Achieving Code Quality using Code Review System

Snehal Bhosale
1st author's affiliation
SVPM College of engineering,
Malegaon(Bk)

Shraddha Bhosale
2nd author's affiliation
SVPM College of engineering,
Malegaon(Bk)

Varsha Bhalerao
3rd author's affiliation
SVPM College of engineering,
Malegaon(Bk)

Ketan Bhagwat
4th author's affiliation
SVPM College of engineering,
Malegaon(Bk)

ABSTRACT
The purposes for code review are as diverse as the environments in which they are conducted. However, almost all code reviews have these goals in common: 1. Defect-free, well-documented software. 2. Software that complies with enterprise coding standards. 3. Teaching and sharing knowledge between developers. Other objectives often include: maintainability, security, and consistent end-user documentation, adequate comments in code, complete unit tests, and scalability.

General Terms
Your general terms must be any term which can be used for general classification of the submitted material such as Pattern Recognition, Security, Algorithms et. al.

Keywords
Keywords are your own designated keywords which can be used for easy location of the manuscript using any search engines.

1. INTRODUCTION
Code review is the single greatest way of noticing and killing bugs, increasing overall understanding, fixing design problems and learning from one another. A code review involves one or more developers examining source code they didn't write and providing feedback to the authors, both negative and positive. Ideally the reviewers are completely disengaged from the project they are reviewing as this maximizes objectivity and ensures the code is readable and maintainable even by those not already well-versed in that project. Typically the reviewers will have a standard checklist as a guide for finding common mistakes and to validate the code against the company's coding standards. As IEEE Transaction On software engineers states that, 'Inspection of a 20000 line program at IBM saved more than 85% of programmer effort by detecting major defects through code review instead of testing'[1].

Maintainability is generally achieved by code organization and adequate comments. A reviewer can provide the ignorance and objectivity necessary to ensure these goals. Code review can facilitate the communication of institutional knowledge as it relates to code written by the newbie. Experienced team members have the opportunity to impart their wisdom and advice. Code reviews are just one part of a more broad-reaching inspection program. As defined by the IEEE Standard Glossary of Software Engineering Terminology, an inspection is a formal evaluation technique in which software requirements, design, or code are examined in detail by a person or group other than the author to detect faults, violations of development standards, and other problems[5].

2. HISTORY
Historically the process for conducting code review was pretty "anti-agile". Originally software inspection technique introduced by Michael Fagan in 1976 [2] [3]. In that code inspection was heavyweight code review process that led to an entire generation of software developers who believed meeting were necessary in order to review code. Highest misconception is that meetings are stated by Fagan, but Lawrence Votta of AT&T Bell Labs [2] was not convinced. His study showed that if developers read the code before the meeting in order to find defects, actually having a meeting will only increase the total defects found by 4%. Now a day's code review tools with agile process model is used everywhere such as peer code review tool.

2.1 Code Review Techniques
There are many techniques to review the code of software. One technique is ICR (Iterative Code Review). As we know that code review is considered on efficient method for detecting faults in software. The number of faults not detected by the review should be the small. Current methods for estimating this number assume reviews with several inspectors, but there are many cases where it is practical to employ only two inspectors. Sufficiently accurate estimates may be obtained by two inspectors employing an iterative code review (ICR) process [3]. So these processes may be stopped when a satisfactory result is estimated. More experiments are needed in order to fully evaluate the approach.

Another one method is peer code review method [2]. Peer code review method is one of the most effective ways to improve software quality, because this is an agile process. Research has consistently shown that peer code review produces software with the emphasis on working software. Agile processes promote sustainable development. The

sponsors, developers, and users should be able to maintain a constant pace indefinitely. There are several methods to process code review using agile:

2.1.1 *Over the shoulder*:[2] This is the easiest technique of all: when it is time for a code review, find a developer and sit down with him or her in front of the code. Face to face communication is an easy, high-bandwidth medium for the author's explanation of the code. An obvious drawback is that not all teams have all members in one location. An additional issue is that the reviewer is being interrupted – after the review it will take time for that developer to get back to the same level of productivity.

2.1.2 *Email pass-around*:[2] When the code is ready, send it out over email. One of the advantages of this approach is that reviewers and authors can be in different locations. Another advantage is that the reviewers can do the review at their convenience. One obvious downside is that as the review proceeds and the emails get nested in multiple replies, it becomes more difficult to follow the conversation.

2.1.3 *Pair programming*: [2] One of the Extreme Programming world's key contributions has been pair programming, which in some ways is a continuous code review. The advantages are that no workflow or tools or interruptions get in the way. Further, the review is at a deep level since the developer who is reviewing has the same level of experience with the code.

2.1.4 *Tool-assisted review: [2]* Code review tools exist to help overcome the shortcomings of the approaches listed above. They can package up source files, send notifications to reviewers, facilitate communication, ensure defects are fixed, and more. The obvious downside is that they require at the very least time for installation and configuration, and in the case of commercial products, money for license purchases [2].

Code reviews (including peer reviews, inspections and walkthroughs) are consistently recognized as an effective method of finding many types of software bugs early – yet many software teams struggle to get good value[4] (or consistent results) from their code reviews. Furthermore, code reviews are mostly considered an activity tackled by developers, and not an activity that typically falls within the realm of the test team. Code reviews, however, are an activity that questions software code; and many testers who conduct code reviews question the software code differently than their peers in development [4].

2.2 Bugnizer system

Finding and fixing bugs is 50% of the total efforts in Software industries. Bugnizer is static analysis tools for software defects detection is widely becoming in practice. If some code defect has occurred during programming, developer creates a bug and that can be assigned to another developer for reviewing. Some of tools are being used in industrial area, in which Google's Findbugs [7] tool is popular. Findbugs is an open source analytic tool that analyzes Java class files looking for programming defects [7].

3. PROPOSED SYSTEM

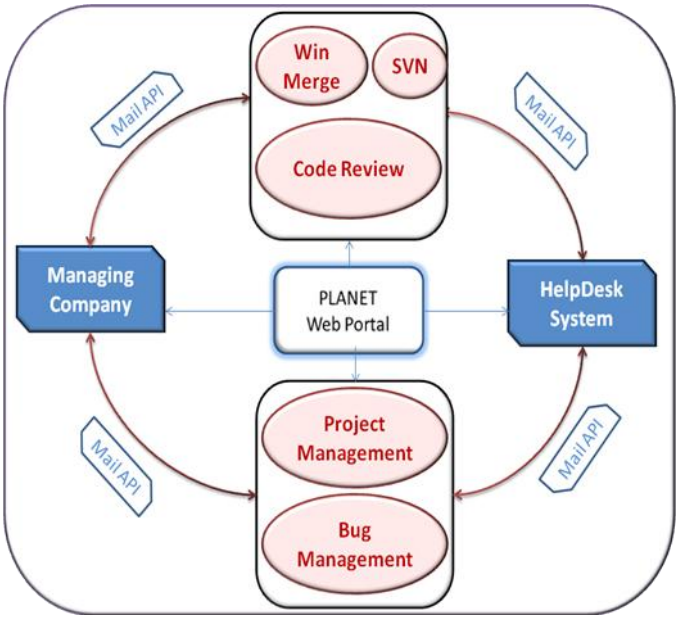

Fig 1: Proposed System

In this Product, we will be providing the features of adding multiple companies into this product. The product will be providing facility for communication between users in company using Mail API.

It will be providing facilities to find and fix the bug. Also this product manages project details and project task.The system defined in Planet systems are:

3.1 Bugnizer (Project Management System)

3.1.1 *Add Company:* Super user has an authority to add multiple companies. The company has added that can use the product and use the services. Super user also provides the specific id to that company and manages their accounts and maintains security and privacy of company.

3.1.2 *Add Project Details:* In project management one administrator who have the authority to manage the all project tasks and manage the all employee's accounts. Administrator first adds the project details. Then next tasks to assign the manager for the project. Manager has a role of managing the team of the project and also assigns the team leaders and S/W engineer to project. All the S/W developers have task that they should update their project tasks/work daily.

3.1.3 User Profile: In Planet system every user in the system has to maintain their profile. Also features provided in the profile are maintained by considering their role. Means administrator have all authority to change the tasks of project management. He also monitors others work. The manager profile has features to add leader and software engineer in project and also he can see the details report of all members related to project. Software developer profile has only feature to add task details and view bugs related to work.

3.2 Code Review System.

In this, System will be providing feature to find and fix the bugs related to project. First after completing one project

module, Developer sends that module for review. In code review system we get the versioning of two files with SVN Repository. Reviewer reviews that code; if he/she satisfies then gives the comment LGTM else he gives the suggestion and report the risk. If developer got the LGTM then it approved those changes.

3.2.1 *Fixing A Bug:* Next work is fixing a bug. This work is done with the help of code review tool. After getting any defects developer changes the code module. After changing that module the new code file without bug and old code file with bug is merged into a file with the help of code review tool and that file is send for reviewing. As Code Review tool get the old version of files from SVN Repository. Reviewer reviews the changed code gives the suggestion if really required else reviewer gives LGTM (Look Good To Me). If developer got the LGTM comment, developer approves the code file. He finalizes the code and replaced old file in repository. Repository gives the two separate file, old files and new files. If code contains some risks, then reviewer can reports the risks like p1, p2, p3… issues. P1 has higher priority risks issues and p2 has middle and p3 has lower level risks. If P1 issues are detected then there will be possibility to failure of project. Hence such if issues are detected then total module is redeveloped and pass it to for reviewing again. This process is continued until at least one LGTM comment is given by reviewer.

3.3 HelpDesk System.

This "HelpDesk" is intended to provide the tool to everyone in the organizations to book a ticket for any IT service required. This tool will also helpful to track the progress of the ticket for completion. It consists of user friendly GUI for data entry, reports, application generated E-mails for ticket transactions. Most enhancing feature is that any novice person can handle it easily and comfortably. HelpDesk provides different reports to different authorities to track each and every aspects of their business. Hence allows taking useful and quick decisions which helps an organization to make foothold in the market.

4. RELEVANT MATHEMATICS ASSOCIATED WITH PROJECT

The mathematical model is composed of three steps. [8]

4.1 The first step : choose a suitable sample project to approximately measure each bug pattern in correctness category.

4.2 The second step: calculate the defect likelihood for bug patterns and bug kinds in correctness category. We denote C as bug category, K as bug kind and P as bug pattern. For each bug category C, it contains bug kinds K1, K2, K3, . . . ,Km. And for any bug kind Ki, it contains bug patterns Pi1, Pi2, . . . , Pin. It is clear that for any bug pattern Pijin Ki, j = 1 . . . n, we have Pij= Pij:F+Pij:S, where Pij:F is the number of false error reports for bug pattern Pij, and Pij:S is the number of true error reports. We can easily calculate the defect likelihood of each bug pattern D(Pij) in the following equation and use D(Pij) to rank the error reports roughly.

$$D(Pij) = \frac{Pij:S}{Pij:F + Pij:S} \quad (1)$$

In order to avoid the inequity of calculating defect likelihood (due to different population size of bug patterns), we consider the variance V (Pij) as an additional indicator for bug pattern Pij.[8]

$$V(Pij) = \frac{D(Pij) * (1 - D(Pij))}{n} \quad (2)$$

Suppose that two bug patterns have the same defect likelihood, the one with larger population will have smaller value of variance, which means the change degree of this bug pattern population is lower and the corresponding error reports should be examined first. Once we have the defect likelihood of each bug pattern D(Pij), we can continue to Calculate the defect likelihood for each bug kind D(Ki).

$$D(Ki) = \frac{Pi1 * D(Pi1) + \ldots + Pin * D(Pin)}{Pi1 + \ldots + Pin} \quad (3)$$

In Equation (3), we ignore the population size of error reports in bug kind. For example, suppose that there are two bug kinds K1 and K2, which both contain two bug patterns. In K1, each bug pattern has 50 false positives and 50 true error reports so that the defect likelihood for K1 is 50%. In K2, one bug pattern has 100 true error reports and 0 false positive, while the other bug pattern has 0 true error report and 100 false positives. As a result, the defect likelihood for K2 is also 50%, and these two bug kinds have the same defect likelihood. However, considering the discrete degree, it is better to examine the bug kind K1 first because of its centralized distribution. The following equation is used to calculate the degree of discretization for bug kind:

$$S2 = \frac{1}{(n-1)} \Sigma (D(Pij) - D(Ki))2 \quad (4)$$

4.3 The final step: assign the value of defect likelihood and variance to bug patterns and bug kinds in correctness category. If error reports are sorted by defect likelihood of bug patterns in Code Review System, we can get a best ranking output. On the other hand, with the sacrifice of precision, it is easier for users to inspect error reports sorted by defect likelihood of bug kinds, because bug patterns in one bug kind focus on one type of defects, which are similar to each other. This statistical observation can also verify the correctness of the code. And also to measure the performance of S/W developer

5. CONCLUSION

The purpose of a code review is for someone other than the programmer to critically go through the code of a module to ensure that it meets the functional and design specifications, is well-written and robust. An incidental benefit is that the reviewer may learn some new programming techniques, and more people in the team become familiar with the module [6].

6. ACKNOWLEDGMENTS

Apart from the efforts of me, the success of any project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project.

I would like to show my greatest appreciation to Prof. Kumbhar H.R. I can't say thank you enough for his tremendous support and help. I feel motivated and encouraged every time I attend his meeting. Without his encouragement and guidance this project would not have materialized.

The guidance and support received from all the members who contributed and who are contributing to this project, was vital for the success of the project. I am grateful for their constant support and help.

7. REFERENCES

[1] White Paper 'Why review code?' by Jason Cohen, Smart Beer Software, inc.

[2] White paper on 'Peer code review: an agile process' This paper was originally published by Smart Bear Software in the proceedings of the Agile Development Practices conference in November 2009.

[3] A. Harel: 'Estimating the Number of Faults Remaining in Software Code Documents Inspected with Iterative Code Reviews' in Computer Science Dep. Technion.

[4] Alan Page: 'Peering Into the White Box: A Testers Approach to Code Reviews' in Microsoft Corp.

[5] A Borland White Paper By Richard C. Gronback: 'Software Remodeling: Improving Design and Implementation Quality' in January 2003.

[6] T.A. Gonsalves & Hema A. Murthy: 'Code Review Guidelines' TeNeT Group, IIT-Madras. In 7/11/2001.

[7] Nathaniel Ayewah, WilliamPugh: 'Using FindBugs On Production Software' University of Maryland

[8] ayewah,pugh@cs.umd.edu and J. David Morgenthaler, John Penix, YuQian Zhou Google, Inc. jdm,jpenix,zhou@google.com.

[9] EFindbugs: Effective Error Ranking for Findbugs. Haihao Sheny_, Jianhong Fangz, and Jianjun Zhaoy

ySchool of Software

Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai 200240, China

[10] Identifying Changed Source Code Lines from Version RepositoriesGerardo Canfora, Luigi Cerulo, Massimiliano Di Penta

RCOST — Research Centre on Software Technology