Improving Bug Tracking Systems

Thomas Zimmermann[1,2] tz@acm.org
Rahul Premraj[3] rpremraj@cs.vu.nl
Jonathan Sillito[2] sillito@ucalgary.ca
Silvia Breu[4] silvia.breu@cl.cam.ac.uk

[1] Microsoft Research, Redmond, USA
[2] Department of Computer Science, University of Calgary, Canada
[3] Vrije Universiteit, Amsterdam, The Netherlands
[4] Computer Laboratory, University of Cambridge, UK

Abstract

It is important that information provided in bug reports is relevant and complete in order to help resolve bugs quickly. However, often such information trickles to developers after several iterations of communication between developers and reporters. Poorly designed bug tracking systems are partly to blame for this exchange of information being stretched over time. Our paper addresses the concerns of bug tracking systems by proposing four broad directions for enhancements. As a proof-of-concept, we also demonstrate a prototype interactive bug tracking system that gathers relevant information from the user and identifies files that need to be fixed to resolve the bug.

1. Introduction

> *Alice, a real person: My ECLIPSE crashed.*
> *Bob, a bug-tracking system: What did you do?*
> *Alice: I clicked on File → New Project and OK.*
> *Bob: Did you choose a Java project?*
> *Alice: No.*
>
> *. . . (a few questions later)*
>
> *Bob: Thanks Alice. The bug is likely in ProjectCreator.java and we will fix it soon.*

The use of bug tracking systems as a tool to organize maintenance activities is widespread. The systems serve as a central repository for monitoring the progress of bug reports, requesting additional information from reporters, and discussing potential solutions for fixing the bug. Developers use the information provided in bug reports to identify the cause of the defect, and narrow down plausible files that need fixing. A survey conducted amongst developers from the APACHE, ECLIPSE, and MOZILLA projects found out which information items are considered useful to help resolve bugs [4]. Items such as stack traces, steps to reproduce, observed and expected behavior, test cases, and screenshots ranked high on the list of preferred information by developers.

Previous research has shown that reporters often omit these important items [4, 6]. Developers are then forced to actively solicit information from reporters and, depending on their responsiveness, this may stall development. The effect of this delay is that bugs take longer to be fixed and more and more unresolved bugs accumulate in the project's bug tracking system. We believe that one reason for this problem is that current bug tracking systems are merely interfaces to relational databases that store the reported bugs. They provide little or no support to reporters to help them provide the information that developers need.

Our on-going work is directed towards rectifying the situation by proposing ideas that can fundamentally transform bug tracking systems to enhance their usability such that relevant information is easier to gather and report by a majority of users (Section 2). Further, we present a simulation of an interactive bug tracking system, using a decision tree, which extracts relevant information from the user to identify the file that contains the bug, as in the above discourse, as a means to help developers. (Section 3). The paper closes with a look at related work (Section 4) followed by conclusions (Section 5).

2. Better bug tracking systems

Having complete information in the initial bug report (or as soon as possible) helps developers to quickly resolve the bug. The focus of our work is on improving bug tracking systems with the goal of increasing the completeness of bug reports. Specifically, we are working on improving bug tracking systems in four ways (see Figure 1):

[1] Many thanks to Sascha Just for creating the template for Figure 1.

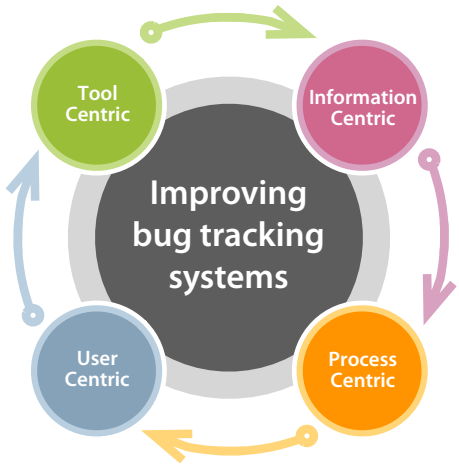

Figure 1. Improving bug tracking systems.[1]

Tool-centric. Tool-centric enhancements are made to the features provided by bug tracking systems. They can help to reduce the burden of information collection and provision. For example, bug tracking systems can be configured to automatically locate the relevant stack trace and add it to a bug report. Also, providing steps to reproduce can be automated by using capture/replay tools or macro-recorders; observed behavior can be easily shown by simplifying screen capture; test cases can be automatically generated [3]. All of the above examples aim to help with the collection of information needed by developers to fix bugs.

Information-centric. These improvements focus directly on the information being provided by the reporter. Bug tracking systems can be embedded with tools such as CUEZILLA [4] that provide real-time feedback on the quality of the information provided and what can be added to increase value. With helpful reminders from the system, reporters may be motivated to go the extra mile and collect more information. Systems can be further modified to perform validity checks such as verifying whether the reported stack trace is consistent and complete, submitted patches are valid, and the like.

Process-centric. Process-centric improvements to bug tracking systems focus on administration of activities related to bug fixing. For example, *bug triaging*, i.e., determining which developer should resolve the bug, can be automated to speed up the task [1, 5]. Other examples are better awareness of the progress made on bug reports (so that reporters are aware of the actions taken as a response for their efforts) or to provide users with an early estimate for when their bug will be fixed.

User-centric. This includes both reporters and developers. Reporters can be educated on what information to provide and how to collect it. Developers too can benefit from similar training on what information to expect in bug reports and how this information can be used to resolve bugs.

The following section presents a prototype developed around the information-centric direction that gathers information from reporters to identify candidate files to be fixed.

3. Asking the right questions in bug reports

When a user submits a bug report she is asked many questions: What is the name of the product? In which plug-in/component? What is the Build ID? What is the bug about? What are the steps to reproduce the bug? Any additional information? However, the initial information provided in a bug report is often incomplete and developers often have follow-up questions: Do you have flash installed? Can you provide a screenshot? Getting replies by users takes time (often weeks) and slows down the progress on open bugs. Furthermore, research has shown that developers get responses to only two out of three questions [6].

Ideally, follow-up questions would be asked immediately after a user has submitted a bug report and is still within reach. We therefore propose to build expert systems that *automatically ask* relevant questions and gather all required information once an initial failure description has been provided. The selection and order of questions should not be static (as in current bug tracking systems), but rather depend on previous responses. We believe that such an expert system can provide better bug descriptions and also narrow down the location of the defect.[2]

To create such a system the following data is needed:

1. *Information that developers use to find the location of a defect.* For each kind of information one can then formulate a question.[3] Some might be general such as asking for screenshots, build identifiers, and stack traces, but other questions might be more specific, for example asking about certain menus or dialogs.

2. *The defect location and question/answer-pairs for a large number of fixed bugs.* One can then use these bugs to build machine learning models (e.g., decision trees, neural networks) which select questions and predict the location of the defect based on the responses.

[2] One can think of this approach to bug reporting as a computerized version of the twenty questions game. The bug tracking system asks the user several questions (one question at a time) and the user provides responses. After this process is finished, the developer would have a description of the failure and ideally the bug tracking system could make a good guess at the location of the defect.

In this paper, we show an early proof-of-concept study that uses data that is readily available in bug reports (such as component, version, reporter). We are currently working on compiling a catalog of information frequently needed by developers. Once we have this catalog we will implement a tool and conduct a fully-fledged evaluation.

3.1. A first experiment

To check whether question/answer-pairs can predict the location where a bug has been fixed, we conducted the following experiment. For the twenty most frequently fixed files in ECLIPSEJDK we selected all related bug reports. We then used the resulting 2,875 cases to train a decision tree to predict the location of the fix (name of the file).[4]

As input features, we used the following questions for which we could extract the answers easily from the ECLIPSE bug database.

- How severe is the bug? *(bug_severity)*
- On what operating system does it occur? *(op_sys)*
- What is the affected component? *(component_id)*
- How important is the bug? *(priority)*
- Which version of ECLIPSE is affected? *(version)*
- What is your name?[5] *(reporter)*
- What platforms are affected? *(rep_platform)*

The resulting decision tree is in Figure 2. The root node shows the defect distribution for all twenty selected files, e.g., 11% of bugs have been fixed in GenericTypeTest.java. For all other nodes, we report only the three files with the highest likelihood.

Out of the seven input features, only three influenced the location of a defect. The feature with the most influence is *component_id*, which comes as no surprise because components almost directly map to source code. The next most influential feature is *version*, which indicates that defect-prone hot-spots in components change over time.

The *reporter* of a bug also predicts the location of the defect, likely because reporters use ECLIPSE differently and thus trigger bugs in different parts. For example in Figure 2, the reporter set R1 mostly reveals bugs related to Java formatting, while R2 reveals bugs related to AST conversions and R3 reveals bugs related to the ECLIPSE Java model.[6]

[3]The user can be an end-user or a tester and both would require different sets of questions. For example, an end-user will not be familiar with the internal structure of a system, while a tester might be.

[4]We decided to use a decision tree for illustrative purposes. It is possible and likely that neural networks or other machine learning models will yield better results.

[5]Decision trees are computationally expensive when an input feature can take many values, which is the case for *reporter*. We considered therefore only the twenty most active reporters and modeled less active reporters as "other".

[6]For privacy reasons, we omit the reporter names in Figure 2.

In Figure 2, the path that leads to JaveEditor.java would ask only about the component (Text). In contrast the path to ASTConverter15Test.java would ask about component (Core, UI), version (3.1 or higher), and the reporter (R4). This justifies the need for interactive bug tracking systems (rather than static web-pages) because for certain responses fewer questions are needed. With more questions and more files we expect the decision tree to become more complex. We also expect that the order of questions will change for different paths in the tree.

3.2. Next steps and challenges

We showed that for some of the questions asked in bug reports (component, version), the responses can potentially predict the location of the defect. Our study is very preliminary, and our next steps will be the following:

- Identify information needs in a large sample of bug reports through manual inspection. This will help to compile a catalog of questions that can be used for the expert system.
- Using this catalog, collect answers and defect locations for another large sample of bug reports. This dataset will be used to automatically learn a prediction model.
- Evaluate the predictions and conduct usability studies.

One of the challenges for this project will be to find a representative catalog of questions that is able to predict defects. In addition, scalability will become an issue once more questions with many unique values are used.

4. Related work

Many researchers have investigated what factors aid or indicate the quick resolution of bugs. Hooimeijer and Weimer [8] observed that bug reports with more comments get fixed sooner. They also noted that bug reports that are easier to read also have shorter life times (also confirmed by our previous work [4]). More recently, Aranda and Venolia have examined communication that takes place between developers outside the bug tracking system [2]. In our earlier work, we discussed shortcomings of existing bug tracking systems [9], which led to the four areas of improvements presented in this paper. Asking the right questions, is a crucial part of debugging and several techniques such as algorithmic debugging [7] and the WhyLine tool [10] support *developers* in doing so. In contrast, we propose to move this process from the developer side to the *user side*. Instead of recorded feedback by many users as it happens in Collaborative Bug Isolation [11], our proposed approach requires explicit feedback by only a single user.

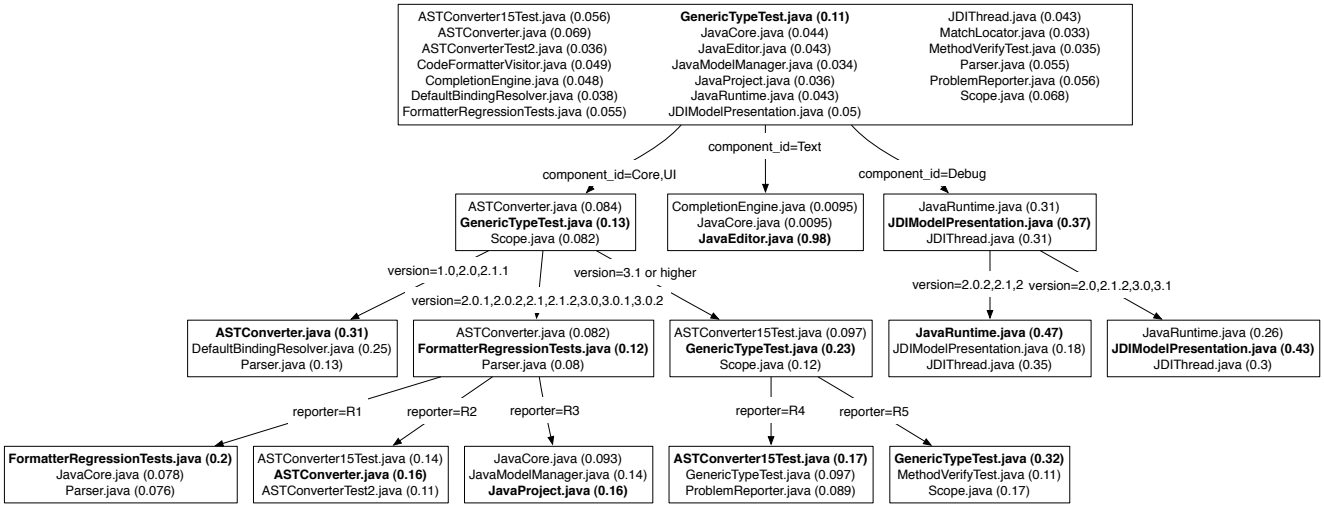

Figure 2. The decision tree illustrates how questions could narrow down the location of a defect. Each path corresponds to a question/answer series. The file in bold is the likely defect location.

5. Conclusions and consequences

Current bug tracking systems do not effectively elicit all of the information needed by developers. Without this information developers cannot resolve bugs in a timely fashion and so we believe that improvement to the way bug tracking systems collect information are needed.

This paper proposes four broad areas for improvements. While implementing a range of improvements from these areas may be ideal, bug tracking systems may instead prefer to specialize, thus providing a rich set of choices. This would be a healthy change to the current situation where they all provide identical functionality.

As an example of the kind of improvements we envision, we have described an interactive system for collecting information from reporters and leveraging that information to locate the source of the bug. To demonstrate the potential of this idea we have conducted an initial study in which we simulated an interactive bug tracking system. The system asks the user context-sensitive questions to extract relevant information about the bug early on to suggest candidate files that will need to be fixed. This is likely to speed up the process of resolving bugs. In the future, we will move from the current prototype of the interactive system to a full-scale system that can deal with a variety of information to gather, as commonly observed in the real world.

References

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *ICSE'06: Proceedings of the 28th International Conference on Software engineering*, pages 361–370, 2006.

[2] J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *ICSE'09: Proceedings of the 31st International Conference on Software Engineering (to appear)*, 2009.

[3] S. Artzi, S. Kim, and M. D. Ernst. Recrash: Making software failures reproducible by preserving object states. In *ECOOP'08: Proceedings of the 22nd European Object-Oriented Programming Conference*, pages 542–565, 2008.

[4] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *FSE'08: Proceedings of the 16th International Symposium on Foundations of Software Engineering*, pages 308–318, November 2008.

[5] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful ... really? In *ICSM'08: Proceedings of the 24th IEEE International Conference on Software Maintenance*, pages 337–345, 2008.

[6] S. Breu, J. Sillito, R. Premraj, and T. Zimmermann. Frequently asked questions in bug reports. Technical report, University of Calgary, March 2009.

[7] P. Fritzson, T. Gyimothy, M. Kamkar, and N. Shahmehri. Generalized algorithmic debugging and testing. In *PLDI'91: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 317–326, 1991.

[8] P. Hooimeijer and W. Weimer. Modeling bug report quality. In *ASE'07: Proceedings of the 22nd International Conference on Automated Software Engineering*, pages 34–43, 2007.

[9] S. Just, R. Premraj, and T. Zimmermann. Towards the next generation of bug tracking systems. In *VL/HCC'08: Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 82–85, September 2008.

[10] A. J. Ko and B. A. Myers. Debugging reinvented: asking and answering why and why not questions about program behavior. In *ICSE'08: Proceedings of the International Conference on Software Engineering*, pages 301–310, 2008.

[11] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable statistical bug isolation. In *PLDI'05: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 15–26, 2005.

To learn more about our work on bug tracking systems, we invite you to visit `http://www.softevo.org/`