CS 846 Course Project Proposal

Xinkai Li
x638li@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

Xueyao Yu
x224yu@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

Zhuangfei Hu
z239hu@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

ACM Reference Format:
Xinkai Li, Xueyao Yu, and Zhuangfei Hu. 2021. CS 846 Course Project Proposal. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

It is widely acknowledged that bug detection and maintenance play a critical role in software engineering. There have been abundant previous works focusing on the factors which could affect bug frequency [1, 3–5]. However, common bugs could be complex in nature, thus introducing difficulties for identifying the bug-inducing factors. Karampatsis, R.M. and Sutton, C [2] first introduced the concept of simple stupid bugs (SStuBs), providing a convenient handle for preliminary study on bug-inducing factors. The ManySStuBs4J dataset [2] collects the commits which contain single-statement bugs from popular Java Maven projects, and classifies them according to common bug patterns in practice. Based on this dataset, we investigate and identify factors that contribute to the appearance of bugs. We assume that the characteristics of the project, the contributors, the commits, and the individual source file are primary factors affecting the SStuBs, and try to determine the effect of each factor by constructing a statistical model with respect to SStuBs frequency. Furthermore, based on these factors, we try to build a prediction model to estimate SStuBs frequency of given commits. We argue that this model could help developers estimate the risk level of given commits and workload of the code review process, thus bringing positive impacts on the quality and work efficiency of the projects.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 Datasets and Tools

Our project is based on the ManySStuBs4J dataset[2] generated by Karampatsis et al., which contains detailed information of one-line bugs from popular Java Maven projects, including bug location, commit identifier and corresponding source code. We also utilize the GitHub platform as a complement for the dataset. Tracing the commit identifiers provided in the ManySStuBs4J dataset, we retrieve commit-related and project-related information, including source files and corresponding contributors' profiles, with GitHub APIs. Machine learning toolkits like PyTorch might be used to assist data analysis and model construction.

3 Research Questions

RQ1. Is the occurrence of SStuBs affected by certain characteristics of commit-related information?

We assume that the frequency of SStuBs is affected by commit-related information, and hope to verify our assumption. We extract commit-related information from the dataset, including timestamps, contributor and project information, and label them as possible factors. We then observe the statistics of each label and determine the most probable factors affecting SStuBs frequency, including four different groups:

- *Project features:* project size, age and reputation.
- *Contributor features:* number of authors and reviewers for the commit, as well as their corresponding experience and reputation.
- *Commit features:* commit time, LOC updated.
- *Source file features:* source file's size and revision times.

We hope to examine each of these factors via statistical model, e.g. linear regression model, estimate the corresponding coefficients of the factors, and provide stronger support for our conclusion by calculating significance levels.

RQ2. Is it feasible to estimate the frequency of SStuBs in given commits via characteristics of commit-related information?

This research question is based on the previous question. If positive results are obtained in **RQ1**, we hope to establish a prediction model for SStuBs' frequency of individual commits based on the verified factors in **RQ1** via statistical method or machine learning technique (e.g. artificial neural network).

Phase	Schedule
Determine explanatory factors	March 1 - March 8
Data extraction	March 9 - March 18
Data analysis	March 19 - March 25
Model construction	March 26 - April 1
Paper submission	April 2 - April 7

Table 1. Schedule of 5 milestone phases

4 Milestones

As Table 1 shows, our milestones consist of 5 phases. The details of each phase are illustrated as below:

Phase 1. Determine explanatory factors

Numerous studies have been conducted in the field of bug prediction, proposing a variety of fault-proneness factors [1, 3–5]. Based on previous work and the provided dataset, we list groups of factors that might affect the occurrence of SStuBs, including project, contributors, commit, as well as individual source file. (March 1 - March 8)

Phase 2. Data extraction

We extend the existing SStuBs dataset by using the commit ID and project name to retrieve the information regarding the research factors from GitHub REST API such as project metadata, contributor' expertise and commit data, etc. This process also includes data pre-processing, which cleanups the data by applying some filters to eliminate the outliers and normalizes the data retrieved from GitHub API so that it can be meaningfully interpreted and directly used for further manipulation.

Phase 3. Data analysis

During this phase, we verify the proposed factors by qualitatively looking into several bugs and explaining the cause of the occurrence, and quantitatively performing statistical analysis such as linear regression. (March 19 - March 25)

Phase 4. Model construction

If we can validate a set of fault-proneness factors from Phase 3, we will build a bug prediction model that can estimate the bug density of a given commit.

Phase 5. Paper submission

The study will be academically recorded and organized in a paper.

5 Threats to validity

Construct validity: As the Java projects studied in [2] are selected by popularity, it is reasonable to assume that they are well designed and maintained by experienced contributors following rigorous development process, yielding relatively low frequency of SStuBs. Thus, the generalizability of the conclusions drawn from the ManySStuBs4J dataset could be questionable. To solve this problem, one possible method is to split the original ManySStuBs4J dataset into disjoint training and testing subsets. Further study and tests on other Java projects of various sizes would also provide stronger promise for our results.

Classification error: The ManySStuBs4J dataset suffers from false-negative and false-positive problems. According to [2], the keyword filter for buggy commit reaches an accuracy of 94%, but there could still exist commits falsely classified as erroneous, or bugs that remain unnoticed. This could be a potential threat to the validity of our conclusions.

Internal validity: Our model does not include all possible factors affecting SStuBs frequency. Certain features or characteristics, such as coding style of developers, culture of development team and interaction between collaborators, are hard to quantify or retrieve and are thus excluded from our model. The effects of these features or characteristics could be studied qualitatively for further understanding of primary factors affecting SStuBs frequency.

Dataset Bias: As the ManySStuBs4J dataset is focused on commits labelled as buggy, the conclusions drawn from this dataset cannot be directly generalized to ordinary individual commits. Furthermore, not all buggy commits in the studied Java project are included in this dataset. Thus, we could only report positive risk levels for certain commits but not vice versa. One possible way to solve this problem is to investigate the SStuBs frequency in the provided Java projects while excluding the commits contained by the ManySStuBs4J dataset and try to identify characteristics of bug-free commits.

References

[1] Roberto Abreu and Rahul Premraj. 2009. How developer communication frequency relates to bug introducing changes. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*. 153–158.

[2] Rafael-Michael Karampatsis and Charles Sutton. 2020. How often do single-statement bugs occur? The ManySStuBs4J dataset. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 573–577.

[3] Shane McIntosh and Yasutaka Kamei. 2018. Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction. *IEEE Transactions on Software Engineering* 44, 5 (2018), 412–428.

[4] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. 2008. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on Software engineering*. 181–190.

[5] Kai Pan, Sunghun Kim, and E James Whitehead. 2009. Toward an understanding of bug fix patterns. *Empirical Software Engineering* 14, 3 (2009), 286–315.