Automated Quality Assurance System

Kshirsagar Aniruddha P.
Computer Science & Engg. Deptt.
K. B. P. College Of Engg. & Poly.
Satara.
Email: ani.kshirsagar {at}
rediffmail.com

Balekar Swapnaja S.
Computer Science & Engg. Deptt.
K. B. P. College Of Engg. & Poly.
Satara

Rasal Swati A.
Computer Science & Engg. Deptt.
K. B. P. College Of Engg. & Poly.
Satara.

.

Abstract— **For maintaining the quality, a bug tracking system is a software application that is designed to help quality assurance and programmers keep track of reported software bugs in their work. It may be regarded as a type of issue tracking system. Bug Tracking Systems allow individual or groups of developers to keep track of outstanding bugs in their product effectively. Having complete information in the initial bug report helps developers to quickly resolve the bug. So choosing a good bug tracking system for your product helps you to reduce downtime, increase productivity, raise customer satisfaction, and improve communication between developers.**

Keywords-bug, CUEZILLA, Bugzilla, bug tracking system.

I. INTRODUCTION

A major component of a bug tracking system is a database that records facts about known bugs. Facts may include the time a bug was reported, its severity, the erroneous program behavior, and details on how to reproduce the bug; as well as the identity of the person who reported it and any programmers who may be working on fixing it [13]. Typical bug tracking systems support the concept of the life cycle for a bug which is tracked through status assigned to the bug. A bug tracking system should allow administrators to configure permissions based on status, move the bug to another status, or delete the bug. The system should also allow administrators to configure the bug statuses and to what status a bug in a particular status can be moved. Some systems will e-mail interested parties, such as the submitter and assigned programmers, when new records are added or the status changes.

A. Bug Tracking System

The main benefit of a bug-tracking system is to provide a clear centralized overview of development requests (including bugs and improvements, the boundary is often fuzzy), and their state. The prioritized list of pending items (often called backlog) provides valuable input when defining the product road map, or maybe just "the next release".[9]

Concurrency bugs are synchronization problems in multithreaded and multiprocessed programs. They are extremely difficult to detect because they are nondeterministic.

The state of the art in concurrency bug detection focuses on data races. A data race occurs when more than one thread access the same memory location without proper synchronization and at least one of them is a write. Three classes of tools have been proposed to detect data races.

They are lockset race detection tools [2], [3], happens-before race detection tools [4], [5], [6], and hybrid tools combining the above two [7], [8]. The lockset algorithm reports a race when it finds no common lock protecting accesses to a shared memory location. This happens-before algorithm reports a race when two conflicting memory accesses do not have a strict happens-before relation.

II. EXISTING BUG TRACKING SYSTEMS

A. Bugzilla

Bugzilla is very popular, actively maintained and free bug tracking system, used and developed together with Mozilla, giving it considerable credibility. Bugzilla is based on Perl and once it is set up, it seems to make its users pretty happy. It's not highly customizable, but in an odd way, that may be one of its features: Bugzilla installations tend to look pretty much the same wherever they are found, which means many developers are already accustomed to its interface and will feel they are in familiar territory.

Bugzilla has a system that will send you, another user, or a group that you specify the results of a particular search on a schedule that you specify. Bugzilla has a very advanced reporting system and you can create different types of charts including line graph, bar graph or pie chart.

B. Mantis

Mantis is a free web-based bug tracking system. It is written in the PHP scripting language and works with MySQL, MS SQL, and PostgreSQL databases and a web server. Mantis can be installed on Windows, Linux, Mac OS and OS/2. Almost any web browser should be able to function as a client. It is released under the terms of the GNU General Public License (GPL).

The main complaint is its interface which doesn’t meet modern standards. On the other hand, is easy to navigate, even for inexperienced users. There not exist some advanced features such as charts and reports. In short, the whole system is sloppily done; there are plenty of bugs and very little functionality.

	Search	Email notifications	Reports	Charts	Time Tracking	RSS/Atom Feed	Configurable	Free
Bugzilla	yes	yes	yes	yes	yes	no	no	yes
Mantis	yes	no	no	no	yes	yes	no	yes
BugTracker.NET	yes	yes	yes	yes	yes	no	yes	yes
Redmine	yes	yes	yes	yes	yes	yes	no	yes
Bugzero	yes	yes	yes	no	no	no	no	no

Table 1: Classification criteria. Legend: search, email notifications, reports, charts, time tracking, RSS/Atom Feed, Configurable, Free. Explanation: This table summarizes criteria that are used in decision making process of choosing suitable bug tracking system.

C. *BugTracker.NET*

BugTracker.NET is a free, open-source, web-based bug tracker or customer support issue tracker written using ASP.NET, C#, and Microsoft SQL Server Express. BugTracker.NET is easy to install and learn how to use. When you first install it, it is very simple to setup and you can start using it right away. Later, you can change its configuration to handle your needs. It has a very intuitive interface for generating lists of bugs.

It has two very useful features. First of them is a screen capture utility that enables you to capture the screen, add annotations and post it as bug in just a few clicks. The second feature is the fact that it can integrate with your Subversion repository so that you can associate file revision check in with bugs.

D. *Redmine*

Redmine is a flexible web-based project management web application. Written using Ruby on Rails framework, it is cross-platform and cross-database. Redmine is open source and released under the terms of the GNU General Public License. Redmine is flexible issue tracking system. You can define your own statuses and issue types. He support multiple projects and subprojects. Each user can have a different role on each project. Interface is very simple, intuitive and easy to navigate. Shortly, this is very good product and our recommendation.

E. *Bugzero*

Bugzero is a web-based bug, defect, issue and incident tracking software. Its single code base supports both Windows and Unix (based on Java™) and supports database systems including Access, MySQL, SQL Server, Oracle, and etc. Bugzero can be customized for software bug tracking, hardware defect tracking, and help desk customer support issue and incident tracking. Bugzero have intuitive interface but he lacks form features. The main drawback is the fact that Bugzero is an commercial product and you can find much better product for free.

III. BUG TRACKING SYSTEM AND BUG REPORT

A. *What a good bug report should contain:*

i. **A descriptive title**, one that contains specific key words and terms to enable categorization.

ii. **A summary** of the situation, which describes the problem at a high level, in a way that anyone familiar with the project can understand.

iii. **A set of steps** to recreate the situation. These should start with launching the app, include any relevant configuration steps, be atomic, be specific, and end with the action that causes the problem to be visible.

iv. **The expected behavior** and why the observed behavior is different.

v. **The severity of the issue**.

It's important to be clear on your reasoning as to why this is the worst bug you've seen in your lifetime, so that others can understand and support your decision.

vi. **The frequency of the issue**.

The general nature of a problem is often not enough to characterize that problem. One also needs to know the frequency. Is a crash always a Critical bug? No. Not if it was only seen once in 6 months of development Frequency can make a big difference to the overall severity of a problem.

vii. **A description of the context** in which the problem was found - the specific system, device, or OS, and any relevant configuration options for the test environment.

- build # of the product under development
- type of hardware on which test was run, e.g. iPhone, iPad, Mac
- configuration of hardware, e.g. 8Gb iPhone 3G, 32Gb iPad 3G
- OS version / build # on that hardware any special conditions, e.g. on wifi, after low battery warning.

B. A better bug tracking system

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.

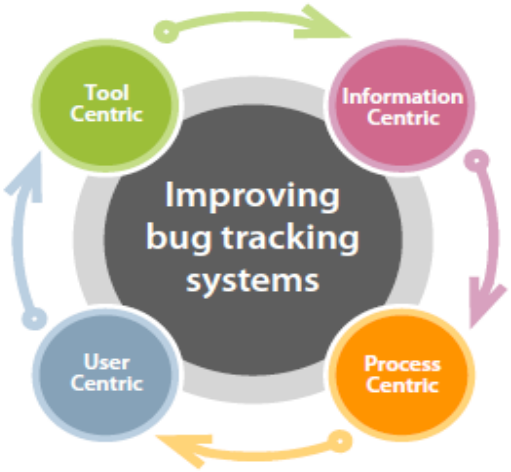

Figure1. Four areas to improve Bug Tracking System

i. Tool-centric improvements are made to the features provided by bug tracking systems. They can help to reduce the burden of information collection and provision. An example of tool-centric enhancements is capture/replay tools, which can provide steps to reproduce automatically [9].
ii. Information-centric improvements focus directly on the information being provided. As an example, the CUEZILLA tool, this provides real-time feedback on the quality of a bug report and what information can be added to increase value.
iii. Process-centric improvements to bug tracking systems focus on administration of activities related to bug fixing. For example, bug triaging, i.e., deciding which bugs get fixed and determining which developer should resolve the bug, can be automated [10], [12].
iv. User-centric improvements include 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.

C. SURVEY CONDUCTED AMONG DEVELOPERS AND USERS

A survey was conducted among developers and users of APACHE, ECLIPSE, and MOZILLA to find out what makes a good bug report. The analysis of the 466 responses revealed an information mismatch between what developers need and what users supply. Most developers consider steps to reproduce, stack traces, and test cases as helpful, which are, at the same time, most difficult to provide for users. Such insight is helpful for designing new bug tracking tools that guide users at collecting and providing more helpful information.

In the first two parts of the developer survey and the first part of the reporter survey, questions share the same items but have different limitations (select as many as you wish versus the three most important). We will briefly explain the advantages of this parallelism using D1 and D2 as examples.

1. *Consistency check.* When fixing bugs, all items that helped a developer the most (selected in D2) must have been used previously (selected in D1). If this is not the case, i.e., an item is selected in D2 but not in D1, the entire response is regarded as inconsistent and discarded.

2. *Importance of items.* We can additionally infer the importance of individual items. For instance, for item i, let N_{D1} (i) be the number of responses in which it was selected in question D1. Similarly, $N_{D1, D2}$ (i) is the number of responses in which the item was selected in both questions D1 and D2. Then, the importance of item i correspond to the conditional likelihood that item i is selected in D2 when selected in D1:

$$\text{Importance (i)} = \frac{N_{D1, D2}(i)}{N_{D1}(i)}$$

Other parallel questions were D3 and D4, as well as R1and R2.

IV. MEASURING BUG REPORT QUALITY WITH CUEZILLA

The bug reporters can provide better reports with similar assistance. As a first step toward assistance, we developed a prototype tool called CUEZILLA that measures the quality of bug reports. CUEZILLA also provides suggestions on how to enhance the quality of a bug report, for example, "Have you thought about adding a screenshot to your bug report?" To encourage reporters to actually provide additional information, CUEZILLA can show did you know facts mined from bug databases; for example, "Bug reports with stack traces are fixed N-times faster." Possible usage scenarios for CUEZILLA are to provide immediate feedback while new bug reports are entered, to solicit information for bug reports that are already in the bug database, or to prioritize bug reports during bug triage. CUEZILLA tool measures the quality of bug reports on the basis of their contents.

For each feature, a score is awarded to the bug report, which is either binary (e.g., attachment present or not) or continuous (e.g., readability).

- Itemizations. In order to recognize itemizations in bug reports, we checked whether several subsequent lines started with an itemization character (such as—,*, or +).To recognize enumerations, we searched for lines starting with numbers or single characters that were enclosed by parentheses or brackets or followed by a single punctuation character.
- Keyword completeness. We reused the data set provided by Ko et al. [16] to define a quality score of bug reports based on their content. In a first step, we removed stop words, reduced the words to their stem, and selected words occurring in at least 1 percent of bug reports.

A. Recommendations by CUEZILLA

The core motivation behind CUEZILLA is to help reporters file better quality bug reports. For this, its ability to detect the presence of information features can be exploited to tip reporters about what information to add. This can be achieved simply by recommending additions from the set of

absent information, starting with the feature that contributes to the quality further by the largest margin.

These recommendations are intended to serve as cues or reminders to reporters of the possibility of adding certain types of information likely to improve bug report quality.

Figure 2(a) illustrates the concept. The text in the panel is determined by investigating the current contents of the report, and then determining that it would be best, for instance, to add a code sample to the report. As and when new information is added to the bug report, the quality meter revises its score.

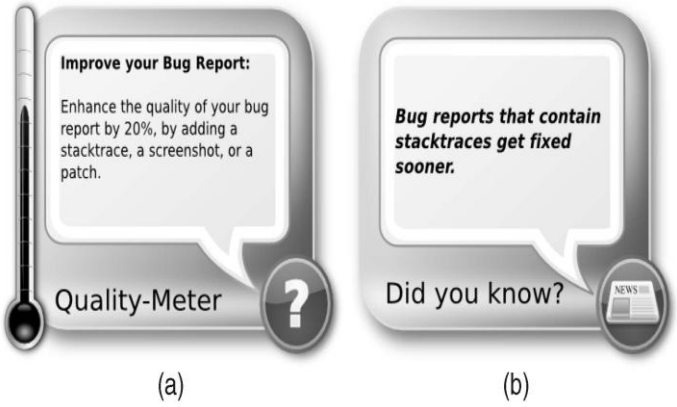

Figure 2. Mockup of CUEZILLA's user interface.
(a) It recommends improvements to the report.
(b) To encourage the user to follow the advice, CUEZILLA provides facts that are mined from history.

The evaluation of CUEZILLA shows much potential for incorporating such a tool in bug tracking systems. CUEZILLA is able to measure quality of bug reports within reasonable accuracy. However, the presented version of CUEZILLA is an early prototype and we plan to further enhance the accuracy before we will conduct user studies to show CUEZILLA's usefulness.

V. ADVANTAGES OF AN EFFECTIVE BUG-TRACKING SYSTEM

In software test management, bug reporting is a complex and complicated process that requires precision, detailing and a whole lot of information. Reporting and tracking bugs manually works fine in case of small projects, whereas for mission-critical or large projects, a paper-based approach can result in chaos and confusion. QA test management teams need an effective defect tracking management system to log the identified bugs and to monitor them.

A bug-tracking system helps the project team to successfully measure the project's status. The measurements, also known as metrics help the QA test management team to assess the quality of the software and in taking business decisions. The software metrics like project metrics, progress metrics, defect metrics and testing metrics also help in evaluating the success ratio of a tester or programmer.

A defect tracking system not only tracks defects but also tracks metrics to make sure everything is going according to the software development plan.

VI. FUTURE SCOPE

Evaluation of a bug tracking system requires the understanding of specific features [14]. These may include configurable workflow and customizable fields. The papers studied provided tips and guidelines for evaluating features of a bug tracking system.

The various factors that must be considered in evaluating a bug tracking system are as follows.

• Roles of the people who will use the system: The understanding of the above point makes the features added to the system relevant and hence increases productivity.

• Workflow for managing bugs: Every company has a different way of doing work. The workflow of the organization must be appreciated and system should work accordingly [15], [14].

• One of the most important things to be seen when developing a bug tracking system is who is responsible for what. A bug tracking system should not ask a data entry operator the expected root cause of the system failure [11], [16].

• Most important is the information that is needed to track the bug. This determines the reports and metrics to be developed.

• The last point is the need to provide different levels of access to different users [16], [14]. This may change the design of the system.

• Bug Tracking systems should have a facility that the Clients have number of problems while executing or running the particular software.If we will provide Change Request facility in Bug Tracking systems so that client will add the problem which he/she is facing while executing or running it,it will be beneficial for us to satisfy customer.

ACKNOWLEDGMENT

Our thanks to Prof. Sayyed S. G. for her valuable guidance regarding the subject.

REFERENCES

[1] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson, "Eraser: A Dynamic Data Race Detector for Multithreaded Programs," ACM Trans. Computer Systems, vol. 15, pp. 391-411, 1997.

[2] J.-D. Choi et al., "Efficient and Precise Datarace Detection for Multithreaded Object-Oriented Programs," Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation (PLDI), 2002.

[3] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson, "Eraser: A Dynamic Data Race Detector for Multithreaded Programs," ACM Trans. Computer Systems, vol. 15, pp. 391-411, 1997.

[4] A. Dinning and E. Schonberg, "An Empirical Comparison of Monitoring Algorithms for Access Anomaly Detection," Proc. Second ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP), 1990.

[5] R.H.B. Netzer and B.P. Miller, "Improving the Accuracy of Data Race Detection," Proc. Third ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP), 1991.

[6] D. Perkovic and P.J. Keleher, "Online Data-Race Detection via Coherency Guarantees," Proc. Second USENIX Symp. Operating Systems Design and Implementation (OSDI), 1996.

[7] R. O'Callahan and J.-D. Choi, "Hybrid Dynamic Data Race Detection," Proc. Ninth ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP), 2003.

[8] Y. Yu, T. Rodeheffer, and W. Chen, "Racetrack: Efficient Detection of Data Race Conditions via Adaptive Tracking," Proc. ACM 20th Symp. Operating Systems Principles (SOSP), 2005.

[9] Wikipedia, http://en.wikipedia.org/wiki/Bug_tracking

[10] S. Artzi, S. Kim, and M.D. Ernst, "Recrash: Making Software Failures Reproducible by reserving Object States," Proc. 22nd European Object-Oriented Programming Conf., pp. 542-565, 2008.

[11] J. Anvik, L. Hiew, and G.C. Murphy, "Who Should Fix This Bug?" Proc. 28th Int'l Conf. Software Eng., pp. 361-370, 2006.

[12] W. Cunningham, "Best of Bugzilla," http://eclipse-projects. blogspot.com/2005/12/best-of-bugzilla.html, 2005.

[13] Multiple (wiki). "Bug report". Docforge. Retrieved 2010-03-09

[14] Stephen Blair, A Guide To Evaluating a Bug Tracking System, White Paper, MetaQuest Software, October, 2004

[15] 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.

[16] 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.