Automatic Contextual Bug Report Modelling

Andrei-Mihai Nicolae (2147392)

1 Research Problem

Background

In recent years, software engineering has been shaping many industries that were previously considered unrelated to technology, ranging from healthcare to public transportation and education. As software has grown and inflicted itself into so many fields, it has inherently become more complex and it now involves teams of even hundreds of developers working remotely or in the same office on applications that can shape even the face of a nation, such as Estonia's data exchange layer called X-Road [8].

Therefore, having such a high number of developers working on complex applications, there has been a need for software engineers to plan, track and manage the workload for increased efficiency. Thus, issue tracking systems have been created, which usually incorporate multiple components into a single, unified interface. The main component, however, of any issue tracking system is called a *ticket*. A ticket is comprised of all the necessary information for either fixing a bug (i.e. specific malfunction of the application) or implementing a new feature (i.e. new functionality for the software). However, after an application is released and it starts to get used by actual end users, the most common ticket found in issue tracking systems is a bug report, stating what went wrong inside the application, how it can be reproduced, even possible locations in the source code where the malfunction might originate from.

Typically, the people who file the bug reports are actual end users who go onto the project's issue tracking system website and complete the necessary data required by that system. Once the bug report is filed in the tracking systen, triagers are responsible for assigning that ticket to developers to get fixed. Bettenburg et. al [4] have conducted interviews with developers and they have gathered valuable data related to what makes for a good bug report and what are the key characteristics that help them fix the bug as quickly as possible. One result presented by this research is that having stack traces (i.e. list of methods in the source code which were called when an exception was thrown) in the description field of the bug report helps developers fix the bug quicker. However, even with this information available, the fields displayed in the bug report for the end users to complete (e.g. summary of the bug, description, attachments) are standard across all the different issue tracking systems.

Having the optimal bug report model/design for end users to complete would be beneficial for both the software development team, as well as the overall costs incurred by the company producing the application. Therefore, the main goal of this proposed project is to create a tool that can automatically generate the optimal bug report model for any specific project taking into account the characteristics of that project or the team behind it (e.g. size of the development team, technologies used, mobile/desktop application). Even though there is state-of-the-art research in this field, such as the work of Zimmermann et al.[22], they only look into what can be done and propose only advice to issue tracking systems developers. However, our work goes one step further and we want to conduct the first research in this area that actually gives the community a tool which can automatically fix this issue.

Key Ideas in a Nutshell

"To automatically create a bug report model tailored specifically for a project, taking into account the whole context of the software as well as the developers behind it."

The goal of the project is to create a tool which, given as input details regarding the project and the developers working on it, can automatically generate a bug report model that will increase the efficiency of the team as well as reduce the costs of the organization. Our proposed approach is to, firstly, determine what components determine the time-to-fix period (i.e. time spent between opening a ticket and closing it) for any given project using both statistical analysis as well as the data already collected in a publicly available appendix [5]. Then, in order to make our tool take into account all the complex characteristics of developing software, we want to both look at how the team of developers is managed and how the work is distributed, as well as how the developers interact with the end users reporting the bugs.

The *main benefit* of our project is that, through fixing a still unresolved issue in the software engineering field, it will improve three key aspects:

- reduce the overall costs of the organization developing the software;
- improve the time-to-fix window for all bug reports, thus reducing maintenance effort;
- reduce the communication friction between end users and developers because having a tailored bug report model will not require engineers constantly wasting time on asking for more details from the users reporting the bug.

Example: The UK government's Health Digital Services department (as well as many other departments) is using Jira as an issue tracking system [14]. If our proposed application would be used for their project, the developers could streamline the bug reports in a timely manner and increase their productivity, while also decreasing the overall costs for the government.

Objectives

Impact Objectives

- To increase the productivity of software developers in any team by giving them bug reports tailored for the project's specific characteristics.
- To increase the productivity of managers and triagers as they would spend less time on managing the bug reports filed on the issue tracking system.
- To decrease the overall costs of the organization financing the software project.
- To decrease the level of communication friction between the developers and the users filing the bug reports.

Supporting Objectives

- To present a novel application in the field to the whole research community and help advance progress on fixing flaws in major issue tracking systems.
- Through making the tool open source, an objective would be to involve developers from any industry/academic institution contribute to the advancement of this technique.

State of the Art

Improving Issue Tracking Systems The studies of Just et al. [16] and Zimmermann et al. [22] have tried to improve the software development process through analyzing what are the flaws of issue tracking systems and how we can fix them.

First of all, in the research conducted by Just et al., they asked a large number of developers about what can be improved about issue tracking systems and ranked the most common flaws through card sort. They propose a couple of fixes, among which there is better tool support (e.g. special UI trackers, internationalization), engage bug reporters in conversations, removing bug report duplicates, as well as improving the search functionality of issue tracking systems.

On the other hand, Zimmermann et. al [22] propose four areas of improvement: tool-centric, information-centric, process-centric and user-centric. The authors of the study believe that in order to improve these areas, we need to ask end users better questions in the bug reports. The solution proposed by the study is the one we want to tackle in our research and the methodology presented in this paper proved very valuable in designing our own experiments and evaluation.

Improving User-Developer Communication Breu et. al [6] propose a rather different approach in their study. They believe that reducing friction between developers and end users filing the bug reports would be of great benefit to the software development process. They revealed that, because bug reports are not optimized and tailored to the specifics of the project, the users are not providing the necessary details to the developers, thus the increased need of engineers continuously discussing with their end users what are the steps to reproduce the bug, what platform was used when the bug occured etc. This study showed again that there is a great need from the community for the tool we are developing which would solve these flaws in the bug reports.

Another study in this area that looked into a more specific scenario is the one conducted by Ko et. al [19]. The authors investigated how projects would benefit if they centered their bug report design towards more experienced bug reporters, also known as *power users*. They show in the study that if issue tracking systems would be more focused on them rather than regular users, which file a small number of reports overall, the software development process would be greatly improved.

Personalizing Issue Tracking Systems In this area, there are two main studies which propose both advice and prototypes - the research of Baysal et al. [2] as well as the study conducted by Kononenko et. al [20]. The research of Baysal et. al [2] proposes a couple of features that could be incorporated into issue tracking systems that would make them more personalized for the developers, such as private watchers list (i.e. each developer could privately watch issues without necessarily being CCed on every comment or change to the ticket), self tracking (i.e. ability of a developer to track his own changes and actions for better self management), a patch log, as well as the ability to assign patch reviewers. These are valuable additions to a issue tracking system that we believe could help improve the development process.

Complementing this research, the study from Kononenko et al. [20] looks into enhancing developer awareness by proposing a prototype of a issue tracking system where developers would have their own tailored dashboard to track and manage bug reports in the project. They admitted some limitations such as the slowness of the application when there were many queries coming from developers to the servers, but they want to improve the prototype by adding ElasticSearch [7] for running all search queries.

Simplifying Bug Reports Herraiz et al. [15] have investigated bug reports in the large database of the Eclipse project. Their research concludes that the standard bug reports from Bugzilla [11], which is a popular issue tracking system, are far too complex and they provide too many fields with too many options to be completed by end users, which was found to drive them away from the

project and abandoning filing bug reports anymore.

Another study that looked into reducing information needs in issue tracking is the one from Baysal et. al [3]. The authors propose that situational awareness, task support and expresiveness are key components in the software development process. They conducted a qualitative study where they interviewed a number of software engineers and then they proposed a model for bug reports that would facilitate a better functioning for task planning and management.

Covering all this state-of-the-art research, as well as numerous other studies in the field, there is none that presents a tool which can automatically create bug report models tailored specifically for any given project.

Our Solution

In order to tackle automatic contextual bug report modelling, our solution is to first create a data mining tool using the recommendations given in the research conducted by Kalliamvakou et. al [17]. We want to scrape bug reports from popular open source repositories and analyze their components, as well as infer common patterns. We believe that targeting large databases of bug reports, such as the ones from the Mozilla project or the Eclipse project, will help us gather enough data to make our tool accurate. Moreover, we want to also use the information available in the appendix created by Breu et. al [5] so that we can infer components and possible options available in bug report models from issue tracking systems. Other data that is available to us is also the collection of statistics on bug reports - for example, we can see how many people successfully managed to file bug reports, how many dropped throughout the process etc. We would like to also store this data inside our database for analysis.

Then, after scraping all the data and having it available in a database, the next step is to measure effectiveness of these bug report models so that we can infer what components or differences in design make the model be more beneficial for the developers. Using the techniques presented in the works of Bettenburg et. al [4] and Aranda et. al [1], we will be able to measure the model's effectiveness in the context of the software development process. In order to measure effectiveness, we are looking at a couple of attributes:

- ratio of people who successfully managed to file bug reports;
- length of the conversation corresponding to the ticket (i.e. number of comments needed to clarify the issue to the developers), as well as running sentiment analysis on the conversation to infer whether the discussion had a negative or a positive tone;
- time period between opening and closing a ticket, taking into account the difficulty of that ticket.

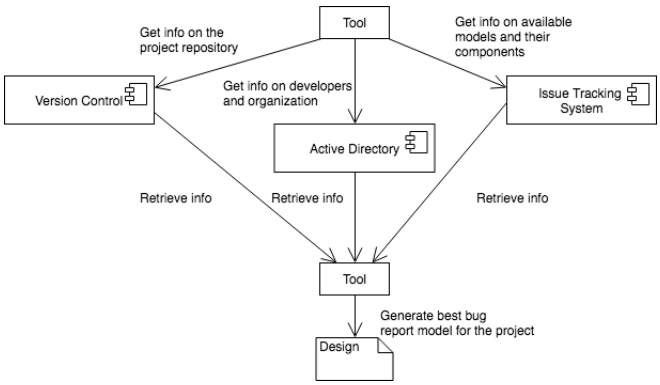

Figure 1: Flow of the application

In order to investigate the conversational characteristics of the discussions surrounding bug reports, we want to use state-of-the-art sentiment analysis techniques. dos Santos et. al [9] looked into how deep convolutional neural networks can help run sentiment analysis of short text, which is suitable for our research as the comment body is rather short, the limit being usually imposed by the issue tracking system. Moreover, another technique which we will try in our project is the one proposed in the research of Khan et. al [18] where they show how lexicon-based and learning-based techniques can infer the sentiment from tweets. In terms of the platform of choice, we decided on using the sentiment analysis APIs available from Google Cloud Platform [13].

After running our analysis on bug report models and their effectiveness, the next step would be to start developing the tool which will automatically generate the best bug report model for the project. Firstly, we connect our application to the version control in order to obtain statistics regarding both the software project as well as the work flows of the developers. Then, we also hook the tool to the active directory in order to infer the size of the team, how it is distributed geographically, typical working hours, languages used etc. Afterwards, we get data from the issue tracking system regarding what type of system it is, what components and options are available in the bug report model and what can the end user reporting the bug see or edit. Finally, we run our tool based on all the data availabe and retrieve the best possible design for the bug report, tailored specifically for the project.

Figure 1 shows the whole flow that we want to apply to our application.

As the development of the tool is ongoing, we would like to run the evaluation process in parallel. We have already discussed with the Mozilla Foundation as well as with the Eclipse project and they have agreed to let us run the tool once it is available. Firstly, we want to conduct interviews with managers, triagers and developers in order to test the validity of the app and if it actually improves the work flow of the project. Then, we also want to track the bug reporters' behavior and see how many abandon reporting the bug, as well as what their feed-

back is on the new format.

Finally, we want to publish our application in the open source world as we believe it would benefit everyone to use it. Moreover, the application could be greatly improved by having a large number of experienced developers contributing to it.

Example: Revisiting the Health Digital Services example, they are using Jira as their issue tracking system. Usinng our tool, we could get all the necessary information to capture the context and then provide them with the best bug report model so that the software development process is improved.

Innovative Aspects

As we have observed while researching the field, there is no tool that would provide the same functionality as ours will. Therefore, the biggest innovation of our study is the application itself which can be used in any project and any issue trackins system.

Moreover, another innovation that we will bring is that we want to make our tool open source for the community to be able to contribute. Even though there are other projects who have chosen the same approach, the number is relatively low - Sonnenburg et. al [21] conducted a research where they concluded that machine learning, one of the most popular areas in the computer science field, suffers from not having many open source tools available to the community.

Another innovative aspect of our application will be that it will use state-of-the-art sentiment analysis on the conversations surrounding the bug reports. There are not many papers in the field who have investigated the conversational aspects of bug reports, thus our research will contribute valuable findings to the community regarding how much does a positive/negative tone in a list of comments affect the lifetime of a bug report.

2 Methodology

Inspection and analysis of bug report models for popular projects (WP1)

We will first select a handful among the most popular open source projects to scrape. We have decided that the first two will be the Mozilla and Eclipse projects which also have their issue tracking systems available to everyone. We will use state-of-the-art techniques to mine them, such as the ones proposed by Kalliamvakou et. al [17] and German et. al [12]. After scraping the issue tracking systems of these projects, we will then store everything in a database for easy access and ability to query.

Then, we will categorize different bug report models based on their components and how they differ. Also, we will look at the different designs between, for example, Jira and Bugzilla. Moreover, we want to scrape the project's specific characteristics, such as:

- number of developers and their organization into teams;
- technologies used;
- geographical and demographics attributes (e.g. geographical position of the developers, languages used);
- typical work schedule and work flow of the developers.

Deliverables: data mining tool capable of scraping multiple components, such as active directories, version controlled repositories and issue tracking systems; database of bug report models and their components.

Measure effectiveness for different bug report models (WP2)

Having the data available from WP1, we will investigate how effective are the bug report models for their project.

As issue tracking systems make available bug report drafts and statistics regarding how many people file bug reports or how many of them decide to abandon throughout the process, we can infer how effective are these designs. Moreover, we can also see which components of the model are completed most often, as well as if there are fields which the users find unnecessary.

Then, we want to inspect the conversational aspects of the bug report, and more specifically the comments on that ticket. We assume that having a large number of comments for clarifying the context of the issue or how it can be reproduced implies that the bug report model is inefficient and it does not force the users to provide all the necessary details for the developers. Moreover, we also assume that having a negative tone attached to the conversation implies that the bug report model is not optimal for the project.

Finally, we want to group all tickets in terms of difficulty and analyze the time it took to get fixed taking into consideration all the completed fields and the options selected.

Deliverable: Data that links the bug report model and its characteristics to a level of effectiveness.

Identity specific components that influenced bug report model effectiveness (WP3)

Having all the data available from WP2, we will to take a closer look at what components influenced the most the effectiveness of the bug report model. More specifically, we will investigate the following fields (they can be found in any bug report model on any issue tracking system):

- summary - short description of the bug;
- description - more detailed description of the bug where users can include stacktraces or steps to reproduce;

- attachments - screenshots or snippets or code that the users can attach to help the developers locate the problem easier;
- comments - discussion between users and developers on the ticket;
- environment - details regarding what platform were the users using when the bug occured, network connection, version of the application etc.

Moreover, we will look into which of these fields actually proved useful for the developers and if they were made required by the issue tracking system (or if the system gives the developers the ability to make them required in a user-friendly manner).

Then, we will investigate if the issue tracking system takes into consideration the characteristics of the project. More specifically, we want to see if, for example Jira, automatically changes the bug report model based on the size of the development team or their typical work flow.

Deliverable: Map of specific components to project's specific characteristics that influence effectiveness the most.

Automatically design bug report models optimized per project (WP3)

The major work product of our research will be the design, implementation and testing of the tool. We have decided that we want to write our tool in Go, mostly because of its simplicity, well defined standards as well as great concurrent capabilities [10].

We will use all the data and statistics collected in the previous work products and produce an app that can successfully take as input all the project's characteristics, the issue tracking system used, the current bug report models, what are the most effective components and then output a design that is tailored specifically for the project.

Deliverable: Tool that can create a bug report model tailored specifically for the project given as input.

Evaluation of the effectiveness of the tool (WP4)

We want to run the evaluation of the tool continuously throughout the whole development phase. As mentioned before, we have already agreed with Mozilla and Eclipse to try out our tool on their projects, therefore we will get valuable insights right from the beginning.

Moreover, we want to team up with several academics from prestigious UK universities in order to improve the tool continuously. Also, we want to conduct interviews with developers on the projects under investigation every month so that they can tell us if the designs generated by our tool are valuable for them. Moreover, we also want to interview triagers and managers on the team to check if our models are helping them track and manage the project easier.

Finally, we want to run statistical analysis and infer, from a quantitative point of view, whether our auto-generated designs are more effective than the standard ones used previously by the projects.

Deliverable: data collected through interviews and statistical analysis in order to infer the quality of our tool.

3 Measurable Outcomes

4 Impact and National Importance

References

[1] J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. pages 298–308, 2009.

[2] O. Baysal, R. Holmes, and M. W. Godfrey. Situational awareness: personalizing issue tracking systems. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1185–1188. IEEE Press, 2013.

[3] O. Baysal, R. Holmes, and M. W. Godfrey. No issue left behind: Reducing information overload in issue tracking. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 666–677. ACM, 2014.

[4] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? pages 308–318, 2008.

[5] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Appendix to information needs in bug reports technical report. Technical report, October 2009.

[6] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: improving cooperation between developers and users. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 301–310. ACM, 2010.

[7] E. Co. Elasticsearch.

[8] Cybernetica. X-road - a secure data exchange layer for building e-governments. Technical report, Mäealuse 2/1, Tallinn 12618, Estonia, 2014.

[9] C. dos Santos and M. Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, 2014.

[10] F. Forsby and M. Persson. Evaluation of golang for high performance scalable radio access systems, 2015.

[11] M. Foundation. Bugzilla.

[12] D. M. German, B. Adams, and A. E. Hassan. Continuously mining distributed version control systems: an empirical study of how linux uses git. *Empirical Software Engineering*, 21(1):260–299, 2016.

[13] Google. Google cloud platform.

[14] U. Government. Health digital services jira.

[15] I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles. Towards a simplification of the bug report form in eclipse. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 145–148. ACM, 2008.

[16] S. Just, R. Premraj, and T. Zimmermann. Towards the next generation of bug tracking systems. In *Visual languages and Human-Centric computing, 2008. VL/HCC 2008. IEEE symposium on*, pages 82–85. IEEE, 2008.

[17] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th working conference on mining software repositories*, pages 92–101. ACM, 2014.

[18] A. Z. Khan, M. Atique, and V. Thakare. Combining lexicon-based and learning-based methods for twitter sentiment analysis. *International Journal of Electronics, Communication and Soft Computing Science & Engineering (IJECSCSE)*, page 89, 2015.

[19] A. J. Ko and P. K. Chilana. How power users help and hinder open bug reporting. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1665–1674. ACM, 2010.

[20] O. Kononenko, O. Baysal, R. Holmes, and M. W. Godfrey. Dashboards: Enhancing developer situational awareness. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 552–555. ACM, 2014.

[21] S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.-R. MÃžller, F. Pereira, C. E. Rasmussen, et al. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8(Oct):2443–2466, 2007.

[22] T. Zimmermann, R. Premraj, J. Sillito, and S. Breu. Improving bug tracking systems. In *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 247–250. IEEE, 2009.