Automatic Contextual Bug Report Modelling

Andrei-Mihai Nicolae (2147392)

1 Introduction

Software engineering, compared to many other engineering fields, is more abstract [9] - in mechanical engineering, for example, if one wishes to see what is wrong with the engine, the components can be touched and manipulated directly by the human operator. However, when taking the example of compiling and running a simple program written in a language, such as Go [19], we go through various layers of abstraction which cannot be directly seen by the developer: the written characters inside the file get compiled by the Go compiler to machine readable code; then, the operating system instructs the kernel to run the newly generated machine code, which in turn communicates with the hardware components responsible for carrying out the necessary computations.

Therefore, having discussed the complex abstraction usually involved in software development, it is not a trivial task to plan and create applications that fit the requirements of the stakeholders. Thus, developers have come up with *issue tracking systems*, applications which help teams plan work, create and assign tasks, monitor progress etc. The basic unit of functionality these systems make use of is called a *ticket*, which can be usually split into two main categories: bug reports and feature requests. These software tickets can include various types of information, such as:

- textual description of the issue/proposed work;
- stacktraces that describe the error occured while using the application;
- attachments which support the bug report's existence;
- story points (i.e. difficulty assigned for that specific ticket, such as how many hours might take a developer to fix it).

Usually, for most projects, bug reports are filed on the application's website by ordinary users, many who do not have technical experience. Thus, a yet unresolved problem arises: how can one design bug reports that are intuitive and which will produce the most value to developers (i.e. help them solve the bug as quickly as possible)? My research tries to find out what changes need to be brought to modern issue tracking systems so that they can automatically change the bug report model based on the context (i.e. type of project and underlying technologies). Thus, a project, such as Mozilla Firefox, which is mainly built using C++, might benefit more from stacktraces in the bug report rathen than Apache Kafka which is a distributed streaming platform written in Java, that might benefit more from steps to reproduce from the users.

There are a couple of research papers that are looking at how to improve bug tracking systems and the bug report models. However, the authors of all of them have either investigated the issue through qualitative approaches, mainly conducting interviews and questionnaires with developers, or through analyzing previous pieces of work and inferring possible solutions from there. However, I believe that these approaches do not yield conclusive results because the developers are biased and also because simply investigating previous progress on this matter does not bring valuable conclusions to the community. Therefore, my research will at first identify what makes for a good quality ticket based on the type of project and underlying technologies, and then automatically re-design the form for filing a bug report using these results.

Report Structure

This literature review presents the most relevant papers in the fields that surround the area of software ticket quality. Section 2 looks at data quality and metrics in the software development area, such as data quality-related methodologies that help organizations develop beneficial data workcycles; Section 3 presents previous work on defining what are the key elements that make for good quality tickets in a specific context, while Section 4 presents previous work on the field that my research is based on (i.e. modelling the bug reports efficiently), as well as what could be improved about issue tracking systems. Finally, Section 5 presents conclusions drawn after completing the literature survey and possible future directions of the research.

2 Data Quality and Metrics

Several authors have investigated the *meaning* of data quality and what characteristics define it.

Bachman et al. [1] conducted a thorough investigation of several software projects, both open source as well as closed source, in order to infer what determines their quality. They selected various sources of information, among which bug tracking databases and version control systems logs, and examined SVN and CVS logs, as well as the content of the bug tracker databases, in the end trying to link the logs with the bug tracker contents as they are not integrated by default.

The study came to several conclusions, among which:

- closed source projects are usually of better quality than open source ones (i.e. better average number of attachments, better average number of commits per bug report); one example is the Eclipse open source project which has more than 20% empty commit messages;

- there is reduced traceability between bug reports and version control logs.

A different approach was taken by Strong et al. [18] who conducted a qualitative analysis instead of quantitative by collecting data from a large number of data quality projects by interviewing custodians, customers and managers. They analyzed each project using the data quality dimensions as content analysis codes.

Complementing the first study, the authors reached the conclusion that representational data quality dimensions are underlying causes of accessibility data quality problem patterns. The authors also found out that three underlying causes for users' complaints regarding data not supporting their tasks are incomplete data, inadequately defined or measured data and data that could not be appropriately aggregated.

Stamelos et al. [17] tried to discuss and examine the quality of the source code delivered by open source projects. They used a set of tools that could automatically inspect various aspects of source code, looking at a release of the OpenSUSE project and its components, written in C. The results show that Linux applications have high quality code standards that one might expect in an open source repository, but the quality is lower than the one implied by the industry standards. More than half of the components were in a high state of quality, but on the other hand, most lower quality components cannot be improved only by applying some corrective actions.

Having defined what data quality is and what are its characteristics, I needed to research what are some methodologies that help measure it efficiently. There are several research papers that present such methodologies, among which the work of Lee et al. [15]. The authors tried to define an overall model along with an accompanying assessment instrument for quantifying information quality in an organization. In the end, the authors applied their proposed methodology to multiple organizations and found out that it is practical and it could be used on a wider scale in the industry.

3 Ticket Quality

Issue quality is an important topic of this literature review as it revolves around software tickets and the characteristics of a well-written and informative bug report, how efficient are bug tracking systems, what defines an efficient bug triaging process etc.

The first and most important sub-topic of issue quality, which is the one I will address in my own research as well, is the quality of bug reports. There are several authors that have tried to defined what makes for a good bug report and what components actually improve the overall quality.

Bettenburg et al. [4] analyzed what makes for a good bug report through qualitative analysis. They interviewed over 450 developers and asked them what are the most important features for them in a bug report that help them solve the issue quicker. They reached the conclusion that stack traces and steps to reproduce increased the quality of a bug report the most, followed by well-written summaries and descriptions. Last but not least, they also created a tool called CueZilla that could predict the quality of a bug report with an accuracy rate of around 50%. Even though this paper is one of the very few that created a tool similar to what my research revolves around, the industry has not been interested in it mainly because it provides quite low levels of accuracy. Therefore, I am trying to come up with a more comprehensive, but precise set of key elements that will increase the accuracy levels considerably, which in turn might be of interest to software organizations.

The authors of [5] examined whether duplicate bug reports are actually harmful to a project or they add quality. They collected big amount of data from the Eclipse project and ran various kind of textual and statistical analysis on the data to find answers. They reached the conclusion that bug duplicates contain information that is not present in the master reports. Moreover, these duplicates could aid developers in finding out more information regarding the issue that needs to be fixed.

In order to model the quality of a bug report, one needs to be able to successfully extract various types of information from such a report. Also, if possible, it would be beneficial to be able to link them to the source code of the project (i.e. source code fragments in bug report discussions should be linked to the corresponding sections in the actual code) or other software artifacts. There are several researches that tried to analyze such techniques and one of them is the paper from Bettenburg et al. [6]. The authors created a tool that could parse a bug report and, using fuzzy code search, extract source code that could then be matched with exact locations in the source code. Eventually, the tool produces a traceability link (i.e. tuple containing a clone group ID and file paths corresponding to the source code files). Compared to the current state-of-the-art technique, change log analysis, this method improved the accuracy by over 20%.

However, Kim et al. [12] propose a rather different technique. The authors employed a two-phase recommendation model that could locate the necessary fixes based on information in bug reports:

- firstly, they did a feature extraction on the bug reports (e.g. extract description, summary, metadata);

- secondly, in order to make successful predictions, this model was trained on the previously collected

bug reports so that, when given a new bug report, it could automatically reveal where the necessary fixes needed to be applied;

- the actual implementation was put into place, and that is the two phase recommendation model, composed of binary (filters out uninformative bug reports before predicting the files to fix) and multiclass (previously filtered bug reports are used as training data and only after that new bug reports can be analyzed and files to be changed recommended); the overall accuracy was above the one achieved by Bettenburg et al. [6], being able to rank over 70%, but only as long as it recommended *any* locations.

4 Bug Tracking Systems and Bug Report Modelling

Bug report modelling based on context is the topic of my research and in this section I will present the related work that has been previously published.

Firstly, I needed to look at how people actually write bug reports, thus the work of Ko et al. [14] proved valuable. The authors inspected how end users tend to write bug reports and developed tools to linguistically analyze them. They parsed and tokenized the report titles so that the engineers could streamline the whole software development process and automate certain tasks. Moreover, being able to analyze report titles and descriptions could also aid in finding duplicates, thus increasing the overall efficiency of the bug tracking system. This paper, even though has limitations in that the number of reports analyzed was not large, proves quite beneficial for my research as if the tool would be made accessible by the authors, I could easily parse report titles and then optimize by automatic bug report modelling.

Just et al.[11] investigated what are the flaws in modern issue tracking systems and what should be changed to make the development process easier. The authors have asked almost 900 developers, of which 156 responded, various questions on what are the most important criterias when designing an issue tracking system with efficiency in mind. They applied card sort to rank the highest comments and they reached several interesting conclusions:

- the most important elements in bug reports, regardless of the type of project or underlying technologies, are stacktraces, steps to reproduce and test cases;

- duplicated bug reports did not affect overall quality of the development process, mainly because they provide additional information;

- even though reporters do know what is important to developers in a bug report, they did not always include that information when they filed the report; the main reason for this is that the reporters consider it is too difficult/time-consuming to gather such data.

In contrast to the research of Just et al.[11], Zimmermann et al.[20] adopted a more empirical approach by looking at previous work in the field. The authors have proposed several ideas to improve modern issue tracking systems, as well as simulate their ideal issue tracking system where the user needs to provide information based on a decision tree. The paper reaches the main conclusion that users need to be asked the right questions in bug reports based on the type of project and what the developers in the team need most when fixing a bug. However, the authors also state that their ideal system is not production-ready yet as it is in a preliminary stage and it did not cover large amounts of data (i.e. 2,875 reports).

Another relevant research is the one of Breu et al.[8], where the authors try to improve the relationship between developers and bug reporters. They have brought several major contributions to the community, among which a *catalogue of frequently asked questions*, split into eight categories (e.g. missing information, clarification, triaging) and 40 sub-categories; *analysis of question time, response rate and time*, which revealed important findings such as correction questions are most likely to be answered quickly; bug reports are *fixed faster if the reporter is engaged in the process*; bug tracking systems should become more *user-centered* and allow developers engage with end users easier. This piece of research comes with conclusions which are very valuable to my work, especially as the authors have also published a technical report [7] where they present all the findings.

The work of Herraiz et al. [10], compared to the ones presented earlier, looks specifically at Eclipse bug reports and concludes, based on statistical analysis, that bug reports should be simplified and only the most vital information should be asked from users, especially as most do not have the time or do not find the task too valuable to complete longer reports. These findings are strenghtened by the work of Panjer [16], where he shows that many bug reports take much longer than necessary due to the overcomplication of the bug report.

A very important paper that is similar to my own research is the one published by Baysal et al.[2]. The authors have looked into how to personalize an issue tracking system based on the developers working in the team, as well as the tasks they usually perform. This work proposes a completely different approach than the research of Breu et al.[8] in that it favors a more developer-centric bug tracking system. The authors presented an enhanced Bugzilla, which is one of the most widely used issue tracking systems, where developers have the ability of tracking their own progress and also maintain progress awareness across the whole project. I will try to develop my own automatic contextual bug report modelling building on top of the findings brought by this paper.

A rather different point of view is the one presented in the research of Bertram et al. [3], where they state that issue tracking systems are not only a database for storing bug reports, but also a communication center where stakeholders discuss with developer about various aspects of the project. The authors conducted interviews with members of small teams where they asked them what did they find most beneficial in such tracking systems. They found that most people involved consider bug tracking systems serve multiple purposes, especially organizational and communication ones, apart from being databses of bug reports. I do aggree with their findings, but I believe that a major limitation of the research is that their target teams have been only small ones, as large ones might answer differently, especially since they dispose of multiple means of communicating (e.g. chat services such as Slack which can be integrated with bug repositories, but are not part of them).

However, the difference in reporters needs to be taken into account. The paper published by Ko et al. [13] looks at how regular contributors to a bug tracking system can help the project move forward. Even more, the authors suggest that projects in general should mostly focus their attention on recruiting such talented reporters rather than deriving value from the large masses of regular reporters. This is valuable to my research because I will design my bug report model based on these findings and try to make it more experienced reporter focused.

5 Discussion and Conclusion

Having analyzed state-of-the-art approaches at how to model quality in bug reports, as well as what makes for good bug reporting systems, I have gained valuable insights for my own research and how I would like to proceed with the grant proposal report.

Firstly, I would like to develop a tool that can identify what is valuable in a ticket using the papers presented in Section 3 using multiple open source repositories such as Apache's projects and Mozilla Firefox. Then, I would like to inspect modern bug tracking systems, such as Jira, and see what fields are necessary when creating a new bug report (e.g. Apache uses Jira for most of its open source projects, thus that would be a good starting point). Afterwards, I will use my knowledge gained from reading the papers depicted in Section 4 to gather the most important questions that the reporter needs to be asked and infer the best format for the bug report based on the specific project and the underlying technologies. The resulting bug report model will then be tested probably using machine learning algorithms in order to test its efficiency.

Undergoing this literature review, I believe I have gathered enough relevant information to proceed with my research and maybe include other collaborators that could aid in testing the approach by giving access to some open source projects and applying the algorithm directly in their bug tracking systems.

References

[1] A. Bachmann and A. Bernstein. Software process data quality and characteristics: a historical view on open and closed source projects. pages 119–128, 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] D. Bertram, A. Voida, S. Greenberg, and R. Walker. Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 291–300. ACM, 2010.

[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] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful... really? pages 337–345, 2008.

[6] N. Bettenburg, S. W. Thomas, and A. E. Hassan. Using fuzzy code search to link code fragments in discussions to source code. pages 319–328, 2012.

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

[8] 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.

[9] F. P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E*. Pearson Education India, 1995.

[10] 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.

[11] 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.

[12] D. Kim, Y. Tao, S. Kim, and A. Zeller. Where should we fix this bug? a two-phase recommendation model. *IEEE transactions on software Engineering*, 39(11):1597–1610, 2013.

[13] 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.

[14] A. J. Ko, B. A. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems. In *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 127–134. IEEE, 2006.

[15] Y. W. Lee, D. M. Strong, B. K. Kahn, and R. Y. Wang. Aimq: a methodology for information quality assessment. *Information & management*, 40(2):133–146, 2002.

[16] L. D. Panjer. Predicting eclipse bug lifetimes. In *Proceedings of the Fourth International Workshop on mining software repositories*, page 29. IEEE Computer Society, 2007.

[17] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris. Code quality analysis in open source software development. *Information Systems Journal*, 12(1):43–60, 2002.

[18] D. M. Strong, Y. W. Lee, and R. Y. Wang. Data quality in context. *Communications of the ACM*, 40(5):103–110, 1997.

[19] G. Team. Golang. `https://golang.org/`. [Online; accessed 06-February-2018].

[20] 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.