The Structure of Software Design Discussions

Giovanni Viviani
University of British Columbia
vivianig@cs.ubc.ca

Calahan Janik-Jones
University of Toronto
cal.janik.jones@mail.utoronto.ca

Michalis Famelis
Université de Montréal
famelis@iro.umontreal.ca

Gail C. Murphy
University of British Columbia
murphy@cs.ubc.ca

ABSTRACT

As a software system is iteratively developed, software developers engage in many discussions, often through written forms. Some of these discussions occur on pull requests and include information about the design of the system to which the code in the pull request is being contributed. Although previous work has shown that design is discussed in forums like pull requests, little is known about the form and content of the discussion about design.

In this paper, we report on an in-depth analysis of three pull requests to better understand the form and content of design information in pull request discussions to enable the development of tools to help humans access this information.

ACM Reference format:
Giovanni Viviani, Calahan Janik-Jones, Michalis Famelis, and Gail C. Murphy. 2018. The Structure of Software Design Discussions. In *Proceedings of CHASE'18:IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software , Gothenburg, Sweden, May 27, 2018 (CHASE'18),* 4 pages.
https://doi.org/10.1145/3195836.3195841

1 INTRODUCTION

Software development is about much more than programming. Before any source code is written, members of a software development team will have engaged in many verbal and written discussions. As a system is iteratively developed, discussions occur frequently. These discussions contain rich information about a project and its evolution. For example, analysis of discussions have shown how they often influence the direction of a project [17], such as playing a critical factor in deciding whether to accept a pull request [7].

Researchers have built on the information in discussions to provide tools to improve software development. For instance, Hipikat indexed mailing list discussions and provided search capabilities for a developer to learn from past discussions related to a current challenge [4]. More recently, research has considered how to recognize and extract useful information out of a discussion: Rodeghero et. al have built a system to identify user stories from verbal discussions between clients of a system and developers of the system [15].

We are interested in whether the discussions that occur between humans involved in the development contain information about the system's *design*. All too often, design information is implicit, particularly for open source systems. If one peruses repositories of open source software, there is no common means of describing the principles of design that are driving the development. When design principles are not understood, the design of the system can erode over time, causing a loss of desired properties [13] and developers can have difficulty making desired contributions to a system [7].

Brunet et. al have shown that design discussions do occur in issue trackers [3]. Although this earlier effort showed that discussions about design occur, it did not provide any detail on the form or content of the design information discussed.

Our aim is to delve into the detail of design discussions as captured through written interactions between developers. By understanding more about the form and content of the discussions, we believe tools can be built to help human developers make use of this latent source of design information about a system. More information about design could lead to faster, higher-quality contributions to a system, particularly for open source systems.

For example, consider a developer joining the Rails[1] project who wonders why the `MailPreviews` component manipulates MIME parts in a particular way. The developer might query system artifacts with this question. A tool that understands written design discussions could find that the discussion over Rails pull request #14519 is relevant and extract for the developer the information that Rails developers have decided that: "it'd probably be clearer (and simpler) to return new MIME parts rather than alter them in-place".[2]

To better understand how developers discuss design, we report on an in-depth analysis of the structure of discussions from three *failed* pull request from the Rails open source project. Failed pull requests represent suggested changes to a project that ultimately were not merged into the system. We study these failed pull requests because design-related concerns are some of the most common factors that impact the choice of whether to integrate a pull request [7]. Our analysis found that particular questions about design are not modularized within a discussion, but rather become spread throughout the discussion with developers posing questions about design that re-surface and are re-considered multiple times with different possible solutions. We found that an individual developer's

CHASE'18, May 27, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5725-8/18/05…$15.00
https://doi.org/10.1145/3195836.3195841

[1] github.com/rails/rails
[2] Rails #14519 line 245

Listing 1: Discussion snippet from Rails pull request #11407.

```
@Alamoz
Alamoz commented Jul 13, 2013
[...]
Someone will have to decide whether to add a default
stylesheet to rails apps. Also, there is discussion to
the effect that inline style elements are not a real
security threat (see http://lists.w3.org/Archives/Public/
public-webappsec/2013Feb/0015.html) and that JQuery
relies on inline style elements for things like tooltips.
 So this warrants further investigation.
;##Q4 Are inline style elements a real security threat?
[...]
@NZKoz
Ruby on Rails member
NZKoz commented Jul 13, 2013
[...]
As for the security issues of inline styles, there are
various dangerous things you can do with expressions and
moz-binding etc, which is why the CSP disallows them.
However there's no issue with the values we're currently
generating,
;##C1::Q4 No.
[...]
@Alamoz
Alamoz commented Jul 14, 2013
[...]
``Inline style elements certainly give the attacker
leverage. The
question isn't
whether it's a vulnerability but rather how severe a
vulnerability.''
;##C2::D4 Gives an attacker leverage.
[...]
@NZKoz
Ruby on Rails member
NZKoz commented Jul 14, 2013
@Alamoz you're missing the point a bit here, there's no
question about whether the code we're generating is
insecure, it's not. There's no vulnerability at all ever
from setting padding and margin on a div.
;##L12 Padding and margin on a div are not vulnerable
;##R C2::Q4 J: L12
```

contributions (i.e., comments) define an overall logical structure for the discussion, but that a determination of the discussion's internal rhetorical structure must be also identified to capture the argumentation between developers. Understanding this rhetorical structure is needed to understand the resulting design of a system.

We begin with a discussion of related work (Section 2) before describing the analyses we performed (Section 3) and what we learned from the analyses. We then present future directions of work based on these early investigations (Section 4). We present our findings as a basis other researchers might use to further explore how to access the content of design discussions.

2 RELATED WORK

Discussions that occur in software artifacts, such as issues and pull requests, are a special case of the generic class of asynchronous, written interactions. The field of computational linguistics studies such forms of discussions using various approaches for capture and analysis (e.g., [5, 8, 18]). In discussions, participants are typically understood to perform communicative acts, such as posing questions, making assertions, and withdrawing suggestions, known as speech acts [18]. Commonly, a participant makes speech acts in response to speech acts of other participants, such as proposing an answer to someone else's question. Such two-part structures are known as adjacency pairs, and their detection is an important computational

Listing 2: Discussion snippet from Rails pull request #14519.

```
@tute commented Mar 28, 2014
[...]
If email is of type 'multipart/relative', then we render
the text/html part contained in it.
;##Q1 Should multipart/relative emails render the text/
html contained in them?
;##C1::Q1 > Q2
;##S C1::Q1 J: L1
We'd like to render the image using a URL [...]
;##Q2 How to render the image?
;##C1::Q2 using a URL
[...]
@jeremy Ruby on Rails member added a note May 1, 2014
Seems we'd need to decode the body before munging it.
;##Q3 Does the body need to be decoded?
;##C1::Q3 Yes.
[...]
@tute added a note May 2, 2014
Calling decode rather than raw_source here didn't work.
It didn't render the images properly, can investigate
further if needed.
;##L6 calling decode doesn't render the images properly.
;##R C1::Q3 J: L6
```

linguistics problem, involving the usage of both local information (at the level of individual sentences) and global (using the discussion history) [8]. The structure of discussions can also be captured using Rhetorical Structure Theory (RST) [16], which was originally developed for representing the internal structure of monologues, but has been extended to discussions [5]. In RST, communicative acts are captured as typed moves, such as "assert", "command", and "support". Relationships between moves, such as "elaboration" and "concession", amongst others, add further structure [10]. RST has been used as the basis for mining various aspects of conversations such as detecting argumentation [12] and disagreement [1].

Generic dialogue modelling systems are not well suited to software design discussions because they have a unique focus on reaching "agreement on what the requirements of the system really are and what features should be implemented in order to meet these requirements" [2, p.2]. Existing work on studying the process of software design has focused on in-person conversations and whiteboard design sessions, with an emphasis on studying design in its "natural setting" and aiming to describe the activities involved in the process of design [14]. To the best of our knowledge, there exist two approaches to modelling the structure of discussions that focus specifically on software design. Mashiyat et al. proposed to capture the structure of online design discussions using a simplified variant of RST, without developing the approach beyond a toy example [11]. Black et al. developed a conceptual framework for capturing events in in-person design sessions as agent moves. The framework was developed by studying a single video-taped design discussion and includes types of moves such as proposals, questions, challenges, justifications, etc. [2]. We used ideas from these approaches as inspiration for our own investigation.

3 DEVELOPER DISCUSSIONS ABOUT DESIGN

To better understand the form and content of how developers discuss design, we posed the following research questions:

RQ1: How does design information occur in written discussions?

RQ2: How are elements of design information related in a written discussion?

As this is an early exploratory study, we decided to focus on a few number of pull requests from one project. We made this choice to enable a rich understanding of the context of the discussions. The project we focused on was Rails, a framework for building web applications. We selected the three failed pull requests at random from a set of 23 Rails discussions extracted with GHTorrent [6]. The three pull requests analyzed were: #14519, #11407, and #3871.

We opted for a coding process that builds on earlier work (e.g., [11]) determining and applying labels to sentences, phrases, and implicit suggestions in any comment of a discussion extracted from a pull request. We refer to these labels as "elements". When coding a discussion, we identified the elements introduced in each comment, and we determined how different elements across the discussion relate to each other.

3.1 GitHub Pull Request Structure

On Github, a pull request discussion consists of multiple time-stamped *comments*. Each comment is made by an *author*, belonging to three possible categories: the *original poster* is the creator of the pull request and author of the first comment in the discussion; we consider authors who are part of the core development as *project members*; otherwise, we consider participants as *other*.

Each comment consists of a number of *paragraphs*, interspersed with occasional code snippets. The discussions also contain occasional *events* such as code commits, cross-references to other pull requests or issues, links to code reviews, etc. As our focus is on developer discussions, and not on the code being modified, we opted to ignore code snippets in our coding process.

3.2 Labelling

Two authors read and applied abstract labels to paragraphs of the discussion for #11407. We started with a small initial set of labels, inspired from previous work [11] and independently annotated the discussion. After consolidating the annotations, we expanded the set of labels to capture additional types of "moves": a move is a part of a comment introducing a new element, a concept inspired from RST and Black et al. [2]. We then used the new label set to annotate the other two pull requests.

Our final set of labels was: Question (Q): the comment author poses a design decision that needs to be addressed; Candidate (C): the author proposes a way to resolve a Q; Support (S)/Reject (R): the author supports/rejects a C; and Justification J and Rationale L: the author justifies the prefixed Support/Reject with a rationale.

Listing 1 contains a snippet of an annotated discussion. Labels are placed below the paragraph to which they refer and are prefixed with the string "; ##". The developers are discussing if inline style elements can be a security threat. In line 8, @Alamoz introduces the discussion by pointing out an ongoing debate whether inline style elements are a threat or not. Hence, the paragraph is annotated with Q4, to indicate that this developer wishes to discuss an open point about the design. We assigned unique numbers to each label to allow cross-referencing. Line 19 has the label "C1 :: Q4", indicating the first candidate solution C1 to question Q4. We use the symbol "::" to link Cs with their respective Qs. Similarly, line 28 has the label "R C2 :: Q4 J : L12", indicating the candidate solution C2 for the question Q4. In this move, the candidate "C2 :: Q4" is being rejected as a solution for Q4, as indicated by the prefix R. The suffix "J : L12" provides further details, indicating that this rejection is justified (J) by the rationale L12.

Listing 2 contains a more complex example. In this case, the developers are discussing a bug fix to an email rendering subsystem. In line 16, @jeremy speculates whether it would help to decode the message body before processing it, and the speculative question is annotated with Q3. However, his phrasing implies that he is also proposing a candidate solution to his own question. That is to say, he thinks that they *should* in fact do the decoding. In this case, line 16 is also labelled "C1 :: Q3" to demonstrate that the same paragraph both poses the question and offers a candidate solution. Later, this candidate is the rejected by @tute at line 25, which is indicated with the label "R C1 :: Q3 J : L6". Again, the suffix "J : L6" shows that this rejection is argued for and justified by the rationale L6.

The snippet also shows labels from the expanded set. line 5 has the annotation Q1, indicating a question. This is followed by line 7, where we have used the label "C1 :: Q1 > Q2" to indicate that question Q2 only makes sense if Q1 is resolved by selecting the candidate solution C1. Deciding whether it is a good idea to render images as a URL (Q2) is only a meaningful question if the team has already decided that multipart emails should be rendered as text (C1 :: Q1). In other words Q2 *elaborates* C1 :: Q1. We indicate the elaboration relationship with the symbol ">").

3.3 Discussion

The coding process provides insight into the two research questions. With respect to RQ1 (how design information appears in discussions) we gained two major insights. First, multiple design concerns may be brought up in the same comment (e.g., @tute's comment between lines 1-14 in listing 2). This suggests that the use of comments as has been considered in earlier work (e.g., [9] and [15]) is too coarse, and a finer level, paragraphs for example, may be a more appropriate granularity for localizing design.

Second, contrary to the intuitions found in previous work on modelling design discussions [2, 11], design concerns do not always appear in the form of design questions (Q) that are subsequently answered by respective candidate solutions (C). It is not always straightforward to unambiguously classify an element as either Q or C, because discussion participants may propose ideas that do not answer some previously posed question.

Specifically, we observed two cases: often, someone first proposes a solution, which can then be understood as posing a new question about design (i.e., whether their particular solution be accepted). For example, in line 16 of listing 2 @jeremy is *asserting* that the email bodies should be decoded, a position rejected by @tute in line 25. The implication is that in line 16, @jeremy *simultaneously* poses a Q (albeit in the form of an assertion) and proposes a C to resolve it, which is later rejected by @tute. In other cases, an element could be a candidate solution to a previous question, while continuing the discussion with a further question at the same time. While our approach would annotate them as both Qs and Cs, these cases seems to indicate that our initial division of elements between Qs and Cs may have been too naive.

With respect to RQ2 (the relationship between design elements) we observe that developer comments are almost always related to

each other, effectively defining a structure for the discussion. These relationships can be logical and rhetorical. On the one hand, the logical structure of a discussion captures the relation between Qs and Cs, as well as the interdependencies between design concerns (either Qs or Cs), such as Q2 being dependent on C1 :: Q1 in line 10 of 2. On the other hand, the rhetorical structure of the discussion captures the argumentation that takes place between developers, including support/reject elements and expositions of rationale. Dependencies between rhetorical elements can be complex, such as conditionally supporting a C, or critiquing someone else's rationale for supporting or rejecting a position.

4 FUTURE WORK

Our analyses of the pull requests indicates a number of avenues worthy of further investigation.

Argumentative Structure. Extracting the decided upon design elements out of a developer discussion, either by a human or a software tool, requires an understanding of the argumentative structure of a discussions. Some parts of the structure of a discussion are relatively clear. For example, we found general patterns of coreferential *questions* and *candidates*. However, there is also a much less ordered side to design discussions. We found *moves* to *support* and *reject* sometimes appear to reference multiple *candidates*, often lack *justification*, and can be counter-rejections to another *reject*.

Developing a model of this less ordered structure to a design discussion is necessary to be able to gain a full understanding of the design elements. The structure of *questions* and *candidates* alone does not capture the full picture of the discussion; a list of possible solutions alone does not predict which solutions are to be accepted. Further work on design discussions will need to not only capture the state of working solutions, but how these are related and argued for and against by the participants in the discussion.

Visible design space. We labelled the discussions post-hoc. Any on-line automated analysis approach would need to consider that any design element can only be backwards referenced by later posts, since design elements introduced in the future may be hard to foresee. We define this concept as the Visible Design Space (VDS) of the discussion. The VDS contains all design elements thus far introduced in the discussion. Comments that introduce new elements can be seen as expanding the VDS of the discussion, until the discussion has terminated and the space contains all the elements. However, the VDS is hardly well defined. If we merely defined it as all the elements introduced previously, this may oversimplify the potential nuances of the actual design space as the actors interact with it. For example, should all elements be considered equal? Elements introduced by major actors, such as core developers for a project, may play a prominent role in the VDS, while *questions* never followed by a *candidate* may not need to be visible in practice.

As tools are developed to help synthesize and make use of design information in discussions, different considerations of VDS will need to be explored to understand the effects on minformation extracted.

5 SUMMARY

The importance of design within a software development project is well understood. However, despite the effect design has on many system properties, it is still unclear what information constitutes design and how to best capture and communicate it. As a result, a significant amount of design information remains implicit in projects. Written developer discussions have been identified as a source of such information. In this paper, we report on an exploratory investigation into the form and content of such design discussions. We have studied three pull requests from the Rails project, showing how to capture the logical and rhetorical structure of design discussions. This dataset provides a starting point for the community to build upon in order to determine how best to identify, extract, synthesize and represent design information captured in written developer discussions. Understanding how design information appears in these kinds of interactions provides an opportunity to build tools to help software developers model, comprehend, document and preserve crucial design information.

ACKNOWLEDGMENTS

We thank Georgios Gousios for providing us with the corpus of failed pull requests from GitHub. We also acknowledge an NSERC Discovery Grant that was used to fund this research.

REFERENCES

[1] K. Allen, G. Carenini, and R. T Ng. 2014. Detecting Disagreement in Conversations using Pseudo-Monologic Rhetorical Structure.. In *EMNLP*. 1169–1180.
[2] E. Black, P. McBurney, and S. Zschaler. 2014. Towards Agent Dialogue as a Tool for Capturing Software Design Discussions. In *2nd Int'l Workshop on Theory and Applications of Formal Argumentation*. 95–110.
[3] J. Brunet, G. Murphy, R. Terra, J. Figueiredo, and D. Serey. 2014. Do developers discuss design?. In *Proc. of the 11th Working Conf. on Mining Software Repositories*. 340–343.
[4] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth. 2005. Hipikat: a project memory for software development. *IEEE Trans. on Soft. Eng* 31, 6 (2005), 446–465.
[5] R. P. Fawcett and B. L. Davies. 1992. *Monologue as a turn in dialogue: Towards an integration of exchange structure and rhetorical structure theory*. 151–166.
[6] G. Gousios and D. Spinellis. 2012. GHTorrent: GitHub's Data from a Firehose. In *Proc. of the 9th IEEE Working Conf. on Mining Software Repositories*. 12–21.
[7] G. Gousios, A. Zaidman, M. Storey, and A. van Deursen. 2015. Work Practices and Challenges in Pull-based Development: The Integrator's Perspective. In *Proc. of the 37th Int'l Conf. on Soft. Eng*. 358–368.
[8] S. Joty and E. Hoque. 2016. Speech act modeling of written asynchronous conversations with task-specific embeddings and conditional structured models. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics*. 7–12.
[9] E. Knauss, D. Damian, J. Cleland-Huang, and R. Helms. 2015. Patterns of continuous requirements clarification. *Requirements Engineering* 20, 4 (2015), 383–403.
[10] D. Marcu. 1997. The Rhetorical Parsing of Natural Language Texts. In *Proc. of the 8th Conf. on European Chapter of the Assoc. for Comp. Linguistics (EACL '97)*. 96–103.
[11] A. S. Mashiyat, M. Famelis, R. Salay, and M. Chechik. 2014. Using Developer Conversations to Resolve Uncertainty in Software Development: A Position Paper. In *Proc. of the 4th Int'l Workshop on Recommendation Systems for Soft. Eng*. 1–5.
[12] R. M. Palau and F. Moens. 2009. Argumentation Mining: The Detection, Classification and Structure of Arguments in Text. In *Proc. of the 12th Int'l Conf. on Artificial Intelligence and Law*. 98–107.
[13] D. E. Perry and A. L. Wolf. 1992. Foundations for the Study of Software Architecture. *SIGSOFT Softw. Eng. Notes* 17, 4 (1992), 40–52.
[14] M. Petre and R. Van Der Hoek. 2013. *Software Designers in Action: A Human-Centric Look at Design Work*. Chapman and Hall/CRC.
[15] P. Rodeghero, S. Jiang, A. Armaly, and C. McMillan. 2017. Detecting User Story Information in Developer-Client Conversations to Generate Extractive Summaries. In *Proc. of Int'l Conf. on Soft. Eng*. 49–59.
[16] Maite T. and William C. M. 2006. Applications of Rhetorical Structure Theory. *Discourse Studies* 8, 4 (2006), 567–588.
[17] J. Tsay, L. Dabbish, and J. Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *Prof. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Soft. Eng*. 144–154.
[18] D. Twitchell, M. Adkins, J. F. Nunamaker, and J. K Burgoon. 2004. Using speech act theory to model conversations for automated classification and retrieval. In *Proc. of the Int'l Working Conf. on Language Action Perspective Communication Modelling*. 121–130.