Changeset Based Developer Communication to Detect Software Failures

Braden Simpson
Department of Computer Science, University of Victoria
Victoria, BC, Canada
braden@uvic.ca

***Abstract*—As software systems get more complex, the companies developing them consist of larger teams and therefore results in more complex communication artifacts. As these software systems grow, so does the impact of every action to the product. To prevent software failure created by this growth and complexity, companies need to find more efficient and effective ways to communicate. The method used in this paper presents developer communication in the form of social networks of which have properties that can be used to detect software failures.**

I. Introduction

Miscommunication among contributors during software development can lead to software failures, causing billions of lost dollars or even lethal accidents [1]. Larger products and teams are inherently complex, making achieving high quality more difficult. To ease this difficulty developers use collaborative development environments (CDE) such as BugZilla, Jira etc. These systems provide bug reports and other communication artifacts which establish a medium for better contributor communication. For this study, we define a *failure-inducing* communication pattern to be a pair of contributors that have a high chance of creating a *fix-inducing change*. In addition, we define a *changeset* as a set of one or more changes to resources by one developer. After building social networks, we study the participants and resultant snapshot of the project at that changeset level. We do so to determine if there are any *failure-inducing* patterns of communication present.

In order to create the required networks, we link changesets to communication, similar to Wolf et. al where they link developer networks to build failures [2]. Other researchers such as Bettenburg et al. have done work to link communication artifacts to source code [3]. As well, Sliwerski et al. have tried to link bug-fix changesets to their corresponding bug reports [4]. These studies can assist us in understanding the direct impact of the communication, and all the effected artifacts.

We construct social networks on a changeset level over a project's lifetime as described in Section II-B; then analyze the history of networks for potential patterns, defined as an unordered pair of contributors of communication. This drives the question *"Can we identify failure-inducing communication patterns in social networks based on changesets and communications artifacts?"*

This paper details our approach towards linking social networks to changes, and evaluating their usability for the detection of potentially failure-inducing communication patterns. These include certain problematic groups of participants, or problem areas of a project that require more communication that others due to a more complex technical background.

II. Approach

Our goal is to detect potentially failure-inducing patterns in social networks. In the following sections we describe our methods used to detect these patterns by examining changeset based social networks, resulting in a repository and tool independent process. The social networks are created by using contributors as nodes and communication as edges, constructed using the parsed social data in Section II-A2.

A. Linking Social Data to Changesets

Our goal is to isolate all the communication artifacts that are related to each changeset.

1) Analyzing Changeset Message: First, the changeset comments are checked for communication related identifiers, usually unique to the specific project. When a strict match to this identifier is found, we construct a link between this changeset and the resulting issue (bug), similar to Sliwerski et al. [4].

2) Parsing Communication Artifacts: Second, after parsing changeset comments, another method is used to link the communication artifacts to source files. A set of communication artifacts are chosen that are within a certain project-specific date-range from the selected changeset. Next, the communication artifacts are parsed for *Stacktraces, Patches*,

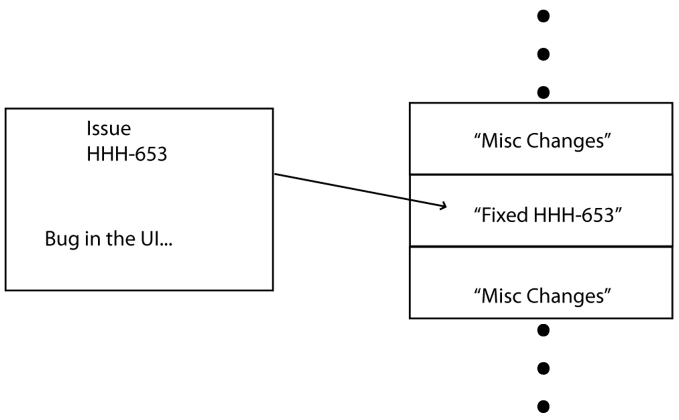

Fig. 1. The unique issue identifier found in a changeset comment.

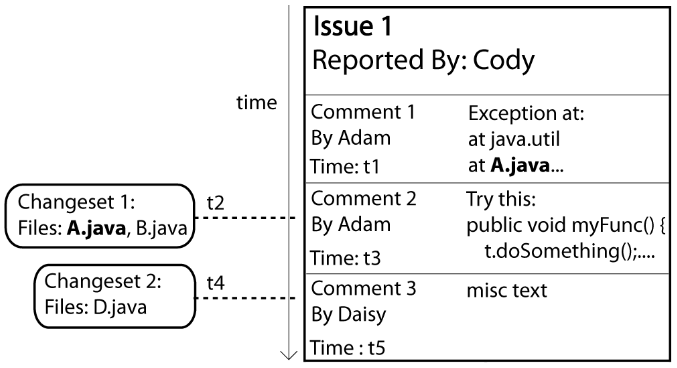

Fig. 2. Timeline of issue, comments and changesets.

and *Code Snippets* using similar methods to Bettenburg et al. [3]. Finally, a link is created with a confidence based on the percentage of matched content. This confidence is calculated by taking the ratio of matched code from the code snippet versus the total size of the code snippet. Once the links to files are created, the changesets that alter those files are linked to the communication artifacts.

In Figure 2, for example, there are links created between *changeset 1* and *Comment 1*, since there is a *Stacktrace* in *Comment 1* that contains a match to the filename *'A.java'* in *changeset 1*. This will create a link between artifact and changeset, allowing us to determine the impact of the social network created by these communication artifacts surrounding a changeset.

B. Creating Social Networks

Since our goal is to create the networks that are related to a single changeset, our strategy to construct the networks is similar to Wolf et al [2]. The approach taken is repository and tool-independent, and results in networks composed of nodes and edges where the nodes are unique people involved in the communication and the edges are a representation of the communication between those people at the given changeset. We create these edges like *(Adam, Bart)* where *Adam, Bart* are given by the creators of the artifacts that were linked.

C. Determining Fix-Inducing Changes

We qualify each changeset as either a pass or fail, pass meaning that it was a changeset that did not introduce a bug, and conversely a fail is a changeset that introduced a bug. We used the method from Sliwerski et al [4] to infer fix-inducing changes.

III. Results

We used Hibernate-ORM[1] as a case study. It is hosted on GitHub, and managed by Jira[2]. Our project requires that both the repository information and the CDE data is able to be mined in order to create the social networks.

[1]An open source Java library that is used for creating object-relational mappings for data-modeling, http://hibernate.org

TABLE I
Five selected changeset patterns, ordered by Failure Index.

Pattern	Passed	Failed	FI	P-Value
(Bob, Drew)	0	3	1.0000	0.04910
(Carl, Drew)	0	3	1.0000	0.04910
(Alex, Frank)	19	25	0.6947	0.006908
(Frank, Elaine)	32	35	0.6541	0.009886
(Alex, Elaine)	63	57	0.6101	0.01534

We classify a failure-inducing pattern by running the statistics through the following formula to calculate the *Failure Index*, FI of a pattern.

$$\text{FI} = \frac{\text{pattern}_{failed}/\text{total}_{failed}}{\text{pattern}_{failed}/\text{total}_{failed} + \text{pattern}_{success}/\text{total}_{success}} \tag{1}$$

Before calculating the FI of a pattern, we perform the Fisher Exact Value test to filter out insignificant patterns by selecting only P-Values smaller than *0.05*, then, the significant patterns found with the FI values greater than *0.5* are classified as failure-inducing patterns. In Table I, we select five failure-inducing patterns, and for those patterns show the number of passing changesets, the number of failing changesets, and the resulting FI.

For the Hibernate-ORM project, which had a total of 3968 changesets, we were able to link 3076 of them to communications artifacts. From these links, we could create 4537 distinct patterns of communication from the 5040 distinct contributors to the project. Of those patterns, 178 were significant, and 55 could be identified as failure-inducing.

IV. Limitations

One limitation of our study was that we used a large open source project, which means the number of contributors will be a lot larger, but the contributions may be smaller than in closed source projects. This is reflected heavily in Hibernate, as we had over 5000 contributors, and this resulted in a large amount of insignificant patterns.

V. Conclusion and Future Work

There are several other studies that suggest that better communication practices result in a more successful build rate [1][5][2][6], however they perform these studies at a build level, rather than changeset. Our study suggests that each changeset can be individually analyzed and that failure-inducing patterns of communication (either present or absent) can be extracted. These patterns can then be used to offer potential recommendations when they are found. In the future we want to perform another study on a closed source project with a much smaller group of contributors, as the patterns would be more significant, and allow for more analysis. As well, we plan on evaluating the congruence between technical networks and the social networks created in this study to identify other possible failure-inducing patterns.

References

[1] T. Wolf, A. Schröter, D. Damian, and T. Nguyen, “Predicting Build Failures Using Social Network Analysis on Developer Communication,” in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070503

[2] T. Wolf, A. Schröter, D. Damian, L. Panjer, and T. Nguyen, “Mining task-based social networks to explore collaboration in software teams,” *Software, IEEE*, vol. 26, no. 1, pp. 58 –66, jan.-feb. 2009.

[3] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, “Extracting Structural Information from Bug Reports,” in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, ser. MSR ’08. New York, NY, USA: ACM, 2008, pp. 27–30. [Online]. Available: http://doi.acm.org/10.1145/1370750.1370757

[4] J. Śliwerski, T. Zimmermann, and A. Zeller, “When Do Changes Induce Fixes?” in *Proceedings of the 2005 International Workshop on Mining Software Repositories*, ser. MSR ’05. New York, NY, USA: ACM, 2005, pp. 1–5. [Online]. Available: http://doi.acm.org/10.1145/1082983.1083147

[5] A. Schröter, “Predicting Build Outcome with Developer Interaction in Jazz,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ser. ICSE ’10. New York, NY, USA: ACM, 2010, pp. 511–512. [Online]. Available: http://doi.acm.org/10.1145/1810295.1810456

[6] T. Zimmermann and N. Nagappan, “Predicting defects Using Network Analysis on Dependency Graphs,” in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE ’08. New York, NY, USA: ACM, 2008, pp. 531–540. [Online]. Available: http://doi.acm.org/10.1145/1368088.1368161