Finding and Using Design Information in Discussions

Giovanni Viviani
Department of Computer Science
University of British Columbia
Vancouver BC, Canada
vivianig@cs.ubc.ca

***Abstract*—A software system's design determines many of its properties. An understanding of design is needed to maintain system properties as changes to the system are made. When developers lose track of the overall design, the system may not conform to its intended properties. I hypothesize that it is possible to solve the problems of design evaporation and erosion by recovering design information from written developer discussions and by leveraging the recovered information to help developers make better decisions. As part of investigating this hypothesis, I have built an automated classifier that is able to locate design information in discussions, at the paragraph level, by learning from manual annotations of discussions extracted from open source pull requests. I next plan to extract and represent the design information from the identified paragraphs and to show the usefulness of the information by creating tools to help notify developers of design information relevant to a task at hand.**

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

I. Introduction

A software system's design is an important component of any project that defines how the system should be built. The design defines certain characteristics of the system that developers need to uphold to guarantee the final product will be what is intended. Unfortunately, it is not always easy to maintain design information or to make sure that developers are aware of the overall design, potentially causing severe repercussions on the project [6].

During the development process, developers continuously update project artifacts, mostly in the form of source code. Design information is created as part of this process and becomes embedded in these artifacts. Design can also be created during other activities than coding, such as through discussions. This process is depicted in Figure 1.

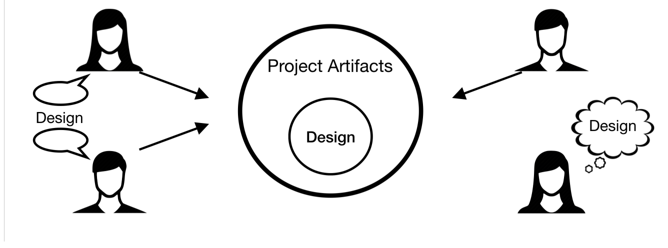

Fig. 1: Depiction of the development process. Developers update project artifacts, stored in a repository. At the same time, developers can also modify the design. Design can also be created or updated during conversation between developers.

Software projects often contain multiple development teams, each composed of multiple developers. The involvement of many developers, combined with the process described above, causes the system design knowledge to become fragmented amongst multiple developers, leading to a situation in which developers do not know who holds the knowledge about the design of a particular part of the system. This can become a major problem since developers might not know how to access the design of an aspect of a system: over the long term this can cause the software to drift away from the original intended product due to the developers lacking access to the information [5].

If we could recover design from project artifacts and provide this information to developers, we could help them make better decisions, preventing the system from drifting away from the intended result. Existing techniques of design recovery have attempted to recover the lost information from code artifacts [2]–[4]. These approaches tend to focus on understanding *how* the software works and help a software developer change the system, but they do not provide information on the reasoning behind the design. My research focuses on recovering design information from discussions between developers to understand not only *how* the software works, but also *why* certain choices were made.

I hypothesize that *it is possible to recover information regarding the design of a software system, and the motivation behind the design, by mining written discussions between developers that have occurred as part of their workflow. Having recovered this information, it is possible to provide developers with tools that bring forward design information appropriate to a task, enabling developers to make better decisions based on up-to-date information about the system's design.*

To validate this hypothesis, I pose the following research questions:

RQ1 Can we localize which parts of a written discussion between developers contain design information?

RQ2 Can we extract and represent design information contained in discussions?

RQ3 Can we help developers make better choices and avoid errors by providing them access to recovered design information?

The following sections will explore the motivation for each of these research questions, together with the plans for investigating and evaluating them.

II. RQ1: Localizing Design Information

To begin extracting design information from discussions, we first need to be able to locate the design information within a discussion. The goal of **RQ1** is to localize where the design information is in discussion between developers. We opted to focus on discussions from pull requests because they have been shown to often contain design information [1] [7].

We created a dataset of discussion fragments containing design information. We extracted and annotated 10,790 paragraphs whereter they are, or are not, a design points. These paragraphs came from 34 pull requests chosen from 3 major oepn source projects: Node.js, Rails and Rust. Of these paragraphs, 2,378 turned out to contain a design point. To make this approach more scalable, we buit a classifer that, once trained, is able to automatically localize design points.

III. RQ2: Extracting Design from Discussions

The goal of RQ2 is to be able to recovert the pertinent design information from the discussion segments. To represent the design information, I intend to determine important attributes about design that I can then represent for particular design points, such as the kind of structure intended by the design point or the presence of code elements in the paragraph.

The combination of these attributes creates an abstract representation of the design information that we can recover. Design points can then be processed and represented. The relationships between the design points can then be investigated to understand the motivations behind the changes.

IV. RQ3: Make design information easily available

The goal of **RQ3** is to use the abstract representation of design information proposed in the previous section to develop tools to help developers in maintaining their awareness of the design and make design that conform to the design of the the system.

As an example, consider a tool which monitors a developers' work and infers an abstract structure of the modifications that he is making. This can then be used to notify of information which he may not be aware of. Another approach consist of storing the representation of design information and allow developers to navigate and search it.

V. Expected contribution

The goal of my PhD thesis is to both enhance design recovery techniques and to develop new ways to present it back to developers. The expected contributions of this project can be summarized as follows:

1) Introduce the concept of a Design Point, as a way to facilitate the capture of design information appearing in written developers discussion.
2) Provide a curated database of paragraphs containing design information on which a supervised learner can be trained.
3) Build a classifier able to localize the presence of design information in piece of discussion.
4) Develop a process to automatically extract and represent design information from discussions.
5) Develop a tool to provide developers with feedback regarding the design of the system and show that it helps them make better decision.

Acknowledgment

I wish to acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] 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*, MSR 2014, pages 340–343, New York, NY, USA, 2014. ACM.

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

[3] 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 *Proceedings of the 2000 International Conference on Software Engineering*, pages 439–448. ACM, 2000.

[4] 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. ACM, 1988.

[5] David Lorge Parnas. Software aging. In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pages 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[6] 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. ACM, 2016.

[7] 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. ACM, 2014.