CS846-001 Week 4 Reviews

Jun Qing Lim
University of Waterloo
jq3lim@uwaterloo.ca

1 No Issue Left Behind: Reducing Information Overload in Issue Tracking

1.1 Problem Being Solved

The paper addresses issue of information overload in software development due to large amount of issues that are difficult to keep track of, resulting in certain critical issues get neglected or missed. These issues are not just code reviews but also bug reports from emails, code management tools, release management tools, etc. The authors instead proposed a model of a customized issue-tracking dashboards that allow developers to customize their issues according to different priority and needs. It argues that it helps to improve the effectiveness and efficiency of issue tracking while developing developers' situational awareness of managing issues.

1.2 New Idea Proposed

The authors proposed a model of a customized issue-tracking dashboard tool, that on a high-level, contains the following features:

1. dashboard that provides developers with public watch lists, patch logs and an activity history feature helps in creating better awareness of issues that are in-progress
2. contains two main panels: **"Issues"** and **"Patches and Reviews"**
3. the `"Issues"` focuses on issues reported by a developer, whose bugs need to be fixed
4. the `"Patches and Reviews"` focuses on displaying list of patches, code reviewers, and current status of the patch

This proposed model is then further tested in an industrial-setting and it was discovered that developers were glad that most of the focused tasks get to be displayed easier with this prototype, and that it helped to improve the overall productivity of issues-resolution. They are also able to have a quick start on which issues to focus on each day, which saves issues browsing time.

1.3 Positive Points

Firstly, the paper addressed a problem in a typical software development issue tracking system which is relevant in current industrial context (and not outdated). Secondly, it proposed a new model of keeping track of issues with categorization of "Issues" and "Patches and Reviews", making bugs and issues to be easily sourced or queried, saving developers' time. Thirdly, the paper does not just propose a solution to a problem, but also verified the practicality and effectiveness of its solution through an industrial trial, this makes the ideas proposed to be practical in a real-world context.

1.4 Negative Points

Firstly, the new model proposed may not be generalized to other types of issue tracking systems in a different software development context (i.e., the authors created a model based on Mozilla bugs survey only and did not consider other possible context of issue tracking system). Secondly, there may be insufficient support on how practical the proposed model of issue tracking system, because there were limited industrial trial evaluation being conducted.

1.5 Future Work

One possible extension is conduct an extensive study of the impact of the model towards existing software development process, this could help us understand not just from the model's productivity and effectiveness point of view, but also from the standpoint of issues missed, workload affected, etc. The other extension is to evaluating the model in a much broader context, this could help understand the effectiveness and how generalizable the model is to other types of systems and context.

1.6 Rating

3 out of 5. The ideas proposed do not seem to have a large impacts in the real-world.

1.7 Discussion Points

- Are there any issue tracking systems currently which are similar to what this paper has proposed?
- What types of companies usually face information overload of issues? Can we identify any particular attributes that caused the problem?

2 Characterizing and Predicting Which Bugs Get Fixed: An Empirical Study of Microsoft Windows

2.1 Problem Being Solved

The paper presents the result of an empirical study to characterize factors that affect which bugs get fixed in a Microsoft Windows operating system. The authors used the factors discovered to build a statistical model to predict the probability of a new bug being fixed. It aims to use the outcome of the statistical model to help engineers to prioritize which bugs get fixed first, resulting in better use of engineers' time and resources.

2.2 New Idea Proposed

The authors proposed that engineers should focus on bugs that will actually get fixed and should not be spending unnecessary attention on redundant bugs. It identifies the following factors that affects if a bug will be fixed:

1. bug is opened by someone that has track record of successful bug fixing
2. bug report that is heavily edited by multiple engineers
3. bug re-assign to more people for bug fixing
4. bug re-opened le number of times (less negative effect)
5. bug assigned to team with similar geographical location

Using these factors and some mathematical formulas, the authors proposed a logistic regression model to predict the probability of an event occuring (if a bug will be fixed) with a use of various regression coefficients. These models are then tested with a performance of about 60-70% accuracy.

2.3 Positive Points

Firstly, the paper first presents the factors that affect if a bug is fixed, this provides useful information (even if the regression model has poor accuracy) as it gives insightful information to not just engineers making decision but also on a managerial-level on setting priorities for bugs. Secondly, the authors used real-world data to determine the regression coefficients during both training and evaluation phase, which gives a realistics picture of how the model would function in an actual practice. Thridly, the paper proposed a model that has a significantly high accuracy, this is useful for engineers because it allows the team to make informed decisions to prioritize the bugs to be fixed, allowing the team to reduce time spent on fixing bugs (that may not necessarily be fixed at the end of the process).

2.4 Negative Points

Firstly, the findings of the paper may be outdated as it is based on a study of Microsoft Windows Visa and Windows 7, which are not used commonly in the present years. Secondly, the prediction model is trained based on information that is associated upon the bugs are reported and that it is possible that some information may not yet be present at the start of bug-fixing process but rather arise during the process itself; as such, the model may not be a true representation of all kinds of bugs in the context of software development.

2.5 Future Work

One extension is that to conduct a study that is more relevant in the current years (2020s, with Microsoft Windows 10 and 11 and/or MacOS), this would allow us to better understand the applicability in the current context. Also, instead of training models with factors that affect if a bug is fixed, we could extend the factors to include element that affect if a bug is not fixed, allowing us to build a more inclusive model with higher accuracy. The other possibility is to we can expand such form of study to other domains (instead of specific only to operating systems), giving us a much comprehensive understanding on bugs-fixing in various domains.

2.6 Rating

2 out of 5. A little outdated in current context.

2.7 Discussion Points

- Is there any difference between bug-fixing in OS context and other context (such as mobile apps, systems development, etc.)?
- How helpful and useful will this be if teams in the software development field adopt practices of bugs-prediction before working on fixing a bug?
- Is there any specific field of bugs that are difficult to be predicted by the model?