IJCSE International Journal of Computer Sciences and Engineering Open Access
Technical Paper Volume-4, Issue-6 E-ISSN: 2347-2693

A new Proposition for Software Code Review Process

Suvra Nandi[1*], Suvankar Dhar[2]

[1]*Department of Information Technology, Jadavpur University, Kolkata India*
[2]*School of Material Science & Technology, Jadavpur University, Kolkata India*

www.ijcseonline.org

Received: May /23/2016 Revised: Jun/05/2016 Accepted: Jun/25/2015 Published: Jun/30/ 2016

Abstract— this paper provides a new theoretical approach of code review, considering its existing challenges in current software industry with upward trend in agile methodology adoption. This article captures both Process aspects and Technical aspects of Code Review. It tries to establish the importance of Ownership, Authority, and Transparency in Process. Technically this solution tries to identify most important four deciding factors in generating function vulnerability score with Red-Amber-Green criteria for all the four factors. It formulates easy steps of determining values for those four factors which are feasible to utilize in real life scenario. Also it explains process of identifying the fifth deciding factor based upon the outcome of a project's defect prevention analysis. It explains ways of capturing review effectiveness by appropriate metric values which can be used for quantified reporting to senior management on a pre-defined interval

Keywords— Code Review Effectiveness, TDCE, RE, Cyclomatic Complexity, Time Complexity

I. INTRODUCTION

Internal Code Review is one of the most common and important process area in Software Industry. This process step is an unavoidable step in Software Engineering with Organizational and Customer expectations of more efficiency in conducting internal review process. Empirical studies provided evidence that up to 75% of code review defects affect software evolution rather than functionality, making code reviews an excellent tool for software companies with long product or system life cycles. Purpose of coed review is having a second set of eyes look over code before it gets checked in caught bugs. This is the most widely cited, widely recognized benefit of code review.

Normally there are procedures to perform internal review which mostly deal with standard code review checklist creation, maintenance and logging review findings as defects in some system. With advent of agile methodology the notion of code review changes. As Agile deals with short cycle time deliveries, there is always insufficient time to perform checklist based code review and naturally concept of different lightweight code review appears. The most famous lightweight code review process involves following re-view techniques:

1. Over-the-shoulder: One developer looks over the author's shoulder as the latter walks through the code.
2. Email pass-around: The author emails code to reviewers
3. Pair Programming: Two authors develop code together at the same workstation.
4. Tool-assisted: Authors and reviewers use specialized tools designed for peer code review.

However, new trend in Test Driven Development model focuses more on testing the applications rather than reviewing the code quality without executing the code. Even while trying to adopt any of lightweight code review process, quite often reviewers are not sure about focus review parameters, review coverage etc.

Hence it is very necessary in software industry to identify an efficient code review process which will be able to cover significant code components with minimum review time. This article tries to formulate a custom review process for a Software Organization taking these factors into consideration

II. ROOT CAUSE ANALYSIS FOR POTENTIAL INEFFICIENCY

Root Cause analysis is performed on a fishbone diagram, based on inputs gathered from different projects through brainstorming sessions in one Organization and also based on experience.[1]
The fishbone diagram is provided below.

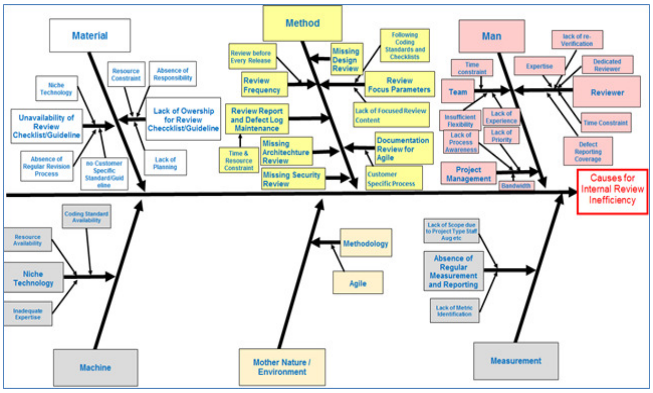

Figure 1: Fishbone Diagram - Causes for Internal Review Inefficiency

Here are the description for each and every cause with sub-causes:

A. Causes pertaining to Man or People

a. Team Issues

1. Time constraint for team members due to stringent delivery deadline
2. Often team members are not flexible enough to accept negative feedback
3. Due to lack of experience, team members are not matured enough to realize significance of review process

b. Project Management Issues

1. Often leadership team is not aware of standard software engineering process steps
2. Due to lack of planning, there is lack of priority setting among required activities
3. Lack of priority setting by senior management leads to team's bandwidth crisis, subject to improper utilization of available bandwidth

c. Reviewer Issues

1. Lack of expertise in efficient review process
2. Time constraint, as apart from review employees are more involved in their owned activities
3. Normally reviewers are not dedicated employees, rather reviewer changes for every review – this leads to lack of expertise
4. After incorporating review comments, reviewers miss to re-verify the proposed changes
5. Reviewers avoid logging identified defects in common repository-which lead to defect reoccurrence by other team members

B. Causes pertaining to Method or Process

a. Review Frequency

Projects fail to maintain the standard of review before every release, mostly for Agile process as release frequency is high

b. Maintenance of Defect Log or Review Record

Defect log or Review Record maintenance is a challenge sometimes, due to time and resource constraint

c. Missing Architecture Review

d. Missing Security Review

e. Missing Design Review

f. Review Focus Parameters

1. Challenges in following code standards for every code review, mainly due to time constraint
2. Lack of focus review content in case of Light weight review processes adopted for Agile

g. Documentation Review for Agile Methodology projects

Less documentation and Customer defined review process is often not matured enough

C. Causes pertaining to Material

a. Unavailability of Review Checklist

1. Niche technology challenges in creating review checklist, due to knowledge gap
2. Absence of Customer supplied standard review checklist
3. Sometimes review checklist is available either with Organization or provided by Customer – however no specific planning for revision of review checklist in order to maintain up to date standards

b. Lack of Ownership for maintenance of Review Checklist

Normally there is no ownership identified at project level for maintenance of review checklists and standards

D. Causes pertaining to Measurement

a. Absence of Review Efficient Measurement and Reporting to Senior Management

1. Lack of reporting leads to lack of transparency about team's efficiency in performing internal review
2. Potential gaps in process remain unnoticed

E. Causes pertaining to Machine or Technology

a. Niche Technology

1. Lack of expert resources in Niche technologies
2. Lack of coding standards

F. Causes pertaining to Mother Nature or Environment

Issues arise mainly in project environments following Agile methodology

III. PROPOSITION OF NEW PROCESS

[2]-[8]The proposed process considers all possible existing pitfalls as identified in Root Cause Analysis stage and tries to cover them to highest extent possible. Mandating process of checklist based internal review and defect logging does not suit well for all projects, as their nature of execution are different. As current trend in Software Industry is adopting Agile Methodology for its multidimensional usefulness, we need to keep our solution of efficient review process in line with that trend. Hence Time Constraint needs to be considered as one of the major factors.

Apart from time constraint lack of ownership and lack of authority are the two factors which acted as significant causes of review inefficiency in some real time scenarios.

The solution can only be effective if it can ensure minimum time and effort consumption with maximum efficiency while maintaining the transparency of potential gaps in each and every team.

The solution contains three main steps:

1. Structured Review Process
2. Structured Reporting Process
3. Process for Identification of Most Vulnerable Functions and Code Snippet
4. Customization of Solution based on Defect Prevention Analysis

Details for each step are provided below:

A. Structured Review Process

Step A. Team needs to choose and finalize manual code review types from below options. As per statistics, sticking to a fixed review type leads to more effective outcome.

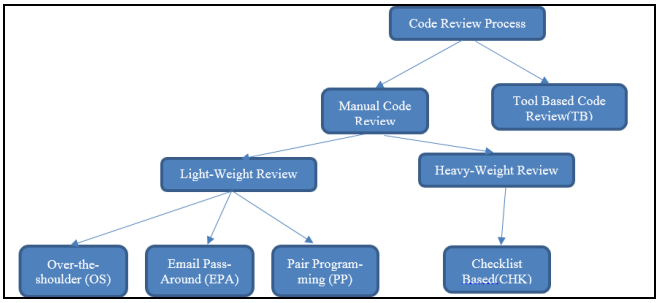

Figure 2: Review Process Options

Lightweight (LW) code review process includes:

1. Over-the-shoulder(OS) – A developer standing over the author's workstation while the author walks the reviewer through a set of code changes; defects are listed & resolved/tracked offline.

2. Pair Programming (PP) – It describes continuous code review process where two developers writing code at a single work-station with only one developer typing at a time and continuous free-form discus-sion and review

3. Email Pass-Around (EPA) - Entire set of files or changes are packaged up by the author and sent to re-viewers via e-mail. Reviewers examine the files, ask questions and discuss with the author and other developers, and suggest changes.

Heavyweight (HW) code review process:

4. Checklists based with record maintenance (CHK) - Reviewers provide review report in standard check-list format and defects are maintained in some system using defect logging tools. On completion of resolution task reviewer verifies the fix and keeps records of verification in the same checklist

The projects following this review process should provide dedicated Owners for Review Checklist, Coding Standard maintenance, and define frequency of updating.

Step B. Identification of Dedicated Reviewers for each project:

As per statistics, dedicated ownership leads to more effective outcome. Team needs to select a group of at least two dedicated reviewers for all types of review in the project. The dedicated reviewer names need to be stored in some common repository, project wise

Step C. Organization's Quality or Process Team to maintain common review process repository with below structure:

1. Project Identification Number
2. Project Name
3. Methodology
4. Review Type (OS, PP, EPA, CHK, TB)
5. Dedicated Reviewer Names

*this repository creation is one time and subject to change as required

Step D. Review Checklist & Review Record Maintenance (only for CHK Review process)

Project teams selecting Checklist based Heavyweight code review, need to maintain code review checklist, review report, defect log etc for all reviews conducted

On the other hand, project teams selecting any of the Lightweight code review, need not maintain review report and defect log mandatorily as their purpose is completing the review cycle with minimum effort.

B. Structured Reporting Process

Reporting on a regularized frequency and to proper audience ensures Authority and Transparency Maintenance in any process area for any type of Organization. Efficiency of reporting depends upon mainly three factors:

- Frequency of Reporting
- Content & Representation of Content in Reports
- Audience of the Reports

Hence for the purpose of showcasing different team's efficiency in executing internal reviews, first we need to decide on above three factors. Our solution proposes reporting in below fashion:

Frequency of Reporting – Monthly

For Waterfall Methodology, delivery to Customer depends upon stage wise milestones which may cover more than one month for one particular stage; similarly for Agile Methodology more than one small re-lease cycles are contained in one month. So in order to maintain the balance between these two we pro-pose monthly reporting as a standard.

Content & Representation of Content in Reports [10]

Metric Review Efficiency (RE) or Total Defect Containment Effectiveness (TDCE) with targets and RAG (Red, Amber, Green) Criteria

The formulae of RE and TDCE are given below:

1. Review Efficiency = No. of Defects Captured in Review Process / No. of Defects Captured in Testing Process

2. Total Defect Containment Effectiveness = No. of Defects Captured till Pre-QA Review and Test-ing / Total No, of Defects including Pre-Delivery Defects, QA Defects and UAT Defects

Standard values can be defined as 80% and 95% for RE and TDCE metrics respectively, for a standard process.

However for maintaining these review metrics as per their definition, it is necessary for each and every project to log defects in some repository or tool. Considering the situations of projects with short cycle time of delivery, mandating defect logging can be more heavyweight. The motive of this article being identifying the most efficient way of review process in minimum effort and time, defect logging process cannot be mandated.

Hence as an outcome of quality check one project may consider no. of external defects in any month, to be a measure along with existing review efficiency measurement metrics. This proposed solution recom-mends using count of external defects (QA + UAT) represented as variable "countEx" along with TDCE or RE to be used for representing internal review effectiveness for any project.

As visual representations carry more value in communicating information to senior management, it is necessary to set the Red-Amber-Green criteria for the reported metric.

Count of External Defects (countEx) + RE metric RAG criteria could be set as follows:

(RE<65%) OR (RE = 0% AND countEx >5) – RED

(65%<=RE<80%) OR (RE=0% AND 0<countEx <=5) - AMBER

(RE >=80%) OR (RE=0% AND countEx = 0) - GREEN

Count of External Defects (countEx) + TDCE metric RAG criteria could be set as follows:

(TDCE<80%) OR (TDCE = 0% AND countEx >5) – RED

(80%<=TDCE<95%) OR (TDCE=0% AND 0<countEx <=5) - AMBER

(TDCE >=95%) OR (TDCE=0% AND countEx = 0) - GREEN

Based on maturity of the process these limits could be set more stringently, while moving towards perfection.

Audience of the Reports

Reporting right content to right group always makes difference. Audience of the report should be all management stakeholders who need to be informed about the performance and efficiency of the team's review process as a whole, with project leader being a common stakeholder. On receipt of the review efficiency report respective project leaders may dig into the details of the raw data to identify most important cause that puts the project in RED or AMBER status. Accordingly project level improvement scopes can be derived in an effective way.

C. Process for Identification of Most Vulnerable Function

From our Root Cause Analysis of potential review inefficiency event displayed on a fish-bone diagram, Time Constraint seems to be one of the important cause which occurs as sub-cause to multiple causes. To take care of that often projects need to be narrow down their review focus areas especially when time or resource crunch is significant. As per standard root cause analysis processes available, Pareto analysis is one which helps in identifying the most important causes using 80-20 rule. The target becomes identification of vital 20% code snippets which may cover 80% of potential defects. Again, as elementary unit of code, we recommend using functions. As per our study and detailed analysis functions make identification of impact factors easier.

Impact of a function depends mainly on below factor as these may decide the performance of whole code base:

1. Structural Complexity – factor for Maintainability, Complexity, Portability

2. Speed of Execution – factor for Complexity, Performance

3. Involvement of the Function in the codebase – factor for Reusability

Below standard measures are identified for each of the factors:

Measure for Structural Complexity of a Function is Cyclomatic Complexity

Measure for Speed of Execution of a Function is Time Complexity

Measure for Involvement of the Function are: Fan-In (No. of other functions calling this function) and Fan-Out (No. of other functions getting called from this function)

So this proposed solution consider these four factors at first step for identification of most vital functions, review of which must not be escaped.

Below are standard definitions and quick calculation algorithms for each of these four factors:

Cyclomatic Complexity Derivation Steps[9]:

- Count of all logical operations in a function is denoted as condCount

  (Note: In any programming language Logical operations include '=', '>=', '<=", '>', '<', '!=')

- In any programming language Switch-Case constructs are counted in variable caseCount

  (Suppose there are three Switch-Case constructs denoted as Switch(1), Switch(2), Switch(3). These Switches have no. of Cases denoted as caseCount(1), caseCount(2), caseCount(3) respec-tively)

- Formula for Cyclomatic Complexity: 1 + condCount + Σ(i=1 to n) (caseCount(i) -1)

Time Complexity Derivation Steps:

- For Single loop with Additive increment (SLA) time complexity is O(n)
- For Single loop with Multiplicative increment (SLM) time complexity is O(Log n)
- For Single loop with Exponential increment (SLE) time complexity is O(Log Log n)
- For Nested loop, time complexity is product of each every single loop time complexity, from out-er to inner direction

Fan-In Derivation Steps:

Fan-In of a function is count of other functions calling current function

Fan-Out Derivation Steps:

Fan-Out of a function is count of other functions being called from current function

After deciding on the impacting factors and identifying their derivation steps, our solution creates a Re-view Process Prioritization Matrix which works as follows:

1. Prioritization Matrix creates value wise weightage factors for each of the four deciding factors as listed below:

RAG is denoted by color coding of each table:

Cyclomatic Complexity Weight Assignment	
Values	Weightage

1-10	1
11-20	2
21-30	3
31-40	4
41-50	5
51-60	6
61+	7

Time Complexity Weight Assignment	
Values	Weightage
O(1)	1
O(log n)	2
O(n)	3
O(n log n)	4
O(n^2)	5
O(2^n)	6
O(n!)	7

Fan-In Weight Assignment	
Values	Weightage
1	1
2	2
3	3
4	4
5	5
6	6
7+	7

Fan-Out Weight Assignment	
Values	Weightage
1	1
2	2
3	3
4	4
5	5
6	6
7+	7

2. Vulnerability Score of a function is the summation of all these four weightage factors for that function. Hence a function with high Vulnerability Score needs more focus compared to low scored functions

Usage for the Scoring Tool:
Developers / Reviewers may use the scoring tool following listed steps

1. Each function under review need to be listed with respective Cyclomatic Complexity, Time Complex-ity, Fan-In and Fan-Out values; Automatically Vulnerability score will be generated

2. If any RED score exist, that part must be revised

3. If count of function to be reviewed is less than 5, it is recommended that reviewer review all of them in detail

4. If count of functions is more than 5, then topmost 20% of the functions need detailed review. Top-most 20% can be decided from Vulnerability Scores assigned to each function

D. Customization of Solution based on Defect Prevention Analysis

The proposed solution is based on normal trend of projects and common vulnerabilities experienced. However, new deciding factors may need to be included by project teams based on their defect prevention analysis outcome. The most important defect cause or the most important defect type for one particular project may also be added as fifth deciding factor with proper weightage and RAG assignment.

One factor once identified as fifth deciding factor, may be continued until defects from that particular category are prevented successfully. Then it can be replaced by another factor which turns out to be most important at that point of time, in defect prevention analysis.

This way project may continue its rigor of internal review process while simultaneously preventing defects.

IV. BENEFITS

The proposed solution helps in achieving listed benefits:

1. Creates structured process in project
2. Creates transparency & authority by quantified reporting to management
3. Creates ownership by dedicated reviewer concept
4. Helps in identifying most vulnerable functions in a code by measuring with Cyclomatic Complexity, Time Complexity, Fan-In and Fan-Out
5. Helps utilizing defect prevention analysis process outcome by introducing fifth deciding factor
6. Helps in improving overall software code quality

V. FUTURE SCOPE OF IMPROVEMENT

Future scope includes sharing case study results after implementing this theoretical solution on practical projects. Pre-improvement and Post-improvement data with statistical outcome can be showcased as next version of this paper.

REFERENCES

[1] Qualiteers – Defending Software Quality, 2005 Qualiteers | info@qualiteers.com
http://www.qualiteers.com/symptom_ineffective.php
[2] Dr. Aviel D. Rubin, Dr. Seth J. Nielson, Dr.Sam Small, Dr. Christopher K. Monson; "Guidelines for Source Code Review in Hi‐Tech Litigation"; Harbor Labs White Paper; http://harborlabs.com/codereview.pdf
[3] Yanqing Wang, Bo Zheng, Hujie Huang; "Complying with Coding Standards or Retaining Programming Style: A Quality Outlook at Source Code Level";
J. Software Engineering & Applications, 2008, 1:88-91 published Online December 2008 in SciRes
[4] "Modernizing the Peer Code Review Process"; KLOCWORK | WHITE PAPER | APRIL 2010;
[5] "Five Types of Review";
Pros and cons of formal, over-the-shoulder, email pass-around, pair-programming, and tool-assisted reviews
www.ccs.neu.edu/home/lieber/courses/cs4500/f07/lectures/code-review-types.pdf
[6] Jason Cohen, Steven Teleki, Eric Brown; "Best Kept Secrets of Peer Code Review";
Collaborator by SMARTBEAR;http://smartbear.com
[7] Archana Srivastava, S.K.Singh and Syed Qamar Abbas; International Journal of Computer Sciences and Engineering; "Proposed Quality Paradigm for End User Development";International Journal of Computer Sciences and Engineering, Review Paper, Volume-4 Issue-4, E-ISSN: 2347-2693;

[8] Suvra Nandi; "Quality Maintenance Effort Optimization in Software Industry"; International Journal of Computer Sciences and Engineering, Case Study, Volume-4 Issue-5, E-ISSN: 2347-2693;
[9] THOMAS J. McCABE; "A Complexity Measure";
IEEE Transactions On Software Engineering, Vol. Se-2, No.4, December 1976;
[10] "LIST OF SUCCESS INDICATORS AND METRICS";
http://www.bth.se/com/mun.nsf/attachments/Metric%20examples_pdf/$file/Metric%20examples.pdf

AUTHORS PROFILE

Ms Suvra Nandi has completed her B.E in "Computer Science & Technology" from Bengal Engineering & Science University, Shibpur, Howrah during the year 2006. Her M.E stream is "Software Engineering" from Jadavpur University, Kolkata and she completed M.E drgree during the year 2015. She has 10 years of Software Engineering experiences with multi-national companies in different Development, Maintenance and Production Support projects. Also she is a Soft Engineering Process and Quality Facilitator, who is also facilitating different real life projects in executing and maintaining expected Quality Standards. She has significant years experiences in Quality Audits and Performance Improvement practices pertaining to CMMI Level 5 organization standards. She is Six Sigma Green Belt certified professional.

Mr. Suvankar Dhar has completed graduation in Electronics & Communication from West Bengal Univerity of Technology and Masters in Nano Technology from Jadavpur University. He has worked on ZnO Nano-particle synthesis in different chemical techniques and has 8 years of experience on Software Quality & Process areas.

