ISSN (Online) : 2348 - 2001
International Refereed Journal of Reviews and Research
Volume 5 Issue 4 July 2017
International Manuscript ID : 23482001V5I4072017-121
(Approved and Registered with Govt. of India)

Software Testing Life Cycle and Code Evaluation

Tamanna
Research Scholar
Shri Venkateshwara University
Gajraula, Amroha, Uttar Pradesh

Dr. K. P. Yadav
Research Supervisor
Shri Venkateshwara University
Gajraula, Amroha, Uttar Pradesh

Abstract. The Source Code quality and understandability completely relies upon the remarks indicated at suitable areas. The present ideal models don't propose any metric or technique that is helpful for checking the source remarks quality. The proposed display and observational parser based execution underline the proficient and significant utilization of code remarks in the source code paying little respect to the dialect or content. In this exploration work, the exact and down to earth assessment of the source understandability and the heightening is finished utilizing review based investigation. The source code remarks and understanding variables are taken into the thought with the goal that the reusability of the source code can be expanded. The key parameters for assessment of source code incorporate union, coupling and the kind of source code. The sort of source code is

ISSN (Online) : 2348 - 2001
International Refereed Journal of Reviews and Research
Volume 5 Issue 4 July 2017
International Manuscript ID : 23482001V5I4072017-121
(Approved and Registered with Govt. of India)

having the kind of procedural or question arranged worldview so any kind of source code with its innate component focuses can be examined and anticipated.

Keywords: Source Code Understanding, Software Metrics, Software Quality, Software Reusability.

1 Introduction

In the context of software engineering, software quality refers to two related but distinct notions that exist wherever quality is defined in a business context:

- Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications. That attribute can also be described as the fitness for purpose of a piece of software or how it compares to competitors in the marketplace as a worthwhile product. It is the degree to which the correctsoftware was produced.
- Software structural quality refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability. It has a lot more to do with the degree to which the software works as needed.

Many aspects of structural quality can be evaluated only statically through the analysis of the software inner structure, its source code, at the unit level, the technology level and the system level, which is in effect how its architecture adheres to sound principles of software architecture outlined in a paper on the topic by OMG. But some structural qualities, such as usability, can be assessed only dynamically (users or others acting in their behalf interact with the software or, at least, some prototype or partial implementation; even the interaction with a mock version made in cardboard represents a dynamic test because such version can be considered a prototype). Other aspects, such as reliability, might involve not only the

software but also the underlying hardware, therefore, it can be assessed both statically and dynamically (stress test).

Functional quality is typically assessed dynamically but it is also possible to use static tests (such as software reviews).

Historically, the structure, classification and terminology of attributes and metrics applicable to software quality management have been derived or extracted from the ISO 9126-3 and the subsequent ISO 25000:2005 quality model, also known as SQuaRE. Based on these models, the Consortium for IT Software Quality (CISQ) has defined five major desirable structural characteristics needed for a piece of software to provide business value: Reliability, Efficiency, Security, Maintainability and (adequate) Size.

Software quality measurement quantifies to what extent a software program or system rates along each of these five dimensions. An aggregated measure of software quality can be computed through a qualitative or a quantitative scoring scheme or a mix of both and then a weighting system reflecting the priorities. This view of software quality being positioned on a linear continuum is supplemented by the analysis of "critical programming errors" that under specific circumstances can lead to catastrophic outages or performance degradations that make a given system unsuitable for use regardless of rating based on aggregated measurements. Such programming errors found at the system level represent up to 90% of production issues, whilst at the unit-level, even if far more numerous, programming errors account for less than 10% of production issues. As a consequence, code quality without the context of the whole system, as W. Edwards Deming described it, has limited value.

To view, explore, analyze, and communicate software quality measurements, concepts and techniques of information visualization provide visual, interactive means useful, in particular, if several software quality measures have to be related to each other or to components of a software or system. For example, software maps represent a specialized approach that "can express and combine information about software development, software quality, and system dynamics".

Software Testing Life Cycle (STLC) refers to the process of testing that is executed in planned and systematic manner. In this process, assorted activities are performed to improve the overall quality of product. Here a number of objects and activities are involved.

Following is the list of steps which are involved and associated with the STLC. Every step is having the unique deliverable.

1. Requirement Analysis
2. Test Planning

ISSN (Online) : 2348 - 2001
International Refereed Journal of Reviews and Research
Volume 5 Issue 4 July 2017
International Manuscript ID : 23482001V5I4072017-121
(Approved and Registered with Govt. of India)

3. Test Case Development
4. Environment Setup
5. Test Execution
6. Test Cycle Closure

"More specifically, the further step is relying on the previous step or it can be said that the next step is not started till the previous step is not finished. This is possible in very ideal point, but very practically not always true.

Requirement Analysis - Requirement Analysis is the very first step in Software Testing Life Cycle (STLC). In this step Quality Assurance (QA) team understands the requirement in terms of what we will testing & figure out the testable requirements. If any conflict, missing or not understood any requirement, then QA team follow up with the various stakeholders like Business Analyst, System Architecture, Client, Technical Manager/Lead etc to better understand the detail knowledge of requirement.

From very first step QA involved in the where STLC which helps to prevent the introducing defects into Software under test. The requirements can be either Functional or Non-Functional like Performance, Security testing. Also requirement and Automation feasibility of the project can be done in this stage (if applicable)

Test Planning - Test Planning is most important phase of Software testing life cycle where all testing strategy is defined. This phase also called as Test Strategy phase. In this phase typically Test Manager (or Test Lead based on company to company) involved to determine the effort and cost estimates for entire project. This phase will be kicked off once the requirement gathering phase is completed & based on the requirement analysis, start preparing the Test Plan. The Result of Test Planning phase will be Test Plan or Test strategy & Testing Effort estimation documents. Once test planning phase is completed the QA team can start with test cases development activity.

Test Case Development - The test case development activity is started once the test planning activity is finished. This is the phase of STLC where testing team write down the detailed test cases. Along with test cases testing team also prepare the test data if any required for testing. Once the test cases are ready then these test cases are reviewed by peer members or QA lead.
Also the Requirement Traceability Matrix (RTM) is prepared. The Requirement Traceability Matrix is an industry-accepted format for tracking requirements where each test case is mapped with the requirement. Using this RTM we can track backward & forward traceability.
Test Environment Setup - Setting up the test environment is vital part of the STLC. Basically test environment decides on which conditions software is tested. This is independent activity and can be started parallel with Test Case Development. In process of setting up testing environment test team is not involved in it. Based on company to company may be developer or customer creates the testing environment. Mean while testing team should prepare the smoke test cases to check the readiness of the test environment setup.
Test Execution - Once the preparation of Test Case Development and Test Environment setup is completed then test execution phase can be kicked off. In this phase testing team start executing test cases based on prepared test planning & prepared test cases in the prior step.
Once the test case is passed then same can be marked as Passed. If any test case is failed then corresponding defect can be reported to developer team via bug tracking system & bug can be linked for corresponding test case for further analysis. Ideally every failed test case should be associated with at least single bug. Using this linking we can get the failed test case with bug associated with it. Once the bug fixed by development team then same test case can be executed based on your test planning.
If any of the test cases are blocked due to any defect then such test cases can be marked as Blocked, so we can get the report based on how many test cases passed, failed, blocked or not run etc. Once the defects are fixed, same Failed or Blocked test cases can be executed again to retest the functionality.
Closure of Test – It refers to the calling out of testing team so that the overall performance and bugs report can be fetched out and further predictions can be done.

Metrics

History of SDP studies started from 1970s; number of metrics are used in SDP models. Mainly metrics are categorized as code and process metrics. Code metrics is source code metrics and also called as product metrics.

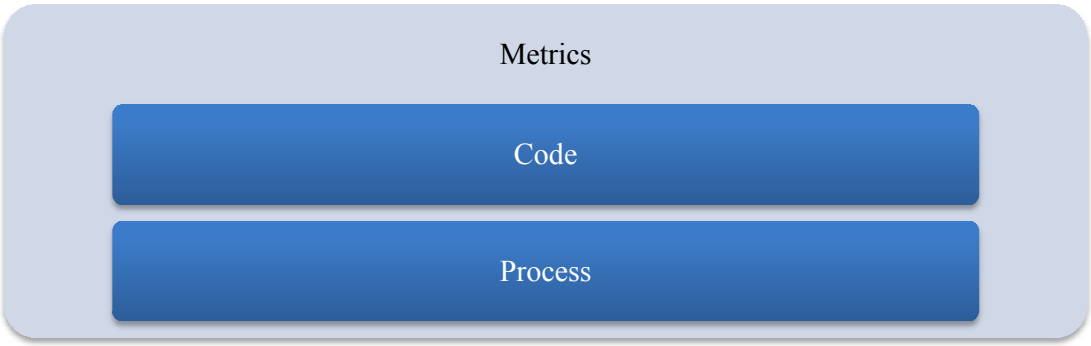
ISSN (Online) : 2348 - 2001

International Refereed Journal of Reviews and Research

Volume 5 Issue 4 July 2017

International Manuscript ID : 23482001V5I4072017-121

(Approved and Registered with Govt. of India)



Code metrics: Code Metrics used to measure the complexity. Code metrics states that complex source is more bug-prone. Data collected from the existing source code.

Size Met-ric	Halstead Metric	McCabe Metric	CK Metric	OO Metric

Types of Process Metric

1. Size metrics: Size metrics measure volume, length, quantity, and overall magnitude of software products The size metrics is represented by lines of code (LOC).The Complexity shown by LOC is very simple and very easy to understand.
2. Halstead metrics: Metrics based on the number of operators and operands proposed several size metrics operators and operands. This metrics measures the program length, difficulty, time, vocabulary, volume and effort. The majority of metrics are associated to quantity or size.It is a widely used metric.

- x = the number of distinct operators
- y = the number of distinct operands
- a = the total number of operators
- b = the total number of operands

From these numbers, several measures can be calculated:

- Program vocabulary: z=x+y
- Program length: c=a+b
- Calculated program length: L= xlogx+ylogy
- Volume: V= c+logz
- Difficulty : D=x/2+b/y
- Effort: D*V

Advantage: Data flow is considered.

Disadvantage: It does not consider the control flow.

3. McCabe metric: Cyclomatic metric was proposed by McCabe. Complexity of software products is represented by Cyclomatic metric. In control flow graphs of source code the cyclomatic metric is measured by the arcs, connected components and number of nodes code. It also measures the number of complexcontrol path. The complexity of code structure is measured by McCabe's cyclomatic metric. The Cyclomatic metric is different from Halstead and size metrics.

   Limitation: Data flow is not considered.
4. Object-oriented Metric: OO metric is widely used metric because it maps to real world entity. The development process improves by using OO metric.
5. CK Metrics:CK Metrics is used to represent OO metrics. Characteristics such as cohesion, inheritance and coupling of OO metric are used to design the CK metric. A DPM can be built by using CK metric.

Process metrics: The process metric is achieved from software repository that has development history. The software repositories are issue control system and version control system.

1) Relative code change churn: A relative code churn metrics measuring the total changes in code. It is computed by the number of lines added and delete divided by total LOC. It is an excellent predictor to give explanationof the defect densityof a binary and bug-proneness.

2) Change metrics: It measures the amount of changes in thehistory recorded in version control systems. Like relative code change churn, it also includes deleted LOC and added LOC. It did not consider the file count and total LOC. It considers maximum values and average values of change churn metrics.

Change metric includes age metrics and average and maximum of change set. Its conclusion is that change metrics are good predictors than code metrics.

ISSN (Online) : 2348 - 2001

International Refereed Journal of Reviews and Research

Volume 5 Issue 4 July 2017

International Manuscript ID : 23482001V5I4072017-121

(Approved and Registered with Govt. of India)

Process Metric	Metric Source
Relative Code change churn	Version control
Change Metric	Version control
Change Entropy	Version control
Code metric churn, Code Entropy	Version control
Popularity	Version control
Ownership	Version control
Micro Instruction Metric	Mylyn

Process Metric (Nam Jaechang)

3) Change Entropy: A history complexity metric (HCM) was proposed when Shannon entropy was applied to discover the how changes are complex. To authorize the HCM, a statisticallinear regression models were built based on two change metrics or HCM.The two change metrics is defined as number of previousmistake and previous alteration on six opensource projects. Their assessment on six opensourceprojects illustrate that prediction models constructed using HCM do better than the twochange metrics. The Entropy is used to computhe change complexity isdifferent but comparing models by HCM to those by only two change metrics disclose the limitationof the assessment of HCM andestimation was performed in the subsystem-level.

4) Code metric churn, Code Entropy: A comparison studyof defect prediction metrics was conducted, in that metric comparison; there is only study about change Entropy and code churn metrics but no study about code Entropy and code metric churn. Thus code metric churn (CHU)and code Entropy (HH) metrics were proposed.

CHU computes the changein biweekly basis of code metrics like CK metrics and OO metrics. Since code metric churncomputes the amount of changes in biweekly basis unlike to code change metrics based on the quantity of lines. CHU captures the amount of changes-

more accurately than code change churn that computes the amount of changes between a new and anold (base) revision. By applying decay functions four variants of CHU were proposed. HHis computed based on the count of involved files when a certain code metric is changed.

5) Popularity: E-mail archives were analysedand popularity metric was proposed bydevelopers in a group mailing list. It is more discussedsoftware artifacts that are more bug-prone in e-mail archives. Most metrics measure how many number of times a class are discussed in mails. Their assessment of the metrics shows that process metrics and code metric do better than popularitymetrics.

Conclusion

Metric estimation is the primary part of the examination. Estimation is characterized as the estimation of characteristic in direct estimation, yet no one but esteem can't be considered for each software or errand. In this way the utilization of direct estimation is less. In Indirect estimation n-tuple space are considered, for instance Module Defect Density, RS and System Spoilage and so forth. A metric is likewise an estimation capacity. Nitty gritty study is required to think about what and how metric play out the required undertaking (Kener and Bond 2004).

Causal model made by a group of experienced and very talented trough to foresee the deformities. Few elements must be viewed as like expense and quality. Causal model is likewise called as Bayesian Net which is made of two sections first is coordinated non-cyclic diagram and second likelihood disseminations. Dataset from thirty one software genuine undertakings was assembled by utilizing a survey. The consequence of survey is in language specialist term for this, an extent is characterized for every inquiry. Deformity Prediction Causal model comprises of a colossal system having a few subnetworks. Each subnetwork is a causal model or a bayesian net and the result of each created in etymological term.

References

1. T. Tenny "Program Readability: Procedures Versus Comments " IEEE Trans. Softw. Eng. vol. 14 no. 9 1988.

2. S. N. Woodfield H. E. Dunsmore and V. Y. Shen "The effect of modularization and comments on program comprehension " ser. ICSE '81 1981.

3. S. C. B. de Souza N. Anquetil and K. M. de Oliveira "A Study of the Documentation Essential to Software Maintenance " ser. SIGDOC '05 2005.

ISSN (Online) : 2348 - 2001
International Refereed Journal of Reviews and Research
Volume 5 Issue 4 July 2017
International Manuscript ID : 23482001V5I4072017-121
(Approved and Registered with Govt. of India)

4. C. S. Hartzman and C. F. Austin "Maintenance productivity: Observations based on an experience in a large system environment " ser. CASCON '93 1993.

5. B. P. Lientz "Issues in Software Maintenance " ACM Computing Surveys vol. 15 no. 3 1983.

6. M. J. B. García and J. C. G. Alvarez "Maintainability as a Key Factor in Maintenance Productivity: A Case Study " ser. ICSM '96 1996.

7. P. Oman and J. Hagemeister "Metrics for Assessing a Software System's Maintainability " ser. ICSM '92 1992.

8. I. S. Microsystems Code Conventions for the Java Programming Language 1997. [Online]. Available: http://www.oracle.com/ technetwork/java/codeconv- 138413.html

9. N. Khamis R. Witte and J. Rilling "Automatic Quality Assessment of Source Code Comments: the JavadocMiner " ser. NLDB '10 2010.

10. M.-A. Storey J. Ryall R. I. Bull D. Myers and J. Singer "TODO or To Bug: Exploring How Task Annotations Play a Role in the Work Practices of Software Developers " ser. ICSE '08 2008.