This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2924006, IEEE Transactions on Software Engineering

Locating Latent Design Information in Developer Discussions: A Study on Pull Requests

Giovanni Viviani, Michalis Famelis, *Member, IEEE,* Xin Xia, *Member, IEEE,*
Calahan Janik-Jones, Gail C. Murphy, *Member, IEEE Computer Society,*

Abstract—A software system's design determines many of its properties, such as maintainability and performance. An understanding of design is needed to maintain system properties as changes to the system occur. Unfortunately, many systems do not have up-to-date design documentation and approaches that have been developed to recover design often focus on how a system works by extracting structural and behaviour information rather than information about the desired design properties, such as robustness or performance. In this paper, we explore whether it is possible to automatically locate where design is discussed in on-line developer discussions. We investigate and introduce a classifier that can locate paragraphs in pull request discussions that pertain to design with an average AUC score of 0.87. We show that this classifier, when applied to projects on which it was not trained, agrees with the identification of design points by humans with an average AUC score of 0.79. We describe how this classifier could be used as the basis of tools to improve such tasks as reviewing code and implementing new features.

Index Terms—Design Discussions, Latent Design, Conversations, Prediction Model, Design Recovery

1 Introduction

The design of a software system dictates many of its properties, such as performance, extensibility, robustness, and maintainability. To uphold desired properties in a system, developers must be aware of, and make choices consistent, with its design. When developers are not aware of a system's design, choices they make can cause desired system properties to erode [58].

Unfortunately, developers often do not have access to information about a system's current design. Despite many notations and approaches for expressing design, it is often not captured explicitly [44]. Even when recorded, a project's design is often not kept current with the evolving system [36]. When design documentation is not explicit or not current, newcomers to a project find it difficult to contribute to the system [48], [50]. In open source projects, delays in making contributions and failed contributions can slow growth and impact the health of the project [10].

One approach to providing design information is to recover design automatically from project artifacts [4]. Most of the existing recovery approaches focus either on inferring the system's structure (e.g., [35]) or its behaviour (e.g., [8]). These approaches generally aim to describe large parts of a system's design and typically focus on describing *how* a system currently works. Such information can help a software developer change a system, but it does not provide the developer information about other aspects of the design, such as the intent guiding *why* certain choices were made.

Such information is available in other project artifacts. Recently, Brunet et al. and Tsay et al. have identified that developer discussions, captured in project artifacts, such as issue reports, include discussions of design [6], [56]. Tsay et al. have further showed that these discussions can be a major factor in deciding how a system evolves, suggesting that the discussions include information that goes beyond *how* a system works. Our recent work also indicates that the design discussed in developer discussions considers *why* certain choices were made, such as to address robustness [60].

Imagine if it was possible to locate and interpret the design information in developer discussions automatically. Consider, for instance, pull request #12081 submitted to the Rust[1] project. This pull request was submitted on February 6, 2014 and, after a discussion involving 11 individuals posting 303 comments, was merged on March 12, 2014. If a developer spends only 2 minutes to read each comment in this pull request to access design information, reading the pull request would consume 10 hours of a developer's time. Given the pressures on developers, it is unlikely that amount of time would be spent, causing any design information contained in such discussions to go unused and developers to work without being aware of it. Instead of a developer spending the hours to read discussions to understand the intent of the design, what if it was possible to learn the design properties of interest for a system from the discussions? When a new pull request is made to the

- *Giovanni Viviani and Gail C. Murphy are with the University of British Columbia, Vancouver, Canada. Email: vivianig@cs.ubc.ca, murphy@cs.ubc.ca*
- *Michalis Famelis is with University of Montreal, Montreal, Canada. E-mail: famelis@iro.umontreal.ca*
- *Xin Xia is with Monash University, Melbourne, Australia. E-mail: xin.xia@monash.edu*
- *Calahan Janik-Jones is with University of Toronto, Toronto Canada. E-mail: cal.janik.jones@mail.utoronto.ca*

[1] https://github.com/rust-lang/rust/pull/12081

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2924006, IEEE Transactions on Software Engineering

system, a tool could draw on this information to generate a comment automatically of the form:

> "Thank you for your contribution to Rust. To help facilitate the timely review of your pull request, please consider if your changes affect any of the following project specific design concerns: *maintainability* and *robustness*".

This prompt would provide the contributor an opportunity to alter their contribution to reflect the comments or to add a comment to the pull request discussion specifically addressing the concerns. Either of these actions may help decrease the time to process a pull request.

To make this scenario a reality, we need to be able to locate latent design information in developer discussions. But, what is *design*? In the literature, design manifests in many ways, including as large-scale architecture [22], as low-level design patterns [13], as compliance to international standards [1] and as the choice of specific algorithms [14], to name just a few.

In this paper, we do not presuppose any single notion of design, focusing instead on separating design information from other kinds of information present in a discussion. For this separation, we introduce the concept of a *design point*, which we consider as a piece of a discussion relating to a decision about a software system's design that a software development team needs to make [60]. Design points thus include a wide variety of design as outlined above, encompassing structural, functional and non-functional aspects of the system.

We investigate the idea of design points on a dataset in which 10,790 paragraphs appearing in pull request discussions from multiple systems were manually labeled as having, or not having, a design point. We focus on pull request discussions as a target as they represent a kind of discussion where it has been shown that developers consider design [56]. To support the identification of design points automatically, the annotation of each design point includes a set of labels describing attributes and context of the design point (Section 3).

Using this dataset, we built a classifier that can locate design points, at the granularity of paragraphs, in pull requests across multiple systems (Section 4). This classifier can locate design points with an AUC score of 0.87, meaning that our classifier will always rank paragraphs with design points higher than these without design points. Romano et al. concluded that a prediction model with an AUC score above 0.7 is often considered to have adequate classification performance [46].

Given that design is inherently a slippery concept, a classifier for locating design points can only be the basis for tools if it recognizes places in a discussion that human developers agree discuss design. To investigate this question, we compared the classifier to a gold standard formed based on the agreement of at least three human developers on the presence or absence of design point in five systems on which the classifier had not been trained. We found that the classifier agreed with the human developers with an AUC of 0.79 (Section 5).

In this paper, we make the following contributions:

1) We provide a data set about design points in 10,790 paragraphs from 34 pull requests from three different open source projects.
2) We present a classifier that can locate when a design point occurs in a paragraph of a pull request discussion, constituting a fundamental building block for tools that make use of latent design information in design discussions.
3) We report on the classification features that are most important for automated design point location at the paragraph level in pull requests.
4) We show that the classifier finds similar design points as identified by human developers.

Data detailing the annotation of the 10,790 paragraphs and all the remaining data used in Sections 3 is available at https://www.cs.ubc.ca/~vivianig/annotation.zip.

2 Related Work

In the context of software engineering, design is traditionally understood both as a process [12] in which a development team engages and as the resulting specifications [40] that the team produces. Researchers have investigated how, over the natural evolution of a software system, small changes accumulate, causing the actual design of the system to differ from what was originally documented, a process known as design erosion [58]. At the same time, the overall knowledge of a system's design by software developers is often subject to "evaporation," which causes the developers' to gradually lose knowledge of the design over time [43].

To help combat design erosion and evaporation, techniques for design recovery have long been discussed in the literature [4]. Often, design recovery has focused on reverse engineering [7] and has been centered around inferring structural (e.g., [35]) or behavioural (e.g., [8]) designs from largely code-related or execution-related software artifacts, such as code and logs. Design can also be recovered outside the code itself; for instance, by establishing traceability links between code and documentation [3]. More recently, there have been attempts to recover architectural design by combining source code with commits and issues [47].

Our approach complements these earlier approaches by looking beyond technical software artifacts and attempting to recover design information from the social context of software development. An important component of this social context is developer discussions, in which design is a frequently discussed topic [6]. Concerns related to design are often raised during pull request discussions [56], and are in fact crucial factors for deciding whether to accept pull requests [16].

The idea that useful design information may occur in discussions is also supported by the literature on design rationale. Toulim introduced a model to describe argumentation [55] that later led to the description of design rationale as "the explicit listing of decisions made during a design process and the reasons why those decisions were made" [20, p. 577]. Initial research on design and its rationale focused on how to create a useful representation [30], generally involving either an observer being present at the moment the design was discussed or an act of reconstruction executed after the design phase had been completed [25].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2924006, IEEE Transactions on Software Engineering

Listing 1: Example of a paragraph containing a design point from pull request #12422, `Node.js` project

```
This PR means that we're unable to safely rely on the existing error handling semantics, namely not swallowing errors by defaultor affecting post-mortem debugging. Again, I can't stress howcritical the existing error handling semantics are regarding operating our Node stack in the critical Netflix streaming path at scale. It's imperative to us that the Node runtime continues to work with the existing error handling best practices. After all, we're relying on Node in some of our most critical systems here at Netflix, where reliability and debuggability are our top priority.
```

Listing 2: Example of a paragraph where a design point is not present from pull request #12422, `Node.js` project

```
@yunong I appreciate you speaking up on this matter, but I've replied to Julien's comment above as well at yours at nodejs/CTC#12 (comment). As noted above, that's a better place for more general concerns about improved Promise support in Node core.
```

Our work complements these earlier approaches by considering an automated approach applied to on-line developer discussions.

Developer discussions about design have been studied in two contexts to date. One context is written discussions as occur in issue reports. Brunet and a colleague manually coded 1,000 discussions and trained a classifier to identify which discussions included design. The identification of whether design was discussed was at the level of the entire discussion. As we have reported, design can be isolated to smaller parts of a discussion, with roughly 22% of the paragraphs appearing in a discussion containing design information [60]. In this paper, we focus on this context and consider how to automate the detection of those paragraphs dealing with design in written developer discussions as they appear in pull requests. The second context of design is in-person conversations and whiteboard design sessions. Within this context, researchers aim to study software design in its "natural setting" and aim to describe the activities involved in the process of design [38]. This paper does not consider this context, leaving investigations to future work.

Discussions involving developers have also been studied to extract other kinds of information related to software development. Knauss et al. created a catalogue of communication patterns that are characteristic to clarification of requirements in online discussions and developed automation to detect them [21]. Rodeghero et al. studied transcripts of discussions between developers and their clients to automatically generate requirements in the form of user stories [45]. We use some of their ideas in the development of our own automated classifier, described in Section 4.

3 A Design Points Dataset

Our goal is to locate design points automatically. To study the presence of design points, we create a dataset consisting of 10,790 paragraphs from 34 pull requests from the `Node.js` (14), `Rails` (10) and `Rust` (10) projects, in which each paragraph was manually annotated as to whether it includes a design point. Each paragraph was labelled by one of three annotators, all of whom are authors of this paper.

Listing 3: Discussion snippet from Rails pull request #505. This paragraphs contains a design point and the enhanced labels are shown

```
;##D54 We should jsut add an option per patch and deal with the default later
;##ROLE PM
;##INV T
;##FORM SOL
;##REL ELAB D43

As I mentioned earlier, we should simply add an option for users who want to use PATCH to be able to use it, and move on with our lives. We can address the defaults in some far distant release once we actually have real experience with the benefits.
```

Before starting the annotation, we developed a codebook to guide the annotation process. This codebook was developed in three steps. In an initial phase, two authors annotated independently the same three pull request discussions. Each one created a codebook to document their annotations. This was followed by a consolidation phase, where the same authors merged the two codebooks. During this phase, we opted to delimit the search of design points to paragraphs. On the one hand, working with entire comments is too coarse, since different paragraphs within a comment sometimes discuss different topics. On the other hand, working with single sentences is too fine, as a design point can easily span over multiple sentences. As part of the iterative development of the codebook, the annotators identified guidelines to recognize which paragraphs are design points and which are discussing other issues. Examples of these guidelines are the use of speculative language, or the presence of a rationale to justify the statements being made.[2]

Following this phase, another author joined the annotation and each annotated four pull requests, and computed the Fleiss's Kappa Coefficient over the resulting annotation. This results in a value of 0.52, commonly understood as "moderate agreement" level [23], [24]. Finally, we proceeded to annotated the 34 pull requests previously mentioned. None of the pull requests used in the first phase to create the codebooks were included in the final set.

To provide a sense of this dataset, Listing 1 provides an example of a paragraph with a design point from pull request #12422 from the `Node.js` project. Listing 2 provides an example of a paragraph without a design point from the same pull request.

The mere presence or absence of a design point does not provide much context about the design point. As a result, during the annotation process, we also labeled three kinds of additional information about each paragraph: the developers' level of expertise, the form of the language used to express a design point, and the logical relationships between design points. We elaborate on each of these kinds of information below.

First, given that in a software development project, developers typically have areas of the code in which they have more expertise than others, we wanted to capture informa-

[2]The codebook can be found with the dataset at https://www.cs.ubc.ca/~vivianig/annotation.zip

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2924006, IEEE Transactions on Software Engineering

tion about the role of the paragraph author in the project. We annotated three values: whether the paragraph author was the owner of the pull request, whether the author was a core project member, or playing another role. We also labeled whether the author was invited to the discussion, which was only possible if the author was not the pull request owner. Listing 3 shows a paragraph from `Rails` pull request #505 annotated with this information: each annotation starts with `;##`. The line marked `;##D54` marks the paragraph below has a design point, identified with id 54, with a short summary of the design point. The next two lines are information about the author of the comment: the line marked `;##ROLE PM` indicates the paragraph was penned by a project member whereas the line marked `;## INV T` indicates that the project member was invited to the discussion. (Table 1 summarizes the labels annotated.[3])

To give a sense of the final dataset, Table 2 reports on the distribution of labels across the paragraphs. Interestingly, the roles associated with design points are roughly split between the three categories: `Owner` (27%), representing the users that opened the pull request, `Member` (38%) representing users that are identified as core members of the project, and `Other` (35%), representing users without any affiliation to the project. Only a minority of paragraphs containing design points were generated by users invited (`INV`) in the discussion (37.3%). This seems to indicate that the actors participating in design discussion are both core members of the projects and generic contributors, not officially recognized as part of the team, who join the discussions on their own accord.

Second, we considered the ways in which developers use language in the discussions. Based on our reading of developer discussions, we chose to label whether a design point takes the form of: 1) an assertion of a solution, 2) an open question, or 3) a closed question, which generally takes the form of an enumeration. In Listing 3, the design point takes the form of a solution (i.e., `;##FORM SOL`). Table 2 shows that the majority (76.6%) of design points appear as `Solutions`, followed by `Questions` (21.4%). We found very few (2%) cases of `Enumerations` (2%), indicating that there was little evidence of developers brainstorming multiple solutions when posting a comment.

Finally, we were interested in whether there is logical structure in a discussion [59]. We used the `REL` label to capture the logical structure of the discussion by making explicit the relationships between design points. Because design points in an asynchronous discussion should only be related to design points previously added, we annotated without looking ahead in the discussion. Some design points may have no relationships to previously seen design points, and are labeled "new". Design points may "elaborate" on a previous point and respond to concerns or expand on premises, or "generalize" previous design points and come to a conclusion. Finally a design point can "reframe" another design point if it highlights different aspects of it, without generalizing or elaborating it. In Listing 3, the design point annotated elaborates a previous design point paragraph that is labeled as `D43`.

As Table 2 shows, we found that large numbers (43%) of design points are either unrelated to others or `Elaborate` on previous design points (36.2%); one interpretation of these values is that a large number of design points have at least one followup. `Reframes` are less common (18.1%) and `Generalize` are rare (3.9%).

[3]The codebook can be found with the dataset at https://www.cs.ubc.ca/~vivianig/annotation.zip

4 Locating Design Points Automatically

To be able to access and leverage design points in written developer discussions, we need to be able to automatically locate them. We consider two research questions:

RQ1 Can we effectively locate design points automatically in discussions on pull requests?

RQ2 Which features are most important to locate design points?

Building on the dataset described in the previous section, we describe a machine learning approach for locating which paragraphs in a pull request discussion contain a design point. We begin by describing the features—measurable properties that are potential indicators of the presence of a design point in a paragraph (Section 4.1). We then describe the approach we took to build and evaluate various classifiers (Section 4.2) before presenting the results (Section 4.3). In evaluating classifiers, we focus on the first research question (**RQ1**) which asks whether design points can be *effectively* located automatically. For us, *effective* means whether the design points can be located accurately and whether the classifier can apply across different software projects. A classifier that locates design accurately, but requires training

TABLE 1: Labels applied to paragraphs from pull requests

Code	Description	Values
`D[ID][Summary]`	Identify a design point using a unique number and a descriptive summary.	
`ROLE[Role]`	Define the role of the author of the paragraph containing the design point.	Pull request Owner, Core Project Member, Other
`INV[Invited]`	Indicate if the author of the design point been invited to the discussion (i.e., was tagged in the discussion before her first comment).	True or False
`FORM[Form]`	Define the form the design point takes inside the paragraph.	Solution, Open Question or Enumeration
`REL[Relationship]`	Indicate whether the design point is related to a previously posted design point.	Elaborates, Generalizes or Reframes another design point, or is New

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2924006, IEEE Transactions on Software Engineering

TABLE 2: Annotation Distribution

Annotation		Distribution	
Dimension	**Label**	**Count**	**Percentage**
ROLE	Owner	669	27%
	Member	949	38%
	Other	857	35%
INV	True	924	37%
	False	1551	63%
FORM	Solution	1897	77%
	Question	530	21%
	Enumeration	48	2%
REL	New	1065	43%
	Elaborate	889	36%
	Generalize	74	3%
	Reframe	447	18%

for every new project is not effective, as it would not be applicable to real-world scenarios. To ensure effectiveness, we evaluate our classifier on projects that were not used for training it.

4.1 Features

When we annotated paragraphs manually, we found a variety of information of interest to characterize a design point, such as the role of the author of the comment containing a paragraph. We use this analysis, as well as knowledge of the literature of similar techniques to define 19 features on which to investigate a machine learning approach. Table 3 summarizes the 19 features that we investigate, grouped in 4 dimensions: process, position, text, and content.

4.1.1 Process Dimension:

Similar to discussions in other contexts, such as emails [49], pull request discussions involve multiple individuals in different roles. For example, a participant may be heavily involved in the project and considered a core developer [64] or the individual may have been asked to join the discussion because someone already involved in the discussion believes their expertise is needed. We observed how these process aspects varied per paragraph of a discussion through the annotations we performed (Section 3). We investigate three process features based on our earlier analysis: whether the paragraph author is a project member (*isProMem*), whether the paragraph author was invited to the discussion (*IsInvited*) and whether the paragraph author is the pull request originator (*IsOriginal*). The fourth feature we investigate, *NumCom*, relates to the level of activity of the author of the paragraph in the discussion; this feature is similar to one used by Rodeghero et al. when building a classifier for turning conversations into user stories [45].

4.1.2 Position Dimension:

Machine learning-based classification has been applied to discussions to perform other tasks, such as extractive summarization (e.g., [37]). As well, the features used for general email discussions have also been found to be useful for text produced as part of the software engineering process (e.g., [41]). The features we define in the position dimension are inspired by the positional features used by such classifiers. *PosInCom* measures the position of the paragraph in a comment. *PosInPR* measures the position of the comment containing the paragraph in the entire pull request discussion.

4.1.3 Text Dimension:

The features of this dimension measure textual characteristics of a paragraph as indications for the presence of design point. Building on features defined by Correa and Sureka, we consider the size of the paragraph (*NumOfWords*) and the presence of capitalized words (*HasCapWord*) [9]. Based on heuristics found to be useful by Li et al., we also consider the presence of words indicative of speculative language (e.g., *HasShould*, *HasMay*, etc.) [27]. Finally, we also measure the density of interrogative sentences directly, by counting the number of questions in a paragraph (*NumQues*) using the heuristic approach developed by Li et al. [27].

4.1.4 Content Dimension:

The vocabulary that an author uses in a paragraph can also indicate the presence of a design point. With the features in the text dimension, we considered simple vocabulary features, such as using specific speculative words. We also want to consider whether a wider set of words might indicate the presence of a design point. Given the broad vocabulary used in any discussion, we cannot simply treat every word that appears in a pull request discussion as a separate feature. In the data mining literature, the large amount of words indicates the problem of the curse-of-dimensionality [19]. To attack this problem, we follow previous studies (e.g., [57], [63]), by converting the words in a paragraph to a simple textual score, called a "content score". The higher the value of the content score, the higher the chance that the paragraph will contain a design point.

To compute the content score, we use the following procdure, also depicted in Figure 1. First, we pre-process the text in paragraphs using standard approaches, including removing stop-words (e.g., "and" and "the") and stemming (e.g., reduce "reading" and "reads" to "read"). Second, for each paragraph in the dataset (Section 3), we extract the tokens and create a word frequency table, which records the number of times each token appears across all paragraphs. Third, to help avoid biasing the design point classifier (hereafter referred to as the "main" classifier), we train three helper classifiers C_1, C_2, and C_p, each of which predicts a content score from the tokens of a given paragraph. Specifically, C_1 and C_2 are used to compute the content score for half of the paragraphs respectively in the training set T used to train the main design point classifier, while C_p is used to compute the content score for paragraphs in the test set P of the main design point classifier (label C in Figure 1). This strategy of multiple classifiers avoids bias from the training sets [57]; otherwise, our models may lead to optimistic (unrealistic) values for the textual scores.

To compute the content score for paragraphs in T, we split it into two subsets t_1 and t_2 using stratified random sampling so that they each have the same distribution and number of paragraphs with and without design points. Then, C_1 is trained using t_1 and used to predict the content scores of the paragraphs in t_2 to produce t_2' (label A in Figure 1). For each paragraph in t_1, C_1 computes the content score based on its word frequency table and whether the paragraph contains a design point or not. Conversely, C_2 is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2924006, IEEE Transactions on Software Engineering

TABLE 3: Features used to identify design points in a paragraph

Dim	Feature	Type	Definition	Rationale
Process	IsProMem	Boolean	Whether the author is a core project member	The role of the author of the comment may influence the likelihood that the paragraph contains design point. A core project member [64] or an invited developer is more likely to have the expertise needed for involving in the pull request discussion
	IsInvited	Boolean	Whether the author was invited to the discussion	
	IsOriginal	Boolean	Whether the author submitted the pull request	
	NumCom	Numeric	Number of comments posted by the author in the discussion	The number of comments relates to the level of the activity of the author, which can characterize the role of the author [45].
Position	PosInPr	Numeric	Position of the containing comment in the discussion	Positional features were found to be useful for classifying discussions [37], [41]. We assume that paragraphs appearing early will have a higher chance to contain design points.
	PosInCom	Numeric	Position of the paragraph in the containing comment	
Text	NumOfWords	Numeric	Number of words in the paragraph	A larger paragraph is more likely to be informative and it is more likely to contain design point.
	HasCapWord	Boolean	Whether the paragraph contains any capitalized words	Developers may use capitalized words to highlight the key points they want to express.
	HasShould	Boolean	Whether the paragraph contains the word "shall/should"	As observed in Section 3, developers may use assertive or speculative language when discussing design. The words "should", "how" "what" "why" "can" or "may" are indicative of speculative language [27], and their presence in a paragraph may increase its likelihood of containing a design point.
	HasHow	Boolean	Whether the paragraph contains the word "how"	
	HasWhat	Boolean	Whether the paragraph contains the word "what"	
	HasWhy	Boolean	Whether the paragraph contains the word "why"	
	HasCan	Boolean	Whether the paragraph contains the word "can/could"	
	HasMay	Boolean	Whether the paragraph contains the word "may/might"	
	NumQues	Numeric	Number of questions in the paragraph	The presence of a high number of interrogative sentences [27] indicates the presence of speculative language (cf. previous).
Content	NBScore	Numeric	Naive Bayes score of the paragraph	The textual content of the paragraph may indicate the presence of a design point. NBScore, NBMScore, COMPScore and RFScore are the likelihood scores of a paragraph containing a design point calculated based on the textual content of the paragraph by four classifiers [19], [57], [63] (see Section 4.1.4).
	NBMScore	Numeric	Naive Bayes multinomial score of the paragraph	
	COMPScore	Numeric	Complement Naive Bayes scores of the paragraph	
	RFScore	Numeric	Random forest score of the paragraph	

trained using t_2 and used to predict the content scores of the paragraphs in t_1 to produce t_1' (label B in Figure 1). This way, we get the new training dataset $T' = t_1' \bigcup t_2'$, where paragraphs are represented by their content score. In other words, the content score of each paragraph in the training set is generated from either C_1 or C_2.

To compute the content score for paragraphs in the test set P, we train the helper classifier C_p. Specifically, C_p is trained on T and used to predict the content score for a new unseen paragraph from P, that we identify as P' (label A in Figure 1). The difference with C_1 and C_2 is that when predicting a new content score, C_p does not know whether the new paragraph contains a design point. In other words, the main design point classifier uses the content scores in T' created by C_1 and C_2, and invokes C_p to get the content scores of paragraphs not seen in the training set, as show in label D in Figure 1.

We investigate four different algorithms for the classifiers, using the same algorithm for each of C_1, C_2, and C_p: a naive Bayes classifier [33], a naive Bayes multinomial classifier [33], a complement naive Bayes classifier [42], and a random forest classifier [5]. We use the implementations of these algorithms provided by the default Weka distribution [18]. We denote their respective features as *NBScore*, *NBMScore*, *COMPScore*, and *RFScore*.

The four classifiers can leverage the textual content of a paragraph in different ways. We opted to use the different classifiers to better characterize the textual content of a paragraph, combining the likelihood scores from different classifiers. This approach can also help dealing with the inherent biases of each classifier, such as the strong independence assumption of the naive Bayes classifier.

4.2 Approach

To determine whether a paragraph contains a design point, we investigate four different classifiers: Random Forest (RF)[4] [5], Naive Bayes (NB) [33], Support Vector Machine (SVM) [19] and K-Nearest Neighbor (KNN, with $K = 5$) [19]. We use Weka to implement the algorithms.

In our study, we use the Area Under the Receiver Operating Characteristic Curve [11] (AUC) to evaluate the performance of our prediction models. AUC scores range from

[4]To reduce the effect of overfitting, in our study, we set the depth of the decision trees built in random forest as 10.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2924006, IEEE Transactions on Software Engineering

TABLE 4: Average AUC for classifiers built on all of the proposed features (Random Forest, Naive Bayes, SVM, and KNN), and content classifiers (Content^{NB}, Content^{NBM}, Content^{COMP}, and Content^{RF}) (mean ± Standard variance)

Approach	AUC	Approach	AUC
Random Forest	0.87±0.06	Content^{RF}	0.80±0.04
Naive Bayes	0.84±0.08	Content^{NB}	0.73±0.04
SVM	0.67±0.15	Content^{NBM}	0.67±0.09
KNN	0.77±0.10	Content^{COMP}	0.56±0.12

0 to 1, with 1 representing perfect prediction performance. A classifier using random prediction has an AUC score of 0.5; thus if the AUC score for a classifier is larger than 0.5, it performs better than random prediction. AUC measures the prediction performance across all the thresholds and it is insensitive to cost and class distributions [39]. Lessmann et al. recommended that AUC should be used as the primary accuracy indicator to compare the performance of prediction models [26]. Romano et al. concluded that a prediction model with an AUC score above 0.7 is often considered to have adequate classification performance [46]. In a recent work, Tantithamthavorn and Hassan recommended that threshold-independent measures (e.g., AUC) should be used in lieu of threshold-dependent measures (e.g., F1-score), since F1-score is sensitive to the used probability threshold [52]. In our context, the AUC score measures the probability the classifier will rank a randomly selected paragraph with a design point higher than a randomly selected paragraph without a design point.

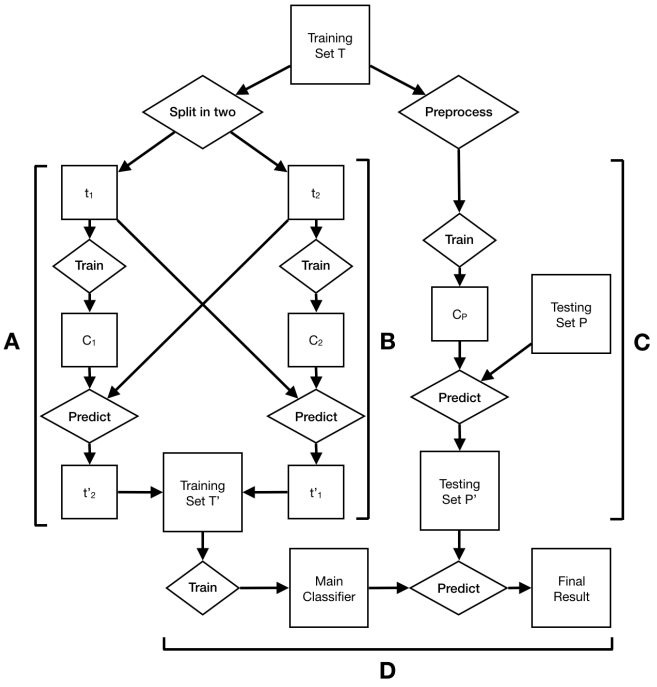

Fig. 1: Diagram of how our approach works, including the calculation of the content scores using C_1 C_2 and C_p

4.3 Results

To determine if the classifier can effectively locate design points in developer discussions (**RQ1**), we evaluate the performance of each of the four classifiers using a cross-project approach. In this approach, we train a prediction model by using annotated pull requests from two projects and test the classifier using the pull requests in the remaining project. This approach helps consider the evaluation of the classifier under realistic conditions as it is conceivable that data from a small number of projects might be manually labeled (as we have done for the dataset presented in Section 3) as long as the classifier trained from such data is applicable across a range of projects.

To help determine if a simpler set of features might suffice for detecting design points, we also built predictions models based on term frequency scores computed by treating each paragraph as a bag-of-words. Following Section 4.1.4, we leverage naive Bayes, naive Bayes multinomial, complement naive Bayes, and random forest to build the prediction models, and we refer to these four classifiers based on the content of paragraphs as content classifiers, and denote them as Content^{NB}, Content^{NBM}, Content^{COMP}, and Content^{RF}, respectively.

Table 4 shows the average and distributions of AUC for each of the four classifiers (RF, NB, SVM, and KNN) based on all of the features we proposed, compared with the four content classifiers on the 10,790 paragraphs from the 34 pull requests in our dataset. The AUC values are averages from running and averaging over all combinations of cross-project validation. Overall, Random Forest (RF) based on all of the proposed features achieves the best results with AUC of 0.87. We confirmed the dominance of the RF approach using a Wilcoxon signed-rank test with Bonferroni correction; the improvement of RF based on all of the features over other approaches is statistically significant at the 95% confidence level. Compared with our proposed approach with the content classifiers, we still find that RF based on all of the features improve them statistically significantly. In practice, RF based on all of the proposed features is the best choice for identifying design points in pull requests.

We believe this result is due to the nature of random forests. A random forest is constructed using a multitude of decision trees, each learned separately at training time, and outputs the mode of the prediction of each tree. By doing this, RF is able to overcome overfits and is robust against noises and outliers in the training dataset.

Our second research question (**RQ2**) asks which features are important in enabling the classifier to discern which paragraphs have a design point. To answer this question, we use a four step process.

Step 1: Correlation Analysis. We first look for collinearity among the features by using variable clustering analysis. Out of the 19 features, there are 4 features belonging to two groups of variables that have correlations larger than 0.7. The two groups are {*NBScore*, *RFScore*}, and {*NBMScore*, *COMPScore*}. Randomly removing one feature from each group (*RFScore* and *COMPScore* respectively), we are left with 17 features.

Step 2: Redundancy Analysis. Having reduced collinearity among the features, we use R to determine redundant features, which do not have unique signal relative to the other features. With this analysis, we did not find any redundant features.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2924006, IEEE Transactions on Software Engineering

TABLE 5: Importance of the 17 features as ranked according to the Scott-Knott ESD test results. The second and third columns correspond to P-values, and Cliff's Delta for the features. Statistically significance at confidence level of 95% and non-negligible effect size are in bold

Group	Feature	P-value	Cliff's Delta
1	**IsInvited**	**<0.001**	**0.36 (Medium)**
2	**NumOfWords**	**<0.001**	**0.56 (Large)**
3	**PosInPR**	**<0.001**	**-0.15 (Small)**
4	NBMScore	>0.05	0.00
5	NumCom	<0.001	-0.10
6	**PosInCom**	**<0.001**	**-0.17 (Small)**
7	NBScore	>0.05	0.00
8	**HasCan**	**<0.001**	**0.16 (Small)**
9	NumQues	<0.01	0.10
10	IsProMem	>0.05	0.02
11	HasShould	<0.001	0.10
12	IsOriginal	<0.001	-0.09
13	HasCapWord	<0.001	0.13
14	HasWhat	<0.001	0.07
15	HasMay	<0.001	0.06
16	HasWhy	<0.001	0.04
17	HasHow	<0.001	0.05

Step 3: Important Feature Identification. Next, we apply "out-of-bag" (OOB) estimation to estimate the internal error of a random forest used to estimate the internal error of a random forest classifier [61]. The main idea is to permute each feature randomly one by one and see whether the OOB estimates will be reduced significantly or not.

For each run of a 10 times 10-fold cross validation, we get an importance value for each feature. To determine the features that are the most important for the whole dataset, we take the importance values from all 10 runs and apply the Scott-Knott Effect Size Difference (ESD) test [28], [53], [62].

Table 5 presents the importance of the 17 features as ranked according to the Scott-Knott ESD test. We find that *IsInvited* (whether the paragraph author is invited to the discussion), *NumOfWords* (number of words in the paragraph), *PosInPR* (position of the paragraph in the pull request), *NumCom* (number of comments the developer posted previously in the pull request), *NBMScore* (naive Bayes multinomial score of the paragraph), and *PosInCom* (position of the paragraph in the comment) are the six most important features that influence the random forest model. These features help most in discriminating paragraphs with design points from these without design points.

Step 4: Effect of Important Features. To understand the effect of the six most important features, we compare their values in paragraphs with design points and in paragraphs without design points. To analyze the statistical significance of the difference between the two groups of paragraphs, we apply the Wilcoxon rank-sum test [31] at 95% significance level. To show the effect size of the difference between the two groups, we calculate Cliff's Delta[5] [29], which is a non-parametric effect size measure. A positive effect indicates that a higher level of a feature corresponds to an increase in the likelihood of a paragraph containing design points, while a negative effect indicates that a higher level of a feature corresponds a decrease in the likelihood of a paragraph containing design points. Table 5 presents the p-values and Cliff's Delta for the 17 features. We notice that *IsInvited*, *NumOfWords*, *PosInPR*, and *PosInCom* show non-negligible effect size on the two groups of paragraphs with and without design points, while *NBMScore* shows negligible effect size. Based on the findings in Table 5, we conclude that:

- *IsInvited* has a medium positive effect: invited developers are more likely to propose Design Points.
- *NumOfWords* has a large positive effect: longer paragraphs have a higher chance to contain Design Points than shorter ones.
- *PosInPR* has a small, non-negligible negative effect: paragraphs posted earlier in the discussion have a higher chance of containing Design Points than those posted later.
- *PosInCom* has a small, non-negligible negative effect: developers tend to express Design Points at the beginning of a comment.

In summary, we extracted 19 features from pull requests, categorized in four dimensions: process, position, text and content. Experimental results show that random forest achieves the best performance with an average AUC value of 0.87. Moreover, statistical tests show that *IsInvited*, *NumOfWords*, *PosInPR*, and *PosInCom* are the four most important features to locate paragraphs with design points; these features are in the process, text, and position dimensions. It might beneficial to focus in the future on investigating these four features to both simplify and improve the classifier.

5 Developers and Design Points

An approach for automatically locating design points is much more useful if it can find design points in pull requests for projects on which it was not trained and that still agree with what a human developer would identify as a design point. To investigate whether the classifier we developed generalizes to pull requests from other projects and whether the design points automatically identified agree with a broader set of human developers than the three annotators used to build the classifier, we performed an experiment. The experiment consisted of creating a gold standard dataset based on pull requests from projects not used to develop the classifier and then comparing the results of applying the classifier to the dataset.

5.1 Gold Standard

To generate our gold standard dataset, we selected 5 among the most popular projects on Github: Bootstrap, React, FreeCodeCamp, TensorFlow, Electron. From each project, we selected the 10 most commented pull requests and randomly sampled 50 paragraphs for each project, to a total of 250 paragraphs.

We then recruited five Computer Science students to annotate the presence of design points in these paragraphs. Four of the students had between three and six years of

[5]Cliff defines a delta of less than 0.147, between 0.147 to 0.33, between 0.33 and 0.474, and above 0.474 as negligible, small, medium, and large effect size, respectively.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2924006, IEEE Transactions on Software Engineering

experience working in software development, averaging 4.75 years; the remaining student had no significant experience in industry, but has been a teaching assistant for a software engineering course involving a large codebase. Each evaluator was assigned three samples from the dataset, or 250 paragraphs each. Overall, these assignments were made in such a way that each sample was assigned to three evaluators. Each evaluator was given the definition of a design point (Section 1) and asked to mark whether each paragraph contained a design point or not.

From these annotations, we formed a gold standard by selecting only the paragraphs for which all three evaluators agreed either did or did not contain a design point. The resultant gold standard consists of 181 paragraphs (out of 250 paragraphs) for which there was agreement by all annotator. 19 of these paragraphs were identified as design points and 162 as paragraphs without design information.

5.2 Classifier Evaluation

Having the gold standard as ground truth, we proceeded to evaluate the classifier against it. We applied the previously trained classifier to the 50 pull request discussions. The classifier was used to predict the presence of design information in every paragraph in the discussions, rather than just those from the gold standard, as the classifier uses information from an entire discussion and cannot be applied to paragraphs independent from their context. We then extracted whether the classifier predicted (or not) if each of the 181 paragraphs of interest were design points, and compared the resultant predictions to the gold standard.

TABLE 6: Results of the comparison between the classifier and the gold standard

Paragraphs			
181			
Design Points		**Not Design Points**	
19		162	
True Positive	False Negative	True Negative	False Positive
13	6	150	12
AUC		0.81	

Table 6 shows the results of this evaluation. Overall, the classifier was able to correctly identify 163 paragraphs out of 181 as either being design points or not. The classifier failed to identify correctly 18 paragraphs: 6 of which were missed as design points and 12 which were incorrectly identified as design points. The classifier tends to create false positives - incorrectly identifying a paragraph as having a design point when it does not - when the language used is similar to design points but is really about implementation details. Listing 4 shows an example of a paragraph from the false positives. Note that all three annotators agreed this paragraph does not discuss design. The language used in this paragraph is similar to the one normally seen in design points (such as having a implicit question followed by some rationale), but this particular paragraph is discussing an implementation detail that is not a piece of design information.

Listing 4: Example of a paragraph containing a design point from pull request #12422, `Node.js` project

```
1 You may want to explain somewhere why you put
  parenthesis here. The only reason is to prevent
  automatic semi-colon insertion in return.
```

The classifier also misses identifying some paragraphs as a design point (false negatives); several of these missed design points consist of short sentences, which might be difficult cases for the classifier to identify features as indicating a design point.

The evaluation resulted in AUC of 0.81, which is comparable to the results of experiments on the classifier reported earlier in this paper Section 4.3. This result provides evidence that the classifier can generalize to pull requests from other projects.

6 Discussion

We have shown that the automatic location of design points at the level of paragraphs in pull request discussions is possible. Locating the design points is useful only if tools can be built to then extract and represent the design point information in ways that improve tasks performed by software development personnel. In this section, we take the first steps towards investigating tools built on design points by sketching tools that might build on paragraphs identified as containing design points. We also consider how design point paragraphs might be interrelated and the implications that the structure that exists between design points may have on techniques needed to extract design information. Finally, we discuss how the techniques described in this paper might apply to detecting design points in other forms of written developer discussions.

6.1 Tool Ideas

We describe three kinds of tools that could be built if design points can be located automatically.

As one kind of tool, design point paragraphs might be used to enhance current approaches for recommending reviewers of pull requests. Existing approaches focus on the expertise of reviewers based on the source code they have contributed to the system (e.g., [54], [65]). This information might be augmented with what kinds of design discussions the developers participate in so that if a particular review discussion begins to talk about that design consideration, the developer could be invited. An advantage over source code based recommenders for reviewers is more ability to provide recommendations within the current focus and topic of the review discussion.

Another possible tool would be to reconstruct the state of the design of the project. Open source software projects often feature thousands of pull requests over their lifespan: for example, the *Rust* project has had nearly 25,000 pull requests on GitHub. Learning about the design of the system by reviewing these pull requests is impossible for a newcomer. For an existing contributor, it is also hard to keep up with the state of the design based on latent information in the pull requests. One possible approach is to investigate the extraction of design topics. In earlier work, we described what design topics might be determined from design points identified manually [60], such as *maintainability*, *testing* or *performance*. With an automated approach to design point

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2924006, IEEE Transactions on Software Engineering

location, design topics could be extracted from more discussions and used to create an easily accessible archive of design information.

Finally, links could be introduced automatically between pull requests to help developers traverse similar design topics in discussions. At present, developers can be unaware of relevant discussions. For instance, sometimes, pull requests are closed due to being duplicates of other pull requests, potentially losing access to valuable discussions [15]. At other times, the discussion on a pull request can transition over time to a topic already covered in detail in the past; contributors might not be aware of the other discussions. To aid developers in following and finding relevant discussions, a tool could be built to automatically reference previous discussions that might be relevant to the current pull request or topic being discussed.

6.2 Other Forms of Latent Design

In this paper, we have been using information extracted from each paragraph to attempt to detect where design is being discussed automatically. However, we have not considered how such information is structured across an entire discussion. A few researchers have attempted to model the structure of design discussions. Mashiyat et al. used a simplified version of Rhetorical Structure Theory [51] to model a small design discussion [32]. Black et al. studied recordings of design sessions and created a framework of "agent moves" for expressing events such as a participant initiating a discussion on a design point or supporting a particular design alternative. In previous work, we studied three pull request discussions and identified different types of relationships between various design related events, such as "rationalization", "elaboration", and "justification" [59]. These studies point to the conclusion that developer discussions contain rich structures of latent design information.

We envision using the automated location of design points as a building block of a comprehensive approach to building models of design discussions, which can in turn be used to manage design discussions in multiple ways. On the one hand, similar to the work of Knauss et al. [21], such models can be used to focus design related communication patterns and to monitor the progress of design discussions to detect problems such as stalling, bikeshedding, and groupthink. On the other hand, structuring relationships between different design points can help in identifying design alternatives and more generally extracting design spaces from discussions. Such information can be used to assist the design discussion by leveraging techniques such as design space exploration [2], or be used for future reference, e.g., by allowing developers in future discussions to reevaluate previously discarded designs. Finally, they can be used to recover and document design rationale [34].

6.3 Design Points in Other Discussion Forms

Our focus in this paper is on discussions generated from pull requests. Pull requests have been frequently studied in the context of investigating the presence of design information [6], [56]. Nonetheless, there are other kinds of discussions that we would like to investigate in the future. Recently, Gousios et al. surveyed developers about their preferred channel of communication to propagate changes: the top two responses were *Issue Trackers* and *Pull requests*, with *Emails* claiming a third spot.

While issues and pull requests are fairly similar entities in Github, emails might provide a new way to expand this investigation. Previous work has already shown how mailing lists can be a fertile ground for gathering information. Gousios et al. found that many experienced contributors use mailing lists to maintain awareness on the state of a project [15], and Sorbo et al. developed a semi-supervised approach to identify the purpose of an email [49]. While we believe that the language utilized by developers does not change significantly, email differs from pull requests discussions in not having clearly defined discussion boundaries and in having a smaller presence of core members in the discussions [17].

7 Threats to Validity

To investigate latent design information in developer discussions, we created a dataset of design points (Section 3) and used the resultant dataset as a base for a supervised learning approach (Section 4). We then compared the results of the classifier to a gold standard formed from human considerations of discussions on projects on which the classifier had not been trained (Section 5). There are threats to validity that arise for each of these approaches.

Internal and Construct Validity. For the annotation process, we had a total of three individuals involved in labeling the existing design points (Section 3). A shared codebook was developed and used to annotated over ten thousand paragraphs with five labels. This annotation process helped inform the occurrence of various kinds of information that could be used in a supervised learning process. For the classifier, we relied on information, such as `ROLE` extracted automatically from the discussions. We leave the investigation of more difficult to extract information, such as the language used and structure of design points, for future work (Section 6).

We formed the gold standard used to evaluate the classifier (Section 5) with five students. To mitigate any risk of a lack of development knowledge in these students, we chose individuals with extensive coding and design experiences; four out of five had spent many years working as software developers.

External Validity The dataset annotated and used by the classifier comprises 10,790 paragraphs across 34 pull requests from 3 projects. By considering pull requests from 3 projects, we mitigate bias to any particular project.

With the classifier, we considered 19 features categorized into 4 dimensions. In choosing the features to investigate, we focused on features related to development process, discussion structure and the content of discussions to avoid features that might only apply to a specific project. Features related to the content dimension depend on the range of vocabulary represented by the projects on which the classifier is trained; further study is needed to determine how these features perform across a wider range of projects.

8 Summary

Developers discuss design issues both in-person [38] and in on-line discussions [6], [56]. In this paper, we focus on one kind of online discussion, pull requests, and investigate how to locate the points of the discussion where developers discuss design. We introduced the concept of *design points*, defined as "a piece of a discussion relating to a decision about a software system's design that a software development team needs to make". We apply supervised machine learning to build a classifier that can locate design points automatically in pull requests at the level of paragraphs. This classifier can locate design points with high accuracy (average AUC score is 0.87).

To demonstrate that the classifier locates design points beyond the dataset we created, we apply the classifier to pull requests on which it was not trained or tested and compared to a gold standard created by five students with development experience using five generic projects from GitHub. We found that the classifier was still able to locate design points in projects it was not trained with high accuracy (average AUC score is 0.79).

This paper shows that there is useful design information latent in on-line developer discussion and provides a means to locate this information at a coarse granularity. Future research can determine how to locate more specific and nuanced design information and investigate how to semantically model the information to produce even more useful tools for developers.

Acknowledgments

We thank Georgios Gousios for providing us with the corpus of failed pull requests from GitHub. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] Ieee standard for information technology–systems design–software design descriptions. *IEEE STD 1016-2009*, pages 1–35, July 2009.

[2] Hani Abdeen, Dániel Varró, Houari Sahraoui, András Szabolcs Nagy, Csaba Debreceni, Ábel Hegedüs, and Ákos Horváth. Multi-objective optimization in rule-based design space exploration. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 289–300, 2014.

[3] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, October 2002.

[4] T. J. Biggerstaff. Design recovery for maintenance and reuse. *Computer*, 22(7):36–49, 1989.

[5] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[6] João Brunet, Gail C Murphy, Ricardo Terra, Jorge Figueiredo, and Dalton Serey. Do developers discuss design? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 340–343, 2014.

[7] E. J. Chikofsky and J. H. Cross. Reverse engineering and design recovery: a taxonomy. *IEEE Software*, 7(1):13–17, 1990.

[8] James C Corbett, Matthew B Dwyer, John Hatcliff, Shawn Laubach, Corina S Pasareanu, Hongjun Zheng, et al. Bandera: Extracting finite-state models from java source code. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 439–448, 2000.

[9] Denzil Correa and Ashish Sureka. Chaff from the wheat: characterization and modeling of deleted questions on stack overflow. In *Proceedings of the 23rd international conference on World wide web*, pages 631–642, 2014.

[10] Kevin Crowston, Hala Annabi, and James Howison. Defining open source software project success. In *Proceedings of the International Conference on Information Systems*, page 28, 2003.

[11] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

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

[13] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. 1995.

[14] Giorgio Giacinto and Fabio Roli. Design of effective neural network ensembles for image classification purposes. *Image and Vision Computing*, 19(9):699 – 707, 2001.

[15] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. Work practices and challenges in pull-based development: The contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 285–296, 2016.

[16] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, pages 358–368, 2015.

[17] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. van Deursen. Communication in open source software development mailing lists. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 277–286, May 2013.

[18] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[19] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. 2011.

[20] A. P. J. Jarczyk, P. Loffler, and F. M. Shipmann. Design rationale for software engineering: a survey. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, volume ii, pages 577–586 vol.2, Jan 1992.

[21] Eric Knauss, Daniela Damian, Jane Cleland-Huang, and Remko Helms. Patterns of continuous requirements clarification. *Requirements Engineering*, 20(4):383–403, 2015.

[22] Philippe Kruchten. Architectural blueprints — the "4+1" view model of software architecture. 12:42–50, 11 1995.

[23] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[24] RJ Landis and GG Koch. The measurement of interrater agreement. *Statistics methods for rates and proportions*, 2:212–236, 1981.

[25] J. Lee. Design rationale systems: understanding the issues. *IEEE Expert*, 12(3):78–85, May 1997.

[26] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008.

[27] Baichuan Li, Xiance Si, Michael R Lyu, Irwin King, and Edward Y Chang. Question identification on twitter. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2477–2480, 2011.

[28] Heng Li, Weiyi Shang, Ying Zou, and Ahmed E Hassan. Towards just-in-time suggestions for log changes. *Empirical Software Engineering*, pages 1–35, 2016.

[29] Guillermo Macbeth, Eugenia Razumiejczyk, and Rubén Daniel Ledesma. Cliff's delta calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica*, 10(2):545–555, 2011.

[30] A. MacLean, R. M. Young, and T. P. Moran. Design rationale: The argument behind the artifact. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '89, pages 247–252, 1989.

[31] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 18(1):50–60, 1947.

[32] Ahmed Shah Mashiyat, Michalis Famelis, Rick Salay, and Marsha Chechik. Using developer conversations to resolve uncertainty in software development: A position paper. In *Proceedings of the 4th International Workshop on Recommendation Systems for Software Engineering*, pages 1–5, 2014.

[33] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48, 1998.

[34] Thomas P Moran and John M Carroll. *Design rationale: Concepts, techniques, and use*. 1996.

[35] H. A. Müller and K. Klashinsky. Rigi-a system for programming-in-the-large. In *Proceedings of the 10th International Conference on Software Engineering*, ICSE '88, pages 80–86, 1988.

[36] Gail C Murphy, David Notkin, and Kevin J. Sullivan. Software reflexion models: Bridging the gap between design and implementation. *IEEE Transactions on Software Engineering*, 27(4):364–380, 2001.

[37] Gabriel Murray and Giuseppe Carenini. Summarizing spoken and written conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 773–782, 2008.

[38] Marian Petre and re Van Der Hoek. *Software Designers in Action: A Human-Centric Look at Design Work*. 2013.

[39] Gopi Krishnan Rajbahadur, Shaowei Wang, Yasutaka Kamei, and Ahmed E Hassan. The impact of using regression models to build defect classifiers. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 135–145. IEEE Press, 2017.

[40] Paul Ralph and Yair Wand. *A Proposal for a Formal Definition of the Design Concept*, pages 103–136. 2009.

[41] Sarah Rastkar, Gail C. Murphy, and Gabriel Murray. Automatic summarization of bug reports. *IEEE Trans. Software Eng.*, 40(4):366–380, 2014.

[42] Jason D Rennie, Lawrence Shih, Jaime Teevan, David R Karger, et al. Tackling the poor assumptions of naive bayes text classifiers. In *ICML*, volume 3, pages 616–623, 2003.

[43] Martin P. Robillard. Sustainable software design. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 920–923, 2016.

[44] Martin P Robillard and Nenad Medvidović. Disseminating architectural knowledge on open-source projects: a case study of the book architecture of open-source applications. In *Proceedings of the 38th International Conference on Software Engineering*, pages 476–487, 2016.

[45] P. Rodeghero, S. Jiang, A. Armaly, and C. McMillan. Detecting user story information in developer-client conversations to generate extractive summaries. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 49–59, May 2017.

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

[47] A. Shahbazian, Y. Kyu Lee, D. Le, Y. Brun, and N. Medvidovic. Recovering architectural design decisions. In *2018 IEEE International Conference on Software Architecture (ICSA)*, pages 95–9509, April 2018.

[48] Susan Elliott Sim and Richard C Holt. The ramp-up problem in software projects: A case study of how software immigrants naturalize. In *Software Engineering, 1998. Proceedings of the 1998 International Conference on*, pages 361–370, 1998.

[49] A. D. Sorbo, S. Panichella, C. A. Visaggio, M. D. Penta, G. Canfora, and H. C. Gall. Development emails content analyzer: Intention mining in developer discussions (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 12–23, Nov 2015.

[50] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proc. of the 18th ACM Conference on Computer Supported Cooperative Work*, pages 1379–1392, 2015.

[51] Maite Taboada and William C. Mann. Applications of rhetorical structure theory. *Discourse Studies*, 8(4):567–588, 2006.

[52] Chakkrit Tantithamthavorn and Ahmed E Hassan. An experience report on defect modelling in practice: Pitfalls and challenges. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 286–295. ACM, 2018.

[53] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 43(1):1–18, 2017.

[54] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. i. Matsumoto. 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)*, pages 141–150, March 2015.

[55] S.E. Toulmin. *The Uses of Argument*. 1958.

[56] Jason Tsay, Laura Dabbish, and James Herbsleb. Let's talk about it: evaluating contributions through discussion in github. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, pages 144–154, 2014.

[57] Harold Valdivia Garcia and Emad Shihab. Characterizing and predicting blocking bugs in open source projects. In *Proceedings of the 11th working conference on mining software repositories*, pages 72–81, 2014.

[58] Jilles van Gurp and Jan Bosch. Design erosion: problems and causes. *Journal of Systems and Software*, 61(2):105 – 119, 2002.

[59] Giovanni Viviani, Calahan Janik-Jones, Michalis Famelis, and Gail C. Murphy. The structure of software design discussions. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '18, 2018.

[60] Giovanni Viviani, Calahan Janik-Jones, Michalis Famelis, Xin Xia, and Gail C. Murphy. What design topics do developers discuss? In *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, pages 328–331, 2018.

[61] David H Wolpert and William G Macready. An efficient method to estimate bagging's generalization error. *Machine Learning*, 35(1):41–55, 1999.

[62] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E Hassan, and Zhenchang Xing. What do developers search for on the web? *Empirical Software Engineering*, pages 1–37, 2017.

[63] Xin Xia, Emad Shihab, Yasutaka Kamei, David Lo, and Xinyu Wang. Predicting crashing releases of mobile applications. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 29, 2016.

[64] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation open source software developers. In *Proceedings of the 25th international conference on software engineering*, pages 419–429, 2003.

[65] M. B. Zanjani, H. Kagdi, and C. Bird. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering*, 42(6):530–543, June 2016.

Giovanni Viviani received his MSc in computer science in 2016 from the University of British Columbia. He is currently a PhD candidate at the Department of Computer Science at the University of British Columbia. His research focuses on investigating how developers communicate and how to recover information from developers discussions.

Michalis Famelis received his PhD in computer science in 2016 from the University of Toronto. He is currently an assistant professor at the Department of Computer Science and Operations Research at the Université de Montréal, where he is a member of the GEODES software engineering research team. His research interests include model-driven engineering, formal methods, software analysis and design, and empirical software engineering.

Xin Xia is a lecturer at the Faculty of Information Technology, Monash University, Australia. Prior to joining Monash University, he was a post-doctoral research fellow in the software practices lab at the University of British Columbia in Canada, and a research assistant professor at Zhejiang University in China. Xin received both of his Ph.D and bachelor degrees in computer science and software engineering from Zhejiang University in 2014 and 2009, respectively. To help developers and testers improve their productivity, his current research focuses on mining and analyzing rich data in software repositories to uncover interesting and actionable information. More information at: https://xin-xia.github.io/

Calahan Janik-Jones Calahan Janik-Jones is a recent graduate from the University of Toronto in Linguistics. Calahan does research in Linguistics, Language Revitalization, and Software Engineering.

Gail C. Murphy is a Professor in Computer Science and Vice-President Research and Innovation at the University of British Columbia. She is also a co-founder and member of the board at Tasktop Technologies, Inc. She received her B.Sc. in Computing Science from the University of Alberta and the M.S. and Ph.D. degrees in Computer Science and Engineering from the University of Washington. Her research interests include software developer productivity and software evolution. She is a Fellow of the Royal Society of Canada, a Fellow of the ACM and a member of the IEEE Computer Society.