Bug Tracking and Reliability Assessment System (BTRAS)

V. B. Singh
University of Delhi
71 PUBLICATIONS **818** CITATIONS
SEE PROFILE

Krishna Kumar Chaturvedi
ICAR-Indian Agricultural Statistics Research Institute
78 PUBLICATIONS **578** CITATIONS
SEE PROFILE

Bug Tracking and Reliability Assessment System (BTRAS)

V.B. Singh[1], Krishna Kumar Chaturvedi[2]

[1]Delhi College of Arts and Commerce, University of Delhi, Delhi, India
singh_vb@rediffmail.com

[2]Indian Agricultural Statistics Research Institute, Delhi, India
chat_krish@rediffmail.com

Abstract

Tracking of a reported bug for fixing is a fascinating area of research in software engineering. Many open source, free and commercial bug tracking tools have been developed and are currently under development. The industry needs criteria to select the best tool among the available set of tools that will help in fixing and tracking the progress of bug fixes. In this paper, we use BugZilla, Jira, Trac, Mantis, BugTracker.Net, Gnats and Fossil for comparative study. We present a comprehensive classification criteria to review the available tools and propose a new tool named Bug Tracking and Reliability Assessment System (BTRAS) for the bug tracking/reporting and reliability assessment. BTRAS helps in reporting the bug, assigning the bug to the developer for fixing, monitoring the progress of bug fixing by various graphical/charting facility and status updates, providing reliability bug prediction and bug complexity measurements, and distributing fixes to users/developers.

Keywords: *Bug/Issue tracking system, Comparison of tools, Resolution of bugs, Classification criteria, Machine learning techniques, Reliability assessment.*

1. Introduction

No software is perfect, it means that software may require additional module or enhancement of the existing module or some of the module may contain some unnoticed or unchecked bug that are left in the software from time to time. The bug may get introduced in any phase of the software development, i.e. requirement analysis (RA), design (SD), coding (SC), testing (ST), implementation (SI) and maintenance of the system (SM). With the proliferation of the developer in open source projects who are continuously contributing toward the project development and improvement, there is a possibility of introduction of new bugs in the project because of the incorporation of the current submission. There are a number of bugs daily submitted to the project website that may be using some software configuration management (SCM) tools for the version and release management of the software. The SCM tools may not provide any idea about the bug reporting as well as progress of fixing of bugs. There is an urgent need to plan and implement the best bug/issue tracking and reporting system. In open source environment generally, when the bug is submitted, any person can start doing work towards its fixing. But at the same time, other people may also start work in fixing the same bug. Therefore the owner or moderator of the project will get confused as to which solution to implement in the system. Recently, business software has conducted a study to compare ten best defect/bug tracking software vendors and created a profile of these vendors based on some qualitative parameters [9]. There is neither any specific time schedule for fixing of bug nor a person/team responsible for timely fixing of the same.

In proprietary software, the fixing of bug takes place in a phased manner because each phase has separate team responsible for the work being done in that phase. While in the open source project, anyone can pick up the bug and start working on that. Dependency may become major issue in calculating the time for fixing of the bug. If the bug is identified before the implementation phase then it can be very well resolved by development team or owner of the module. But once the software is released and implemented, it becomes very difficult to fix the bug. It has become a cumbersome job for the moderator to find a suitable developer who can fix a particular bug because the description of the reported bug may not generally provide complete information [6]. Many organizations generally rely on email with or without attachment/feedback based bug reporting system [1],[5] and these bugs will be maintained in spreadsheet or in any document editing software. It again becomes very difficult for the owner or developer to track the bug and its progress of fixing. There are many tools developed and currently used in industry for reporting and tracking progress of the reported bugs in small as well as large project. The tools were developed by open source community as well as proprietary software development organizations. It is getting very difficult to choose a particular bug reporting and tracking system that may provide us a very effective, advisable, helpful and cost effective tool in monitoring the progress of the bug fix schedule [2],[3]. A survey was conducted to find out the commonly faced problems in the bug reporting system by the developers and users [8]. Modelling effort was also made in finding the bug report quality in the past [7]. Recently, a survey was conducted over bug tracking tools on the basis of presentation, analysis and trends [10]. Marko and Aleksandar [10] made a decision tree based on some of the characteristics.

Various software reliability growth models have been developed in the literature to measure the reliability and testing progress of the product [18], [19] and [20]. These reliability growth models have been used during testing/debugging phase of the software.

Rest of the paper is organised as follows: Section 2 of the paper describes bug tracking system and bug life cycle. Section 3 describes different criteria for classifying bug tracking system. Section 4 presents a detail comparative analysis of different bug tracking system on the basis of various comparison criteria. Section 5 and 6 presents issues and challenges along with detail description of the proposed Bug Tracking Reliability Assessment System (BTRAS). Finally, the paper concludes with future research direction in section 7.

2. Bug Tracking System

The bug reporting/tracking system should provide an interactive web based platform for bug reporting and tracking the progress. The system may involve a generic process or specific schedule of bug reporting process. The generic process of the reporting a bug may generally contain the following information:

i. Title: Title of the bug.
ii. Description: Detailed description of the bug including what, where, why, how and when the bug occurs. Actual message that appears during the operation may be included with the actual set of input and expected output.
iii. Version: Specify the version of the project.
iv. Component: Specific component need to be specified.
v. Screenshot/Attachment: Corresponding screenshot can also be uploaded as a .jpg or .gif file by capturing the actual operation/output/message.
vi. Priority: Priority may be assigned for its urgency.
vii. Severity: Specify frequent occurrence and its impact in the system.
viii. Status: Current status of the bug (new, opened, confirmed, closed, etc.).

ix. Created by: Name of the person or id already registered with the system who is reporting the bug.

x. Assigned to: The bug reporter may also assign bug to specific person, if one knows about a particular person who can solve this problem otherwise assigned by the moderator.

xi. Revision History: If the bug is earlier reported, show the historical changes.

xii. Estimated time: The estimated time may be specified, generally used in case of closed team environment not in the open source environment.

xiii. Comments: Any other information that will be helpful in identifying the bug.

Once the bug is submitted to the system, a moderator will see its status and look for the details of the report and its various parameters. Moderator can also reset or update some of the parameters and correspondingly updates the values of status and assignee. The average bug reporting rates are quite high in large open source software projects. For example, in the eclipse project [23], average bug reporting rate is 50 issues per day [24]. Thus, it is very difficult for the moderator to look into each and every bug and then update its status and assignment of the bug to a particular person. Even though every system/tool has its own life cycle of the bug, here a generic bug life cycle (figure 1) can be described as follows.

When the bug is first reported, its status must be set to “Unconfirmed” as in the case of bugzilla. After its first review, the status can be updated to “New” or “Assigned” based on the previous bug history databases. Once the assignee has corrected the bug, he or she can update its status as “Resolved” by specifying how it was resolved i.e. “fixed”, “invalid (not a bug)”, “duplicate”, “won't fix”, and the (in) famous “works for me”. A resolved bug, will not be “closed” until someone else will confirm it out or it will be confirmed by the moderator. Once it is confirmed, the bug status becomes “verified” otherwise the status must be set to “Reopen”. It remains in “Verified” state until it is incorporated in the release version of the product and its status will be updated to “Closed”. The closed bug can be reopened with its status as “New” or “Unconfirmed”. There might be many bugs that are currently being fixed and some of bugs that are waiting for resolution set to be “closed”. A status report can be generated from time to time with number of bugs pending in each category as a status and emailed to the assignee as well as submitter of the bug. So the moderator or assignee can see the workload on individual and may postpone the bugs for sometime or reassign it to others for its resolution. In this way, the bug can be effectively resolved in an effective way.

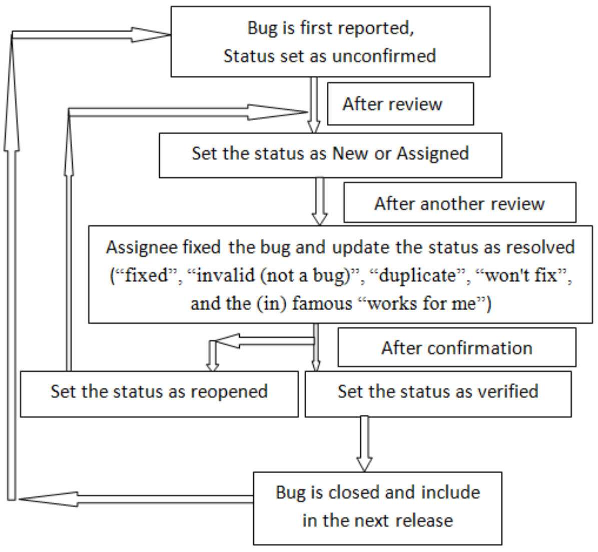

Bug is first reported,
Status set as unconfirmed
After review
Set the status as New or Assigned
After another review
Assignee fixed the bug and update the status as resolved (“fixed”, “invalid (not a bug)”, “duplicate”, “won't fix”, and the (in) famous “works for me”)
After confirmation
Set the status as reopened
Set the status as verified
Bug is closed and include in the next release

Fig. 1. Generic Bug Life Cycle

There are many bug tracking systems available in the industry to use. Bug tracking systems are also called as issue tracking system or issue reporting system or defect tracking system or defect reporting system, etc. Bug tracking systems are developed by open source community as well as closed source organizations as a proprietary software. Open source means the source code is being shared with everybody under the General Public License (GPL) policy. Anyone can contribute to the code voluntarily. There is no restriction towards the submission of code. The moderator will see and verify the reputation of the code submitter through his code submission pattern and its status can also be changed as moderator. While in closed source community, the source code is the property of the organization and the people who are outside the project may not be able to see/browse the code. Outsiders can submit only bugs through the feedback or e-mail to the sales/support person specified by the organization. In this article, we will consider the open source project development scenario except one or two closed source products.

Some of bug tracking tools are from open source communities and some of them from closed source communities or commercial organizations. There are organizations which can also provide support for the open source solutions. For example, Redhat Inc. [21] provides Red Hat Enterprise Linux (RHEL) and sponsors open source Fedora project [22] communities. Fedora comes under the category of open source while the RHEL is the commercial version which is supported and sponsored by Redhat.

3. Classification Criteria

The bug tracking system varies from general purpose to the specific purpose. The generic system will have many features while the specific purpose systems may have limited features. An suitable tool can be selected based on user's requirement. The requirement generally varies from the platform, support, size, reporting, tracking and other features of the project. These broad criteria can be further classified or categorized in three or more sub-categories as given in table 1.

Platform refers to license policy, architecture, operating system, web server, back-end database, programming language and clients. Support refers to language, multiple projects, web interface, database, email notification and localization. Size and usage refer to the size of the project, number of downloads, number of releases and number of clients/usage. Reporting refers to form based, email, attachments, web and feedback. Tracking refers to timely update status of the bug, graphs for number of pending bugs, assigned, fixed over a period of time in each category and search facility for new as well as existing bugs to know about their status. The other feature refers to any other feature that is not a part of the above mentioned categories.

Table 1: Classification of Criteria in Various Categories

Platform	Support	Size and Usage	Reporting	Tracking	Other Features
1. License 2. Architecture 3. Operating system 4. Web Server 5. Back-end database 6. Programming language 7. Client	1. Language support 2. Multiple projects 3. Web interface 4. Database 5. Email notification 6. Localization	1. Size of the project 2. Number of downloads 3. Number of releases 4. Number of clients/Usage	1. Form based 2. Email 3. Attachments 4. Web (online) 5. Feedback	1. Timely updates 2. Graphical display/ reporting 3. Search facility	Any other feature that may not come under any specific category

Many more differentiating features of bug/defect tracking systems may vary from issue based, wiki based, minimalistic support, integration with subversion, and may also include some other important features such as decision support parameters, source code repository, suggest co-change, automatic bug assignment/ reassignment, etc.

4. Comparative Analysis of Bug Tracking Systems

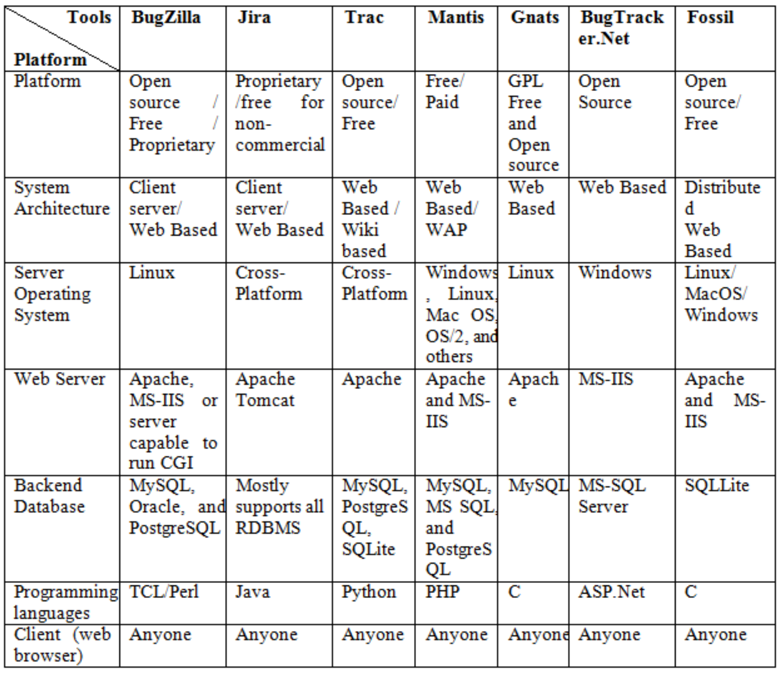
Plenty of bug tracking or reporting systems are available in the industry. We have chosen the most popular, most used and still in the improvement process, i.e. whose release keeps on coming and development work is going on for the enhancement of its feature and inclusion of the new features. Tools considered in the review are BugZilla [11], Jira [12], Trac [13], Mantis [14], GNats [15], BugTracker.Net [16] and Fossil [17].

4.1. Classification Based on Platform Characteristics

The comparison criteria based on platform characteristics are shown in table 2. In this table, most of these tools are available in open source environment. Mantis is closed source while its free version is available for download and use. Some of the tools are available as a licensed version for which support can be asked from the owner. Paid version of Jira comes in three different variants as Standard, Professional and Enterprise based on the size of the project and support.

Most of these tools work in web based environment while their early versions were available as client/server based environment. Trac is developed on wiki based architecture, i.e. anyone can see bugs, fix bugs and edit any part of it. Operating environment for most of the projects are cross-platform, i.e. they can be installed on any machine. BugZilla and GNats are available on Linux while BugTracker.Net is available on windows environment. The clients are browser based, so they are independent of the platform. For web server, most of the tools required apache/tomcat except BugTracker.Net which requires MS-IIS. Most of these tools support MySQL as a backend database except BugTracker.net which requires MS-SQL Server and Fossil requires SQLLite. These tools may vary on the basis of programming language used in coding. Tools are developed in C, Python, PHP, Perl, Java and C#.Net programming languages.

Table 2: Comparative Analysis of Features Based on Platform Characteristics



Tools / Platform	BugZilla	Jira	Trac	Mantis	Gnats	BugTracker.Net	Fossil
Platform	Open source / Free / Proprietary	Proprietary /free for non-commercial	Open source/ Free	Free/ Paid	GPL Free and Open source	Open Source	Open source/ Free
System Architecture	Client server/ Web Based	Client server/ Web Based	Web Based / Wiki based	Web Based/ WAP	Web Based	Web Based	Distributed Web Based
Server Operating System	Linux	Cross-Platform	Cross-Platform	Windows, Linux, Mac OS, OS/2, and others	Linux	Windows	Linux/ MacOS/ Windows
Web Server	Apache, MS-IIS or server capable to run CGI	Apache Tomcat	Apache	Apache and MS-IIS	Apache	MS-IIS	Apache and MS-IIS
Backend Database	MySQL, Oracle, and PostgreSQL	Mostly supports all RDBMS	MySQL, PostgreSQL, SQLite	MySQL, MS SQL, and PostgreSQL	MySQL	MS-SQL Server	SQLLite
Programming languages	TCL/Perl	Java	Python	PHP	C	ASP.Net	C
Client (web browser)	Anyone	Anyone	Anyone	Anyone	Anyone	Anyone	Anyone

4.2. Classification Based on Support Characteristics

The important component for comparison of tools is support features which are shown in table 3. The general characteristics for support features are Language, Web Interface, Maintenance support, Email notification and Localization. Most of the tools support multiple languages based on Unicode system. Only Fossil and GNats do not have the support of multiple language features. Again in the same line, most of the tools support web based interface. People are more fascinated to use browser based interface because it is independent of operating system environment. Fossil does not have the complete web based interface. Maintenance support is also available for free of cost or with a paid service with nominal charges. Whenever, a new bug or update is reported, an email will be automatically sent to all users who are in the registered mailing list with the project. The email will also be sent to all registered users even when there is a small update. Bug localization, i.e. finding the most relevant part of source file hierarchy of software in the bug repository database is another important feature provided by BugZilla, Trac and Mantis while others does not have.

Table 3: Comparative Analysis Based on Support Characteristics

Tools / Support	BugZilla	Jira	Trac	Mantis	Gnats	BugTracker.Net	Fossil
Language Support	Yes	Yes	Yes	Yes	No	Yes	No
Web Interface	Yes	Yes	Yes	Yes	Yes	Yes	No
Maintenance Support	Yes	Yes	Yes	Yes	No	Yes	Yes
Email Notification	Yes	Yes	Yes	Yes	Yes	Yes	No
Bug Localization	Yes	No	Yes	Yes	No	No	No

4.3. Classification Based on Size and Usage Characteristics

Another component of the feature to compare is the size/usage of the tools. In this category, size of the project, support of multiple projects, number of downloads, number of releases and clients per usage are the important parameters for which the statistics can be collected over different time period that may be from the date of first release to the recent date of release. The authors have collected statistics by browsing the corresponding project website wherever it is available as shown in the table 4.

Table 4: Year of First and Latest Release

Tools / Release Status	BugZilla	Jira	Trac	Mantis	Gnats	BugTracker.Net	Fossil
First Release	1998	2004	2006	2000	1992	2002	2006
Latest Release	2011	2010	2010	2010	2005	2011	2011

Looking at the table, conclusion can be drawn about the current development status of the project, i.e. whether it is active or dead. It will also inform about the life of project that will be helpful, regarding its integrity and maturity.

The numbers of releases are shown in the table 5. Here, it is clearly shown that there are many releases for the project BugZilla. The reason may be the frequent releases which occur and early fixes are being incorporated and updated version is being made available to the users.

Table 5: Year Wise Number of Releases

Tools / Year	BugZilla (Version 2.0 onwards)	Jira (Version 3.0 onwards)	Mantis (Version 0.18.0 onwards)	Bugtracker.Net (Version 2.5.6 onwards)
2011	4	1	-	-
2010	14	2	5	14
2009	15	1	6	28
2008	10	1	9	36
2007	7	5	8	15
2006	8	3	11	-
2005	9	4	10	-
2004	3	1	6	-

*Gnats, Trac and Fossil, this statistics is not available

The number of downloads are also collected. The time based historical downloads are available for the BugTracker.Net only. The number of downloads are quite encouraging that shows the usage of the tool in the industry are increasing day by day. The trends can be seen in the following fig. 2. The trend is increasing because for the year 2011(till March 15, 2011), the data is for only 2 and half months. This trends shows the popularity of the tools in the industry.

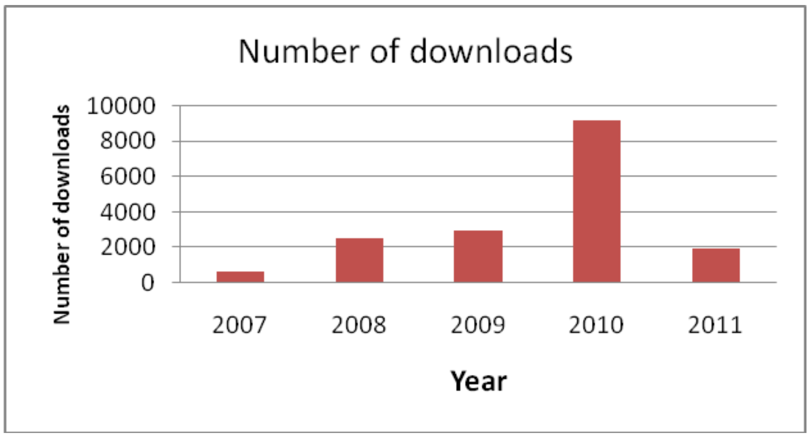


Fig. 2: Number of Downloads for BugTracker.Net

4.4. Classification Based on Reporting Characteristics

Another important component is reporting of bug to the project. This can be done by either of these methods, i.e. form based, email, attachments, web (online) and feedback. This is shown in table 6.

Table 6: Classification Based on Reporting Characteristics

Tools / Reporting	BugZilla	Jira	Trac	Mantis	GNats	BugTracker.Net	Fossil
Form Based	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Email	Yes	Yes	No	Yes	Yes	Yes	No
Attachments	Yes	Yes	No	Yes	Yes	Yes	No
Web (Online)	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Feedback	Yes	Yes	Yes	Yes	Yes	Yes	Yes

A well designed reporting feature will attract users in submitting the bug effectively so the project will not miss any of the failure report instance. For example, in the Eclipse project, the submission of bug reports can be done automatically as well as its usage. The user will click only once in the checkbox and later one can click on a button to submit the bug to the project that will get updated in the system.

Therefore, if the bug reporting system contains all these methods, then it becomes very efficient to know about the bug because user will not hesitate in submitting the bugs if he follows a click and proceed approach. BugZilla, Jira, Mantis and BugTraccker.Net have almost every reporting facility but Trac and Fossil are lagging behind in attachments and email based reporting.

4.5. Classification Based on Tracking Characteristics

The tracking feature will contain subcategories likely timely updates, graphical display and search facility. Most of the systems have these functionalities as shown like in table 7. This can be further categorized to one more level. Whenever the new updates are received, the registered members will receive an email regarding the updates as well as new release of the system. The graphical and charting facility is one of the very important features that tell user about the number of pending bugs, number of open bugs, number of closed bugs, time taken by each bugs in the form of interactive charts and estimated time to resolve the bug. To track the bug, the tools have the facility to search for a bug that is already submitted in the system. This will be helpful in identification of duplicate bugs [4]. Current day's product must have all these facility with dynamic graphical and charting options. The search facility is also one of the important considerations. Generally, the systems have full text search facility on title, description and summary of the bug report.

Table 7: Classification Based on Tracking Characteristics

Tools / Tracking	BugZilla	Jira	Trac	Mantis	GNats	BugTracker.Net	Fossil
Timely updates	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Graphical display/ reporting	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Search facility	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Most of the systems are able to provide the search facility on any of the parameters such as bug-id, severity, priority, time of submission, author, assignee, number of reassignment/history of assignment, etc. There are number of parameters on which the search facility can be provided. Jira has the limited parameters search options but powerful. Mantis provides the bar chart based result of the search to show the progress of a particular bug.

4.6. Classification Based on Other Features

Any other feature may incorporate many more features that are very important but may not have one special category. These are bug localization, change prediction, browsing of source code repository, version control, and Subversion facility. Some of the systems have the capability of these features but they are not mature.

Overall the above mentioned subcategories can be further sub divided into specific subgroups. It will enable to provide a better picture of the bug reporting and tracking system. It becomes very helpful in choosing a tool that will meet the user specified criteria and need.

5. Issues and Challenges

There are many issues that arise after studying these categories and sub-categories. The user may be used to compromise or willing to adopt the combination of one or more systems that require extra effort, resource and duplication of some of the task. Challenges are in front of user in choosing a better system that will be able to cater the need of the project manager or mentor or developer in tracking the progress and submission of reporting system. The systems are not capable of providing the source code change prediction, automatic code browsing, automatic bug assignment/reassignment, LDAP (Lightweight Directory Access Protocol) support for single sign-on, identification of actual module/files for bug fixing, intelligent build and release systems and specialized visualization capability. Various research efforts have been made in the above mentioned areas but they are not integrated as a single system. The system may also be capable of doing things automatically, i.e. ask less input from the user and produce more values to the system as well as to the user. The user can be either a developer or a manager. To the best of our knowledge, no bug tracking system talks about bug prediction or remaining number of bugs yet to be removed from the software and bug complexity from developer's point of view/bug distribution. In the following section we are proposing Bug tracking and reliability assessment system by incorporating the above concern.

6. Proposed Bug Tracking and Reliability Assessment System(BTRAS)

By looking into the complexity of the bug reporting features or various parameters required during the reporting of the bug, user may or may not submit the bug report. It is his choice or right to report the bug with its well defined and explained description. There should not be any ambiguity in reporting bug. If bug reporting system is perfect, tracking the buggy code or module will be much easier. The review of various bug or issue tracking system tells us that the following specific features need to be in the proposed system.

It should be web browser based system that will satisfy the users from diverse platforms, i.e. independent of the operating system. The multilingual support enables in the system by using the Unicode based system. It must have the support of multiple projects, the project may be of any platform or language or size. It should have the browser based interface with bug localization feature and able to predict the source code change through the repository. The bug reporting must be enabled in all possible ways and have the capabilities to contain the historical records related to reported bug, fix, daily downloads, usage, and many more. Release and version history of the source code needs to be maintained which will be helpful in browsing the source code changes, predicting the future bugs, source code change prediction, etc. Ideally, the system should have the capability of wiki-web based system, with MySQL or PostgresSQL as a backend database, Java, C, PHP, Python as a programming language and client will be a web browser. The web based system can work effectively in distributed environment. In addition to the above mentioned features, the proposed Bug Tracking and Reliability Assessment System (BTRAS) in figure 3 will be able to take decision by incorporating the mathematical models, reliability models, statistical models and machine learning techniques. The task will be accomplished automatically. We categorized BTRAS into two steps process, i.e. reporting process and tracking the progress / reliability process. We will describe these processes one by one.

The reporting process will have the following guidelines

i. It should have all the ways and means of reporting the bug, i.e. by using minimum field form, email support, attachments, web based and feedbacks, etc. It should have the capability of asking minimal things from the user and generate automatically most of the parameter's values using mathematical models, reliability models, statistical models and machine learning techniques. This will encourage the user to report the bug.

ii. It should have automatic assignment facility by showing the list of probable fixers of this type of bug [25] that will save the moderator's time in searching the developer.

iii. It should be able to identify the duplicate bug. If it is a duplicate, it must be able to provide the previously submitted/fixed bug to the user.

iv. It must be capable to provide the LDAP support to enable single sign-on process. The status of the user from reporter to developer and developer to moderator can be updated based on his previous activities. Single user-id sign-on must be used to report/fix/monitor the bug in/from the system.

v. User profiling can be done on the basis of their bug removal efficiency. The threshold values of number of submitted bugs, number of assigned bugs and number of removed/fixed bugs by a particular user should be able to put the user in the privileged category of the user. The privileged category of user is the active and reliable user who is continuously contributing in the development of the system. It must have the database of all the developers who are useful for fixing of the future bugs. It means that if the bug/fix is submitted by these groups of user, this should be accepted immediately and requires minimal interventions in incorporating their solutions in the next build or release.

vi. It should be able to provide the exact classification of errors/bugs/patches/features using the past report submitted. The classifiers can be developed by using the supervised classification techniques.

vii. It should have the web based interface capability as mentioned earlier. The availability of the system must be ensured on 24*7 manners to avoid reporting at any point of time.

viii. It should be fully integrated with version control and able to communicate with most of the available version control system which will be helpful in browsing in source code history, predicting source code change and future bug prediction using suitable data mining or statistical techniques.

ix. It should have the capability to provide the web services for integration with the other system. Support of web services will be helpful in sharing the data from the diverse platform.

Tracking the progress of fixing the bug process/reliability assessment requires the following in the system:

i. It should be able to provide current status of the bug in a graphical way. Status of the bug can be depicted using some color coding scheme. For example, when the bug is first submitted, the color of status can be set as Red, if the bug is assigned and in progress; the color of status can set as yellow. If the bug is fixed, the color of status will be set to green. The color coding scheme can use some more colors based on the value of status. A very powerful phrase says "Picture speaks thousands of words", It means by looking at the color of status, the state of the bug can be decided very easily.

ii. It should be able to estimate the time taken by the developer to resolve the bug. The system must be able to calculate the time to resolve bug using the history of related bugs or developer fixing the bug.

iii. It should able to produce a timeline graphs for time taken and remaining time required to fix the bug. By looking at the charts, it can be easily figures out that when moderator can plan for the next release.

iv. Mathematical models can be incorporated in the system that will predict the number of bugs yet to be fixed and their complexity. On the basis of reported bugs, the priority and severity can be classified from developer's perspective. The complexity of the bugs and their distribution will be helpful in determining the optimal effort required in assigning and fixing of bug.

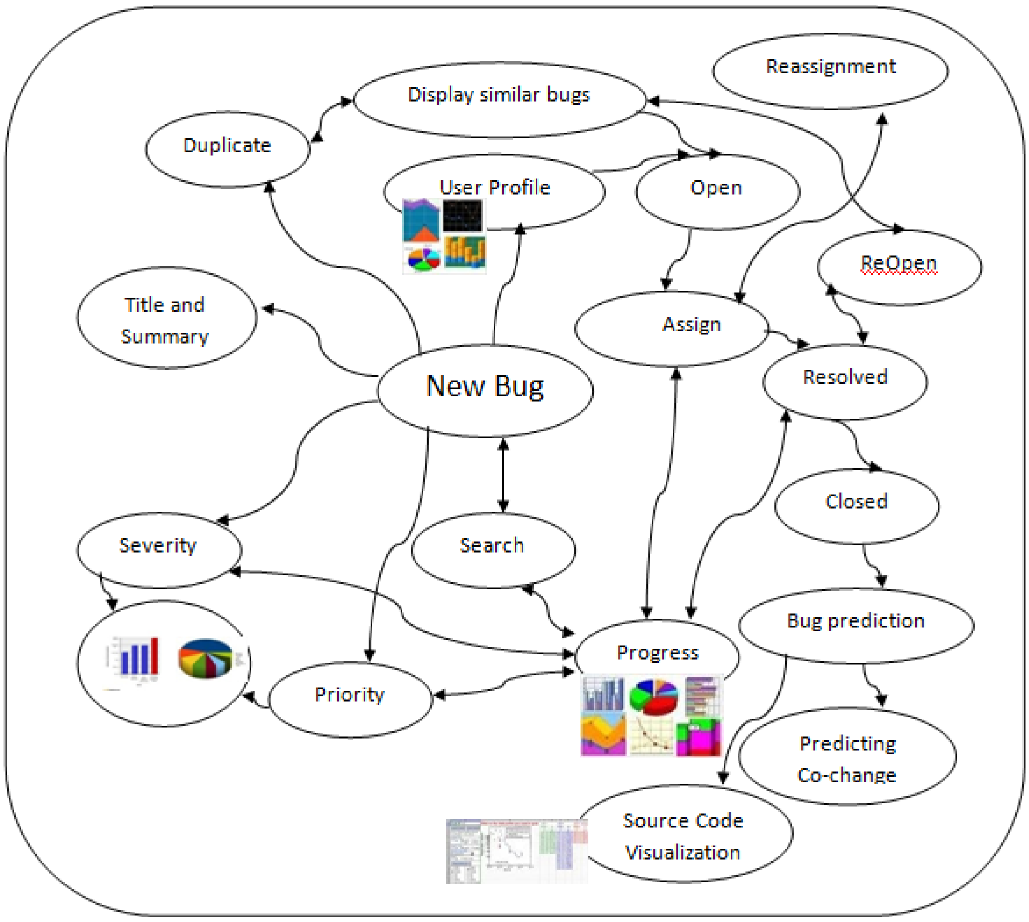


Fig. 3: Proposed BTRAS Tool

v. Imperfect debugging and error generation can also be incorporated along with the bug prediction because there may be chances of introduction of some new bugs during bug fixing or the bug may not be perfectly fixed due to incompetency/lack of complete knowledge about the bugs.

vi. Software reliability growth models [18, 19, 20, 26 and 27], other statistical and machine learning models [28, 29 and 30] may be incorporated in the bug prediction. Bug predicting tool, which may be an integral part of the bug tracking and assessment system, will able to provide the useful information to the active user or developer in making plans to remove of bugs, tracking, and releases, etc.

vii. It should be able to identify the module for which bug/fix is related in a graphical way. It should have the capability of source code visualization and co-change prediction facility. The graphic based source code browser will be helpful in co-change prediction.

viii. It should be able to manage the change management system, build release management system, and intelligently automatic reporting system whenever there is some submission.

ix. Automatic email notification whenever there is any significant updates or submission of reports. It enables the users to get updates that are fixed recently.

x. It should support the inclusion of test cases in the system itself. It enables the system to check for the left out bugs due to this fix.

xi. Role based user interfaces should be provided based on access label of the user.

xii. Intelligent search facility should be provided in the system which will be helpful in detecting the duplicate bugs or related bugs.

xiii. It should be able to provide the graphical view of number of clients, number of releases, number of downloads, current as well past build releases with respect to time. It should also be able to calculate some of the derived parameters behind the box and display on the user's screen to facilitate the user regarding the role and value of those parameters.

xiv. Online help facility must be hyperlinked with the contents on every page.

xv. Installation and configurations should be plug and play manner using a GUI based installer.

xvi. The required component must be packaged in the release versions/package.

If the above mentioned process/features included in the system, the reporting, assignment and reliability assessment finding of the bug will become quite easier.

7. Conclusion

A good automated bug-tracking tool should streamline the process of reporting, managing and fixing issues. In this paper, authors reviewed some of the bug tracking systems and then present a comparative analysis of different bug tracking system based on different classification criteria. It will help the developers to choose bug tracking tool as per their requirement and constraint. We have also presented a theoretical framework for developing bug tracking and reliability assessment system (BTRAS) based on different consideration mentioned in section 6. The proposed BTRAS will assess the reliability of software and give information about bug complexity, bug distribution to developer apart from reporting and tracking.

7.1. Future Research Agenda

The given lists of the categories/sub categories are quite exhaustive. The weights can be assigned based on the requirement and cumulative score can be calculated to find out the best features for the implementation and use. The proposed tool once developed, will able to deliver the required and useful features, necessary for the developer as well as the triager. To ease out the process, the tools can be further incorporated the visualization and machine learning techniques by which assignment of the bug to the concern person can be done automatically during the reporting of the bug itself. The status can also be updated whenever there is a new submission of fix or build or release takes place.

Acknowledgement

First author acknowledges with thanks the financial support provided by University Grants Commission, India, under the project No.F.8-1(77)2010(MRP/NRCB).

References

[1] 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, 2009.

[2] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, “Quality of bug reports in Eclipse”, In eclipse’07, Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange, pages 21–25, 2007.

[3] N. Bettenburg, S. Just, A. Schroter, 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] “Top 10 Defect tracking software vendors revealed”, http://www.business–software.com/DefectTracking, 2011.

[10] Trajkov Marko, Smiljkovic Aleksandar, “A Survey of Bug Tracking Tools: Presentation, Analysis and Trends”, aleksland.com/wp-content/uploads/2011/01/Survey.pdf, 2011.

[11] The BugZilla, http://www.bugzilla.org/releases/ (bugzilla, accessed on 17/03/2011)

[12] The Jira, http://confluence.atlassian.com/display/JIRA/JIRA+Release+Summary (JIRA, accessed on 17/03/2011)

[13] The Trac, http://trac.edgewall.org/

[14] The Mantis, http://www.mantisbt.org/bugs/changelog_page.php (mantis, accessed on 17/03/2011)

[15] The BugTracker.Net, http://btnet.codeplex.com/releases/view/14423 (BugTracker.Net, accessed on 17/03/2011)

[16] The Gnats, http://www.gnu.org/software/gnats/ (GNats, accessed on 17/03/2011)

[17] The Fossil, http://www.fossil-scm.org/

[18] Singh V.B., Kapur P.K. and Abhishek Tandon, “Measuring Reliability Growth of Software by Considering Fault Dependency, Debugging Time Lag and Irregular Fluctuation”, ACM SIGSOFT, Software Engineering Notes Vol. 35 , No.3 pp.1-11, May 2010.

[19] Kapur P.K., Garg R.B. and Kumar S., “Contributions to Hardware and Software Reliability”, World Scientific, Singapore, 1999.

[20] Pham H., Software reliability, Springer, 2000.

[21] The Redhat Inc., http://www.redhat.com/.

[22] The Fedora project, http://www.fedoraproject.org/.

[23] The eclipse Bug Repository, http://bugs.eclipse.org/bugs.

[24] Moin, H.A. and Khansari, Mohd., “Bug Localization Using Revision Log Analysis and Open Bug Repository Text Categorization”, P. Agerfalk et al., (Eds.) OSS2010, IFIP AICT 319. Pg. 188-199, 2010.

[25] Baysal, O., Godfrey, M.W. and Cohen, R., “A Bug You Like: A Framework for Automated Assignment of Bugs”, In proceedings of 17th IEEE International Conference on Program Comprehension (ICPC), 2009.

[26] Smidts, C., Li, B., Li, M. and Li, Z., “Software Reliability Models. Encyclopaedia of Software Engineering”, Vol. 2. J.J. Marciniak, Ed., 2nd Edition, New York, John Wiley & Sons Inc., 2002. PP. 1594-1610.

[27] Yamada, S., Ohba, M. And Osaki, S., “S-Shaped Reliability Growth Modelling for Software Error Prediction”, IEEE Transactions on Reliability, Vol. 32, 1983. pp. 475-478.

[28] Lyu, M.R., “Handbook of Software Reliability Engineering”, McGraw Hill, 1996.

[29] Zou, F. and Davis, J., “Improving Software Reliability Modelling Using Machine Learning Techniques”, International Journal of Software Engineering and Knowledge Engineering, 2008 Vol. 18(7), PP. 965-986.

[30] Costa, E.O., de Souza, G.A., Pozo, A.T.R. amd Vergillo, S.R., “Exploring Genetic Programming and Boosting Techniques to Model Software Relaibaility”, IEEE Transcations on Reliability, 2007, Vol. 56(3), PP. 422-434.

Authors

Dr. V. B. Singh is currently as Associate Professor in the Department of Computer Science at Delhi College of Arts and Commerce, University of Delhi, Delhi (India). He received his M.C.A. degree from M. M. M. Engineering College, Gorakhpur, U.P. (India) and Ph.D. degree from University of Delhi in Software Reliability. He has published more than twenty five research papers. His research interests include software testing and software reliability engineering. He is member of ACM.

Krishna Kumar Chaturvedi is a Scientist, Computer Applications at Indian Agricultural Statistics Research Institute (Indian Council of Agricultural Research), New Delhi, India. He involved in the development of Agriculture Data Warehouse, Expert System on Wheat Crop and other Application Software. He published around fifteen papers in various journals and conferences. His research interests include application of Data Warehousing, Data Mining, Software Configuration Management and Open Source Software. Currently doing Ph.D. in Computer Science from Department of Computer Science, Delhi University, New Delhi, India.