Volume 5, Issue 5, May 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Bug Tracking and Project Management with Crawling, Parsing & Tracking

Manjusha Shelke*, Rahul Kapse
Computer Science, Mumbai University
Maharashtra, India

Abstract— The number of issues related to the software projects is increasing and the developers have started to use bug tracking systems in order to manage their bug reports more competently. However, acquisition of useful information from the huge and unorganized set of reports is still difficult problem because most of bug tracking systems provide the data via web interfaces. So we have developed a bug tacking and project management solution with the help of data mining technique such as baysian theorem. Application works in three steps crawl, parse and filter the bugs. Also most of the bug tracking system doesn't have project management and audit trail facility.

Keywords— bug,crawling,JSONparsing,filtering,report generation

I. INTRODUCTION

As software programs become increasingly huge and composite, it is more important to improve the quality of software conservation. Numerous software programs rely on bug reports to correct errors in maintenance undertakings. Bug tracking systems were developed to guide maintenance activities of software developers. However, due to the extreme number of duplicate bug reports, developers employ abundant time to recognize these bug accounts. In order to save developer's time in software maintenance, we propose a bug rule centered classification technique to catalogue bug reports. By utilizing developer feedback mechanism in the technique, it differentiates duplicate and valid bug reports and is expected to improve the accuracy of bug reports retrieval. With huge amounts of software projects, it becomes more difficult to check bugs from a large number of source code files. Essential activities in software maintenance include bug reporting and fixing. Maintenance activities of many software projects rely on bug reports to correct defects in source code files. Recently, plenty of software companies use bug tracking systems during software development to track bugs, where developers can correct these bugs quickly. Bug tracking systems allow developers to upload, define and comment on bug reports. Bugzilla, a web-based bug tracking tool, is a famous open source bug tracking system. Developers can acquire bug reports to Bugzilla at any time. These bug reports are used to guide corrective maintenance activities and improve software quality. Free-form text fields of bug reports include title and description. Bugzilla also allows developers to track status and comment of bug reports and submit related attachments such as screenshots. Therefore, in this study we propose a technique to classify bug reports for identifying duplicate reports.This classification technique based on bug rule, combines textual resemblance and taxonomy algorithm. The classification involves identifying duplicates and invalid bug reports to give developers a better result of bug reports retrieval. In order to help developers to find applicable bug reports, we apply developers' feedback to refine the result of classified bug reports. In practice, giving feedback is a generous of filtering between developers and incoming bug reports.

A bug tracking system or defect tracking system is software that is planned to help keep way of described software bugs in software growth efforts. It may be considered as a type of issue tracking system. Several bug tracking systems, as those used by most open source software projects, permit users to enter bug reports directly. Other systems are used only within in a company or organization doing software development. Typically bug tracking systems are combined with other software project management applications.

Having a bug tracking system is exceptionally valuable in software development, and is used extensively by companies emerging software products. Constant use of a bug or issue tracking system is measured one of the "trademarks of a upright software crew". Details may include the time a bug was reported, its severity, the erroneous program performance, and details on how to replicate the bug; as well as the identity of the person who reported it and any programmers who may be functioning on fixing it. document is a template. An electronic copy can be downloaded from the Journal website. For questions on paper guidelines, please contact the journal publications committee as indicated on the journal website. Information about final paper submission is available from the conference website.

II. LITERATURE SURVEY

Many researchers have investigated what factors aid or indicate the quick resolution of bugs. Hooimeijer and Weimer 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.Aranda and Venolia have examined communication that takes place between developers outside the bug tracking system.Runeson et al. proposed an approach to identify duplicate bug reports by using Natural

Language Processing (NLP) techniques. They developed a prototype tool for evaluating and analyzing bug reports. The evaluation shows that about two-thirds of the duplicates can be found possibly by using the NLP techniques. Since these bug reports are not classified in their work, it is not easy to find related bug reports. Moreover, if some reports among duplicates which the prototype found are invalid, developers spend much time in finding appropriate valid bug reports.

Some web-based tracking systems have been used to provide useful troubleshooting advice. However, it is still difficult to process semi-structured data in bug tracking systems. H. M. Tran et al. proposed a semantics-based bug search system for extracting semi-structured data from other bug tracking systems.

N. Jalbert and W. Weimer proposed a system that automatically classifies duplicate bug reports for saving developers' time. They applied surface feature, texture semantics, and graph clustering to detect duplicate bug reports. They showed that the system reduced development cost by filtering out 8 percent of duplicate bug reports. Their study described that inverse document frequency was not useful in this task and title similarity was the most useful element for detecting duplicate bug reports. However, the work does not recognize that experienced developer' knowledge is also very effective for classifying bug reports.

While many detecting techniques of duplicate bug reports are exploiting only natural language information to implement the detecting task, X. Wang et al. present a new method which combines natural language information and execution information. When a new bug report is submitted, the method retrieves the most similar bug reports to the new bug report by comparing natural language information and execution information of the report with those of existing bug reports. Then developers examine the suggested bug reports to determine whether the new bug report duplicates an existing bug report. They showed that the method detected from sixty-seven to ninety three percentage of duplicate bug reports in the Firefox bug repository. The number of detected bug reports is more than that of the method which uses only natural language information. However, even if execution information helps the method to improve the accuracy of detecting duplicate bug reports, it leads to increase in workload of the method.

Table 1 Different bug tracking & project management application

Application	**Bug reporting**	**Filtering**	**Project Management**
Bugzilla[4]	Yes	No	No
Mantis[6]	Yes	No	No
Trac	No	No	Yes
Red mine[5]	No	No	Yes

Before we move to our proposed mechanism will first understand the process of bug tracking i.e. life cycle of bug.

The typical life of a bug goes something like this: A tester (or user) creates a new bug. New bugs are formed either as unverified, new, or allocated. Normally, a developer will accept a bug (or assign it to someone else). When the developer has fixed the bug, they can mark it as fixed, postulating how it was resolved: fixed, invalid (not a bug), duplicate, won't fix, and the (in) famous "works for me." A fixed bug isn't formally closed until someone from QA (quality assurance) checks it out. General Workflow of bug tracking system based on comments of developer and tester, if comments are long than priority of bug is high, Which is technically incorrect. Comments may be related to understanding of bug.

- Bug description and no provision for extra information of bug such as screen shots etc.
- Some tools don't have collaboration facility for communicate.
- Tools have the facility that end user can report bug to tool.
- Bug categorizations are not appropriate manner.
- Bug tracking between developer and tester is also not in proper manner.
- Duplication of bugs. There is no proper way to sort the bugs.
- Most of the bug tracking tools not have natural language processing techniques.
- Bug report generated by most of the tools is not good and not in proper manner. As reports among duplicates which the prototype found are invalid, developers spend much time in finding appropriate valid bug reports.
- Difficult to process semi-structured data in bug tracking systems.
- Most of the tools work does not recognize that experienced developer knowledge is also very effective for classifying bug reports.

III. PROPOSED SYSTEM

Recently, the number of issues related to the software projects is increasing and the developers have started to use bug tracking systems in order to manage their bug reports from the huge and unorganized set of reports is still difficult problem because most of bug tracking systems provide the data via web interfaces. Thus, in order to process and analyze the issues reported and discussed in bug tracking systems, raw data should be crawled from the web pages and organized with a proper schema in advance.

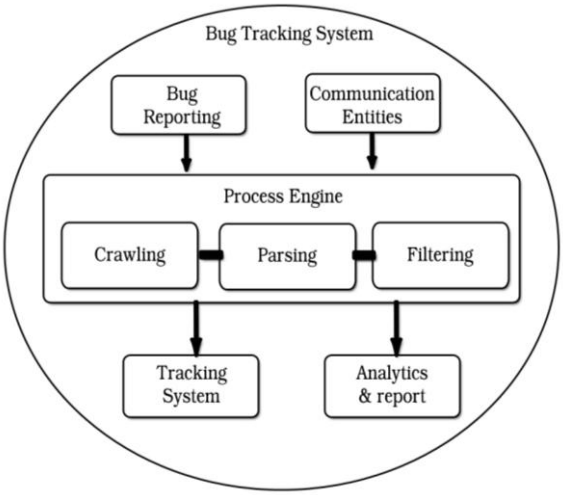

Figure 3.1 High level architecture

Generally, the extraction cost or complexity of bug tracking systems is considered to be high because of the difficulties in predicting the structure of the bug reports. Even if they are crawled, some parts of data are difficult to be identified except trivial fields such as priority or severity.

We describe the overall process for extracting data from bug tracking systems as follows –

1. Bug reporting
2. Communication entities
3. Process engine
   - Crawling
   - Parsing
   - Filtering
4. Tracking system
5. Analytics and report

Our proposed system main focus is project management and bug filtering. For project management consist of user management, role based access control system, project detail level management and project assignment management. This is most important module of bug tracking system. This very helpful to track the bug status. Communication entities consist of who is involved in the current product testing i.e. developer list, testers list and project managers. Our project management provides removes the communication gap between manager, developer, TL & tester.Our main focused concept is Process Engine, which crawl, parse and filter the bug data.Process engine is main module of bug tracking system. Process engine consist of Crawling, Parsing and filtering modules in system[1]. Crawling gets the bug list and classifies it by using Bayes probability and removes the duplicate. Crawling returns or maintains result in JSON format that need to parse it by JSON Parser. Here Parser module of process engine parse it accordingly and provides data to filtering module.

Process engine consists of three different main modules such as crawling, parsing and filtering. Crawling is first and very important module of bug tracking system. Crawling takes bug list as input and returns JSON data as output.

It works on bug list, which consist of all reporting data. There may be possibility that tester can make the mistake while reporting the bugs. So system should be intelligent that it need to be classifying bugs by using classification technique.

Bayesian theorems used for make the decision that bug are duplicate or not. Bayesian theorem is a probabilistic efficient algorithm for make the decisions.

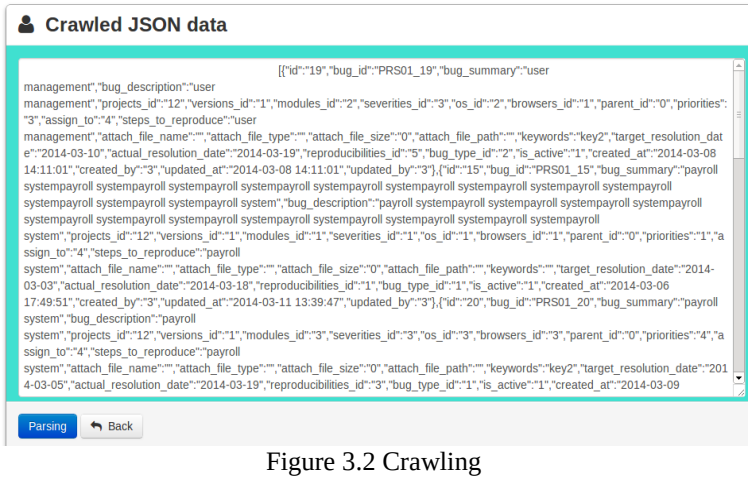

Figure 3.2 Crawling

Crawling yields result in JSON format after grouping by crawling. So that JSON need to parse for reading of data. JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is stress-free for individuals to read and write. It is easy for machines to analyse and generate. It is constructed on a subgroup of the JavaScript Programming Language, JSON is a script format that is totally language independent but uses conventions that are familiar to computer programmer of the C-family of languages, counting C, C++, C#, Java, JavaScript, Perl, Python, and several others. These properties make JSON an ideal data-interchange language.A group of name value pairs in various languages; this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array. In most languages, this is comprehended as an array, vector, list, or order. Our aim is to take JSON data to array which is handling by application.

Figure 3.3 Parsing

Filter matches bug occurrences against a set of norms. By defining a filter, one can first-rate bug instances for distinct treatment; for instance, to exclude or include them in a report.
Filtering caries array data and then match each bug details with each calculate its matching percentage with probability and accordingly delete the bug whose is crossing the threshold limit calculated by the system.

```
static void php_similar_str(const char *txt1, int len1, const
char *txt2, int len2, int *pos1, int *pos2, int *max)
{
char *p, *q;
char *end1 = (char *) txt1 + len1;
char *end2 = (char *) txt2 + len2;
int l;
*max = 0;
for (p = (char *) txt1; p < end1; p++) {
for (q = (char *) txt2; q < end2; q++) {
for (l = 0; (p + l < end1) && (q + l < end2) && (p[l] == q[l]);
l++);
if (l > *max) {
*max = l;
*pos1 = p - txt1;
*pos2 = q - txt2;
}
}
}
}
```

Algorithm 1. Filtering

After crawling and parsing application will filter the bug data and discard those bug who are matching more than 60% (this general assumption made by use for filtering).

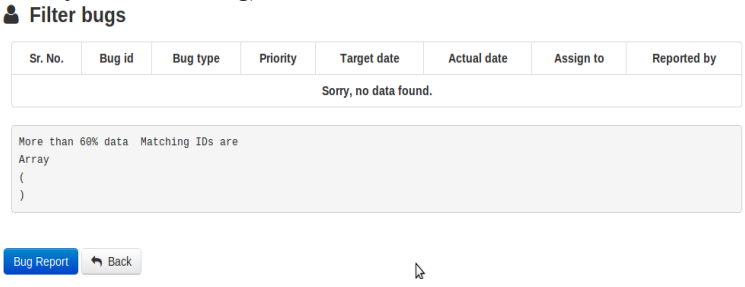

Figure 3.4 Filtering

System tracks the project management with proper communication with project manager, developer, tester and team leader. This Issue Tracking Life Cycle is an adopted from other Bug Life Cycle. The system design had been proposed according to the user requirements and the detail new system. Each issue in the system will have a related status, precedence, and severity. The status indicates the current progress with respect to an issue.

Once a bug is found, an issue is generated in the issue tracking system; also the issue will be in Open state. When a developer starts functioning on the issue, the issue will transfer to In-progress state. When the developer offers a solution, the issue is then; the fix needs to be confirmed. If the confirmation fails, the issue will be re-opened and will drive back to the Open state. If the authentication is passed, the issue can be closed, which signposts that the bug is fixed. Priority specifies how soon the issue needs to be appeared to. Extraordinary priority issues are to be attended before the others. Severity specifies the significance of the issue. In other words, severity indicates the influence of the issue on the system.

Each communication entity such as manager, developer, TL or tester has its own dashboard. They are responsible for manage their assigned bugs. Each entity can give a feedback and take respective action on assigned bug such as open, close, in progress & resolved.

Assigned bug status are shown by application according to graphical and version wise, so communication entity can take action on it no messy work.

Figure3.5. Dashboard bug status according to project version

This way we have achieved the project management and bug tracking system with process engine such as crawling, parsing and filtering.

IV. CONCLUSION

By we proposed a bug rule based bug report classification technique with feedback for duplicate Identification and bug reports retrieval. By performing textual analysis, clustering and classification, the bug reports which are crawled in bug tracking system have been classified into a taxonomy structure model. In order to improve the accuracy of duplicate identification and bug report retrieval, we applied the crawling, parsing & filtering process to decide these bug reports. Developers decide whether a bug report is a duplicate, the report is valid and it belongs to an appropriate bug category in the filtering process.

REFERENCES

[1] Yongsoo Yuk, Woosung Jung, Comparison of Extraction Methods for Bug Tracking System Analysis, National Research Foundation of Korea(NRF)

[2] Tao Zhang & Byungjeong Lee*, A Bug Rule based Technique with Feedback for Classifying Bug Reports, 2011 11th IEEE International Conference on Computer and Information Technology.

[3] Thomas Zimmermann, Rahul Premraj, Jonathan Sillito & Silvia Breu, Improving Bug Tracking Systems.

[4] https://www.bugzilla.org/

[5] http://www.redmine.org/

[6] https://www.mantisbt.org/

[7] Tomi Prifti, Sean Banerjee, Bojan Cukic, Detecting Bug Duplicate Reports through Loca References

[8] Anahita Alipour, Abram Hindle and Eleni Stroulia, A Contextual Approach towards More Accurate Duplicate Bug Report Detection, IEEE, 2013.

[9] Naresh Kumar Nagwan & Pradeep Singh, Weight Similarity Measurement Model Based, Object

[10] Oriented Approach for Bug Databases Mining to Detect Similar and Duplicate Bugs, International Conference on Advances in Computing, Communication and Control (ICAC3'09).

