can be.

1. INTRODUCTION

Tough time-to-market constraints and unanticipated integration or evolution issues lead to design tradeoffs that usually cause flaws in the structure of a software system [Marinescu 2012]. Software developers that work under tight delivery deadlines are often induced to deliver program code that, in some aspects, deviates from standard design principles and good practices of software development. In the structure of a program code, such shortcuts in the development process can be detected as code smells, a program code that serves its purpose but does not seem to be just right. Cunningham describes [Cunningham 1992] the phenomenon of rushing implementation to meet pending deadlines as a technical debt. Extensive deviation from principles and good practices in software development process does not come without consequences for future software development nor can it be easily ignored. Namely, code smells can indicate some issues with code quality, such as understandability and changeability, which can lead to the introduction of faults [Fowler and Beck 1999]. Therefore, short term benefits related to faster code delivery are traded for the long-term goals of software development project. In general, the compromised long-term project goals are usually expressed in the form of extra time, effort, and cost to address future changes in a project [Conejero et al. 2018; Abad and Ruhe 2015].

From a broader perspective, schedule pressure has not been identified as the only cause of code smells in the source code of software systems. Besides tight time constraints, design flaws can also be produced by carelessness, lack of education, poor processes, lack of verifications or, even, basic incompetence [Conejero et al. 2018]. In general, code smells can

This work is supported by the Slovenian Research Agency (research core funding No. P2-0057)
Author's address: M. Gradišnik, University of Maribor, Faculty of Electrical Engineering and Computer Science, Institute of Informatics, Koroška cesta 46, SI-2000 Maribor, Slovenia; email: mitja.gradisnik@um.si; M. Heričko, University of Maribor, Faculty of Electrical Engineering and Computer Science, Institute of Informatics, Koroška cesta 46, SI-2000 Maribor, Slovenia; email: marjan.hericko@um.si.

In: Z. Budimac (ed.): Proceedings of the SQAMIA 2018: 7th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Novi Sad, Serbia, 27–30.8.2018. Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

be introduced into programing code intentionally, with the use of not-quite-right solutions in order to speed up the development process, or on the other hand completely unintentionally, via a lack of developer skills and competences that are required in the software development process.

High software quality is evidently a required attribute for successful software products [Plosch et al. 2010]. Therefore, software development teams must constantly monitor the quality of their source code, also by controlling the rate of code smells presented in the source code. The quality monitoring activities require extra attention especially in cases where the software system must be able to evolve and be properly maintained for decades [Steidl et al. 2014]. Thus, different approaches, which help development teams maintain a high quality source base, can be applied. For example, some development teams manage the source base quality by reviewing source code quality at the end of each development cycle. The code review is performed by the most skilled developers in the development team and can be assisted by continuous code quality assurance tools. The aim of the activity is to detect bad design decisions and prevent them from being introduced into the source base.

If not remedied during a code review, code smells can also be removed in later phases of a software development cycle. For instance, if a code smell is detected during development, the developer can easily refactor the detected code smell. The described scenario is especially the case when low quality program code has been traded for faster development. In practice, it turns out that if not automated, the source code quality assurance process can be time consuming, error prone, and costly [Choudhary et al. 2018]. To avoid an additional loss of time and extra costs for a manual code review, a tool for automatic quality control can be introduced into the development process.

In this paper, we focus our attention on the relationship between the introduction of code smells in source code and the low quality of software products. The goal of this study is to determine whether all types of code smells in a project have an equal impact on software quality or if there is a subset of code smells, which are proven by studies to be particularly harmful for a software project or could even jeopardize a software system in the future. Refactoring is also one of the points of interest in this study. Refactoring detected code smells in source code comes with additional work, which consequently results in higher costs for software development. Refactoring can therefore be expensive, in particular if it is used to remedy code smells that have little, or even zero, impact on low software quality. Code smells must be removed, but only if they represent real quality problems [Fontana, Ferme, Zanoni, and Roveda 2015]. To answer our research goal, we performed a literature review.

This paper is structured as follows. The introduction establishes the scope and purpose of the paper and gives the necessary background information relevant to the research. Following the introduction, a second section explains the procedure of the literature review. We then describe the analysis of studies selected in the process of the literature review. Section 5 outlines the key findings of our research. Finally, in the conclusion, we summarize our research and give concluding remarks.

2. LITERATURE REVIEW OF THE RESEARCH AREA

Code quality has always been an interesting topic of discussion within the programming community. It is the general belief that good code is the kind of code that can be easily reused years after it has been written. Such code can be easily taken over and extended by other programmers. On the other hand, it is believed that low quality code is meant to solve a problem that exists at the time of its writing and completely ignores any long-term consequences. Some authors link technical debt caused by the low quality of programming code with severe maintenance difficulties or even project cancellations [Zazworka et al. 2014].

Despite the fact that these beliefs are deeply rooted and commonly accepted in the software engineering community, there is very little known about formal methods that can be applied to measure tangible effects of poor design decisions on software quality. Similarly, for the development process, a precise prediction of the influence of refactoring activities on software quality can be equally crucial. Without formal approaches to evaluate the benefits of refactor activities, it is hard to justify that these activities are economically viable and worth the resources invested. To dissolve this ambiguity, we performed a literature review, whose main objective is to find studies that analyze the relationship between code smells in source code and the quality of software systems. In our study, we were particularly interested in studies that analyze the impact of code smells on the rate of detected defects in a software product. To the best of our knowledge, this research goal, in that iteration, has so far remained unaddressed.

Based on our research goals, the following four research questions were formed:

RQ1: do all of the code smells equally impact software quality in terms of detected software defects?

RQ2: which code smells are directly correlated with low software quality in terms of detected software defects?

RQ3: does refactoring detected code smells in the source code directly improve software quality measured by defect rate?

RQ4: how active is the current research field of studies investigating the influence of code smells on software quality in terms of detected software defects?

In accordance with our research questions, the search was conducted in digital libraries. According to the research questions, we formed the following search string: *("bad smell*" OR "code smell*" OR "design flaw*" OR "antipattern*") AND ("error probability" OR "defects" OR "bugs" OR "anomalies")*. With it, the search was conducted in two digital libraries, *Web of Science* and *IEEE Xplore*. The number of search results and number of selected studies can be found in Table I. In our research we only included studies available in the selected digital libraries, which were written in the English language. The search was conducted for both journal and conference articles. All the studies used in our research were published in the domain of software engineering.

Table I. Digital Libraries, Search Results and Selected Studies

Electronic base	URL	No. of Results	No. of Selected Studies
Web of Science	http://www.webofknowledge.com/	128	4
IEEE Xplore	https://ieeexplore.ieee.org	45	2

The research field that addresses quality issues related to code-based technical debt is relatively broad. In general, four different approaches can be found to software quality management based on an analysis of structural issues of source code - code smells, automatic static analysis, grime buildups, and modularity violations [Zazworka et al. 2014]. For the purpose of our study, we will only focus on techniques and approaches that study the direct influence of code smells on the occurrence of software defects in classes. Studies that did not address this topic directly were not included in our research. In the process of the literature review, we identified six relevant studies, which were included upon further data extraction. Basic information about the selected studies are summarized in Table II.

Table II. Basic Information about Selected Studies

Study	Published in	Research method	Software studied in the research	No. of studied code smells
[Bán and Ferenc 2014]	conference	experiment	34 Java based systems from PROMISE database	8
[D'Ambros et al. 2010]	conference	experiment	Apache Lucene, Apache Maven, Mina, Eclipse CDT, Eclipse PDE UI, Equinox	5
[Li and Shatnawi 2006]	journal	experiment	Eclipse Platform, Eclipse JDT, Eclipse PDE	6
[Marinescu and Marinescu 2011]	conference	case study	three releases of Eclipse IDE (2.0, 2.1, 3.0)	4
[Olbrich et al. 2010]	conference	experiment	Apache Lucene, Apache Xerces 2 Java, Log4J	2
[Zazworka et al. 2014]	journal	case study	Hadoop	9

For each study, we specified a research method used in the study and a publication type of the study. In the majority of studies, an experiment was used as a research method. A case study was used as a research method in only two selected studies. The selected studies were published in both journal and conference proceedings. Although all of the selected research study software was based on the Java platform, this was not a condition for inclusion into our study. Table II shows the number of code smell types analyzed for each study and the software from which the observed code smells were extracted.

In this literature review, we were especially interested in studies that research a direct, tangible and measurable impact of code smells in source code on an occurrence of quality issues in software products under the study. According to some authors [D'Ambros et al. 2010], software defects are a tangible effect of poor software quality. Therefore, correlation between code smells and software defects is a popular topic of research. Additionally, all of the studies in Table II study the link between the presence of code smells in source code and a frequency of defects in software. In general, many measures have been proposed to predict software maintainability, but empirical qualification linking maintainability and measure attributes of software, such as code smells, remain elusive [Sjøberg et al. 2013]. In practice, it turns out that studying the relationship between code smells and defects is more common among studies due to the availability of necessary data, which can be relatively easily extracted from bug tracking systems. Other effects of poor code quality seem to be much more elusive, harder to measure, collect, and study.

3. ANALYSIS OF SELECTED LITERATURE

In order to analyze the impact of code smells on software quality in terms of detected bugs, all authors except one rely on freely available source code from open source projects. Open source projects are suitable for these kinds of studies especially because all data related to a development process of software products are freely available online and can be easily obtained by researchers. On the other hand, studying open source software solutions has an advantage from the perspective of the relevance of the derived conclusions. Because open source solutions are built for use in real business environments, the study of them can give us quite relevant study results.

In the case of a study performed by Bán and Ferenc in 2014, the PROMISE bug database [Menzies et al. 2012] was used as a source of required data for the study. Because the PROMISE database is well structured, the researchers had easier work extracting all the information required to study. For this reason, the researchers managed to analyse more pieces of software (34 in total) comapared to other researchers, who have extracted data

manually. In studies that used manual data extraction directly from the source code of open sources projects, up to six open source soulutions were analyzed in a study.

To extract information about the defects detected in the studied software solutions, the researchers relied on issue tracking systems, used by the developers of studied software solutions to track detected defects in software components. In the case of research performed by Olbrich, Cruzes, and Sjoberg, 2010 information on defects for the studied solutions Apache Lucene and Apache Xerces was extracted from Jira, while in the case of the studied tool Log4J, Bugzilla was a source of information about defects.

To prove the impact of code smell occurrence in source code to a higher rate of detected bugs in software, source-code studies identified two to nine different types of code smells. The majority of observed code smells were either defined in Refactoring: Improving the Design of Existing Code [Fowler and Beck 1999] or in Object-Oriented Metrics in Practice [Lanza and Marinescu 2006]. The majority of studies reference well known and common code smells, so that in only one study on code smells did we encounter ones that are not so common in the existing literature.

Despite the fact that code smell catalogs define a larger set of code smells, for example [Fowler and Beck 1999] define 22 code smell types in total, the researchers included in their studies use only a subset of code smells defined in the literature. The researcher's choice of code smell selection was not explained in detail in any of the studies. The chosen subset of code smells was most likely a compromise between the ability of tools to detect code smell in the source code of studied solutions and the predictions of researchers regarding which type of code smell has the potential to lead to quality problems in software. Sets of code smells used in the analyzed studies can be seen in Table III.

Table III. Set of code smells analyzed in selected studies

Code smells	[Bán and Ferenc 2014]	[D'Ambros et al. 2010]	[Li and Shatnawi 2006]	[Marinescu and Marinescu 2011]	[Olbrich et al. 2010]	[Zazworka et al. 2014]
Brain Class				X	X	X
Brain Method		X				
Code Duplication						
Data Class			X	X		X
Data Clumps						
Dispersed Coupling		X				X
Feature Envy	X	X	X	X		X
God Class			X	X	X	X
God Method			X			
Intensive Coupling		X				X
Large Class Code	X					
Large Class Data	X					
Lazy Class	X					
Long Method	X					
Long Parameter List	X					
Refused Bequest	X		X			
Refused Parent Bequest						X
Schizophrenic Class						
Shotgun Surgery	X	X	X			X
Temporary Field	X					
Tradition Breaker						X

The main point of interest of our literature review is to answer the question of whether the presence of code smells in the code base impacts software quality. We are also interested in the impact of code smell removal on software quality. To answer our research goals, we had to analyse studies selected in the previous stage of the study. Despite the fact that approaches to address this research question were quite similar, the results of the studies are not so conclusive and unambiguous.

The first study we analysed was conducted by Li and Shatnawi in 2006. The authors of the study conducted an experiment in which they studied whether classes containing code smells, i.e. Data Class, Feature Envy, God Class, or God Method, are more prone to software defects. A statistical analysis showed that the correlation between code smells and bugs can be confirmed in case of the code smells God Class, God Method and Shotgun Surgery. A correlation with software defects in the cases of Data Class, Refused Bequest and Feature Envy was not confirmed.

Researchers D'Ambros, Bacchelli, and Lanza study an impact of code smells Brain Method, Shotgun Surgery, Feature Envy, Intensive Coupling, and Dispersed Coupling on the occurrence of defects identified in software solutions containing mentioned code smells. The impact of code smells on the low quality of software was studied on six open-source software tools. The results of the study showed that program code, which contains design flaws, do correlate with a higher rate of software defects, but research was unable to correlate a specific code smell type with a higher rate of software defects across all software systems under the study.

Furthermore, a study conducted in 2010 by Olbrich, Cruzes, and Sjoberg analysed the correlation between the code smells God Class and Brain Class with the frequency of defects detected in the post-release software product. The study took into consideration the size of classes containing the code smells. The results showed that without taking the class size into account there is a higher defect rate in the God and Brain classes compared to other classes. But when normalizing a defect rate with respect to class size, God and Brain Classes had even smaller values of defect rates when compared to other classes in the software system. Therefore, the result of the study showed a negative correlation between code smells under the study and low software quality. The authors of the study claimed that, according to the results, analyzed code smells can be, in some cases, even beneficial for the software system.

Next, the correlation between code smells and their impact on the frequency of detected bugs in released software were investigated by R. Marinecu and C. Marinecu in their study published in 2011. In the study, the authors analysed types of code smells, namely, Data Class, God Class, Brain Class, and Feature Envy. Influence of code smells on the defect-proneness of software was observed in three versions of Eclipse IDE (versions 2.0, 2.1, and 3.0). The analysis did not confirm any correlation between the observed code smells and an increased likelihood of exhibiting software defects compared to classes that do not reveal design flaws. However, a direct correlation between specific code smell types and defect rates in studied software was not confirmed; instead, the study came to an interesting conclusion. The study showed that when code smell affected classes are used by their clients, the likelihood for the clients to exhibit software defects greatly increases, especially in the case of post-release defects. The research therefore provided an explanation as to why the presence of code smells in source code do statistically correlate with a higher defect rate in software despite the fact that classes affected by code smells cannot be strongly related to software defects. The author of the research came to similar conclusions in one of the following studies [R. Marinescu 2012].

The same research problem was addressed by Zazworka et al. in a study conducted in 2014. The authors address the impact of structural flaws in the program solution on software quality. They studied four different approaches to structural flaw detection, namely: modularity violations, design patterns and grime buildups, code smells, and automatic static analysis. In the study, they observed if there was a correlation between detected structural

design flaws and the software defect rate. For the purpose of the case study, the authors considered 13 releases of Hadoop software. In the research, the authors observed 10 different types of code smells, namely: God Class, Brain Class, Refused Parent Bequest, Tradition Breaker, Feature Envy, Data Class, Brain Method, Intensive Coupling, Dispersed Coupling, and Shotgun Surgery. Of all the observed code smell types, only classes affected by Dispersed coupling and God Classes were correlated with higher defect-proneness in the context of the source code of the Hadoop library.

Finally, a study performed by Ban and Ferenc in 2014 came to a similar conclusion as a previous study conducted by [Marinescu and Marinescu 2011]. In the study they analysed 9 code smells (antipatterns), namely: Feature Envy, Lazy Class, Large Class Code, Large Class Data, Long Function, Long Parameter List, Refused Bequest, Shotgun Surgery, and Temporary Field. At the end of study, the authors did not state explicitly, which code smell was correlated with a higher defect rate. However, the study, which was based on an analysis of 34 systems from the PROMISE dataset, showed significant positive correlation between the rate of code smells (antipatterns) in the source code and the number of detected bugs in the software system.

Table IV. From the studies extracted correlations between studied code smells and defects

Code smells	[Bán and Ferenc 2014]	[D'Ambros et al. 2010]	[Li and Shatnawi 2006]	[Marinescu and Marinescu 2011]	[Olbrich et al. 2010]	[Zazworka et al. 2014]
Brain Class				O	–	O
Brain Method		O				
Code Duplication						
Data Class			O	O		O
Data Clumps						
Dispersed Coupling		O				+
Feature Envy	?	O	O	O		O
God Class			+	O	–	+
God Method			+			
Intensive Coupling		O				O
Large Class Code	?					
Large Class Data	?					
Lazy Class	?					
Long Method	?					
Long Parameter List	?					
Refused Bequest	?		O			
Refused Parent Bequest						O
Schizophrenic Class						
Shotgun Surgery	?	O	+			O
Temporary Field	?					
Tradition Breaker						O

The sign + stands for positive correlation, the sign – stands for negative correlation, the sign O denotes an unconfirmed correlation, and the sign ? denotes that the correlation was unclear according to the final report of a study.

A literature review shows that previous studies analysed the variety of different code smell types, but that they were rarely able to confirm a positive correlation between classes containing code smells and a higher rate of software defects in these classes. Furthermore, a

literature review shows that there is not a clear pattern in results, which can correlate with a specific type of code smell with the defect-proneness of a system. Additionally, the results of some studies (e.g. Olbrich, Cruzes, and Sjoberg 2010) indicate a negative correlation between code smells and bugs. In other words, the study showed that code smells can be beneficial in some cases when it comes to the perspective of software quality. On the other hand, some studies confirmed that the higher number of code smells in the source code does lead to a higher rate of software defects in the software product in general. Table IV summarizes extracted correlations in the existing literature between code smells and the defect-proneness of software.

4. DISCUSSION

The literature review of selected studies allowed us to answer our research questions. In the context of **RQ1**, our results showed that researchers analyzed, for the purpose of the studies, a variety of code smells, which can be found in the literature. Most of the used code smells were defined in [Fowler and Beck 1999] or [Lanza and Marinescu 2006]. On the other hand, a literature review showed that researchers did not study code smells defined in literature systematically and holistic. Rather, they focused on a subset of code smells defined in the literature.

None of the analyzed studies debated the choice of the code smells. It presumably depended on the researcher's predictions of which code smells could lower software quality. Nevertheless, some code smells occur in experiments and case studies more often than others. Our study shows that Feature Envy was expected to be correlated with a low quality of software in most studies. These types of code smells were included in five out of the six analyzed studies. The code smells God Class and Shotgun Surgery occurred in four of the analyzed studies. Brain class and Data Class could be found in three of the analyzed studies. In all of the studies included in our literature review, researchers analyzed 21 different code smells. The use of code smells in research by various authors is summarized in Table III. Therefore, from the analyzed studies it is evident that all code smells do not affect software quality equally.

After a literature review, we are able to provide an answer to RQ2. The summarized view on the collected data is shown in Table IV. The collected data of our analysis indicate a weak correlation between code smells in the source code and an increased incidence of defects in the studied software. In the analyzed studies, the impact on software defects was confirmed for only a few code smells. Additionally, the correlations that were confirmed are quite contradictory when considering all the analyzed studies that address this research question. For example, a positive correlation between God Class and the high rate of detected defects in studied software was identified in two studies. In one study, the authors did not confirm the studied correlation, and in one study the correlation between God Class and defect rate in software was positive, which means that code smells of that type are likely beneficial for software systems from the perspective of studied aspects of software quality. For other types of code smell, the results of the literature review are even more inconclusive. In answer to **RQ2**, we have to agree with the authors who claimed that none of the analyzed code smell classes could be undoubtedly correlated with an increased incidence of defects in studied software by itself.

According to the analyzed studies, an answer to the **RQ3** about the meaningfulness of refactoring of systems affected by code smells is not straightforward. Despite the fact that the studies show that the presence of an individual code smell in source code classes does not predict a higher defect rate in these classes, the increased existence of code smells in the source code of a software system can predict a higher rate of software defects in a software system.

According to some of the analyzed studies, when classes that are affected by code smells are used by client classes in a software system, a higher incidence of software defects is detected for these client classes. In other words, classes containing code smells are not the source of defects by themselves. Rather, these smelly classes could be related to low quality issues for client classes in a software system, which use classes affected by code smells. In addition to the study by [Marinescu and Marinescu 2011], a study conducted by [D'Ambros et al. 2010] also confirmed that a high rate of code smells in the source code increased the incidence of defects in the software.

On the other hand, a study by [Olbrich et al. 2010] advocates a different position. Namely, the study indicates that it may be tolerable and even good for a software system to have some code smells, e.g. God Classes and Brain Classes. According to the analyzed studies, it remains quite unclear, whether the removal of classes affected by code smells increases software quality directly in terms of detected bugs. Namely, quality issues rooted from the use of classes affected by code smells in a software system can remain, even after the removal or refactoring of these classes because of their impact on client classes.

With regard to **RQ4**, our analysis of selected studies showed that studies interested in the correlation between code smells and software defects were conducted in the years between 2006 and 2014. Later research in this field was not found despite the fact that many research questions remain open.

5. CONCLUSIONS

This paper presents the results of a literature review in which we addressed the impact of code smells in source code on software quality and the effectiveness of related refactoring activities. In the software development community, it is commonly believed that code smell in the source code can be correlated with low software quality. For instance, the existence of code smells in the source code can be manifested in the post-release era of software products as an increased rate of identified defects. As extracted from the literature, the correlation between code smell classes and the occurrence of defects in software in these classes is weaker than first assumed. Our findings also show that none of the code smells that were analyzed in the studied literature can be explicitly linked to a higher defect-proneness for these classes. Therefore, classes containing code smells are not a distinct carrier of defects in software systems. The latter has a strong impact on refactoring activities, which cannot be pointed only towards classes affected by code smells.

Despite this, some studies detected an increased occurrence of software defects in pieces of software with an increased incidence of code smells in general, which links code smells with low quality software. For future research, an unanswered research question remains: how exactly do code smells help increase the incidence of defects in other classes in a system, and more importantly, how the negative impacts on quality can be neutralized.

REFERENCES

ABAD, Z.S.H. AND RUHE, G. 2015. Using real options to manage Technical Debt in Requirements Engineering. *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, IEEE, 230–235.

BÁN, D. AND FERENC, R. 2014. Recognizing Antipatterns and Analyzing their Effects on Software Maintainability. .

CHOUDHARY, G.R., KUMAR, S., KUMAR, K., MISHRA, A., AND CAGATAY, C. 2018. Empirical analysis of change metrics for software fault prediction. *Computers and Electrical Engineering 67*, 15–24.

CONEJERO, J.M., RODRÍGUEZ-ECHEVERRÍA, R., HERNÁNDEZ, J., ET AL. 2018. Early evaluation of technical debt impact on maintainability. *Journal of Systems and Software 142*, 92–114.

CUNNINGHAM, W. 1992. The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger 4*, 2, 29–30.

D'AMBROS, M., BACCHELLI, A., AND LANZA, M. 2010. On the impact of design flaws on software defects. *Proceedings - International Conference on Quality Software*, 23–31.

FOWLER, M. AND BECK, K. 1999. *Refactoring : improving the design of existing code*. Addison-Wesley.

LANZA, M. AND MARINESCU, R. 2006. *Object-Oriented Metrics in Practice*. .

LI, W. AND SHATNAWI, R. 2006. An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. .

MARINESCU, R. 2012. Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development 56*, 5, 9:1-9:13.

MARINESCU, R. AND MARINESCU, C. 2011. Are the clients of flawed classes (also) defect prone? *Proceedings - 11th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2011*, 65–74.

MENZIES, T., CAGLAYAN, B., KOCAGUNELI, E., KRALL, J., PETERS, F., AND TURHAN, B. 2012. The PROMISE Repository of empirical software engineering data. .

OLBRICH, S.M., CRUZES, D.S., AND SJOBERG, D.I.K. 2010. Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems. *2010 IEEE International Conference on Software Maintenance*, IEEE, 1–10.

PLOSCH, R., GRUBER, H., KORNER, C., AND SAFT, M. 2010. A Method for Continuous Code Quality Management Using Static Analysis. *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, 370–375.

SJØBERG, D.I.K., YAMASHITA, A., ANDA, B., MOCKUS, A., AND DYBÅ, T. 2013. Quantifying the Effect of Code Smells on Maintenance Effort. *IEEE transactions on Software engineering*.

STEIDL, D., DEISSENBOECK, F., POEHLMANN, M., HEINKE, R., AND UHINK-MERGENTHALER, B. 2014. Continuous software quality control in practice. *Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014*, 561–564.

ZAZWORKA, N., ANTONIO VETRO, B., CLEMENTE IZURIETA, B., ET AL. 2014. Comparing four approaches for technical debt identification. *Software Qual J 22*, 403–426.