Revealing the Social Aspects of Design Decay

A Retrospective Study of Pull Requests

Caio Barbosa, Anderson Uchôa, Daniel Coutinho
PUC-Rio, Rio de Janeiro, Brazil
{csilva,auchoa,dcoutinho}@inf.puc-rio.br

Filipe Falcão, Hyago Brito, Guilherme Amaral
UFAL, Alagoas, Brazil
{filipebatista,hpb,gvma}@ic.ufal.br

Vinicius Soares, Alessandro Garcia
PUC-Rio, Rio de Janeiro, Brazil
{vsoares,afgarcia}@inf.puc-rio.br

Baldoino Fonseca, Marcio Ribeiro
UFAL, Alagoas, Brazil
{baldoino,marcio}@ic.ufal.br

Leonardo Sousa
Carnegie Mellon University, California, USA
leo.sousa@sv.cmu.edu

ABSTRACT

The pull-based development model is widely used in source-code environments like GitHub. In this model, developers actively communicate and share their knowledge or opinions through the exchange of comments. Their goal is to improve the change under development, including its positive impact on design structure. In this context, two central social aspects may contribute to combating or adversely amplifying design decay. First, design decay may be avoided, reduced or accelerated depending whether the *communication dynamics* among developers – who play specific roles – is fluent and consistent along a change. Second, the *discussion content* itself may be decisive to either improve or deteriorate the structural design of a system. Unfortunately, there is no study on the role that key social aspects play on avoiding or amplifying design decay. Previous work either investigates technical aspects of design decay or confirms the high frequency of design discussions in pull-based software development. This paper reports a retrospective study aimed at understanding the role of communication dynamics and discussion content on design decay. We focused our analysis on 11 social metrics related to these two aspects as well as 4 control technical metrics typically used as indicators of design decay. We analyzed more than 11k pull request discussions mined from five large open-source software systems. Our findings reveal that many social metrics can be used to discriminate between design impactful and unimpactful pull requests. Second, various factors of communication dynamics are related to design decay. However, temporal factors of communication dynamics outperformed the participant roles' factors as indicators of design decay. Finally, we noticed certain social metrics tend to be indicators of design decay when analyzing both aspects together.

CCS CONCEPTS

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

KEYWORDS

pull request, design decay, social aspects, social metrics

ACM Reference Format:
Caio Barbosa, Anderson Uchôa, Daniel Coutinho, Filipe Falcão, Hyago Brito, Guilherme Amaral, Vinicius Soares, Alessandro Garcia, Baldoino Fonseca, Marcio Ribeiro, and Leonardo Sousa. 2020. Revealing the Social Aspects of Design Decay: A Retrospective Study of Pull Requests. In *34th Brazilian Symposium on Software Engineering (SBES '20), October 21–23, 2020, Natal, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3422392.3422443

1 INTRODUCTION

Open-source environments, such as GitHub, promote social coding activities. These social activities consist of developers continually communicating and sharing their knowledge along a code change until it is submitted as a pull request [14, 15]. This is known as the pull-based development model [7]. This model allows developers, who play different roles, to submit, review, comment and discuss code contributions to a software project [39]. The pull-based development model is widely used by open-source communities.

One of the goals of pull-based development model is to encourage the improvement of the change under development, including its beneficial impact on the design structure. In fact, recent studies consistently report discussions about design structure are frequent in this development model [6, 28, 51]. However, social activities in pull-based development may contribute to avoiding, reducing or accelerating design decay. Design decay is a phenomenon in which developers progressively introduce code with poor design structures into a system [13, 40].

Two social aspects are central to open-source coding environments: *communication dynamics* and *discussion content* (or *communication content*) [3, 47]. The former determines how the communication flows among developers, who also play specific roles along the change under development. The latter determines characteristics of the contents of comments exchange among developers. These two social aspects may be related to design quality for various reasons, such as two examples described as follows. First, design decay may be avoided, reduced or accelerated depending whether

SBES '20, October 21–23, 2020, Natal, Brazil
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8753-8/20/09…$15.00
https://doi.org/10.1145/3422392.3422443

the communication dynamics of developers with specific roles is fluent and consistent along a change. Second, the characteristics of content of comments can determine the quality of the discussion, and therefore be decisive to either improve or deteriorate the structural design of a system.

Unfortunately, the relationship between these key social aspects and design decay have not been studied so far. Prior works either investigate technical aspects of design decay [11, 18, 27] or confirms the high frequency of design discussions in pull-based software development [34, 44]. Studies of social aspects focus on investigating their relations with post-release defects [3, 12, 17], pull request acceptance [4, 35], and software vulnerabilities [24]. Despite this vast body of knowledge, no study has performed a retrospective investigation on the relationship between key social aspects in pull-based development and design decay. Hence, it is unclear if social aspects, such as communication dynamics and discussion contents, have a positive or negative relationship with design decay.

This paper reports a retrospective study aimed at understanding the role of communication dynamics and discussion content on design decay. We focused our analysis on 11 social metrics related to these two aspects as well as 4 control technical metrics typically used as indicators of design decay. We analyzed more than 11k pull request discussions mined from five large open-source systems. Our findings reveal that many social metrics can be used to discriminate between design impactful and unimpactful pull requests. Second, various factors of communication dynamics are related to design decay. However, temporal factors of communication dynamics outperformed the participant roles' factors as indicators of design decay. Finally, we noticed certain social metrics tend to be indicators of design decay when analyzing both aspects together.

Section 2 provides background information and related work. Section 3 presents our motivating example. Section 4 describes our study settings. Section 5 presents the study results. Section 6 discusses threats to validity. Finally, Section 7 concludes the paper and suggests future work.

2 BACKGROUND AND RELATED WORK

2.1 Pull Request Discussion and Social Aspects

Pull requests contributions are increasing on open-source environments, such as GitHub. Many open-source projects have guidelines to ensure the use of pull requests for this task [14, 15]. The pull request mechanism is basically started by a developer who submits his code contributions to a repository describing his changes, e.g., new features, bug fixes, improvements. Next, the changes are scrutinized by code reviewers of the organization until they are merged or abandoned. In parallel, both contributors and reviewers can interact through different discussions [39]: (i) discussions on the pull request itself, (ii) discussions on a specific line within the pull request, and (iii) discussions on a specific commit within the pull request. Moreover, these discussions may be related or not to the complexity and impact of the changes submitted.

These discussions are not trivial since it may involve different social aspects that capture communication activity among developers and measures the impact of interpersonal relations [47]. Examples of social aspects are *communication dynamics* (or discussion dynamics) and *discussion content*. Communication dynamics represent the role of participants involved in a discussion or temporal aspects of the messages. For instance, a discussion that involves developers with different roles (e.g., members, contributors, or newcomers). On the other hand, the discussion content represents the interaction of developers during the exchange of messages and obtained information about the content of each message. For instance, the presence of code snippets, and the number of words per comment.

2.2 Design Decay and its Symptoms

Software design results from a series of decisions made during software development [41, 42]. However, along with software development, a software design may decay due to the progressive introduction of poor structures into the system, i.e., decay symptoms [13, 40, 45]. Such design decay is caused by a successive increase in the density of symptoms along with software evolution.

Aimed at minimizing and removing decay symptoms, developers need to identify and to refactor source code locations impacted by design decay. Previous studies [25, 40, 49] have identified five categories of symptoms upon which developers often rely to identify structural decay. Such studies observed that developers tend to combine multiples decay symptoms by considering dimensions such as density, and diversity to determine if there is code decay.

In this work, we focus on two categories of symptoms: low-level and high-level structural smells [38]. *Low-level structural smells* are indicators of fine-grained structural decay; their scope is generally limited to methods and code blocks [38]. For instance, the Long Method smell. *High-level structural smells* are symptoms that may indicate structural decay impacting on object-oriented characteristics such as abstraction, encapsulation, modularity, and hierarchy [21, 38]. An example of high-level structural smells that may be used for finding design decay is Insufficient Modularization [38]. This symptom occurs in classes that are large and complex, possibly due to the accumulation of responsibilities. Symptoms of such categories can be automatically detected using a state-of-the-practice tool called DesigniteJava [36].

2.3 Related Work

Social aspects in code review. There are multiple studies about the influence of social aspects during code review [23, 24, 35, 45]. For instance, Ruangwan *et. al.* [35] 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. Nevertheless, Meneely *et. al.* [24] attempted to investigate Linus' law to identify if it applies to security vulnerabilities empirically. In this study, the authors concluded that code files reviewed by more reviewers are more likely to be vulnerable. While those works focus on analysing aspects surrounding code review tasks, our study aims at investigating the social aspects occurring in discussions that are not directly related to source-code.

Social aspects in open-source environments. Previous studies [43, 50] have investigated different social aspects in open-source environments, such as GitHub. Yu *et. al.* [50] not only identified social patterns among developers by mining the follow-networks, but also found that those repositories provide transparent work environments for developers, promoting innovation, knowledge sharing, and community building. Additionally, Tsay *et. al.* [43]

found that the social connection between submitters and core members has a strong positive association with pull-requests acceptance. Previous work focused on open-source environments and their relations. In order to complement these studies, we will use some of the social aspects observed, such as communication dynamics, to assess their influence on design decay.

Effect of social aspects on code quality. Bettenburg and Hassan [3] investigated the impact of social interaction measures on post-release defects. The authors observed that social information can not be used as a substitute to traditional product and process metrics used in defect prediction models. Bird et al. [5] examined the relationship between ownership and failures in two large industrial software projects. They found that the number of developers has a strong positive relationship with failures. Falcão et al. [12] investigated the relationship between social, technical factors, and the introduction of bugs. The authors identified that both social and technical factors can discriminate between buggy and clean commits, such as ownership level of developers' commit, and social influence. These works use social and technical metrics to understand the influence over the presence of code defects. Our work differs from these studies, by analyzing if and what extent metrics related to two social aspects are good indicators of the design decay.

Our work differs from the existing ones in several ways: (i) we analyzed social aspects not only related to code review tasks; (ii) while most studies are focused on analyzing the influence of social aspects on software defects and vulnerabilities, we investigated the influence of social aspects on design decay; and (iii) we used a multiple logistic regression model technique to evaluate which social aspects indicate design decay separately and together.

3 MOTIVATING EXAMPLE

This section presents a real example in which design decay was introduced in the system after merging a pull request. We aim to highlight possible aspects surrounding the discussion that could have possibly indicated that the changes would influence a design decay. For this purpose, we adopt merged pull request #8153 from the Elasticsearch project, to motivate our study. Figure 1 illustrates this pull request in two parts that represent two periods: before and after the merge. We discussed our example step-by-step as follows.

This pull request was titled "*Add inner hits to nested and parent/child queries*" and its main goal was the addition of a new feature. As seen in step ①, the discussion starts with the author reporting that his pull request is related to Issues #3022 and #3152. This discussion evidence the existence of multiple concerns being addressed in the same pull request. Moreover, after a few rounds of review by two different reviewers, this pull request was also mentioned on another Issue (# 761). Further in the discussion (step ②), we can observe that another participant, an user, joins the discussion by asking the author for information about the performance of the feature being developed, in his use case. The author promptly answers the user and returns to discussing his code changes.

Perhaps due to the precedent set, another participant, also an user, joins the discussion (step ③) and also asks something about the feature, and again, the author answers it; but this time, reaches the conclusion that the question relates to another unreleased feature. After this, the user who asked the first question comments again (step ④) giving suggestions about how the feature should be implemented, which this time are never answered by the author.

A few rounds of review later, and the Pull Request is merged. The changes contained in this pull request ended up being large, with one reviewer later saying "*If you perform some changes, please do them as separate commits so that I don't need to review everything again, this change is BIG!*" (shown in step ⑤). When the change was merged, a new issue (#8153) was also associated to the pull request, further indicating that new concerns were added during the development of the code changes (step ⑥).

After the Pull Request was merged, four more users commented asking questions related to the feature being introduced: "*can I bump our use case against this to see if I am understanding this feature correctly?*", "*Has this work be slated for a particular release yet?*", and so on. The discussion in this pull request continued for seven months after the merge, containing multiple questions and suggestions for possible improvements.

In this example, we can observe many social aspects surrounding the discussion, such as: (i) different participant roles (core developers of the organization, contributors and users); (ii) temporal aspects (the time span of the pull request); and, (iii) size and content of comments (snippets being used). One could argue that, some of those social aspects could indicate or even be responsible for changes that lead to introduction of the design decay symptoms into the system. For instance, the presence of users (i.e. participants that never committed on the repository) commenting and raising extra concerns to a pull request, could have raised the complexity of the changes. As a consequence, an increase in the design decay symptoms could have happened.

This example shows a scenario where several confounding aspects created a situation where design decay symptoms were unknowingly increased in the system. By analyzing some of those aspects, such as process metrics relating to the changes and metrics about social aspects related to the discussion, we hope to improve our understanding on some of the situations and behaviors that guide those discussions and the development contained within them, and by doing that, we hope to also improve our knowledge about how those aspects can influence design decay.

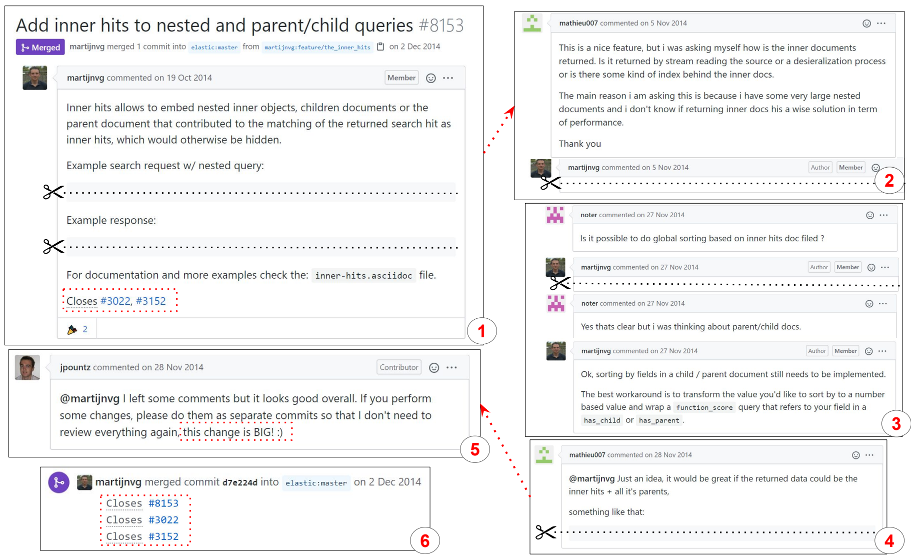
4 STUDY SETTINGS

4.1 Goal and Research Questions

We relied on the Goal Question Metric template [48] to describe our study goal as follows: *analyze* social aspects; *for the purpose of* understanding their impact on design decay; *concerning* changes in structural design quality; *from the viewpoint of* software developers when performing code changes; *in the context of* five open-source systems. We introduce each research question (RQs) as follows.

RQ$_1$: *Are social metrics related to design decay?* – **RQ$_1$** aims at investigating if there is a statistically significant difference between social measures for impactful pull requests and unimpactful ones. We consider that a merged pull request is impactful when an *increase* or *decrease* in design decay was observed as a result of merging the pull request changes. Conversely, unimpactful pull requests are merged pull requests that do not affect on the design decay. Thus, by answering **RQ$_1$**, we will be able to understand which social metrics are more related or not with impactful pull requests.

Figure 1: Discussion in pull request #8153 from Elasticsearch project


Add inner hits to nested and parent/child queries #8153

Merged martijnvg merged 1 commit into `elastic:master` from `martijnvg:feature/the_inner_hits` on 2 Dec 2014

martijnvg commented on 19 Oct 2014 Member

Inner hits allows to embed nested inner objects, children documents or the parent document that contributed to the matching of the returned search hit as inner hits, which would otherwise be hidden.

Example search request w/ nested query:

Example response:

For documentation and more examples check the: `inner-hits.asciidoc` file.

Closes #3022, #3152

mathieu007 commented on 5 Nov 2014

This is a nice feature, but i was asking myself how is the inner documents returned. Is it returned by stream reading the source or a desieralization process or is there some kind of index behind the inner docs.

The main reason i am asking this is because i have some very large nested documents and i don't know if returning inner docs his a wise solution in term of performance.

Thank you

martijnvg commented on 5 Nov 2014 Author Member

noter commented on 27 Nov 2014

Is it possible to do global sorting based on inner hits doc filed ?

martijnvg commented on 27 Nov 2014 Author Member

noter commented on 27 Nov 2014

Yes thats clear but i was thinking about parent/child docs.

martijnvg commented on 27 Nov 2014 Author Member

Ok, sorting by fields in a child / parent document still needs to be implemented.

The best workaround is to transform the value you'd like to sort by to a number based value and wrap a `function_score` query that refers to your field in a `has_child` or `has_parent` .

mathieu007 commented on 28 Nov 2014

@martijnvg Just an idea, it would be great if the returned data could be the inner hits + all it's parents,

something like that:

jpountz commented on 28 Nov 2014 Contributor

@martijnvg I left some comments but it looks good overall. If you perform some changes, please do them as separate commits so that I don't need to review everything again, this change is BIG! :)

martijnvg merged commit `d7e224d` into `elastic:master` on 2 Dec 2014

```
Closes #8153
Closes #3022
Closes #3152
```

RQ$_2$: ***To what extent the communication dynamics influence the design decay?* – During software development, the influence of different social aspects may contribute to design decay. Thus, differently from the previous research question, **RQ$_2$ aims at investigating to what extent multiple social metrics related to the communication dynamics aspect influence design decay. Thus by answering **RQ$_2$** we can evidence whether each social metric, by considering the presence of others, can be used as indicators of design decay.

RQ$_3$: ***To what extent the discussion content influence the design decay?* – Similar to RQ$_2$, we investigate as multiples social metrics related to the discussion content aspect influence design decay. By answering **RQ$_3$, we also can compare which social aspects, i.e., communication dynamics and discussion content when analyzed in isolation, are sufficient or not to indicate a design decay. Furthermore, we can reveal if the combination of these multiple social aspects results in a better indication of the design decay. Our goal is to provide a set of metrics that can be used, in the context of social aspects on discussions inside pull requests, to indicate the increase or the decrease of design decay symptoms. By doing this, we can shed light on future work on social aspects and design decay.

4.2 Study Steps and Procedures

Step 1: Selecting open-source systems. From GitHub, we selected five open-source Java projects that widely adopt pull request-based development. We selected only open-source projects to allow study replication. To select them, we followed criteria based on related studies [12, 43]. We selected systems that matched with the following criteria: (i) systems that use pull request reviews as a mechanism to receive and evaluate code contributions; (ii) systems that have at least 1k commits and pull requests; (iii) systems that are at least 5 years old and are currently active. Moreover, we selected this criteria to avoid known mining perils [19]. Finally, we focused on Java systems due to constraints of the DesigniteJava tool [36] (see Step 2). Table 1 provides details about each selected system. The first column shows the names of each selected system and the remaining columns present: system's domain; number of commits; number of pull-requests; and period considered in this study.

Table 1: Software systems investigated in this study

System	Domain	# Commits	# Pull-requests	Time span	LOC
Elasticsearch	Search Engine	17,251	4598	2011-2018	734,514
Presto	Query Engine	1,958	1,542	2012-2019	635,760
Netty	Framework	4,071	147	2011-2019	279,572
OkHttp	HTTP client	9.690	4,013	2012-2019	36,686
RxJava	Library	4,140	1,299	2013-2016	103,609

Step 2: Detecting multiple design decay symptoms. We used the DesigniteJava tool [36] to detect a total of 27 decay symptoms types: 17 high-level structural smells, and 10 low-level structural smells. Hence, for each system, we identified these decay symptoms by considering each pull request that has been submitted and merged during the project history. For each merged pull request, we have downloaded a snapshot of each commit related to this pull request and its parent. Then, we accessed the difference between them, by following this methodology, we are guaranteeing that the introduced design decay symptoms were solely introduced by the code change in the pull request, this way we avoid the Rebase effect [30, 31]. This is due to such a pull request being the only potential point in time in which the code could be changed. Table 2 lists the 27 symptoms types investigated in our study, where the high-level structural smells and low-level structural smells are presented in the upper and bottom halves of the table, respectively. The descriptions, detection strategies, and thresholds for each symptom are available in our replication package [2].

Step 3: Computing design decay indicators in terms of density symptoms. Based on previous studies [8, 26, 29, 40, 45], we have selected the density of symptoms as indicators of design decay[1]. For this purpose, for each target system, we computed the difference of the indicator for each decay symptom, i.e., high-level

[1]We also compute diversity as an indicator. However, we did not observe any difference in the results of density and diversity. Thus, we decided to use only density.

Table 2: Degradation symptoms investigated in this study

High-level symptoms
Imperative Abstraction, Multifaceted Abstraction, Unutilized Abstraction, Unnecessary Abstraction, Deficient Encapsulation, Unexploited Encapsulation, Broken Modularization, Insufficient Modularization, Hub Like Modularization, Cyclic Dependent Modularization, Rebellious Hierarchy, Wide Hierarchy, Deep Hierarchy, Multipath Hierarchy, Cyclic Hierarchy, Missing Hierarchy, Broken Hierarchy.
Low-level symptoms
Abstract Function Call From Constructor, Complex Conditional, Complex Method, Empty Catch Block, Long Identifier, Long Method, Long Parameter List, Long Statement, Magic Number, Missing Default.

and low-level structural smells, by considering all merged pull requests collected. We computed the density as a sum of the aggregate value of the number of instances of symptoms types in each smelliness file for each version of the system before and after the merged pull request. In summary, a positive difference in the density of symptoms indicates an *increase* in the design decay as a result of the merged pull request, therefore, there is a worsening on the design. Similarly, a negative difference in density of symptoms indicates a *decrease* of the design decay as a result of the merged pull request. Finally, a difference equal to zero in the density of symptoms indicates that there has been no structural design change. In total, we have computed the four indicators for 11,599 merged pull requests. We provide all computed indicators in our replication package [2].

Step 4: Calculating control metrics and social aspects. Table 3 shows the 15 metrics that we have used to measure certain social aspects occurring parallel to the code development. The first part of Table 3 describes the control variables that we computed to avoid some factors that may affect our outcome if not adequately controlled. As control variables, we used *product* and *process* metrics, which have been shown by previous research to be correlated with design decay [20, 32]. The second part of Table 3 describes the metrics that we considered as independent variables to measure certain social aspects. We have grouped each metric in two categories, each one representing a social aspect. *Communication dynamics* represent the dynamic of the discussion activity, such as the role of participants involved in a discussion or temporal aspects of the messages. Finally, *discussion content* represents the interaction of developers during the exchange of messages and obtained information about the content of each message. For instance, the number of snippets written in a discussion. We emphasize that these metrics are extensively used by previous works as reported in [47] to measure the social aspects. Moreover, all two categories investigated in our study suggest social aspects that may be favorable or not the structural design change.

Step 5: Assessing the relationship between social aspects and impactful pull requests. We use a statistical approach to determine which social metrics are able to discriminate between impactful pull requests and the unimpactful ones. We observe that the social metrics are not normally distributed [22]. Thus, we use the *Wilcoxon Rank Sum Test* [46] to decide whether a social metric is *statistically different* for impactful pull requests when compared to the unimpactful ones. The test was conducted using the customary .05 significance level.

Step 6: Evaluating the influence of multiple social aspects on design decay. We assess the influence of each social aspect over the design decay. For this purpose, we created a *multiple logistic regression* model for each aspect, by considering each metric that composes an aspect in the presence of each one other. Additionally, we also created a *multiple logistic regression* model that combines all social aspects and their related metrics together. All the social aspects and their related metrics presented in Table 3 are predictors in the model, and the outcome variable is whether there was decay on the design symptoms related to the merged pull request. We choose a *multiple logistic regression* approach due to the fact that we are studying the effect of multiple predictors (i.e., the metrics) in a binary response variable. We remove from our models the metrics that have a pair-wise correlation coefficient above 0.7 [9] to avoid the effects of *multicollinearity*.

Furthermore, we measure the relative impact to understand the magnitude of the effect of the metrics over the possibility of a merged pull request degrading the system design. We estimate the relative impact using the odds ratio [10]. In our study, odds ratios represent the increase or decrease in the odds of a pull request degrading the system occurring per "unit" value of a predictor (metric). An odds ratio < 1 indicates a decrease in these odds (i.e., a risk-decreasing effect), while > 1 indicates an increase (i.e., a risk-increasing effect). Most of our metrics presented a heavy skew. To reduce it, we apply a $\log_2$ transformation on the right-skewed predictors and a x^3 transformation on the left-skewed. Moreover, we normalize the continuous predictors in the model to provide normality. As a result, the mean of each predictor is equaled to zero, and the standard deviation to one.

To ensure the statistical significance of the predictors, we employ the customary p-value < .05 for each predictor in the regression models. Finally, we also report the amount of deviance accounted for by our *multiple logistic regression* models, in terms of the *D-squared* [16]. Similar to *R-squared* [22] for linear regression models, the *D-squared* represents the goodness-of-fit of logistic regression model, measured by the residual deviance (i.e., the deviance that is unexplained by the model). A perfect model has no residual deviance and its *D-squared* takes the value of 1.

5 RESULTS AND DISCUSSION

5.1 Social Metrics and Impactful Pull Requests

We address **RQ**$_1$ by understanding which social metrics can discriminate between impactful pull requests and unimpactful pull requests. As described in Step 3 of Section 4.2, we consider that a merged pull request is impactful when it *increases* or *decreases* the design decay. Conversely, unimpactful pull requests do not affect the design decay. Table 4 shows the results of the *Wilcoxon Rank Sum test* [46] grouped by social aspects metric, design decay symptom, i.e., low-level and high-level structural smells, and system. Each row represents the p-values of the metrics obtained as results of the *Wilcoxon Rank Sum test* for each grouping. The last column (All) represents the results from all systems combined. The cells in gray represent the p-values that obtained statistical significance (i.e., p-value < .05), where there is a valid distinction between impactful and unimpactful pull requests.

The relationship with impactful pull requests. Table 4 reveals some interesting conclusions. First, we observed that many metrics were *statistically different* for impactful pull requests when compared to unimpactful ones. This observation is consistent in all projects analyzed. Moreover, note that the Discussion Length

Table 3: Control and independent variables used in our study.

Type	Metrics	Description	Rationale
Control variables			
Product	Patch Size	Number of files being subject of review.	Large patches can be more prone to be analyzed for how the involved classes are designed.
	Diff Size	Difference of the sum of the Lines of Code metric computed on the version before and the version after the review of all classes being subject of review	Large classes are hard to maintain and can be more prone to be refactoring [33]
	Diff Complexity	Difference of the sum of the Weighted Method per Class metric computed on the version before and after review of all classes being subject of review.	Classes with high complexity are potential candidates to be refactored
Process	Patch Churn	Sum of the lines added and removed in all the classes being subject to review.	Large classes are hard to maintain and can be more prone to be subject to refactoring.
Independent variables			
Communication Dynamics Aspect	Number of Users	Number of unique users that interacted in any way in a discussion inside a Pull Requests (opened, commented, merged or closed)	This metric allows us to identify discussions with the presence of common users, constant contributors, experienced developers or core members of the project [3]. The classification method can be found on [2].
	Number of Contributors	Number of unique contributors that interacted in any way in a Pull Request (opened, commented, merged or closed)	
	Number of Core Developers	Number of unique core developers that interacted in any way in a Pull Request (opened, commented, merged or closed)	
	Pull Request Opened By	The type of user that has opened each pull request. The user might be an Employee or Temporary. Employees are active contributors and code developers. Conversely, temporary are developers that do not actively work on the project or does not work for the software organization	Pull Requests opened by temporaries have more risk of increasing design symptoms [47]. The classification method can be found on [2].
	Number of Comments	Number of comments inside a Pull Request.	Discussions with a high number of comments around a code change would find possible design symptoms, improving or maintaining the quality [3].
	Mean Time Between Comments	Sum of the time between all comments of a Pull Request weighted by the number of comments.	A higher time between comments (e.g., a long pause in an otherwise fast-paced discussion) are related to design decay [3].
	Discussion Length	Time in days that a Pull Request lasted (difference of creation and closing days).	The longer is the discussion, the higher the chance of problems being explained and solved, avoiding design decay [47].
Discussion Contents Aspect	Number of Snippets in Discussion	The number of snippets inside each comment of a Pull Requests. Those snippets are detected by the number of ''' (syntax that opens a snippet in markdown) divided by two (opening and closing).	The higher the number of snippets in a discussion, the clearer the users are trying to pass a message. Therefore, avoiding confusion and possibly design decay [3].
	Snippet Size	Sum of the size of all snippets found on comments in a Pull Request.	
	Number of Words in Discussion	Sum of the all words of each comment inside a Pull Request. Here we applied the preprocessing in the text removing contractions, stop words, punctuation, and replacing numbers.	Discussions with a high number of words are related to more complex changes, that may lead to design decay [3].
	Number of Words per Comment in Discussion	Sum of the all words of each comment inside a Pull Request weighted by the number of comments. Here we applied the preprocessing in the text removing contractions, stop words, punctuation, and replacing numbers.	Discussions with a high weighted number of words are related to more complex changes, that may lead to design decay [3].

Table 4: Results of the wilcoxon rank sum test grouped by social metrics, design decay symptom and software system

Social Metrics	Elasticsearch		Netty		Okhttp		Presto		RxJava		All	
	High-level	Low-level	High-level	Low-level	High-level	Low-level	High-level	Low-level	High-level	Low-level	High-level	Low-level
# Comments	<.001	<.001	.019	.019	<.001	<.001	<.001	<.001	.026	.026	<.001	<.001
# Users	.008	.008	.060	.060	.093	.093	<.001	<.001	.156	.156	<.001	<.001
# Contributors	.002	.002	.701	.701	<.001	<.001	<.001	<.001	.006	.006	<.001	<.001
# Core Devs	<.001	<.001	.117	.117	.002	.002	.006	.006	.251	.251	<.001	<.001
MTBC	<.001	<.001	.001	.001	<.001	<.001	<.001	<.001	.001	.001	<.001	<.001
DL	<.001	<.001	.034	.034	<.001	<.001	<.001	<.001	.065	.065	<.001	<.001
# of Snippets	<.001	<.001	.022	.022	.007	.007	<.001	<.001	.021	.021	<.001	<.001
NWD	<.001	<.001	.005	.005	<.001	<.001	<.001	<.001	.003	.003	<.001	<.001
NWPCD	<.001	<.001	.004	.004	<.001	<.001	<.001	<.001	.016	.016	<.001	<.001
Snippets Size	<.001	<.001	.022	.022	.011	.011	<.001	<.001	.018	.018	<.001	<.001

(DL) and the Number of Contributors (# Contributors) metrics are statistically different in 4 out of 5 projects (80%). We also observed that the Number of Users and Number of Core Developers metrics reached statistical significance in 40% and 60%, respectively. Additionally, social metrics related to the participant role (communication dynamics) were the ones that presented a more unstable behavior, only 66% of the cases were statistically different. These results show that social metrics can differentiate impactful pull requests from the unimpactful pull requests. Metrics from both social aspects might be good indicators to identify potential impactful pull requests. This result evidence the rationale that pull requests with high values for these metrics need more attention and concern of software organizations. For such cases, software organizations could monitor significant changes in the values of these metrics. Such changes may indicate an increase or decrease in design quality.

Finding 1: Social aspects are able to differentiate impactful pull requests from the unimpactful ones. Besides, software organizations could monitor significant changes to avoid design decay.

5.2 Communication Dynamics and Decay

We address **RQ**$_2$ by understanding the influence of the *communication dynamics* over the design decay, we assessed the effect of each metric that composes this aspect in the presence of each one other. We have applied a *multiple logistic regression* to support this assessment (Step 6 of Section 4.2). The results of this analysis are summarized in Table 5. Each row contains the results of the metrics for each project, divided by high-level structural smells and low-level structural smells. The last column presents the *D-squared* of the regression model and the percentage increase or decrease in the *D-squared* compared to a model with only control metrics. The last row contains the results for the data of all projects combined. Moreover, the grey cells represent the statistically significant metrics (p-value $< .05$) and the arrows represent the following behavior:

risk-increasing (arrow up) and risk-decreasing (arrow down). Finally, the blank cells represent metrics that were removed from the model for being collinear. We discuss the results as follows.

Risk-increasing effect of communication dynamics. Table 5 shows that one out of the five metrics related to a specific role of the developer (i.e, user) involved in a discussion, i.e., the Number of Users (# Users), has a risk-increasing tendency. Additionally, all metrics related to temporal aspects of communication i.e., Mean Time between Comments (MTBC) and Discussion Length (DL) also presented a risk-increasing tendency. More precisely, the metric # Users was statistically significant when we combined the data of all projects. Conversely, the Discussion Length was statistically significant only for the Netty project. Finally, the Mean Time between Comments was statistically significant for two projects (Netty and OkHttp), and when we combined all data.

These results reinforce two rationales. First, pull requests with a high mean time between comments are related to design decay. In other words, pull request discussions with a high delay between the comments lead to poor workflow with little communication dynamics between developers, indicating a risk-increasing effect. The delay is usually related to either the lack of interest or the fact the proper knowledge is being forgotten along the conversation. Second, a higher number of users participating in a discussion may hinder the communication dynamics, mainly when these users are not familiar with a system feature, or just by filling the discussion environment with meaningless messages. For such cases, the developers involved can be induced to increase the complexity of the change. Such changes can lead to an increase in design decay as observed in our motivating example (Section 3).

Moreover, the result about pull requests with long discussions which presented a decay risk-increasing effect was surprising. One may expect that longer discussion increases the chances of issues being explained and resolved, avoiding design decay. Such a result suggests that long discussions do not increase the developer's engagement on being conscious with the design structure, increasing the chances of design decay. We observe this phenomenon in the pull request #1388 from the OkHttp project, for instance.

This pull request was opened by a contributor that explained their code changes in detail through two big comments (131 and 179 words each). Almost a month later, a core member of the project reviews the code and replies to the author with an apology for the delay in the review. Next, the author replies to the reviewer a week later. After a long review process, the core member asked to author why the tests were failing. Then, the author replied with a big comment (213 words) answering the concern about the test of the core member. Moreover, the pull request merge only would happen three weeks later, when, again, the reviewer apologized for his delay. In the end, this pull request did induce an increase in the design decay, confirming our rationale for `MTBC` and `DL` metrics.

These observations suggest that future empirical studies should focus more on the temporal aspects of communication dynamics, which have been neglected so far. When one has to focus on a reduced set of metrics, such metrics of temporal factor should be part of his priorities. Also, the core members of organizations should pay more attention to this kind of social factor to avoid design decay.

Finding 2: Social metrics related to temporal aspects of communication dynamics work better as indicators of design decay increase than metrics related to the role of participants.

Risk-decreasing effect of communication dynamics. The data presented in Table 5 also allows us to observe that metrics that measure the communication flow among developers during a pull request discussion provide a better indication of the design decay than metrics that measure the role of the developers. By complementing the previous finding, we also observed that metrics related to the communication dynamics aspect, in general, help to characterize only the risk-increasing effect. This happens even in the presence of control metrics.

Finding 3: In general, the aspect of communication dynamics is a better indicator of increase in design decay symptoms.

The strength of communication dynamics metrics compared to current models. Finally, we also assess to what extend the metrics related to communication dynamics are complementary with the control variables. As shown in Table 5, we used four control variables that represent variables widely used in the current models of design decay analysis (Patch Size, Diff Size, Diff Complexity, and Patch Churn). We note that in the three cases where communication dynamics metrics were statistically significant in comparison with the control variables, the *D-squared* of the models only increased, ranging from 7.14% (OkHttp) up to 409.75% (all data combined). Such finding indicates that the use of communication dynamics metrics increases the explanatory power of design decay models when compared to models with only control metrics.

Finding 4: The metrics of communication dynamics can increase the explanatory power of current design decay models. Thus, metrics from this aspect are relevant indicators of design decay.

5.3 Discussion Content and Decay

By following the same procedures of the previous research question, in **RQ**$_3$ we aim to understand the influence of the *discussion content* over the design decay. For this purpose, we also assessed the effect of each metric that composes this aspect in the presence of each one other. The results of this analysis are summarized in Table 6, in which we also used the *odds ratio* technique to explain the effect of this social aspect over the design decay. Similarly to Table 5, the grey cells represent the statistically significant metrics (p-value < .05), and the arrows represent the risk-increasing (arrow up) and risk-decreasing (arrow down) effects. The blank cells are the metrics missing due to *collinearity*. Finally, the last column presents the *D-squared* of the regression models.

Risk-increasing effect of the discussion content. Table 6 presents that only the metric Number of Words in Discussion (`NWD`) presented a risk-increasing tendency when we combined the data of all projects. To understand this result we also need to address another metric of the table, the Number of Words per Comment in Discussion (`NWPCD`), which presented a risk-decreasing effect, since the rationale of these two metrics is linked. The metric `NWD` indicated that the higher the number of words in a discussion the bigger would be the risk of design decay. However, the metric `WPCD`

Table 5: Results of the odds ratio analysis for the communication dynamics

System	Symptom	Control Variables				Communication Dynamics Aspect							D-squared
		PS	PC	DC	DS	# Comments	# Users	# Contributors	# Core Devs	OBTemp	MTBC	DL	
elasticsearch	High Level	0.576 ↓	6.992 ↑			0.805	1.076	1.358	1.1	2.006	1,18	1.023	0.077 (-10.46%)
	Low Level	0.576 ↓	6.992 ↑			0.805	1.076	1.358	1.1	2.006	1,18	1.023	0.077 (-10.46%)
netty	High Level				1.835 ↑	1.038	0.997	0.987	1.037	4.336	1.185 ↑	1.302 ↑	0.432 (42.57%)
	Low Level				1.835 ↑	1.038	0.997	0.987	1.037	4.336	1.185 ↑	1.302 ↑	0.432 (42.57%)
okhttp	High Level		3.572 ↑				1.07	0.921	0.978	0.473	1.227 ↑	1.025	0.240 (7.14%)
	Low Level		3.572 ↑				1.07	0.921	0.978	0.473	1.227 ↑	1.025	0.240 (7.14%)
presto	High Level		10.753 ↑			0.747	0.627	0.68	1.265	4.266	1,914	1.022	0.214 (239.68%)
	Low Level		10.753 ↑			0.747	0.627	0.68	1.265	4.266	1,914	1.022	0.214 (239.68%)
rxjava	High Level				1.875 ↑		0.906	1.176	0.938	1.088	1,223	0.89	0.045 (28.57%)
	Low Level				1.875 ↑		0.906	1.176	0.938	1.088	1,223	0.89	0.045 (28.57%)
all	High Level		3.815 ↑			0.911	1.099 ↑	1.001	0.949	1.046	1.178 ↑	1.071	0.209 (409.75%)
	Low Level		3.815 ↑			0.911	1.099 ↑	1.001	0.949	1.046	1.178 ↑	1.071	0.209 (409.75%)

Table 6: Results of the odds ratio analysis for the discussion content

System	Symptom	Control Variables				Discussion Content Aspect				D-squared
		PS	PC	DC	DS	Number of Snippets	NWD	NWPCD	Snippet Size	
elasticsearch	High Level		4.488 ↑		0.817	7.772	1.187		0.137	0.050 (-41.86%)
	Low Level		4.488 ↑		0.817	7.772	1.187		0.137	0.050 (-41.86%)
netty	High Level			1.882 ↑		1.075		1.125	1.115	0.319 (5.28%)
	Low Level			1.882 ↑		1.075		1.125	1.115	0.319 (5.28%)
okhttp	High Level	2.391 ↑				1.508		1.018	0.764	0.215 (-4.01%)
	Low Level	2.391 ↑				1.508		1.018	0.764	0.215 (-4.01%)
presto	High Level	4.461 ↑						1.141	13.848	0.128 (103.17%)
	Low Level	4.461 ↑						1.141	13.848	0.128 (103.17%)
rxjava	High Level				1.858 ↑	0.332		0.804 ↓	3.542	0.045 (28.57%)
	Low Level				1.858 ↑	0.332		0.804 ↓	3.542	0.045 (28.57%)
all	High Level	2.023 ↑				1.097	9.907 ↑	0.105 ↓	0.968	0.109 (165.85%)
	Low Level	2.023 ↑				1.097	9.907 ↑	0.105 ↓	0.968	0.109 (165.85%)

shows us that the higher is the number of words, but weighted by the number of comments, the smaller would be the risk of design decay. Moreover, these two metrics do not conflict, but they are complementary as indicators of design decay.

We observed that the NWD can indicate an increase on design decay when: (1) a discussion has a high number of comments and a high number of words, however, these words are concentrated only in a few comments; and (2) a discussion have a high number of comments and a low-mid amount of words in each comment. Additionally, these high numbers of words concentrated on few comments may be indicators that only few participants were truly engaged or providing useful pieces of information. Hence, the case (2) happens when the conversation: (i) does not contain a set of relevant information, (ii) message contents do not say much.

To exemplify the cases, we can observe the Pull Request #1388 from the OkHttp project, already discussed in this paper, regarding the case (1) aforementioned. The discussion on this pull request presented seven comments, three of them containing over one hundred words (131, 178 and 213), and they were all made by the author of the pull request. However, the other four remaining ones did not have more than 35 words, which were made by a core member working on reviewing the pull request.

These results imply that the size of a discussion is related to design decay. The developers should evaluate the quality of their comments since their size alone do not avoid the design decay.

> **Finding 5**: Discussions with a high number of words, when it is not accompanied by a high number of words per comment, work as indicators of design decay increase.

Risk-decreasing effect of the discussion content As stated in the previous finding, we also observed that the metric Number of Words per Comment in Discussion (NWPCD) presented a risk-decreasing effect. Moreover, this behavior was observed on the RxJava project, and when we combined all data of projects. This result also suggests that, even in situations where the number of comments is low, but the number of words per comment is high, there is a high volume of information that may be related to the complexity of the change. As well as discussion on changes that may affect the structural quality of the system. In other words, there is a concentration of useful comments.

> **Finding 6**: In general, the discussion content can indicate the increase (number of words in discussion) and the decrease (number of words per comment in discussion) of design decay symptoms.

The strength of discussion content metrics compared to current models. Similarly to RQ$_2$, we assess how discussion content metrics can complement control metrics. Table 6 presents the *D-squared* measure of the studied models and the percentage increase or decrease in this measure when compared to current models. In summary, we observed two cases in which discussion content metrics were statistically significant: the *D-squared* of the models increased 28.57% (RxJava) and 165.85% (all data combined). These results indicate that the discussion content metrics are also able to increase the explanatory power of design decay models.

> **Finding 7**: Discussion content metrics may be significant indicators of design decay due to the increase of explanatory power when included in current models.

The communication dynamics aspect vs. discussion content aspect. By comparing the results obtained by each social aspect in isolation (Tables 5 and 6), we observed that the discussion content metrics help to characterize both risk-increasing and risk-decreasing effects. However, the communication dynamics metrics, in general, help to characterize only the risk-increasing effect. This happens even in the presence of control metrics. In summary, this result indicates the future studies should consider both aspects, i.e., communication dynamics and discussion content. However, the discussion content aspect can be prioritized when there is a need for a reduced number of metrics. The metrics of this aspect when combined with control metrics help to characterize both risk-increasing and risk-decreasing of design decay.

Finding 8: The metrics related of discussion content provide a better indication of design decay than communication dynamics metrics, even in the presence of control metrics. In any case, both social aspects have shown to be good selecting indicators of design decay.

When looking at all social aspects together, which behavior do we see? Table 7 presents the results of the odds ratio analysis when we combined our two social aspects (*communication dynamics* and *discussion content*) and using the projects combined data. This table reveals interesting results. First, when we combine the two social aspects, new metrics related to the communication dynamics appear as statistically significant: Number of Comments and Number of Core Developers. Both metrics presented a risk-decreasing effect. Such a result reveals that the communication dynamics is also a good indicator of a decrease in design decay. Finally, the combination of social aspects did not lead to any of the previously discussed metrics being irrelevant.

Table 7: Results of the odds ratio analysis with both social aspects together for all project data

Metrics	High level	Low level
Control variables		
Patch Size	1.971 ↑	1.971 ↑
Diff Size		
Diff Complexity		
Patch Churn		
Communication Dynamics		
Number of Users	1.109 ↑	1.109 ↑
Number of Contributors	0.885	0.885
Number of Core Developers	0.848 ↓	0.848 ↓
Opened By: Temporaries	0.902	0.902
Number of Comments	0.346 ↓	0.346 ↓
Mean Time Between Comments	1.081	1.081
Discussion Length	1.153 ↑	1.153 ↑
Discussion Content		
Number of Snippets in Discussion	1.334	1.334
Snippet Size	0.76	0.76
Number of Words in Discussion	15.288 ↑	15.288 ↑
Number of Words per Comment in Discussion	0.211 ↓	0.211 ↓
D-squared	0.122 (197.56%)	0.122 (197.56%)

These observations suggest that depending on the models you want to build and the aspects to be observed, new social metrics can rise as indicators of design decay. Table 7 allows us to observe that the addition of both social aspects' metrics to the current models resulted in an increase in the *D-squared* by 197.56%. These results indicate that communication dynamics and discussion content accounted for a major contribution of deviance compared to current models. Such a result highlights the importance of social aspects when studying design decay.

Finding 9: When combining both social aspects in the model, two new metrics appear on the model as risk-decreasing indicators, suggesting that different metrics can emerge as design decay indicators in different situations.

6 THREATS TO VALIDITY

Construct and Internal Validity. This study analyzes a range of 27 types of decay symptoms. Thus, our findings might be biased by these types, even though they are commonly investigated by previous works [26, 36, 38]. We have used the DesignateJava tool to detect these symptoms. Thus, aspects such as precision and recall may have influenced the results of this study. However, Designite-Java has been used successfully in recent studies [1, 26, 36, 45], and previous work [37] indicated a precision of 96% and a recall of 99%. The metrics chosen to represent the social aspects may not fully represent all the possible interactions among developers that could lead to design decay. To mitigate this threat, we choose social aspects already analyzed by previous work and metrics of process and product, that are commonly used for design decay analysis.

Conclusion and External Validity. We carefully performed our descriptive and statistical analyses. Regarding the descriptive analysis, three paper authors contributed to the computation of the merged pull request impact on the density of symptoms. With respect to the statistical analysis, the metrics used in this study did not follow a normal distribution due to high skewness. To mitigate that, we used the non-parametric Wilcoxon Rank Sum Test [46] on RQ_1 and, for the regression analysis, we reduced the heavy skew of our metrics by applying $\log_2$ and x^3 transformations. Moreover, since multicollinearity of predictors may heavily affect the results of a multiple regression model [9], we removed from our models the predictors with pair-wise correlations above 0.7 (see Section 4). We also normalized the continuous predictors in the model to ensure normality. Furthermore, in our regression models, we controlled some factors that may affect our outcomes via control variables (see Section 4.2). Finally, we have investigated design symptoms in Java software systems only. Thus, our study results may be biased by the underlying code structure of Java-based systems, although we highlight that Java is one of the most popular programming languages in both industry and academia.

7 CONCLUSION AND FUTURE WORK

This work investigated the relationship between two social aspects, *communication dynamics* and *discussion content*, and design decay. For that, we collected pull request data from five open-source systems, assessed 27 types of design decay symptoms, 4 control variables and 11 social metrics related to two social aspects.

From that analysis, we reached the following findings: (i) social aspects can distinguishing impactful and unimpactful pull requests; (ii) temporal factors of the communication dynamics are a better indicator of an increase in design decay than the role of developers; (iii) the communication dynamics is a better indicator of design decay; (iv) social aspects should be included in current models of design decay analysis, since they improve the explanatory power of the models. We believe that these findings can benefit both developers and software organizations, as they may be concerned with carefully verifying contributions from developers.

As future work, we intend to assess the importance of contribution that are external to the analyzed projects, to better understand the social aspects that relate to design decay, such as developers' experience and their interactions in other communities. Moreover, we intend to expand our work on design decay to analyze a wider amount of technical (e.g., software metrics) and social (e.g., social network measures) aspects.

ACKNOWLEDGMENTS

This work was partially funded by CNPq (grants 434969/2018-4, 312149/2016-6, 427787/2018-1, 421306/2018-1, 309844/2018-5, 1428 5/2019-2, 131020/2019-6), CAPES/Proex (373892/2019-00, 88887.3739 33/2019-00), CAPES/Procad (175956), and FAPERJ (200773/2019, 010002285/2019).

REFERENCES

[1] Mamdouh Alenezi and Mohammad Zarour. 2018. An empirical study of bad smells during software evolution using designite tool. *i-Manager's Journal on Software Engineering* 12, 4 (2018), 12.

[2] Caio Barbosa. 2020. *Replication Package*. Available at: https://guriosam.github.io/revealing_social_aspects_of_design_decay/.

[3] Nicolas Bettenburg and Ahmed E Hassan. 2013. Studying the impact of social interactions on software quality. *Emp. Softw. Eng. (ESE)* 18, 2 (2013), 375–431.

[4] Christian Bird, Alex Gourley, Prem Devanbu, Anand Swaminathan, and Greta Hsu. 2007. Open borders? immigration in open source projects. In *14th MSR*. 6–6.

[5] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don't touch my code! Examining the effects of ownership on software quality. In *13th FSE*. 4–14.

[6] João Brunet, Gail C Murphy, Ricardo Terra, Jorge Figueiredo, and Dalton Serey. 2014. Do developers discuss design?. In *11th MSR*. ACM, 340–343.

[7] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *15th CSCW*. 1277–1286.

[8] Rafael de Mello, Anderson Uchôa, Roberto Oliveira, Willian Oizumi, Jairo Souza, Kleyson Mendes, Daniel Oliveira, Baldoino Fonseca, and Alessandro Garcia. 2019. Do Research and Practice of Code Smell Identification Walk Together? A Social Representations Analysis. In *13th ESEM*. 1–6.

[9] Carsten F Dormann, Jane Elith, Sven Bacher, Carsten Buchmann, Gudrun Carl, Gabriel Carré, Jaime R García Marquéz, Bernd Gruber, Bruno Lafourcade, Pedro J Leitão, et al. 2013. Collinearity: a review of methods to deal with it and a simulation study evaluating their performance. *Ecography* 36, 1 (2013), 27–46.

[10] Anthony WF Edwards. 1963. The measure of association in a 2× 2 table. *J. Royal Stat. Soc.* 126, 1 (1963), 109–114.

[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.

[12] Filipe Falcão, Caio Barbosa, Baldoino Fonseca, Alessandro Garcia, Márcio Ribeiro, and Rohit Gheyi. 2020. On Relating Technical, Social Factors, and the Introduction of Bugs. In *27th SANER*. 378–388.

[13] Martin Fowler. 1999. *Refactoring*. Addison-Wesley Professional.

[14] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *36th ICSE*. 345–355.

[15] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work practices and challenges in pull-based development: the integrator's perspective. In *37th ICSE*, Vol. 1. 358–368.

[16] Antoine Guisan and Niklaus E Zimmermann. 2000. Predictive habitat distribution models in ecology. *Ecological modelling* 135, 2-3 (2000), 147–186.

[17] Ahmed E Hassan. 2009. Predicting faults using the complexity of code changes. In *31st ICSE*. 78–88.

[18] Mario Hozano, Alessandro Garcia, Baldoino Fonseca, and Evandro Costa. 2018. Are you smelling it? Investigating how similar developers detect code smells. *Inf. Softw. Technol. (IST)* 93 (2018), 130–146.

[19] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Emp. Softw. Eng. (ESE)* 21, 5 (2016), 2035–2071.

[20] Foutse Khomh, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2012. An exploratory study of the impact of antipatterns on class change-and fault-proneness. *Emp. Softw. Eng. (ESE)* 17, 3 (2012), 243–275.

[21] Robert C. Martin and Micah Martin. 2006. *Agile Principles, Patterns, and Practices in C# (Robert C. Martin)*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[22] John H McDonald. 2009. *Handbook of biological statistics*. Vol. 2. Sparky House Publishing.

[23] Rafael Mello, Roberto Oliveira, Leonardo Sousa, and Alessandro Garcia. 2017. Towards effective teams for the identification of code smells. In *10th CHASE*. 62–65.

[24] Andrew Meneely, Alberto C. Rodriguez Tejeda, Brian Spates, Shannon Trudeau, Danielle Neuberger, Katherine Whitlock, Christopher Ketant, and Kayla Davis. 2014. An Empirical Investigation of Socio-Technical Code Review Metrics and Security Vulnerabilities. In *6th SSE*. 37–44.

[25] W Oizumi, A Garcia, L Sousa, B Cafeo, and Y Zhao. 2016. Code Anomalies Flock Together: Exploring Code Anomaly Agglomerations for Locating Design Problems. In *38th ICSE*.

[26] 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*.

[27] Willian Oizumi, Leonardo Sousa, Anderson Oliveira, Alessandro Garcia, Anne Benedicte Agbachi, Roberto Oliveira, and Carlos Lucena. 2018. On the identification of design problems in stinky code: experiences and tool support. *J. Braz. Comput. Soc.* 24, 1 (2018), 13.

[28] 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.

[29] Anderson Oliveira, Leonardo Sousa, Willian Oizumi, and Alessandro Garcia. 2019. On the Prioritization of Design-Relevant Smelly Elements: A Mixed-Method, Multi-Project Study. In *13th SBCARS*. 83–92.

[30] Matheus Paixao and Paulo Henrique Maia. 2020. Rebasing in Code Review Considered Harmful: A Large-Scale Empirical Investigation. In *19th SCAM*. 45–55.

[31] 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*.

[32] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Fausto Fasano, Rocco Oliveto, and Andrea De Lucia. 2018. On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. *Emp. Softw. Eng. (ESE)* 23, 3 (2018), 1188–1221.

[33] Fabio Palomba, Annibale Panichella, Andy Zaidman, Rocco Oliveto, and Andrea De Lucia. 2017. The scent of a smell: An extensive comparison between textual and structural smells. *IEEE Trans. Softw. Eng. (TSE)* 44, 10 (2017), 977–1000.

[34] Mohammad Masudur Rahman and Chanchal K Roy. 2014. An insight into the pull requests of github. In *11th MSR*. 364–367.

[35] 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.

[36] Tushar Sharma, Pratibha Mishra, and Rohit Tiwari. 2016. Designite: a software design quality assessment tool. In *1st BRIDGE*. 1–4.

[37] Tushar Sharma, Paramvir Singh, and Diomidis Spinellis. 2020. An empirical investigation on the relationship between design and architecture smells. *Emp. Softw. Eng. (ESE)* (2020).

[38] Tushar Sharma and Diomidis Spinellis. 2018. A survey on software smells. *J. Syst. Softw. (JSS)* 138 (2018), 158–173.

[39] Marcelino Campos Oliveira Silva, Marco Tulio Valente, and Ricardo Terra. 2016. Does technical debt lead to the rejection of pull requests?. In *12th SBSI*.

[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.

[41] Antony Tang, Aldeida Aleti, Janet Burge, and Hans van Vliet. 2010. What makes software design effective? *Design Studies* 31, 6 (2010), 614–640.

[42] Richard N Taylor and Andre Van der Hoek. 2007. Software design and architecture the once and future focus of software engineering. In *FOSE'07*. IEEE, 226–243.

[43] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *36th ICSE*. 356–366.

[44] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *22nd FSE*. 144–154.

[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*. 1 – 12.

[46] Elise Whitley and Jonathan Ball. 2002. Statistics Review 6: Nonparametric methods. *Critical care* 6, 6 (2002), 509.

[47] Igor Scaliante Wiese, Filipe Roseiro Côgo, Reginaldo Ré, Igor Steinmacher, and Marco Aurélio Gerosa. 2014. Social metrics included in prediction models on software engineering: a mapping study. In *10th PROMISE*. 72–81.

[48] Claes Wohlin, Per Runeson, Martin Höst, Magnus Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering* (1st ed.). Springer Science & Business Media.

[49] 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.

[50] Yue Yu, Gang Yin, Huaimin Wang, and Tao Wang. 2014. Exploring the patterns of social behavior in GitHub. In *1st CrowdSoft*. 31–36.

[51] 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*. ACM, 11.