Unveiling Multiple Facets of Design Degradation in Modern Code Review

Anderson Uchôa
Opus Research Group, Informatics Department, PUC-Rio
Rio de Janeiro, Brazil
auchoa@inf.puc-rio.br

ABSTRACT

Software design is a key concern in code review through which developers actively discuss and improve each code change. Nevertheless, code review is predominantly a cooperative task influenced by both technical and social aspects. Consequently, these aspects can play a key role in how software design degrades as well as contributing to accelerating or reversing the degradation during the process of each single code change's review. However, there is little understanding about such social and technical aspects relates to either the reduction or the increase of design degradation as the project evolves. Consequently, the scarce knowledge on this topic helps little in properly guiding developers along design-driven code reviews. Our goal in this Doctoral research is three-fold: (1) to characterize the impact of code review and their practices on design degradation over time; (2) to understand the contribution of technical and social aspects to design degradation; and (3) to propose a conceptual framework to support design-decision making during code review. Our preliminary results show that the majority of code reviews had little to no design degradation impact, and that technical and social aspects contribute to distinguishing and predicting design impactful changes.

CCS CONCEPTS

• **Software and its engineering → Software evolution**; **Collaboration in software development**.

KEYWORDS

Design Degradation, Modern Code Review, Influential Aspects

ACM Reference Format:
Anderson Uchôa. 2021. Unveiling Multiple Facets of Design Degradation in Modern Code Review. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3468264.3473099

1 INTRODUCTION

Modern code review is a lightweight, informal, asynchronous, and tool-assisted technique aimed at detecting and removing issues introduced during software development [3]. Both industrial [37] and open-source [36] projects have been adopting modern code review process on a daily basis as a means to promote the quality of their systems [3]. However, this process can be affected by both technical and social aspects. For instance, the technical aspects are those that characterize the source code, the modification of the files, and a textual description of the change. Social aspects characterize the developer's experience, collaboration network, and participation in discussions during a code review.

A key concern of all code review stakeholders, including code owners and reviewers, is to remain aware of ongoing changes impacting the design [44, 45, 48]. However, the social and technical aspects can influence – either alone or simultaneously – how design degradation occurs, and to which extent the degradation can be slowed down or accelerated. For instance, the quality of each code change might be influenced by the developer(s) working on it and the change process itself [33, 34, 45, 48].

Software design degrades whenever one or more symptoms of poor structural decisions (i.e., smells) end up being introduced by a change. If not properly avoided, identified, and combated, design degradation has severe consequences to software maintenance and to the project (dis)continuation in the future [5, 7, 18, 22, 30, 40]. Despite its importance, we know little about how the design degradation evolves over time, both across and within reviews. Thus, we are likely to be misinforming the research and practice of modern code review. Since the code review also aims to improve design quality, one could expect that, over time, the reviews will gradually reduce multiple degradation symptoms.

Furthermore, despite prior studies revealing the positive influence of certain technical and social aspects in software engineering [2, 9, 16, 47], we lack evidence on to what extent these aspects can influence – either alone or simultaneously – the design degradation during code reviews. Hence, a better understanding of these aspects might help us on deriving guidelines for avoiding or mitigating degradation during code reviews. This understanding can also help to improve the current code review practices and develop a new generation of tools for assisting developers on becoming early becoming aware about the design impact during code reviews.

In this context, this Doctoral research focuses on *understanding how to support design-driven code reviews and how technical and social aspects can be positively explored to better support the code review process.* Thus, this doctoral research aims to answer: *How to better support code review stakeholders in combating design degradation of their software systems?.* To answer this question, we aim to employ a mix of scientific methods, including mining-based investigation, survey, interview, and quasi-experimental research.

2 RELATED WORK

We overview the related studies under three different perspectives.

Empirical studies on design degradation. There are multiple studies about the perspective of developers about design degradation [11, 23, 30, 40, 46] and others on the use of diversity and density of symptoms (i.e., smells) as characteristics of design degradation [1, 21, 31]. Sousa et al. [40] identified five categories of symptoms upon which developers often rely to identify design problems. Similarly to other studies [30, 46], they observed that developers tend to combine multiple symptoms, by considering dimensions such as density, diversity, and granularity to decide if there is design degradation. Oizumi et al. [31] investigated if degradation symptoms appear with higher density and diversity in classes refactored by developers. The authors observed that despite not being removed by refactorings, some types of symptoms may be indeed strong indicators of design problems. Ahmed et al. [1] analyzed how open source projects get worse in terms of design degradation. The authors identified strong evidence that the density of design problems builds up over time. Mannan et al. [21] have compared the occurrence of degradation symptoms in Android and desktop systems in terms of their variety, density, and distribution.

Empirical studies on the impact of modern code review on design quality. Prior studies have investigated the impact of code review on design quality [26, 29, 33, 35, 48]. Morales et al. [29] studied the impact of code review coverage (proportion of changes code reviewed) and reviewer involvement on smells. They observed that high coverage and review participation can reduce the occurrence of smells. Later, McIntosh et al. [26] suggested that coverage, reviewers participation, and expertise play high impacts on bug introduction. Pascarella et al. [35] observed that active and participative code reviews have a significant influence on the reduction of code smell severity. Zanaty et al. [48] observed that design-related discussion during code review is still rare. Later, Paixao et al. [33] studied the code review impact on the structural high-level design. They observed that only 31% of the reviews with design discussions have a noticeable impact on the structural high-level design.

Empirical studies on technical and social aspects in modern code review. There are multiple studies about the effect of technical and social aspects in code reviews [4, 6, 10, 13, 38, 43]. Bosu et al. [6] studied the impact of interpersonal relations on the patch review. They found that the code author is one of the important factors for reviewers to decide to review a patch or not. In addition, Ruangwan et al. [38] investigated how many reviewers did not respond to a review invitation. The authors found that the more reviewers were invited to a patch, the more likely it was to receive a response. Thongtanunam et al. [43] investigated patches that do not attract reviewers, not discussed, and receive slow initial feedback. They found that the length of the patch description plays an important role in the likelihood of receiving poor reviewer participation or discussion. Ebert et al. [10] found that missing rationale and lack of familiarity with the code are the major reasons for confusion in code review. Recently, Barbosa et al. [4] observed that communication dynamics among developers and discussion content decay contribute to combating or amplifying design decay.

This doctoral research goes a step further than the previous studies, by explicitly investigating the intersection among the aforementioned perspectives. More specifically, we aim at understating the multiple facets of design degradation in modern code review and how social and technical aspects can influence – either alone or simultaneously – how to design degradation occurs.

3 PROBLEM STATEMENT AND LIMITATIONS

This section presents our research problems. Moreover, we also discuss why the related works do not address such research problems.

Empirical evidence of the impact of modern code review and their practices on design degradation is still scarce. Software design is a key concern in modern code review through which developers actively discuss and improve each single code change. However, there is little understanding of the impact of modern code review on reducing design degradation over time. Even though studies have been investigating the impact of modern code review on design quality [26, 29, 33, 35, 48], most of them only address the relation between modern code review – and their practices – and design degradation superficially. In fact, such studies tend to analyze design degradation considering only single events, such as the introduction of a single design problem [40, 46], or simply analyzing the degradation frequency [11, 31, 32, 35].

In addition, such studies have not assessed how the *process of design degradation evolution* is impacted: (i) *within each single review*, and (ii) *across multiple reviews*. Consequently, one cannot understand how certain *code review practices*, related to review intensity, developer participation, and the review duration, consistently reduce or further increase degradation as the project evolves. Hence, it remains unclear if and to what extent code review helps to combat design degradation. Moreover, there is little knowledge about the impact of developers' design discussions on degradation. Additionally, we do not know which practices may strengthen the combat or the acceleration of degradation. Therefore, aims at deriving this knowledge, our first research problem is states as follows.

Research Problem 1. There is little knowledge about the impact of modern code review and – their practices – on design degradation over time.

The role that social aspects play in distinguishing and predicting design impactful changes is rarely studied. The advantage of using technical and social metrics to characterize and predict failures have widely been studied [2, 9, 16, 47]. However, their use to discriminate and predict design impactful changes is rarely studied [4, 45]. In code review platforms, stakeholders have either technical or social information at their disposal to be used as additional information, both before or after each change. Social information includes the developers' experience, their collaboration network, and participation in discussions during a code review [6, 14, 28, 49]. Technical information includes aspects related to the source code, the history of modification of the files, and a textual description of the change [12, 24, 28]. Technical information is available after changes and revisions done during the review.

In fact, such metrics can act as indicators of design impactfulness of ongoing changes along the code review process [4]. However, there is no empirical evidence about which metrics can distinguish the impactfulness of design changes. Moreover, there is little knowledge about the use of supervised machine learning techniques to assist developers to automatically determine whether a code change is impactful or not. Hence, it remains unclear which metrics, used as features, are the best predictor, as well as the effectiveness of combining social and technical features for predicting design (un-)impactful changes. Therefore, aims at deriving this knowledge, our second research problem is states as follows.

Research Problem 2. There is a lack of empirical evidence on the role of social aspects in distinguishing and predicting (un)impactful design changes.

The contextual information to aid making design decisions along code review is unexplored. Researchers have proposed and investigated solutions based on prediction models to support different code review activities, for instance, promptly suggesting code reviewers [20], or predicting defective parts [17]. Although the use of prediction models is a viable solution, developers are reluctant to trust predictions produced by models without understanding the rationale for those predictions [8]. For instance, a developer would like to understand why a prediction model suggests that a specific source file is degraded so that they can fix the issue. In other words, just warning a file as degraded is often not enough. Moreover, such contextual information is especially important when the model produces predictions that are different from the developer's expectations (e.g., generating alerts for parts of the codebase that developers expect to be clean).

By exploring the contextual information, either technical or social, we could understand the review process better; in particular, which contextual information developers consider important to conduct a code review, how those information needs to evolve during code review, and how this information can be presented as an effective alert. Therefore, aiming at deriving this knowledge, our third research problem is states as follows.

Research Problem 3. There is little knowledge about the technical and social information that may support the design-decision making during code review.

4 PROPOSED APPROACH

The research problems aforementioned are essential to provide the knowledge required to understand (i) the impact of modern code review and their practices on design degradation, (ii) the role that social aspects play in distinguishing and predicting design impactful changes, (iii) the useful contextual information to aid code stakeholders in design-decision making; and, finally, (iv) how to provide the support to design intelligent tools that will aid code review stakeholders to avoid design degradation in practice. Given this context, the goal of this thesis is the following: *Understand how to support design-driven code review and how contextual information – technical and social – can be addressed to support the code review process.* To achieve this goal, we mapped each research problem into three research questions (RQ_s) as follows.

RQ_1: *To what extent do modern code review impact the design degradation evolution?* – **RQ_1** aims at providing evidence on the impact of code reviews on the evolution of design degradation. To this end, we conducted a mining-based investigation and reported the first study that characterizes how the process of design degradation evolves within each review and across multiple reviews. Moreover, we analyzed a comprehensive suite of metrics to enable us to observe the influence of certain code review practices on combating or even accelerating design degradation.

RQ_2: *To what extent social aspects contribute to distinguishing and predicting design impactful changes in modern code review?* – **RQ_2** aims at investigating if code review stakeholders, could benefit from approaches to distinguish and predict design impactful changes with technical and/or social aspects. To this end, we reported an investigation on the prediction of design impactful changes in modern code review. We extracted and assessed 41 different features based on both social and technical aspects of the changes involved in each revision of a code review. Based on different features set, we evaluated the use of interpretable Machine Learning (ML) algorithms to predict design impactful changes. Finally, we evaluated the predictive power of the selected features and algorithms to assist developers to determine whether a code change is impactful.

RQ_3: *What contextual information the code review stakeholders consider useful to support the design-decision making?* – **RQ_3** will help us to know which types of contextual information, either technical or social, are the most useful for helping developers in the design-decision making. To answer this question, we will propose and evaluate a conceptual framework for design awareness contextual information. This framework will be built based on: a survey with developers. After that, we will conduct a controlled experiment with post-experiment interviews to validate the framework.

5 RESULTS ACHIEVED SO FAR

5.1 The Impact on Design Degradation (RQ_1)

Approach. To answer **RQ_1**, we conducted an in-depth empirical study [45] in which we retrospectively investigate 14,971 code reviews from seven systems of two large open source communities. We performed three major steps. First, we characterized the impact of code reviews on the evolution of design degradation. To this end, we explored the evolution of two degradation characteristics: density and diversity of symptoms. We analyzed such characteristics in the context of two categories of degradation symptoms, which are the fine-grained and coarse-grained smells [39]. Finally, we analyzed the impact on design degradation caused by two code review factors. The first factor was the presence of explicit intent of improving the design. The second one was the presence of explicit design discussions along with the revisions of a review.

Second, we investigated how degradation characteristics evolve along with the revisions that occur along each code review. To this end, we identified and investigated four different evolution patterns for degradation characteristics (i.e., density and diversity). Such investigation provided us with new insights about the evolution of design degradation along the reviewing process. Finally, we explored the relationship of different code review practices with the evolution of degradation characteristics. By exploring this relationship, we evidenced that certain code review practices can be used as indicators of increased design degradation. Additionally, we reveal if according to previous studies [25, 26, 35, 42] code reviews that are intensely scrutinized, with more team participation, and reviewed for a longer time, usually has a positive effect on design.

Contributions. Our main findings show that: (1) when developers have an explicit concern with design, the effect on design degradation is usually positive or invariant. However, the sole presence of design discussions is not a decisive factor to avoid degradation; (2) during the revisions of each single review, there is often a wide fluctuation of design degradation. This fluctuation means that developers are both introducing and removing symptoms along a single code review. However, at the end of the review, such fluctuations often result in the amplification of design degradation, even in the context of reviews with an explicit design concern; and (3) certain code review practices increase the risk of design degradation, including long discussions and a high rate of reviewers' disagreement. The finding on long discussions shows that long discussions are introducing more than removing degradation symptoms.

5.2 The Role of Social Aspects (RQ_2)

Approach. To answer $\mathbf{RQ_2}$, we conducted a large-scale empirical study [44] in which we analyzed more than 50k code reviews of seven real-world systems. We performed three major steps. First, we mined a comprehensive set of 41 features able to capture both technical and social aspects of the changes involved in each revision of a code review. Next, we evaluated which metrics can distinguish between design impactful changes and unimpactful ones.

Second, we assessed the use of supervised ML techniques to aid developers in automatically make their decisions. Thus, we compare the performance of six interpretable ML algorithms: Logistic Regression, Naive Bayes, SVM, Decision Tree, Random Forest, and Gradient Boosting. We chose these algorithms since they provide an intuitive and easy to explain the classification output [15, 27]. After, we explored how effective are the social and technical features as a proxy to the design (un)impactful changes. To this end, we evaluated and compared the performance of both kinds of features. Thus, we applied the ML algorithm using three feature sets: a set using only social features, a set using only technical ones, and a set using technical and social features together. Finally, we explored what features are the best indicators of impactful design changes across ML models, by considering the different feature sets.

Contributions. Our key findings are: (1) both social and technical metrics are able to distinguish design (un)impactful changes; (2) features related to the code change, commit message, and file history are effective for differentiating (un)impactful changes; (3) *Random Forest* and *Gradient Boosting* have shown to be the most accurate in predicting design impactful changes; (4) the use of technical features results in more accurate predictions when compared to the social ones. Moreover, such kind of features can be used in combination with social features without reducing the performance of the ML algorithms; and (5) the technical features tend to be considered the best indicators across models when compared with social features. However, social features that quantify the developer's experience are also considered important across models.

5.3 The Contextual Information (RQ_3)

Proposed Approach. To obtain richer data to answer $\mathbf{RQ_3}$, we plan to conduct two major steps. First, we will survey developers to capture their perceptions about influential factors, either technical and social, that most likely to strengthen the combat or the acceleration of design degradation. To this end, we will use the Linaker's [19] guidelines for setting up and conducting a survey. Moreover, we will rely on the knowledge obtained about the code review impact on design degradation ($\mathbf{RQ_1}$), and the role that social aspects play in distinguishing and predicting design impactful changes ($\mathbf{RQ_2}$) to build the survey script. Thus, we aim to both confirm and complement our previous studies. We will rely on Grounded Theory procedures [41] to analyze the survey data. As output this survey, we expect to derive a conceptual framework of influential factors that often are perceived by developers.

Second, based on the conceptual framework derived, we will conduct a quasi-experiment with post-experiment interviews to validate the framework. To this end, we plan to evaluate how the contextual information, either technical or social, helped the developers into two different scenarios: (i) when developers are unaware of the design impact of their change; and (ii) when the developers are aware of the design impact, but they still are not able to see all the ramifications and impacts of their changes along with revisions.

Expected contributions. By investigating this information, we can better understand the code review process, as well as drive the research community toward the definition of methodologies, and envision how tool support can make developers more aware of design quality when verifying newly submitted code changes.

6 CONCLUSION AND NEXT STEPS

This research contains three main novel contributions, which map onto the research questions and are related to the artifacts generated along our research: (i) an empirical characterization of modern code review impact on design degradation; (ii) a catalog about the effect of technical and socials aspects on design degradation during code reviews; and (ii) guidelines for designing intelligent tools that will aid both authors and reviewers during a code review.

This doctoral research is currently in the fifth semester. In our first study [45], we characterized the impact of modern code review and their practices on design degradation ($\mathbf{RQ_1}$). The second study [44] we explored and assessed the role of social aspects in distinguishing and predicting design impactful changes in code reviews ($\mathbf{RQ_2}$). We are currently working on the building of the survey script aims to deriving a conceptual framework about influential factors on design degradation that often are perceived by developers during code reviews ($\mathbf{RQ_3}$). Next, after deriving the conceptual framework, we will conduct a quasi-experiment with post-experiment interviews to validate the framework. The deadline for completing the first step of Section 5.3 is August 2021, when we plan to submit a full paper to an ICSE 2022 co-allocated conference. The second step of Section 5.3 is scheduled to end up until February 2022. We also plan to submit the results of this second step in a full paper to the main track of ICSME 2022. Based on such goals, we are expect to defend the thesis until March 2023.

ACKNOWLEDGMENTS

The author is advised by Prof. Alessandro Garcia from PUC-Rio. This study was financed by the National Council for Scientific and Technological Development (CNPq) under grant 141285/2019-2.

REFERENCES

[1] Iftekhar Ahmed, Umme Ayda Mannan, Rahul Gopinath, and Carlos Jensen. 2015. An empirical study of design degradation: How software projects get worse over time. In *10th ESEM*. 1–10. https://doi.org/10.1109/ESEM.2015.7321186

[2] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, and Jean-Marc Jézéquel. 2020. Sampling Effect on Performance Prediction of Configurable Systems: A Case Study. In *11th ICPE*. 277–288. https://doi.org/10.1145/3358960.3379137

[3] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *35th ICSE*. 712–721. https://doi.org/10.5555/2486788.2486882

[4] Caio Barbosa, Anderson Uchôa, Daniel Coutinho, Filipe Falcão, Hyago Brito, Guilherme Amaral, Vinicius Soares, Alessandro Garcia, Baldoino Fonseca, Marcio Ribeiro, et al. 2020. Revealing the social aspects of design decay. In *34th SBES*. 364–373. https://doi.org/10.1145/3422392.3422443

[5] Terese Besker, Antonio Martini, and Jan Bosch. 2017. Time to Pay Up: Technical Debt from a Software Quality Perspective.. In *CIbSE*. 235–248.

[6] Amiangshu Bosu, Jeffrey C Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. 2016. Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft. *IEEE Trans. Softw. Eng. (TSE)* 43, 1 (2016), 56–75. https://doi.org/10.1109/TSE.2016.2576451

[7] João Brunet, Gail C Murphy, Ricardo Terra, Jorge Figueiredo, and Dalton Serey. 2014. Do developers discuss design?. In *11th MSR*. 340–343. https://doi.org/10.1145/2597073.2597115

[8] Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. 2018. Explainable software analytics. In *40th ICSE-NIER*. 53–56. https://doi.org/10.1145/3183399.3183424

[9] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2010. An extensive comparison of bug prediction approaches. In *7th MSR*. 31–41. https://doi.org/10.1109/MSR.2010.5463279

[10] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. 2019. Confusion in code reviews: Reasons, impacts, and coping strategies. In *26th SANER*. 49–60. https://doi.org/10.1109/SANER.2019.8668024

[11] Andre Eposhi, Willian Oizumi, Alessandro Garcia, Leonardo Sousa, Roberto Oliveira, and Anderson Oliveira. 2019. Removal of design problems through refactorings: are we looking at the right symptoms?. In *27th ICPC*. 148–153. https://doi.org/10.1109/ICPC.2019.00032

[12] Todd L Graves, Alan F Karr, James S Marron, and Harvey Siy. 2000. Predicting fault incidence using software change history. *IEEE Trans. Softw. Eng. (TSE)* 26, 7 (2000), 653–661. https://doi.org/10.1109/32.859533

[13] Wenjian Huang, Tun Lu, Haiyi Zhu, Guo Li, and Ning Gu. 2016. Effectiveness of conflict management strategies in peer review process of online collaboration projects. In *19th CSCW*. 717–728. https://doi.org/10.1145/2818048.2819950

[14] Yujuan Jiang, Bram Adams, and Daniel M German. 2013. Will my patch make it? and how fast? case study on the linux kernel. In *10th MSR*. 101–110. https://doi.org/10.1109/MSR.2013.6624016

[15] Sotiris B Kotsiantis, Ioannis D Zaharakis, and Panayiotis E Pintelas. 2006. Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review* 26, 3 (2006), 159–190. https://doi.org/10.1007/s10462-007-9052-3

[16] Sandeep Krishnan, Chris Strasburg, Robyn R Lutz, and Katerina Goševa-Popstojanova. 2011. Are change metrics good predictors for an evolving software product line?. In *7th PROMISE*. 1–10. https://doi.org/10.1145/2020390.2020397

[17] Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, and E James Whitehead. 2013. Does bug prediction support human developers? findings from a google case study. In *35th ICSE*. 372–381. https://doi.org/10.5555/2486788.2486838

[18] Zengyang Li, Paris Avgeriou, and Peng Liang. 2015. A systematic mapping study on technical debt and its management. *J. Syst. Softw. (JSS)* 101 (2015), 193–220. https://doi.org/10.1016/j.jss.2014.12.027

[19] Johan LinÂker, Sardar Muhammad Sulaman, R Maiani de Mello, Martin H^st, and P Runeson. 2015. Guidelines for conducting surveys in software engineering. *Technical report* (2015).

[20] Jakub Lipcak and Bruno Rossi. 2018. A large-scale study on source code reviewer recommendation. In *44th SEAA*. 378–387. https://doi.org/10.1109/SEAA.2018.00068

[21] Umme Ayda Mannan, Iftekhar Ahmed, Rana Abdullah M Almurshed, Danny Dig, and Carlos Jensen. 2016. Understanding code smells in android applications. In *3rd MOBILESoft*. 225–236. https://doi.org/10.1145/2897073.2897094

[22] Robert C Martin. 2002. *Agile software development: principles, patterns, and practices.* Prentice Hall.

[23] Júlio Martins, Carla Bezerra, Anderson Uchôa, and Alessandro Garcia. 2020. Are Code Smell Co-occurrences Harmful to Internal Quality Attributes? A Mixed-Method Study. In *34th SBES*. 1 – 10. https://doi.org/10.1145/3422392.3422419

[24] Shinsuke Matsumoto, Yasutaka Kamei, Akito Monden, Ken-ichi Matsumoto, and Masahide Nakamura. 2010. An analysis of developer metrics for fault prediction. In *6th PROMISE*. 1–9. https://doi.org/10.1145/1868328.1868356

[25] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2014. The impact of code review coverage and code review participation on software quality. In *11th MSR*. 192–201. https://doi.org/10.1145/2597073.2597076

[26] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Emp. Softw. Eng. (ESE)* 21, 5 (2016), 2146–2189. https://doi.org/10.1007/s10664-015-9381-9

[27] Tim Menzies and Martin Shepperd. 2019. "Bad smells" in software analytics papers. *Inf. Softw. Technol. (IST)* 112 (2019), 35–47.

[28] Audris Mockus and David M Weiss. 2000. Predicting risk of software changes. *Bell Labs Technical Journal* 5, 2 (2000), 169–180. https://doi.org/10.1002/bltj.2229

[29] Rodrigo Morales, Shane McIntosh, and Foutse Khomh. 2015. Do code review practices impact design quality? a case study of the qt, vtk, and itk projects. In *22nd SANER*. 171–180. https://doi.org/10.1109/SANER.2015.7081827

[30] Willian Oizumi, Alessandro Garcia, Leonardo Da Silva Sousa, Bruno Cafeo, and Yixue Zhao. 2016. Code anomalies flock together: Exploring code anomaly agglomerations for locating design problems. In *38th ICSE*. 440–451. https://doi.org/10.1145/2884781.2884868

[31] Willian Oizumi, Leonardo Sousa, Anderson Oliveira, Luiz Carvalho, Alessandro Garcia, Thelma Colanzi, and Roberto Oliveira. 2019. On the density and diversity of degradation symptoms in refactored classes: A multi-case study. In *30th ISSRE*. 346–357. https://doi.org/10.1109/ISSRE.2019.00042

[32] Gustavo Ansaldi Oliva, Igor Steinmacher, Igor Wiese, and Marco Aurélio Gerosa. 2013. What can commit metadata tell us about design degradation?. In *13th IWPSE*. 18–27. https://doi.org/10.1145/2501543.2501547

[33] Matheus Paixao, Jens Krinke, DongGyun Han, Chaiyong Ragkhitwetsagul, and Mark Harman. 2019. The Impact of Code Review on Architectural Changes. *IEEE Trans. Softw. Eng. (TSE)* (2019), 1–19. https://doi.org/10.1109/TSE.2019.2912113

[34] Matheus Paixão, Anderson Uchôa, Ana Carla Bibiano, Daniel Oliveira, Alessandro Garcia, Jens Krinke, and Emilio Arvonio. 2020. Behind the intents: An in-depth empirical study on software refactoring in modern code review. In *17th MSR*. 125–136. https://doi.org/10.1145/3379597.3387475

[35] Luca Pascarella, Davide Spadini, Fabio Palomba, and Alberto Bacchelli. 2020. On The Effect Of Code Review On Code Smells. In *27th SANER*. 1–10.

[36] Peter C Rigby. 2012. Open source peer review–lessons and recommendations for closed source. *IEEE Software* (2012). https://doi.org/10.1109/MS.2012.24

[37] Peter C Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In *9th FSE*. 202–212. https://doi.org/10.1145/2491411.2491444

[38] Shade Ruangwan, Patanamon Thongtanunam, Akinori Ihara, and Kenichi Matsumoto. 2019. The impact of human factors on the participation decision of reviewers in modern code review. *Emp. Softw. Eng. (ESE)* 24, 2 (2019), 973–1016. https://doi.org/10.1007/s10664-018-9646-1

[39] Tushar Sharma and Diomidis Spinellis. 2018. A survey on software smells. *J. Syst. Softw. (JSS)* 138 (2018), 158–173. https://doi.org/10.1016/j.jss.2017.12.034

[40] Leonardo Sousa, Anderson Oliveira, Willian Oizumi, Simone Barbosa, Alessandro Garcia, Jaejoon Lee, Marcos Kalinowski, Rafael de Mello, Baldoino Fonseca, Roberto Oliveira, et al. 2018. Identifying design problems in the source code: A grounded theory. In *40th ICSE*. 921–931. https://doi.org/10.1145/3180155.3180239

[41] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded theory in software engineering research. In *38th ICSE*. 120–131. https://doi.org/10.1145/2884781.2884833

[42] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. 2015. Investigating code review practices in defective files: An empirical study of the qt system. In *12th MSR*. 168–179. https://doi.org/10.1109/MSR.2015.23

[43] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. 2017. Review participation in modern code review. *Emp. Softw. Eng. (ESE)* 22, 2 (2017), 768–817. https://doi.org/10.1007/s10664-016-9452-6

[44] Anderson Uchôa, Caio Barbosa, Daniel Coutinho, Willian Oizumi, Wesley KG Assunçao, Silvia Regina Vergilio, Juliana Alves Pereira, Anderson Oliveira, and Alessandro Garcia. 2021. Predicting Design Impactful Changes in Modern Code Review: A Large-Scale Empirical Study. In *18th MSR*. 1–12. https://doi.org/10.1109/MSR52588.2021.00059

[45] Anderson Uchôa, Caio Barbosa, Willian Oizumi, Publio Blenilio, Rafael Lima, Alessandro Garcia, and Carla Bezerra. 2020. How Does Modern Code Review Impact Software Design Degradation? An In-depth Empirical Study. In *36th ICSME*. 511–522. https://doi.org/10.1109/ICSME46990.2020.00055

[46] Aiko Yamashita, Marco Zanoni, Francesca Arcelli Fontana, and Bartosz Walter. 2015. Inter-smell relations in industrial and open source systems: A replication and comparative analysis. In *31st ICSME*. 121–130. https://doi.org/10.1109/ICSM.2015.7332458

[47] Hamed Shariat Yazdi, Mahnaz Mirbolouki, Pit Pietsch, Timo Kehrer, and Udo Kelter. 2014. Analysis and prediction of design model evolution using time series. In *26th CAiSE*. 1–15. https://doi.org/10.1007/978-3-319-07869-4_1

[48] Farida El Zanaty, Toshiki Hirao, Shane McIntosh, Akinori Ihara, and Kenichi Matsumoto. 2018. An empirical study of design discussions in code review. In *12th ESEM*. 11. https://doi.org/10.1145/3239235.3239525

[49] Marcelo Serrano Zanetti, Ingo Scholtes, Claudio Juan Tessone, and Frank Schweitzer. 2013. Categorizing bugs with social networks. In *35th ICSE*. 1032–1041. https://doi.org/10.1109/ICSE.2013.6606653