On the Relationship between Design Discussions and Design Quality: A Case Study of Apache Projects

Umme Ayda Mannan
Oregon State University
Corvallis, Oregon, USA
mannanu@oregonstate.edu

Iftekhar Ahmed
University of California, Irvine
Irvine, California, USA
iftekha@uci.edu

Carlos Jensen
Oregon State University
Corvallis, Oregon, USA
Carlos.Jensen@oregonstate.edu

Anita Sarma
Oregon State University
Corvallis, Oregon, USA
anita.sarma@oregonstate.edu

ABSTRACT

Open design discussion is a primary mechanism through which open source projects debate, make and document design decisions. However, there are open questions regarding how design discussions are conducted and what effect they have on the design quality of projects. Recent work has begun to investigate design discussions, but has thus far focused on a single communication channel, whereas many projects use multiple channels. In this study, we examine 37 Apache projects and their design discussions, the project's design quality evolution, and the relationship between design discussion and design quality. A mixed method empirical analysis (data mining and a survey of 130 developers) shows that: I) 89.51% of all design discussions occur in project mailing list, II) both core and non-core developers participate in design discussions, but core developers implement more design related changes (67.06%), and III) the correlation between design discussions and design quality is small. We conclude the paper with several observations that form the foundation for future research and development.

CCS CONCEPTS

• **Software and its engineering → Software development process management**; **Maintaining software**; **Designing software**.

KEYWORDS

Design discussion, Design quality, Code smells, Empirical Analysis

ACM Reference Format:
Umme Ayda Mannan, Iftekhar Ahmed, Carlos Jensen, and Anita Sarma. 2020. On the Relationship between Design Discussions and Design Quality: A Case Study of Apache Projects. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3368089.3409707

ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7043-1/20/11…$15.00
https://doi.org/10.1145/3368089.3409707

1 INTRODUCTION

Good design is key to ensuring the performance, robustness, and maintainability of software [75]. To be able to create a "high quality" product, developers need to not only be aware of the design of the system but also about the underlying design decisions. If developers are unaware of the design decisions, they can make changes contrary to what was intended, setting others up for clashes, which can degrade the overall design quality of the system [74].

Design decisions are often decided and shared informally among developers of a project [40], making it a key challenge to document and share design decisions [40]. This becomes a bigger challenge for open source software (OSS) for several reasons. First, these projects are distributed, involving 100's of developers who are geographically or temporally separated. This means that purposeful discussions are the only way for developers to coordinate and build together. Second, because of their often volunteer nature, OSS projects rarely produce updated design documents and, in some cases, lack formal design documentation altogether [19]. In such cases, design discussions are the only form of design documentation.

Recent work has started to investigate how OSS developers discuss the design and how they find these design discussions later. For example, Brunet et al. [19] and Viviani et al. [75, 76] examined how design discussions are embedded in pull request comments and how it can be difficult for developers to piece together these discussions. Researchers have also developed techniques for detecting design discussions from a (single) communication channel using Machine Learning (ML) techniques [19, 45, 75, 76].

However, we still have gaps in our understanding of how design discussions are conducted in OSS projects—What channels are used? Who participates in these discussions? Who implements design-related changes? Answers to these questions are especially pertinent for OSS projects since they are known to use different channels (project mailing list, issue tracking systems, pull requests, code, etc. [19]) to discuss their work. Design discussions fragmented across channels can make it difficult for developers to retrieve and reconcile the scattered pieces of design discussions. This was indeed the case as one of the respondents in our survey of 130 software developers mentioned "*discussions are fragmented between several[sic] systems and not structured for easy searching*".

Even if developers are able to find the design discussions, it is yet unknown to what extent design discussions in OSS relate to the design quality of the project. Without such knowledge, OSS projects might be discussing design under incorrect assumptions (of how design decisions can be retrieved and used), and researchers might miss opportunities for improving state of the art. To the best of our knowledge, no prior work has investigated how design discussions are fragmented across multiple OSS channels and the relation between design discussions and design quality.

In this work, we seek to understand two broad aspects of design discussions in OSS projects:

RQ1: How are design related discussion conducted in OSS projects?
RQ2: What is the relationship between design discussions and design quality in OSS projects?

To answer our research questions, we performed a mixed-method empirical study of 37 Apache projects. These projects use multiple communication channels (according to their official documentation), making them ideal candidates for our study. We opted to investigate Apache projects because these projects have well-documented discussion procedures. Apache projects initiate discussions on mailing list, and once decided on the high-level design, the discussion moves to issue tracking system, and finally, the code is submitted using a pull request, where another round of discussion may occur [30].

We started by extracting design discussions from multiple channels (developer mailing list, issue tracking systems–Jira and Bugzilla, and pull requests). This gave us a corpus of 1, 661, 922 distinct discussions. We then developed an ML classifier that can identify design discussions from either of the four aforementioned communication channels. Applying the classifier on our corpus, we found that overall 9.34% of all discussions touch on some design aspect. We then validated our findings via a survey of 130 developers from the Apache projects.

Finally, we evaluated the association of design discussions with design quality. Design quality could mean different things to different people. For example, it could mean poor feature selection, terrible UX design, or bad architecture design. While trying to identify a suitable metric for measuring design quality, researchers have investigated different metrics such as Coupling Between Object (CBO), COupling Factor (COF), Weighted Method per Class (WMC), and code smells [13, 20, 24, 44, 47, 56]. We opted to use the occurrence of code smell as a proxy for (poor) software design quality since a large number of studies have used and validated code smells for the aforementioned purpose [12, 13, 24, 25, 44, 46, 47, 52, 52, 56, 69, 70].

Our paper makes the following contributions:

- We present a first study investigating the relationship between design discussions and design quality.
- We present the results of a survey of 130 software developers examining the difficulties that developers face in locating and acting on design discussions in OSS projects.
- Based on our results, we outline implications for developers and researchers, make suggestions for improving existing tools and guidelines to better support design discussion.

The paper is structured as follows Section 2 provides a review of prior research efforts. In Section 3, we present our methodology, the demographics of our corpus, approach of mining discussions, machine learning classifier to classify discussions, data collection process, and surveying developers for answering our intended research questions. In Section 4, we present our findings. Section 5 discusses the results and outlines implications for developers and researchers. Section 7 concludes with a summary of the key findings and future work.

2 RELATED WORK

In the context of software engineering, design is defined both as a process [33] and an artifact [62]. Researchers have investigated both aspects. While investigating the process aspect, researchers looked into how and where developers discuss the design. Previous studies [19, 67] show that in OSS projects design discussions takes place in platforms like email communications, issue trackers, pull request etc. Brunet et al. [19] found that 25% of the discussions are about design. Though many channels are available for discussion, all prior studies focused on analyzing discussions in only one channel [19, 75, 76]. Since many channels are available, discussions can be fragmented, and an in-depth analysis involving all channels is required to gather a better understanding of how and when design discussions happen. To the best of our knowledge, ours is the first study analyzing multiple communication channels in this context.

Researchers have also looked into the artifact aspect of design. Though OSS projects share most of the information in written form, it is very often to find an updated design document in a project's archive. Brunet et al. [19] examined 90 popular projects from GitHub and found that 68% of these did not have any kind of design documents. Cherubini et al. found that developers mostly convey design decisions through temporary drawings [21]. Soria et al. [68] found that design knowledge generated during verbal meetings was not captured in the project archive. Researchers also found that design decisions are lost over time [21].

Since design in OSS is not well documented, recovering it becomes difficult with time. To help recover design decisions, researchers have proposed many techniques. One technique is recovering design by reverse engineering [22] or inferring structural designs from code and logs [53]. Antoniol et al. [14] showed that design could also be recovered by tracing links between code and documentation. Another study recovered architectural design from source code with commits and issues [66].

Researchers have also applied ML based techniques to retrieve design discussions. Brunet et al. applied a machine learning classifier to automatically label design related discussions [19]. Viviani et al. [75] applied a classifier to automatically locate paragraphs in pull request discussions related to design. Mahadi et al. [45] trained a classifier on the dataset created by Brunet et al. [19] and tested it on the dataset of Viviani et al. [75]. However, both of the dataset include discussions only from pull requests. Our study builds a machine learning classifier that automatically labels design-related discussion from different communication channels (project's mailing lists, issue tracking systems, and pull requests).

Different ways of measuring design quality have been investigated. Many researchers [12, 13, 24, 25, 44, 46, 47, 52, 56, 69, 70] have used code smells [32] to quantify design quality. Code smells are symptoms of poor design and implementation choices [32]. Initially, it was introduced to identify potential maintainability issues

in the software system; however, later on, it has been associated with bugs [41, 44, 55], and fault-proneness [34, 79]. Researchers have also investigated how design quality evolves over time and found that small changes accumulating over time cause the design drift [74]. Ahmed et al. [13] found that software design quality, measured using code smell, gets worse over time. Researchers have also studied the relationship between project activities and design quality [52, 59] and found that code reviews have a significant influence on reducing code smells. However, no studies are investigating the relationship between design discussions and design quality. Our study takes the first step towards filling this gap in research.

3 METHODOLOGY

The goal of this study is to understand how design discussions take place in OSS projects and the relationship between design discussions and design quality. In the following subsections we describe the pipeline we followed to collect, process and analyze the data.

3.1 Data Collection

3.1.1 Project Selection. Our overarching goal is to identify design discussions in OSS projects and investigate their relationship with design quality. We start by selecting 37 Apache projects written in Java. We selected Apache projects because they use multiple communication channels (mailing list, issue tracking system, pull request, etc.) and have specific instructions for using these channels. Selecting random projects from GitHub without a well-documented discussion procedure would make it difficult to ensure that we have identified all communication channels. Since our goal was to understand how discussions are scattered across all channels, identifying all channels is of the utmost importance. We selected projects written in Java since it is one of the most popular programming languages [73], and there is more design quality (code smell) detection tools available for Java than other languages [28]. We downloaded the git repositories for these projects to collect various information such as code smell, project duration, developer information etc. Table 1 provides a summary of our selected projects.

Table 1: Project Statistics

Dimension	Max	Min	Average	Median
Line count	18,474,542	82,303	1,770,088.82	1,074,659.50
Duration (weeks)	1,063	214	606	560.14
# Developers	1,852	21	226.44	105.50
Total Commits	80,277	3,561	22,688.05	18,083.50
Total Discussions	117,995	594	86,599.33	38,195

3.1.2 Discussion Collection. For these 37 Apache projects, we found the list of communication channels used by them. From the developer mailing list, we collected 1, 437, 753 emails as discussions. We also collected discussions from Bugzilla (67027), Jira (134, 312), and pull requests(22, 362). The entire thread of interactions (emails-grouped by the subject header, comments grouped by issue-ID) was the analysis unit and considered a (single) discussion. In total, our initial data set contained 1, 661, 922 of such discussions.

3.1.3 Unit of Measure Selection. To investigate how design discussions correlate with design quality over time, we could use different ways of partitioning time. Some researchers [36, 37] have used releases as the unit of time; others have used individual commits, or discrete-time units (years, months, weeks, days) [13, 48]. Individual commits are the only "level" measure but would be too fine-grained for our purpose. Thus, similar to Ahmed et al. [13], we selected the week as our unit of measure because it gives us enough detailed insight into the evolution of projects. We then checked the distribution of commits across the projects' history and found that the majority (90%) of the projects had an active history of 560 weeks or less. We cut off our analysis at 560 weeks to prevent extremely long-lived projects from skewing the results.

3.2 Building the Discussion Classifier

To answer our research questions, we need to separate design discussions from other discussions. As manual classification is not a practical option to classify 1, 661, 922 discussions, we use machine learning techniques. We followed the protocol of Brunet et al. [19] with some improvisations suggested by Mahadi et al. [45].

3.2.1 Step 1: Manual Classification of Sample Data. To ensure that our manually labeled dataset contained discussions from all four channels, we took the following steps: (1)an initial random selection of 500 discussions from each channel so that each channel is "represented", and (2) an additional random selection of 3, 000 discussions agnostic of the channel type. In table 2, column three represents the total number of discussions taken from each channel after these two steps. This resulted in a total of 5, 000 discussions, which were used to build and evaluate our ML classifier.

Next, two of the authors manually labeled 2, 000 discussions independently (40% of the corpus). We did not discuss any specific rules except for focusing on the structural design aspects of the code to classify the discussions. This was to avoid bias in our independent classification of the discussions. Also, our initial manual investigation of the discussions found that keywords used in design-related discussions are diverse. As a result, only focusing on keywords would not capture all the design discussions. Therefore, instead of focusing on keywords, we focus on the semantics in the discussions. While reading the discussion, the researchers paid attention to discussion regarding "structural design", "code architecture", anything regarding the "restructuring of code", and such kinds of high-level topics. To remove subjectivity, two researchers labeled each discussion either as a design or non-design discussion. Then we calculated the inter-rater reliability using Cohen's Kappa and found a Kappa value of 0.88. Cohen's kappa is a statistic that assesses the degree of agreement between the codes assigned by two researchers working independently on the same sample [54]. Values of Cohen's kappa fall between 0 and 1, where 0 indicates poor agreement and 1 indicates perfect agreement. According to the thresholds proposed by Landis and Koch [43], kappa value of 0.88 indicates an almost perfect agreement between the researchers.

Once the agreement was reached, one author manually classified the rest of the sample data. In our final sampled data, 998 (20%) discussions are design related and 4, 002 (80%) are non-design related discussions. Table 2 shows the distribution of manually classified sample discussions across channels.

Table 2: Distribution of manually classified sample across different Channels.

Channel	Total Discussion	Total Sample Discussion	# of Design Discussion	# of Non-Design Discussion
Mailing List	1,437,753	3,000	600	2,400
Bugzilla	67,027	647	129	518
JIRA	134,312	797	159	638
Pull Request	22,830	556	110	446
Total	1,661,922	5,000	998	4,002

3.2.2 Step 2: Natural Language Pre-processing. We followed some pre-processing steps to clean the data before applying a machine learning classifier and used the NLTK library [7] for this. We remove stop words and applied Lemmatization to normalize the data. We added some domain-specific words in the stop-word list that are not part of the predefined English stop words list [8]. For example, we added name of the days in week, name of the months, special character sequence such as "»" or "#" etc. The full list of stop words can be found in our companion website [77]. Then we use lower-case letter conversion. After removing stop words and doing the letter-case conversion, we used TF–IDF for vectorizing the natural language words into numerical vectors. More specifically, we used TfidfVectorizer [11] provided by Scikit-Learn.

3.2.3 Step 3: Machine Learning Classifier. Using the manually classified corpus, we train four different machine learning classifiers, a Multinomial Naive Bayes (MNB) [6], a Decision Tree(DT) [3], a Support Vector Machine (SVM) [10] and a Logistic Regression (LR) [5]. We use Python Scikit-learn library [65] to implement the classifiers. We used 10-fold cross validation to train and evaluate the classifiers [38]. This validation approach randomly divide the manually classified data set into 10 groups of equal size. The first group is treated as a validation set, and the classifier is fit on the remaining 9 groups. The mean of the 10 executions is used as an estimation of classifier's accuracy. 10-fold cross validation has been recommended in the field of applied machine learning [42].

After 10-fold cross validation, we compared the classifiers using the Area Under the Receiver Operating Characteristic Curve (AUC) [26]. AUC returns a scores from 0 to 1 to represent prediction performance of a classifier. A classifier with an AUC score higher than 0.5 indicates that it is performing better than random chance. An AUC score above 0.7 is often considered to have adequate classification performance [63]. We choose to use AUC instead of F1-score for comparing classifier performances for several reasons. First, it is independent of prior probabilities [17]. Second, AUC is not biased by the size of test data. Finally, AUC provides a "broader" view of the performance of the classifier since both sensitivity and specificity for all threshold levels are incorporated in calculating AUC [61].

Table 3: AUC for classifiers

Classifier name	AUC
Decision Tree	**0.86**
Logistic Regression	0.77
SVM	0.74
Multinomial Naive Bayes	0.50

Table 3 shows that the Decision Tree achieved the highest AUC score of 0.86 compare to other classifiers. We, therefore, used the Decision Tree classifier to label the remaining discussions automatically. We performed hyper-parameter optimization using randomized search [16] for all four classifiers before selecting the decision tree. Randomized search sets up a grid of hyper-parameter values and selects random combinations to train the model and score on the validation data. The number of search iterations is set based on time/resources; in our case, it was 10. We search for the optimal parameter settings using the Scikit-Learn RandomizedSearchCV function. Our hyper-parameter grid for the best classifier (Decision Tree) includes "max_depth","max_features", "min_samples leaf" and "criterion". The final tuned decision tree parameters for our classifier were "min_samples_leaf: 5, max_depth:3, max_features: 5, criterion: gini".

3.3 Developer Categorization

To answer our research questions, we needed to identify developers' status in the project and their participation in the discussion. We used the number of commits contributed by individual contributors in the code base as the criterion to classify a developer as core or non-core of the project [13, 51].

In order to calculate the status of developers, we start by collecting developer names and corresponding email addresses from the project repository. After doing so, we noticed some miss matches. For example, multiple slightly different names with the same email addresses and same name with different email addresses. In our data set, 45% of the developer names had this issue. To identify the developers' unique list, we did a two-way matching of name and email address. First, we identified all names attached to each email address and all email addresses attached to a name. Then we identify all unique pairs of names and email addresses. We were able to match 98.5% of email addresses with names.

Next, we categorized developers into core and non-core groups according to their code contributions in the projects. Open source contribution follows a power law, where 20% of contributors are responsible for 80% of the contributions [51]. We follow the same rule to identify the core and non-core developers. We consider a developer as *core* if the developer is among the top 20% of developers in that project (calculated by the number of commits authored). Otherwise, the developer is *non-core*.

3.4 Code Smell Collection

To examine the relationship between design discussions and design quality, we collected code smell (an indicator of design quality) for the 37 selected projects.

3.4.1 Code Smell Detection Tool Selection. We used inFusion [2, 57], a commercial tool to identify code smells. We selected inFusion for several reasons: First, inFusion detects a wide range (20) of code smells [57]. As a commercial tool, inFusion is no longer available for download. However, there is an open source version of the tool called iPlasma [1], which is available. It uses static code metrics to identify design related issues. Details of the metrics along with the definition of the code smells are provided in the companion website [77]. Second, a previous study by Ahmed et al. [13] showed that inFusion has a high precision of 0.84, recall of 1.00, and an F-measure of 0.91. Also, inFusion scales well to large codebases. Finally, many other studies [27, 29, 35] used inFusion as their code smell detection tool.

3.4.2 Measuring Code Smells. For each of the 37 projects, we collected code smells by running inFusion on their codebase. First, we collected the number of code smells in each code smell category per week. Then we add all smells across all categories to collect total code smells for each project.

3.5 Data Analysis

We calculated the number of design discussions and code smells for each project over 560 weeks. To compare these two time series, first, we normalized the data since the number of discussions and code smells will vary according to the size of the development team and project. There are many ways of normalization, and the most commonly used one is dividing the data by the lines of code. However, in our case, dividing the number of discussions by total lines of code does not necessarily give us a meaningful measure. Instead, we normalized both the number of design discussions per week and the number of code smells per week using the feature scaling [18], which gives a score between 0 and 1 (Equation 1).

$$\text{Rescaled value} = \frac{x - min(x)}{max(x) - min(x)} \quad (1)$$

Where x = each data point. min(x) = The minimum among all the data points. max(x) = The maximum among all the data points.

Our goal is to identify any correlation between two time series data, which are the number of design discussions per week and the number of code smells per week. We calculate the cross-correlation between design discussions per week and count of code smells per week individually for each project for this purpose [50]. As the first step of time series analysis, we start by checking if there are any visible trends. If a time series exhibits a visible trend, we need to remove the trend before further analysis. This is called the detrending. We applied the first differencing method to detrend both time series [50], and after that, we calculate the cross-correlation.

3.6 Survey

To validate our findings from mining archival data, we conducted an online survey of software developers working in Apache projects. This section describes the survey design, participant selection criteria, pilot testing, data collection, and data analysis.

3.6.1 Survey Design. We designed an online survey to identify the issues associated with accessing and implementing design-related discussions using Qualtrics survey system [9]. We gathered demographic information to understand the respondents' backgrounds (e.g., number of years as a professional developer, number of years contributing to open source project). Next, we asked them how they conduct design-related discussions in the OSS project and what are the problems they faced while finding design-related discussions. We asked the respondents to rate their difficulty level of finding design discussions in each communication channel using a Likert scale where the options are *Extremely easy, Somewhat easy, Neither easy nor difficult, Somewhat difficult, Extremely difficult.* We used multiple choice for some of the questions. A text box option was provided for respondents in the multiple choice questions in case they wanted to share the reason behind their choice. The survey instrument is available in the companion website [77].

3.6.2 Participant Selection. For our survey, we recruited software developers from the 37 Apache projects we analyzed. To build our participants list, we created the list of unique email addresses of individuals from the version control system. In total, we had 7,682 unique email addresses on our list. We selected a random sample of 2,000 for potential participants from the list and sent a targeted email inviting them to our survey.

3.6.3 Pilot Survey. To help ensure the survey's validity, we asked Computer Science professors and graduate students (Two professors and 5 Ph.D. students) with experience in OSS and survey design to review the survey. To make sure that the questions were clear and complete, we conducted several iterations of the survey and rephrased some questions according to their feedback. We also focused on the time limit to ensure that the participants could finish it under 10 minutes. The survey was anonymous, but at the end of the survey, we gave the participants a choice to receive a summary of the study through email.

3.6.4 Data Collection. After sending an invitation email to 2,000 potential participants, 1,980 invites were delivered via Qualtrics [9]. 20 failed to deliver likely due to invalid email addresses. 946 of those emails bounced and we received 22 automatic reply notifying the respondent's absence. Our final count of potential participants were 1,012. According to the Software Engineering Institute's guidelines for designing an effective survey [39], *"When the population is a manageable size and can be enumerated, simple random sampling is the most straightforward approach"*. This is the case for our study with a population of 7,682 software developers from the selected Apache projects.

We received 110 responses from 1,012 email requests during first 10 days. Then we sent a reminder email. After the reminder, we received 30 more responses in the next 10 days. We sent out a second reminder email 10 days after the first reminder and got 12 additional responses. In total, we received 152 responses from 1,012 email requests (15.01% response rate). Previous studies in Software Engineering field have have reported response rates between 5.7% [60] and 7.9% [49]. We discarded 22 partial responses, which left us with 130 responses (12.9% response rate after excluding the

partial responses). Our respondents' software development experience varies from 3 years to more than 20 years, with an average of 15.21 years.

3.6.5 Data Analysis. We collected the ratings our respondents provided for each question, converted these ratings to Likert scores from 1 (Extremely difficult) to 5 (Extremely easy) and computed the average Likert score. We also extracted comments and texts from the "other" fields by the survey respondents explaining the reasons behind their choices. To further analyze the results, we applied Scott-Knott Effect Size Difference (ESD) test [71] to group the difficulty level of finding design discussion in each channel into statistically distinct ranks according to their Likert scores. Tantithamthavorn et al. [71] proposed ESD as it does not require the data to be normally distributed. ESD leverages hierarchical clustering to partition the set of treatment means (in our case: means of Likert scores) into statistically distinct groups with non-negligible effect sizes.

4 RESULTS

We structure our study findings based on our research questions.

4.1 Design Discussions in OSS Projects (RQ1)

As a first step, we investigate how often and where OSS contributors discuss design related matters:

RQ 1.1: What is the prevalence of design related discussion and which channels are used for such discussions?

To answer this research question, we collected all discussions from developer mailing list, issue tracking system (JIRA and Bugzilla) and pull requests. We built a machine learning classifier (discussed in section 3.2) that labeled 155, 175 (9.34%) discussions as design related out of 1, 661, 922 total discussions.

Some example of design discussions labeled by the classifier are:

- *"I do not think this is by design. I think it is so today because... UI should be able to send this every time a call is made - please open a JIRA for this".*
- *"I'm really confused. I have tried looking at the code, but I got lost in the tangle of Contexts, Containers, Wrapper, Valves, Dispatchers...and I gave up. BTW, Craig, is there a design document anywhere for Catalina?".*
- *"did you guys ever come up with any sort of design document? Looking back at the last chatter, we were still in a localfs WAL capability".*

Next, we analyze where these design discussions take place by grouping discussion by communication channel. Table 4 shows the percentage of design discussions distributed across the three types of channels.

Dev-mailing lists contain the vast majority of design related discussions, accounting for 89.51% of the total design discussions from all channels, with 9.34% of the total discussions in that channel. Issue tracking (comments) were the next used communication channel with 9.66% of all design discussions across all channels being conducted there and 7.44% of Issue Tracker discussions being about design. Pull requests were the least used medium—a scant 0.84% of design related discussions being through pull request comments; and 5.69% of all pull request comments being about design.

Table 4: Channel wise design discussions.

Communication Channel	Design Discussions	% of total Design Discussions
Dev-mailing list	138, 891	89.51%
Issue tracking system	14, 986	9.66%
Pull requests	1298	0.84%

Survey Triangulation: Our survey responses (from 130 developers) confirm these findings, which indicates that project mailing list and issue tracking system were the top two communication channels for discussing design. Project mailing list was considered the preferred channel by 57.69% of the respondents, whereas issue tracking system was the preferred choice for 54.62% respondents (see Figure 1).

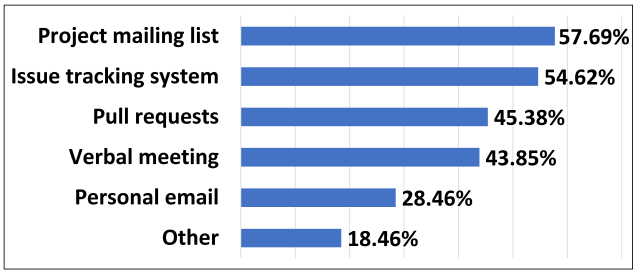

Figure 1: Communication channels used by survey respondents

Observation 1: Design discussions in our selected projects appear across all channels but project mailing list is the top choice among developers to discuss design.

We also ask our participants to rate the difficulty of retrieving design discussion from the three communication channels. To that end, we presented the communication channels and asked the developers to rate them (See Section 3.6.5 for details). Dev-mailing lists ranked the most difficult and issue tracking system ranked the least difficult according to the Scott-Knott ESD test in terms of means of Likert scores for all the respondents.

Observation 2: Mailing list is the most difficult channel in terms of retrieving design discussions.

Respondents from the survey mentioned that they often also use *unofficial channels* like personal emails, verbal meetings, etc. for design related discussion. In fact, 76.21% respondents mentioned using "verbal meeting". This is troubling since these conversations are not archived and inaccessible to others or for later review. Another 23.79% mentioned "personal email" as a medium for design discussions in the project. While this leaves some trace, it is still not recorded in the project archives or available to the community—reducing transparency of decision making in the project.

Observation 3: Design discussions occur via both documented and undocumented channels.

RQ 1.2: Who are involved in design discussions in the projects? We answer this question by categorizing the developers into core and non-core developers according to their amounts of code contribution to the project (See section 3.3 for details). It is important to investigate this question because, if design discussions in a project are controlled by a small, core group of contributors, it can leave out a large group of non-core contributors. In such a situation, non-core members may be unaware of design decisions and implications of these decisions on their proposed changes, making these changes suboptimal or incompatible with the rest of the project.

In our dataset, a majority of the design discussions were from the core group (67.48%); The non-core group was responsible for 28.54% design discussions. At first glance, it seems that non-core groups are in the minority and unable to participate, but recall that by definition non-core groups have fewer overall (code) contributions, which also translates to fewer instances of participation in discussions. Therefore, we normalize the design discussion of each group with the total participation in discussions using the equation below.

$$\text{Rescaled value} = \frac{\text{Total design discussions by the group}}{\text{Total discussions done by the group}} \quad (2)$$

This normalization shows that the mean of design discussions by core developers is 0.12, whereas the mean design discussions of the non-core developers is 0.06. These numbers indicate that after accounting for the amounts of contributions, core members were more involved in the design discussions of the project. While the difference between the groups is not statistically significant (Welch Two Sample t-test, $p > 0.05$), the effect size is small (0.30, Cohen's d [23]).

Survey Triangulation: In our survey, we asked respondents, "Who are more involved in design related decisions?" after defining core and non-core groups. 46% of the respondents answered that both groups were involved in design related discussions in their projects (See Figure 2.a); confirming our results showing that both groups are active in design discussions.

Observation 4: Both core and non-core developers participate in design discussions, with some difference in the amount of contributions between the two groups.

RQ 1.3: Who implements changes resulting from design discussions? Given that both core and non-core developers participate in design discussions, the next question is whether there are any differences in who makes changes resulting from these design discussions. It is possible that, non-core developers participate in discussions, but core developers–who typically have longer tenure and have a broader view of the project—serve as gatekeepers to design related changes.

To investigate this, we first identified the assignee of each issue in the issue tracking systems (JIRA and Bugzilla). In our data set, 14,986 issues were related to design discussions. Of these 7,310(48.77%) were assigned to core developers and 4,696(31.33%) to non-core developers. We did not find any assignee for 2,980 (19.88%) issues.

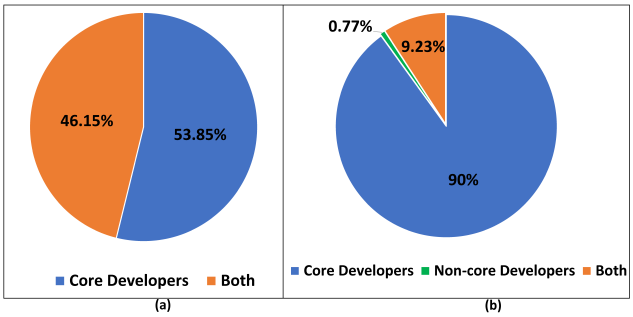

Figure 2: Percentage of survey responders describing participation of core and non core developers in a) design discussion, b) implementing design discussion

In the next step, we linked the assigned issues to the commits in GitHub, so as to discount those issues that did not result in changes to the project (and remain uncommitted to the version history). Out of 12,006 issues (combined issues of core and non-core) we could link 11,228 (93.52%) issues. Of these 11,228 linked issues, 7,530 (67.06%) were authored by core developers and 3,698 (32.93%) by non-core developers. We were unable to link 778 (6.48%) issues due to the missing link problem [15, 64].

To understand whether core developers were more involved in implementing design related changes, we compared the mean number of design-related changes by core developers with that of non-core developers. The results show that the two groups are significantly different (Welch Two Sample t-test, $p < 0.05$), with a medium effect size (Cohen's d [23] of 0.77).

Survey Triangulation: Among the 130 survey respondents, 90% reported that design discussions were implemented mostly by core developers (See Figure 2.b) which triangulates our findings.

Observation 5: Core developers are responsible for implementing a majority of design discussions despite core and non-core developers participate equally in design discussions.

RQ 1.4: How do design discussions evolve? When a project matures, it is possible that it needs fewer design discussions as the design is already stable by then. However, on the other hand, as the project grows, adding new features may require more detailed discussions so as to ensure that different features are compatible. Additionally, in the case of OSS projects, as the project matures and gains visibility, more contributors join the project bringing with them "fresh" design ideas or needing to understand the core design decisions, which can affect how design discussions occur in these projects.

We tracked the design discussion evolution of each project over a period of 560 weeks. First we analyze the overall design discussion trend across all 37 projects, we found that in most projects, design discussions follow a *decreasing* trend, and discussions start to drop after approximately 250 weeks (Figure 3). When comparing across projects, three unique trends emerged. Figure 4 presents examples

of these trends using specific projects from our dataset. Table 5 shows the percentage of projects in each design discussion trend category. The first trend, *Decreasing discussions*, includes projects where design discussions decrease as the project matures, manifests in "Maven" project. The second trend, *Constant discussions* shows the amount of design discussions stay stable over time, despite several peaks; "Log4j" is an exemplar of this trend. *Increasing discussions* trend includes projects where design discussions increase as seen in the project "ant".

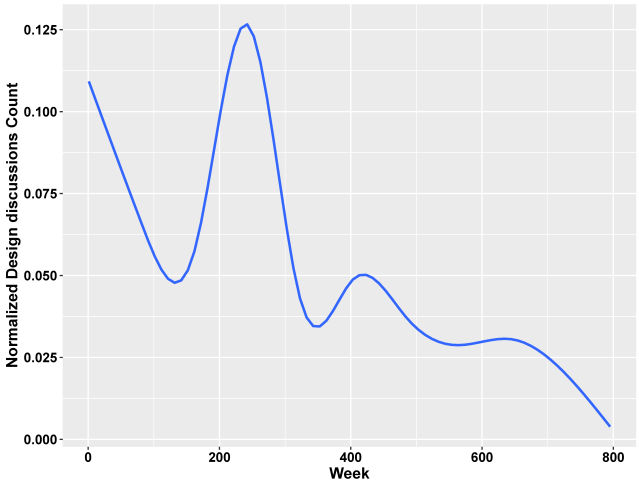

Figure 3: Week-wise average design discussions across all projects.

Table 5: Number of projects in each design discussion trend category

General Trend	Number of projects(%)
Decreasing	21 (56.76%)
Constant	11 (29.73)%
Increasing	5 (13.51)%

Observation 6: Distinct patterns of design discussion trends exists but on an average design discussions decrease over time.

4.2 Association between Design Discussions and Design Quality (RQ 2)

Projects that discuss their design are likely to have fewer design quality issues. However, only discussing about it will not have any positive impact on the design quality of the project. Hence investigating the relationship between design discussions and design quality can help to inform the role of design discussions in improving (or maintaining) design quality.

We start by looking into the general trend of average code smell count across all projects. Figure 5 shows the *increasing* trend of the average number of code smells across all projects. This is similar to the findings of Ahmed et. al [13].

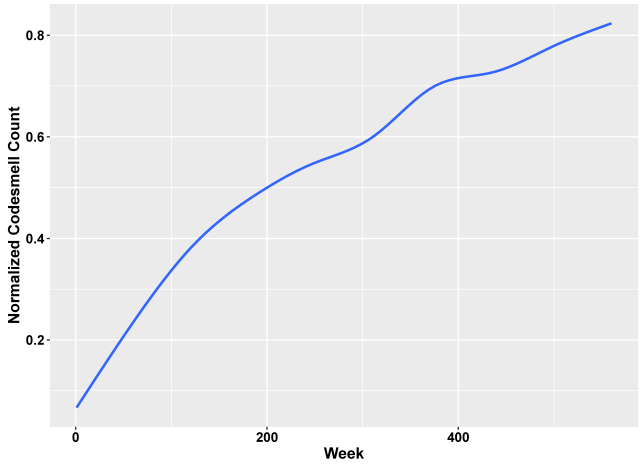

Figure 5: Average code smell count per week across all projects.

Since design discussions per week and code smell count per week both are time series data, a time series analysis is required to identify the correlation between these two, which is called cross-correlation. We do time series analysis individually for each project. Time series analysis requires a preprocessing step before doing the actual correlation analysis [50]. Due to space limitations, we report the results after each preprocessing step in the companion website[77].

Next, we calculate the cross-correlation between these two time series (design discussions per week and code smell count per week). Figure 6 shows the result of this step for qpid project as an example. From the figure we see that the highest correlation value is 0.16 (shown by the circled vertical line) at lag of 20. Next, we calculate the significance of correlation for each project. A correlation is significant when the absolute value is greater than, $\frac{2}{\sqrt{n-|k|}}$, where n is the number of observations and k is the lag [4].

We found that 20 out of the 37 projects (54.05%) have statistically significant cross-correlation between design discussions and code smells count. However, the cross-correlation values are small (Maximum cross-correlation is 0.34 and the minimum one is 0.003). We also looked into the lags between two time series and found that the length of the lag varies from 1 to 22 weeks between two time series. But in most projects the lag between two time series is 13 weeks. Due to space limitations, we report the plots for each of the 37 projects in the companion website [77].

Observation 7: Design discussions and design degradation are weakly correlated.

As design issues are always increasing and design related discussions are not helping project's design quality, we wanted to check if any specific group of developers are responsible for adding design

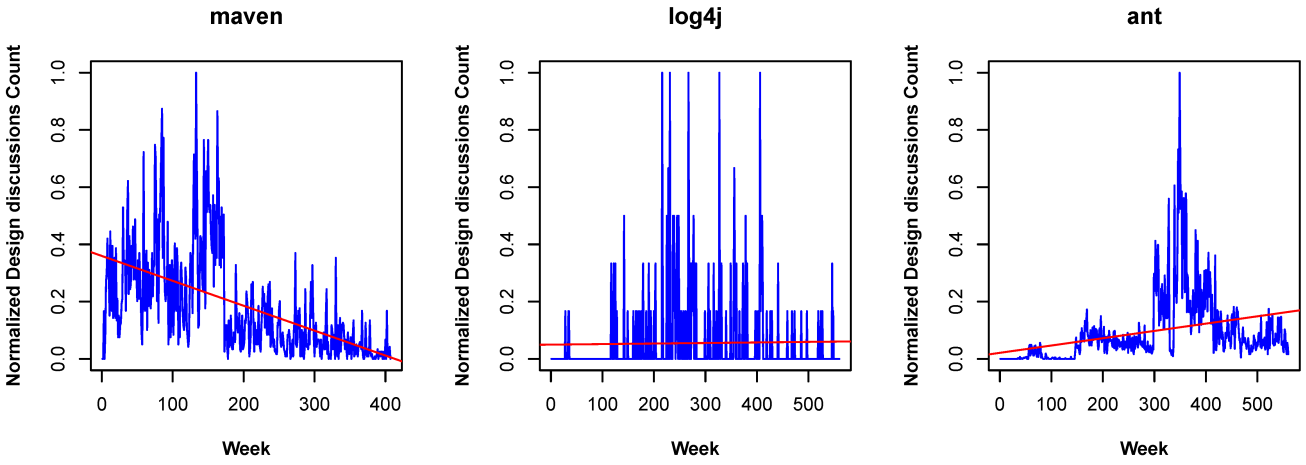

Figure 4: Week wise project's design discussion trend.

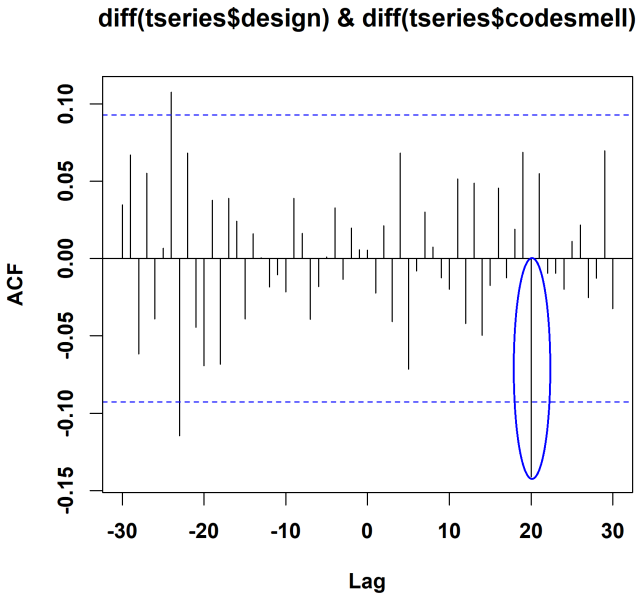

Figure 6: Cross correlation values of two time series

issues in the projects. First we collect the total number of commits for each developer that has added or removed at least one code smell. We normalized the number of commits (that add or remove code smell) for each developers using the following equation.

$$\text{Rescaled value} = \frac{\text{Total number of smell related commits}}{\text{Total number of commits}} \quad (3)$$

We found that, 66.35% of the commits by core developers and 33.65% commits by non-core developers added at least one code smell in the code base. However we can see from figure 7 that core developers remove more code smells than they add, whereas the opposite is true for non-core developers.

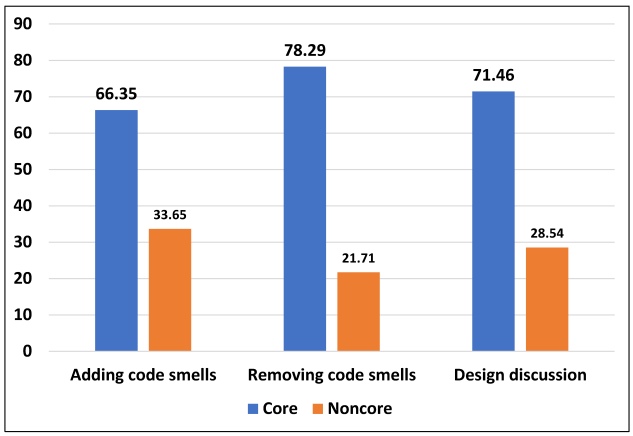

Figure 7: Adding, removing code smells versus design discussion participation.

Observation 8: Core developers are adding and removing more design issues in code base than non-core developers.

We also investigate if there is any difference between adding and removing code smells for each group. We found that there is no statistically significant difference in terms of number of code smell adding and removing for the core developers (Welch Two Sample t-test, $p > 0.05$). We found similar result for non-core group (Welch Two Sample t-test, $p > 0.05$).

5 DISCUSSION

In this section, we discuss the results presented in the previous section and present practical implications of our study for researchers, and tool builders.

The development culture in Apache projects is not very different than that of other OSS projects. If anything, it's more systematic. For example, while merging a change, Apache developers have to provide a reference to the issue related to the submitted change as a part of the commit message. This helps to preserve an activity trail between the issue and the commit [31]. This is not a common practice in most of the OSS projects. Since such well managed projects struggle to find design related discussions in their communication channels, it is high likely that other OSS projects struggle even more.

Through mining we found that despite the presence of multiple channels the vast majority of design discussions occur over email (Table 4). However, our survey results show that both project mailing lists and issue tracking systems are the preferred channel for developers to discuss design related issues (Figure 1). One reason behind this could be that mailing lists allow developers to have lengthier discussions. Developers may therefore initiate discussions in the mailing list. Only after deciding on a design issues, they move to issue tracking system to work on those changes.

Using mailing lists for design discussions has drawbacks. Finding a specific design discussion from email archive can be extremely difficult as one of the survey respondents explained "*existing documentation is about usage and rarely about design, design decisions are lost in mail. It's very hard and time consuming to find a email thread*".

The other problem related to using mailing lists for design discussion comes from the fact that emails eventually need to translate into an issue in the *Issue Tracking System*. There may not be a one-to-one mapping between an email discussion and a corresponding issue. This can lead to fragmented discussions, lost information etc. Making it difficult to retrieve design discussions and decisions from the archive. In our survey, 56.92% of respondents confirmed that retrieving design information from email archives is a difficult and time consuming task. Our study takes the first step towards showing that it is possible to automatically identify (and retrieve) design related discussions from multiple discussion archives (all three communication channels). Further research to develop efficient automated techniques to locate design discussions would be beneficial for end users.

A manual inspection of discussion contents across the three communication channels showed that a majority of design related discussions started on the mailing list and then shifted to the issue tracking system. Therefore, in order to understand the rationale of *why* a design change is made, developers need to be able to link discussion threads across multiple channels. Currently there are no tools to support such discussion tracking. For example, tools could allow context sensitive search on archives and help identify design discussions related to a specific piece of code. Similarly, another useful feature could be the ability to tag design related discussions and link them to the pull requests or to the part of code where the relevant implementation exists.

Identifying design discussions could also help in automated review assignment. Current recommender systems for task assignment or code reviews use developers' expertise [72, 78] to identify the best match. Identifying individuals who have been involved in design discussions and therefore have a bigger picture of the intention of changes can prove beneficial for such recommender systems. Further research can help in the design of approaches for automatically selecting contributors involved in a specific design discussion and assigning them higher priority.

Apart from the issue of design discussions being fragmented across multiple channels, some of the discussions also occur through personal emails, verbal meetings, etc. These unofficial discussions are not recorded in the project archives and are not accessible to developers who were not part of these discussions. We posit that this may be one of the reasons why we don't see a strong association between design quality and design discussions in our analysis. Discussion in unofficial channels can also make it difficult for developers to gauge the impact of their contribution on design. Further research is required to answer many of the questions raised by these observations: what types of discussions occur over the unofficial channels, who participate in these discussions, and what is the impact of such discussions on tracking design decisions and the overall quality of the project.

We also investigated who are mostly engaged in design discussions and design implementation. From both mining and survey we found that though both core and non-core developers are participating in design discussions and core developers are more involved implementing design discussions than non-core developers. Some reasons (as alluded to by the survey respondents) about why this might be occurring are: it is hard to find discussions across multiple channels, lack of knowledge of where to look for information, project culture where non-core developers do not work on design related changes, and design related changes when raised by non-core members are overlooked by the core group. Some of these can be fixed by better tooling, others need changes to the project culture. Researchers need to investigate these reasons and devise monitoring mechanisms to recommend necessary measures to project leaders when needed.

Finally, 79.23% of our survey respondents said that they do not continuously monitor the design quality of a project and even worse they do not check the impact of their contribution on design quality. One reason for this might be the lack of tool support. Current code smell detection tools do not support just-in-time analysis for the most recent changes. These tools analyze the whole code base, which make it difficult to incorporate them into the regular development workflow. Moreover, the tools mentioned by the respondents–SpotBugs, JaCoCo, Simian, Checkstyle, SonarQube etc.– provide information regarding design quality (number of code smells). However, to the best of our knowledge none of these tools provide insight or links to relevant design discussions and how the actual design deviates from those. They also do not provide refactoring options to developers. This might be one of the reasons why we see such accumulations of code smells. Developers do not act upon these warnings unless they have either (i) a high return on investment (e.g., eliminating the smell has an immediate value), or (ii) tools make it easy to eliminate the smells. While developers might not see the immediate benefit of eliminating code smells (as

smells usually have long-term effects on code maintenance), we encourage tool builders to close the gap between detection and correction of code smells.

6 THREATS TO VALIDITY

We have taken care to ensure that our results are unbiased, and have tried to eliminate the effects of random noise, however some biases are unavoidable and in some cases it is possible that our mitigation strategies may not have been effective.

The data set used for the study contained 1, 661, 922 discussions from 37 Apache projects. We analyzed discussions from the official communication channel archives. However, developers also use personal emails, chats, offline discussion, etc. for design related discussions which are not available on these archives and are not part of our analysis. As our goal was perform a case study of a single ecosystem where projects have similar development processes, we selected projects from the Apache Software Foundation and hence our findings may not generalize to all open source projects. Similarly, we surveyed developers only from he Apache Software Foundation. The characteristics of these developers may not be representative of all developers in other open source projects.

We categorized the developers into core and non-core groups using a threshold of number of commits in the code base for each developer. Some of the developers could have been categorized as non-core according to our criteria though they were actually core developers who focus on large contributions rather than frequent contributions, or simply focus on architecture and high-level design (high value contributions).

For code smell detection , we used inFusion [2], which is a static source code analysis tool. Code smells that are intrinsically historical, such as Parallel Inheritance, are difficult to detect by just using static source code analysis [58]. So, the number of occurrences of such "intrinsically historical" smells will be different when historical information based smell detection technique is used. Also, InFusion detects 22 code smells, which is not a exhaustive list of code smell. There could be more smells which InFusion cannot detect.

It is always possible that the participants misunderstand the survey questions. To mitigate this threat, we conducted a pilot study with experts in OSS and survey design. We updated the survey based on the findings of these pilot studies.

7 CONCLUSIONS

In this paper, we present the results of our investigation of design discussions, their evolution, and their association with the project's design quality. Our mixed method empirical study of 37 Apache projects and survey of 130 developers revealed that design discussions are fragmented across multiple communication channels. This fragmentation of design discussions likely makes it difficult for developers to keep track of the agreed upon design decisions. According to the respondents, though mailing list is the most difficult channel (56.92% of respondents) when it comes to retrieving design discussions, it is the most frequently used channel for design discussions in the Apache projects. Interestingly, survey respondents mentioned that other side channels such as personal emails (28.46% respondents) and verbal meeting (43.85% respondents) were also used to conduct design discussions. Further, the average number of design discussions decrease, but code smells increase, as the project evolves. These factors could be playing a role in the low association of design discussions with design quality.

Our work, showcases that further research is needed to: a) understand the state, evolution, and impact of design discussions that are fragmented across multiple communication channels–ours is just a start, b) analyze the disconnect between the design discussions and design quality in OSS projects, and c) build tool support to help developers find and link different design discussions.

8 ACKNOWLEDGMENTS

This work was supported in part by awards 2008089 and 1815486 from the National Science Foundation.

REFERENCES

[1] 2009. iPlasma. http://loose.upt.ro/iplasma/. Accessed: 2019-08-17.
[2] 2017. InFusion. http://www.intooitus.com/inFusion.html. Accessed: 2014-01-01.
[3] 2019. Decision Trees. https://scikit-learn.org/stable/modules/tree.html. Accessed: 2019-08-17.
[4] 2019. Interpret all statistics and graphs for Cross Correlation. https://support.minitab.com/en-us/minitab/18/help-and-how-to/modeling-statistics/time-series/how-to/cross-correlation/interpret-the-results/all-statistics-and-graphs/. Accessed: 2019-08-13.
[5] 2019. Logistic Regression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. Accessed: 2019-08-17.
[6] 2019. Multinomial Naive Bayes classifier. https://scikit-learn.org/stable/modules/naive_bayes.html. Accessed: 2019-08-17.
[7] 2019. Natural Language Toolkit. https://www.nltk.org/
[8] 2019. NLTK stop words. https://pythonspot.com/nltk-stop-words/. Accessed: 2019-08-17.
[9] 2019. qualtrics. https://www.qualtrics.com/. Accessed: 2019-08-17.
[10] 2019. Support Vector Machine. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. Accessed: 2019-08-17.
[11] 2019. TfidfVectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html. Accessed: 2020-08-17.
[12] Iftekhar Ahmed, Caius Brindescu, Umme Ayda Mannan, Carlos Jensen, and Anita Sarma. 2017. An Empirical Examination of the Relationship between Code Smells and Merge Conflicts. In *Empirical Software Engineering and Measurement (ESEM), 2017 ACM/IEEE International Symposium on*. IEEE, 58–67.
[13] 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 *Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on*. IEEE, 1–10.
[14] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. 2002. Recovering traceability links between code and documentation. *IEEE transactions on software engineering* 28, 10 (2002), 970–983.
[15] Adrian Bachmann and Abraham Bernstein. 2009. Software process data quality and characteristics: a historical view on open and closed source projects. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*. ACM, 119–128.
[16] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.
[17] Abraham Bernstein, Jayalath Ekanayake, and Martin Pinzger. 2007. Improving defect prediction using temporal features and non linear models. In *Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting*. ACM, 11–18.
[18] Grady Booch. 2006. *Object oriented analysis & design with application*. Pearson Education India.
[19] João Brunet, Gail C Murphy, Ricardo Terra, Jorge Figueiredo, and Dalton Serey. 2014. Do developers discuss design?. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 340–343.
[20] Eugenio Capra, Chiara Francalanci, and Francesco Merlo. 2008. An empirical study on the relationship between software design quality, development effort and governance in open source projects. *IEEE Transactions on Software Engineering* 34, 6 (2008), 765–782.
[21] Mauro Cherubini, Gina Venolia, Rob DeLine, and Andrew J. Ko. 2007. Let's Go to the Whiteboard: How and Why Software Developers Use Drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '07)*. ACM, 557–566.

[22] Elliot J. Chikofsky and James H Cross. 1990. Reverse engineering and design recovery: A taxonomy. *IEEE software* 7, 1 (1990), 13–17.

[23] Jacob Cohen, Patricia Cohen, Stephen G West, and Leona S Aiken. 2013. *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge.

[24] Ignatios Deligiannis, Martin Shepperd, Manos Roumeliotis, and Ioannis Stamelos. 2003. An empirical investigation of an object-oriented design heuristic for maintainability. *Journal of Systems and Software* 65, 2 (2003), 127–139.

[25] 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 *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 148–153.

[26] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.

[27] Vincenzo Ferme, Alessandro Marino, and F Arcelli Fontana. 2013. Is it a Real Code Smell to be Removed or not?. In *International Workshop on Refactoring & Testing (RefTest), co-located event with XP 2013 Conference*.

[28] Francesca Arcelli Fontana, Elia Mariani, Andrea Mornioli, Raul Sormani, and Alberto Tonello. 2011. An experience report on using code smells detection tools. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*. IEEE, 450–457.

[29] Francesca Arcelli Fontana and Marco Zanoni. 2011. On investigating code smells correlations. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*. IEEE, 474–475.

[30] Apache Software Foundation. 2019. Apache Developers' Contributors' Overview. http://apache.org/dev/. Accessed: 2019-08-17.

[31] Apache Software Foundation. 2019. How Should I Apply Patches From A Contributor. http://www.apache.org/dev/committers.html#applying-patches. Accessed: 2019-08-17.

[32] Martin Fowler and Kent Beck. 1999. *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

[33] Peter Freeman and David Hart. 2004. A science of design for software-intensive systems. *Commun. ACM* 47, 8 (2004), 19–21.

[34] Tracy Hall, Min Zhang, David Bowes, and Yi Sun. 2014. Some code smells have a significant but small effect on faults. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23, 4 (2014), 33.

[35] Mario Hozano, Henrique Ferreira, Italo Silva, Baldoino Fonseca, and Evandro Costa. 2015. Using developers' feedback to improve code smell detection. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 1661–1663.

[36] C. Izurieta and J. M. Bieman. 2007. How Software Designs Decay: A Pilot Study of Pattern Evolution. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. 449–451.

[37] C. Izurieta and J. M. Bieman. 2008. Testing Consequences of Grime Buildup in Object Oriented Design Patterns. In *2008 1st International Conference on Software Testing, Verification, and Validation*. 171–179.

[38] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An introduction to statistical learning*. Vol. 112. Springer.

[39] Mark Kasunic. 2005. *Designing an effective survey*. Technical Report. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.

[40] Michael Keeling and Runde Joe. 2017. Architecture Decision Records in Action.

[41] 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. *Empirical Software Engineering* 17, 3 (2012), 243–275.

[42] Max Kuhn and Kjell Johnson. 2013. *Applied predictive modeling*. Vol. 26. Springer.

[43] J Richard Landis and Gary G Koch. 1977. An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers. *Biometrics* (1977), 363–374.

[44] Wei Li and Raed Shatnawi. 2007. An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. *Journal of systems and software* 80, 7 (2007), 1120–1128.

[45] Alvi Mahadi, Karan Tongay, and Neil A Ernst. 2019. Cross-Dataset Design Discussion Mining. (2019).

[46] Umme Ayda Mannan, Iftekhar Ahmed, Rana Abdullah M Almurshed, Danny Dig, and Carlos Jensen. 2016. Understanding code smells in android applications. In *Proceedings of the International Workshop on Mobile Software Engineering and Systems*. ACM, 225–234.

[47] Radu Marinescu. 2004. Detection strategies: Metrics-based rules for detecting design flaws. In *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*. IEEE, 350–359.

[48] S. McIntosh. 2011. Build system maintenance. In *2011 33rd International Conference on Software Engineering (ICSE)*. 1167–1169.

[49] Flávio Medeiros, Márcio Ribeiro, Rohit Gheyi, Sven Apel, Christian Kästner, Bruno Ferreira, Luiz Carvalho, and Baldoino Fonseca. 2017. Discipline matters: Refactoring of preprocessor directives in the# ifdef hell. *IEEE Transactions on Software Engineering* 44, 5 (2017), 453–469.

[50] Andrew V Metcalfe and Paul SP Cowpertwait. 2009. *Introductory time series with R*. Springer.

[51] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.* 11, 3 (July 2002), 309–346. https://doi.org/10.1145/567793.567795

[52] 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 *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 171–180.

[53] Hausi A Müller and Karl Klashinsky. 1988. Rigi-A system for programming-in-the-large. In *Proceedings of the 10th international conference on Software engineering*. IEEE Computer Society Press, 80–86.

[54] John Noll, Sarah Beecham, and Dominik Seichter. 2011. A qualitative study of open source software development: The open EMR project. In *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, 30–39.

[55] Steffen Olbrich, Daniela S Cruzes, Victor Basili, and Nico Zazworka. 2009. The evolution and impact of code smells: A case study of two open source systems. In *Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement*. IEEE Computer Society, 390–400.

[56] Gustavo Ansaldi Oliva, Igor Steinmacher, Igor Wiese, and Marco Aurélio Gerosa. 2013. What can commit metadata tell us about design degradation?. In *Proceedings of the 2013 International Workshop on Principles of Software Evolution*. ACM, 18–27.

[57] Thanis Paiva, Amanda Damasceno, Eduardo Figueiredo, and Cláudio Sant'Anna. 2017. On the evaluation of code smells and detection tools. *Journal of Software Engineering Research and Development* 5, 1 (2017), 7.

[58] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. 2013. Detecting bad smells in source code using change history information. In *Automated software engineering (ASE), 2013 IEEE/ACM 28th international conference on*. IEEE, 268–278.

[59] Luca Pascarella, Davide Spadini, Fabio Palomba, and Alberto Bacchelli. 2020. On The Effect Of Code Review On Code Smells. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*.

[60] Leonardo Passos, Rodrigo Queiroz, Mukelabai Mukelabai, Thorsten Berger, Sven Apel, Krzysztof Czarnecki, and Jesus Padilla. 2018. A study of feature scattering in the linux kernel. *IEEE Transactions on Software Engineering* (2018).

[61] Gopi Krishnan Rajbahadur, Shaowei Wang, Yasutaka Kamei, and Ahmed E Hassan. 2017. The impact of using regression models to build defect classifiers. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 135–145.

[62] Paul Ralph and Yair Wand. 2009. A proposal for a formal definition of the design concept. In *Design requirements engineering: A ten-year perspective*. Springer, 103–136.

[63] Daniele Romano and Martin Pinzger. 2011. Using source code metrics to predict change-prone java interfaces. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 303–312.

[64] Bilyaminu Auwal Romo, Andrea Capiluppi, and Tracy Hall. 2014. Filling the gaps of development logs and bug issue data. (2014).

[65] scikit learn. 2019. . Accessed: 2019-08-1.

[66] Arman Shahbazian, Youn Kyu Lee, Duc Le, Yuriy Brun, and Nenad Medvidovic. 2018. Recovering architectural design decisions. In *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 95–9509.

[67] T. J. Smiley. 1958. The Uses of Argument. By S. E. Toulmin, Professor of Philosophy, University of Leeds. [Cambridge: at the University Press. 1958. vii, 261 and (index) 2 pp. 22s. 6d. net.]. *The Cambridge Law Journal* 16, 2 (1958), 251–252. https://doi.org/10.1017/S0008197300003937

[68] Adriana Meza Soria and André van der Hoek. 2019. Collecting design knowledge through voice notes. In *Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering*. IEEE Press, 33–36.

[69] 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 *Proceedings of the 40th International Conference on Software Engineering*. 921–931.

[70] Leonardo Sousa, Roberto Oliveira, Alessandro Garcia, Jaejoon Lee, Tayana Conte, Willian Oizumi, Rafael de Mello, Adriana Lopes, Natasha Valentim, Edson Oliveira, et al. 2017. How Do Software Developers Identify Design Problems? A Qualitative Analysis. In *Proceedings of the 31st Brazilian Symposium on Software Engineering*. 54–63.

[71] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering* 43, 1 (2016), 1–18.

[72] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken-ichi Matsumoto. 2015. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 141–150.

[73] TIOBE. [n.d.]. TIOBE Index. https://www.tiobe.com/tiobe-index/.

[74] Jilles Van Gurp and Jan Bosch. 2002. Design erosion: problems and causes. *Journal of systems and software* 61, 2 (2002), 105–119.
[75] Giovanni Viviani, Michalis Famelis, Xin Xia, Calahan Janik-Jones, and Gail C Murphy. 2019. Locating Latent Design Information in Developer Discussions: A Study on Pull Requests. *IEEE Transactions on Software Engineering* (2019).
[76] Giovanni Viviani, Calahan Janik-Jones, Michalis Famelis, Xin Xia, and Gail C Murphy. 2018. What design topics do developers discuss?. In *Proceedings of the 26th Conference on Program Comprehension*. ACM, 328–331.
[77] Companion website. [n.d.]. Companion website. https://fse2020.wixsite.com/designdiscussion. Accessed: 2018-05-20.
[78] Motahareh Bahrami Zanjani, Huzefa Kagdi, and Christian Bird. 2015. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering* 42, 6 (2015), 530–543.
[79] Nico Zazworka, Michele A Shaw, Forrest Shull, and Carolyn Seaman. 2011. Investigating the impact of design debt on software quality. In *Proceedings of the 2nd Workshop on Managing Technical Debt*. ACM, 17–23.