ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 10, December 2012

Open Source Software and Reliability Metrics

Vinay Tiwari[1], Dr. R.K. Pandey[2]

University Institute of Computer Science and Applications, R.D. University, Jabalpur INDIA[1, 2]

ABSTRACT: **Open source software during the last decade has got phenomenal success but still people are hesitant in picking up open source products. Various Studies shows that the adoption rate of open source software is very low especially in the countries like India. Reliability and quality of open source software may be the main concern for the most users. Users not only want high reliable software but often desire to check the quantitative estimation of the reliability of the open source product. Reliability metrics are units of measure for system reliability which are used to quantify the reliability of the software product. Various reliability metrics are exists which measures software reliability in various development phases like requirements, Design, coding and testing phases and helps project managers to assess the ongoing project. OSS development methodology is quite distinct and does not have formal document for Requirement, Design, Testing, and so on. Hence these metrics have to be studied on the open source software point of view. In OSS the end-user is also a part of development community and various statistics like number of contributors, number of commits, usage levels in terms of number of users, project behaviour in terms of faults/bugs reports and fixing time source line of code etc. are publicly available through the repository sites. Therefore the quality and reliability of the OSS code needs to be studied on the scale of these parameters. In this paper exploratory study is made on reliability metrics in view of OSS software and proposes a derived metrics based on these repository metrics of OSS. Although reliability is hard to measure but proposed simple derived user oriented matrix facilitate user for quantitative estimation and helps them to take decision on the adoption of OSS software.**

Keywords: Open Source Software, Software Metrics, Reliability, Reliability Metrics, OSS Contributors, Source Code, Commits

I. INTRODUCTION

Open source software during the last decade has got phenomenal technological success and produces the alternate form of software development methodology. Open Source Project Hosting Websites like SourceForge, Google Code, GitHub, Codeplex, Launchpad etc. not only providing open source packages to the users but also providing development platform to the developers and still thousands of open source projects are in developing stages at these sites. Although developers have produced systems with a functionality that is competitive with similar proprietary software developed by commercial software organizations, people are still hesitant in picking up open source products. Various Studies shows that the adoption rate of open source software is very low especially in the countries like India. Reliability and quality of open source software may be the main concern for the most users. Users not only want high reliable software but often desire to check the quantitative estimation of the reliability of the open source product. Software metrics are the quantifiable measurement of the software attributes and development process that are used to evaluate performance of software and reliability metrics are units of measure for system reliability. Reliability metrics are used to quantitatively express the reliability of the software product. Various reliability metrics are exists for the requirement phase design-coding phase, testing phase which measures software reliability during the software development and testing and helps project managers to assess the ongoing project and take managerial decisions.

Open source software development strategies are quite distinct from that of traditional software development methods. Free Open Source Software Development (FOSSD) is a way for building deploying and sustaining large software systems on a global basis and differs in many interesting ways from the principles and practices traditionally advocated for software engineering [1]. For example this does not have formal document for Requirement, Design, Testing, and so on. Software design before the development and software testing before the release is hardly carried out in the OSS development. Secondly there is no single well defined development process for OSS and it can vary from project to project. In an open source environment no project managers or company exists, there are mainly poll of developers/contributors and users. Hence these metrics have to be studied on the open source software point of view. In OSS the end-user is also a part of development community and they also have an opportunity to see the source code and various other statistics related to software like number of contributors, number of commits, usage levels in terms of number of users, project behaviour in terms of faults/bugs reports and fixing time source line of code etc. which were not possible in the case of proprietary software. These statistical information are publicly available through the project repository sites or code search sites like Ohloh. In a closed source environment user is nothing to do with the metrics since they only have the end-product in hand, but in open source development source code is also available to the user. There is a need that quality and reliability of the OSS code needs to be studied on the scale of parameters publicly available and simple reliability metrics has to be

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 10, December 2012

established on users' point of view. In the subsequent section an exploratory study is made on the reliability metrics in view of OSS software and proposed a derived metrics on the basis of available statistics which support users to take decision on reliability of OSS. Although reliability is hard to measure but this simple derived user oriented metrics helps them to take decision on the adoption of OSS software.

II. SOFTWARE METRIC

Since its inception the definition of software metrics has taken various forms. According to the IEEE standard glossary of Software Engineering terms software metrics are quantitative measure of the degree to which a system, component, or process possesses a given attribute [2]. Software metrics deals with the estimation, measurement and quantify different attributes of different attributes of the software product and the software development process. A metric is a quantifiable measurement of software product, process, or project that is directly observed, calculated, or predicted [3] i.e. they are used to compare software products, processes, or projects or to predict their outcomes. Software metric is a simple quantitative measure derivable from any attribute of the software life cycle [4]. Metrics are quantitative measures that enable software people to gain insight into the efficacy of software process and also pinpoint problem areas [2]. It is a process of measuring the quality of software. Metrics are usually specialized by the subject area and are valid only within a certain domain and cannot be directly benchmarked or interpreted outside it. Essentially, software metrics deals with the measurement of the software product and the process by which it is developed. Software metrics are used to obtain objective reproducible measurements that can be useful for quality assurance, performance, debugging, management, and estimating costs. The term software metrics is a collective term used to describe the very wide range of activities that are related to measurement in software engineering. These activities include [5]:

- Quantitative values i.e. producing numbers that characterize properties of software code (these are the classic software 'metrics')
- Prediction models to predict resource requirements and software quality
- Quantitative aspects of quality control and assurance (this covers activities like recording and monitoring defects during development and testing.

Metrics have been used more and more in making quantitative/qualitative decisions as well as in risk assessment and reduction. They give software professionals the ability to evaluate software process and the importance of software metrics has grown in the software engineering community.

III. CLASSIFICATION OF METRICS

There is no agreement in the literature on how to classify metric. They can be classified in variety of ways on development, observation, measurement, significance and Commercial Perspective under different categories. Depending upon behaviour and characteristics same metrics may belong to more than one category.

Broadly metrics are classified as Process Metric, Product Metric and Project metric. Process metrics are measures of the software development process. It is used to measure the characteristic of the methods, techniques and tools employed in developing, implementing and maintaining the software system. It describes the effectiveness and quality of a process that is used to produce software. Examples are type of methodology used, efforts required, time needed, defect removal average level of experience of the programming staff etc. Product Metric describes the characteristics of the product such as size, design features, performance, quality, etc. It is used to measure the characteristic of the documentation, code, the size of the final program i.e either source or object code. These are measures of the software product at any stage of its development, from requirements to installed system. Project metrics describes the characteristics of a project and also its execution. For example number of software developers, staffing, cost and schedule, etc.

From the measurement perspective metrics are classified as direct measurement metrics, Indirect or derived measurement metrics and prediction measurement metrics. Direct measurement is assessment of something existing [3] e.g. number of lines of code. Indirect/ derived measurement means calculation involving other attributes or entities by using some mathematical model. It always contains a calculation of at least two metrics [3]. E.g. defect density = no. of defects in a software product / total size of product. Prediction System consists of a mathematical model together with a set of prediction procedures for determining unknown parameters and interpreting results [3]. E.g. predict effort required to develop software from measure of the functionality – function point count Software Quality.

From observation perspective, Metrics can also be categorized as Primitive metrics and Computed metrics [6] . Primitive metrics are based on the direct observation such as program size i.e. Line of Code (LOC) metrics, total development time of the project, or number of defects observed in unit testing etc. [7]. Computed metrics as name suggest are derived from the computation in some manner from other metrics. Computed metrics are combinations of other metric values and thus are often more valuable in understanding or evaluating the software process than are simple metrics [7]. Examples of computed metrics are those commonly used for productivity, such as LOC produced per person-month (LOC/person-month), or for product quality,

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 10, December 2012

such as the number of defects per thousand lines of code (defects/KLOC).

IV. SOFTWARE RELIABILITY AND ITS MEASURES

According to ANSI [8] software reliability is defined as "the probability of failure free software operation for a specified period of time in a specified environment". Informally reliability is defined as a measure of how closely a system matches its stated specifications. Software reliability is based on the concept of failures. Failure of software occurs for variety of reasons like defects in code i.e. coding error, faulty software design, irrelevant input data etc. Reliability means the absence of defects which cause incorrect operation, data loss or sudden failures. It is obvious that software product having a large number of defects is unreliable. It is also clear that the reliability of the system improves if the number of defects in it is reduced. Software reliability is comprised of three major mechanisms these are fault prevention, fault detection and removal and measurement to maximize reliability [9].

Reliability growth:

Software reliability is not a direct function of time and it is fundamentally differs from hardware reliability. Software has no aging property i.e software does not deteriorate or physically change in any other way with time. In this sense the factor of time is not explicitly involved. Unlike mechanical or electronic components that are subject to wearing out. Software reliability doesn't decrease with time, because software doesn't wear out. Hardware failure can be "fixed" by replacing a faulty component with an identical one, therefore hardware reliability is stable or constant over time there is no growth in reliability. Whereas software problems can be "fixed" by changing the code in order to have the failure not happen again, therefore there is always a possibility of growth of software reliability. This phenomenon is referred to as reliability growth.

Measuring reliability means the measuring of defects. Identification and removing of software error contribute to increased reliability of software. Software reliability can not be directly measured, so other related factors are measured to estimate software reliability and compare it among products. Development process, faults and failures found are all factors related to software reliability.

V. RELIABILITY METRICS

Reliability metrics are units of measure for system reliability and are used for software reliability evaluation and assurance. Reliability metrics assess the degree to which a software product consistently performs its intended function without failure i.e. it assess the probability of software failure or the rate at which software errors will occurs. Reliability metrics are derived from failure occurrence expressions and data. Software Reliability Measurement is not an exact science and measuring software reliability remains a difficult problem since there is no clear definition to what aspects are related to software reliability. To estimate and predict the reliability of software product Software reliability metrics use statistical methods applied to information obtained during the development or maintenance phases of software. Reliability is also measured by counting the number of operational failures and relating these to demands made on the system at the time of failure. For critical systems, a long-term measurement program is required to assess the reliability. Reliability metrics are important for software reliability because they provide quantitative indicator for reliability management, evaluate and validate reliability; trade-off among cost, schedule, and reliability, monitor testing process and interpret reliability behaviour. The current practices of software reliability measurement are classified as follows:

Requirement Reliability Metrics:

Requirement specifies what features and functionality must be included in the final software. During requirement phase requirement of the customer is gathered and organized in to software requirement specification. Here important point is that the clear understanding between client and developer must exist. For high reliability software the requirement must be structured complete and easy to apply [9]. The requirements should not contain inadequate information. In order to develop reliable software from requirement phase the requirements must be free from multiple meanings [10]. There should not be any ambiguous data in the requirements. If there exists any ambiguous data , then it is difficult for the developer to implement that specification. The requirement must contain the valid structure to avoid the loss of valuable information. Requirement Reliability metrics evaluates the above said quality factors of the requirement document. Some requirement phase metrics are Requirement compliance, Number of conflicting requirement, number of fault remaining, cause and effect graphing, fault day numbers, fault density test, test coverage, Defect indices, error indices.

Design and Code Reliability Metrics

Software requirement specification document gathered during requirement phase is transform into structure during the design phase. The quality factors that exists in design and coding plan are complexity, size and modularity. More complex modules are more difficult to understand and have a higher probability of defects than less complex module

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 10, December 2012

[11]. The complexity has a direct impact on overall quality, so complexity of the modules should be less. Size of the software reflects the complexity, development effort and reliability of the software. LOC (Lines of Code), or KLOC (Lines of Code in thousands is an intuitive initial approach to measuring software size. LOC or KLOC is depends upon the factors such as blank lines, comments, executable statements etc. The reliability will decrease if modules have a combination of high complexity and large size. High complexity and small size will sometimes also decrease the reliability because; the smaller size results in a short code which is difficult to alter. For the object oriented code additional metrics are required to evaluate the quality of the software. Weighted method per class (WMC) is one of them such metrics which is used to predict how much time and effort is required to develop and maintain the class. Some other metrics are RFC (Response for a class), CBO (Coupling between objects), DIT (Depth in Tree), NOC (Number of children), Cylcomatic complexity, software maturity index etc.

Testing Reliability Metrics

Software testing is a process used to identify the correctness, completeness, and quality of developed computer software. The basic function of testing is to detect defects in software and correct it before the release to the end users. Testing Reliability metrics uses two approaches to evaluate the reliability. First approach is to ensure that the system is equipped with the functionality specified in the requirements. This approach minimizes the errors which causes due to the lack of functionality. The second approach is the evaluation of the number of errors in the code and the rate of finding the errors and fixing them [9]. Test coverage metrics are a way of estimating fault and reliability by performing tests on software products. Fault and failure metrics is able to determine when the software is approaching failure-free execution. Failure rate and mean time to failure (MMTF) are the two most used metrics in software system for reliability testing. Test coverage, Failure rate, Defect density, mean time to repair, mean down rate, mean accuracy are some other important metrics of this phase.

Some of the most common reliability metrics used to quantitatively express the reliability of software product are discussed in the following section. Some of these metrics used to measure both hardware and software reliability.

1. MTTF (Mean Time To Failure): MTTF is the average time between two successive failures observed over a large number of failures. It is the average time it takes for a system to fail. In other words we understand it as that it is the time that a system is available i.e. not failed. It is often referred to as 'uptime' in the IT industry. MTTF is a statistical value and is meant to be the mean over a long period of time and a large number of units.
An MTTF of 300 means an average of 300 time units passes between failures i.e. one failure can be expected every 300 time units. The time units are totally dependent on the system and only run-time is considered in the time measurement. Here failure means system does not meet its desired objectives.

2. MTTR (Mean Time To Recover): It is another critical metric and measures the amount of time required to repair a system and bring it back online. Failure causes because of errors and some time required to find and fix the error. MTTR measures the average time it takes to track the errors causing the failure and then to fix them.

3. MTBF (Mean Time Between Failure): The most common failure related metric "MTBF" refers to the amount of time that elapses between one failure and the next. It is the average time between consecutive system failures. Mathematically, this is the sum of MTTF and MTTR, the total time required for a device to fail and that failure to be repaired. MTBF of 300 hours means that once a failure occurs, the next failure is expected to occur only after 300 hours. Here time measurements are real time and not the execution time as in MTTF.

4. ROCOF (Rate Of Occurrence Of Failures): ROCOF corresponds to the failure intensity i.e. it measures the frequency or rate of occurrence of unexpected behaviour, For example ROCOF of 0.02 means 2 failures are likely in each 100 operational time units. For the software product ROCOF measures can be obtained by observing the behaviour of a software product in operation over a specified time interval and then calculating the total number of failure during this interval.

5. POFOD (Probability Of Failure On Demand): POFOD measures the likelihood that a transaction request or system will fail when a request is made. This metrics does not explicitly involve time measurement as in the case of other metrics. For example POFOD of 0.001 means that 1 in 1000 requests may result in failure. Here any failure is important and it doesn't matter how many.

6. AVAIL (Availability): AVAIL measures of how likely a system is available for use, taking in to account repairs and other down-time. It is the likelihood that the system will work satisfactorily at a given period of time. To calculate Software Availability, we need data for the failure time/rates and for the restoration Time / rates. For example availability of .997 means that system is available 997 out of 1000 time units.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 10, December 2012

The above six reliability metrics are used to quantify the reliability of the software product. These reliability metrics are concerned around the probability of occurrence of system failures but take no account of the consequences of the failures.

VI. OSS AND RELIABILITY METRICS

Measurement of reliability is not an exact science since there is no clear definition to what aspects are related to software reliability. Several reliability metrics exists which are primarily based on different aspects like project, process etc. and for the software developed by traditional method. These metrics measures software reliability in various phases like requirements phase, Design and coding phase, and testing phases. Open source software development methodology is completely different and they does not have formal document for Requirement, Design, Testing, and so on. There is no formal requirement analysis, system design and testing before release is hardly carried out. Very few researches have been done on the applicability of reliability metrics on open source software. In a closed source development, the metrics are mainly used by the companies, institutions and project managers in order to get a quantitative view of how they are doing and for improving software and personnel performances. In an open source environment no project managers or company exists, there are mainly poll of developers/contributors and users. The main strengths of open source projects are long pool of developers, early and frequent release, and frequent addition of patches. Although the systematic requirement analysis or not done in open source project but these project starts with the personal need and requirement is clear to the developer. New projects begins with a personal need of a single developer who has a vision and tries to devise solutions for his unmet need calls this "scratching an itch" [12]. So there is no conflicting in requirement and requirement compliance is almost met. Then he or she starts and discussion with his friends and colleagues about the possible solution and making the code base. To assess the quality of the code, existing design and code reliability metrics could be a choice for experienced developers. Typically open source software is under permanent development, testing and bug fixing. Number of errors, number of bugs reported, number of bugs removed, failure intensity etc. data are available through the bug repository of the project. These repositories not only providing bug modification progress to the developers but also ease of reporting bug to the users. The data available through these bug repositories helps to measure MTTF,MTTR,MTBF for the OSS projects. This reliability value can be calculated for the number of errors and number of repairs during a specified time interval. Reliability can also be calculated as:

Reliability = 1 – Number of errors (actual or predicted)/Total number of lines of executable code

Concept of reliability of open source software is not different from normal software reliability and many metrics equally applicable to predict the software behavior and to measure the reliability of OSS. Only difference for reliability of OSS is in collection of data, source of collection and trend of that data for the specific open source software. In a closed source development most of defect related data made available only after the problem is resolved. But the bug repositories of open source system maintain this information from the time the problem was reported until the problem is resolved. So for open source projects two more metrics helps to measure the reliability named MTTPR (Mean Time to Problem Report) and MTTPC (Mean Time To Problem Correction). MTTPR is the mean or average amount of time a system is operational without problems. MTTPC is defined as the average time taken to correct problem reports [13]

Software reliability metrics discussed some times requires many complex parameters and are good for the experienced developer. But for a normal user, who want to take the adoption decisions of OSS have a difficulty to obtain these parameters and measures. Therefore there is a need of simple metrics which will be based on the data publicly available on the project site or repository sites or code search sites like ohloh.

The public nature of open source software development makes a quantitative approach to analyzing open source software development processes possible. Data about the actual behavior of software developers is readily available to researchers in source code repositories, in mailing list archives, and on project websites. This is in sharp contrast to closed source projects, which typically remain hidden behind corporate firewalls[14]. Ohloh is a website which provides a web services suite and online community platform that aims to map the landscape of open source software development. It is a free public directory of free and open source software and the contributors who create and maintain it. Ohloh does not host projects and code. It is a directory. A community and analytics and search services. By retrieving data from revision control repositories (such as CVS, SVN, Git, Bazaar, and Mercurial), Ohloh provides statistics about the longevity of projects, their licenses and software metrics such as source lines of code and commit statistics. The codebase history informs about the amount of activity for each project. The Ohloh database provides the complete configuration management history of each crawled project. Based on the various measures available through sites discussed, we propose two source code reliability metrics which helps users/potential adopters to assess the quality of open source projects and to take adoption decision on open source project.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 10, December 2012

VII. PROPOSED METRICS

No. of contributors per thousand line of code: In an open source projects number of contributors irrespective of their role in project, simply contributing in any form to the development of OSS project plays an important role. Eric Raymond [12] in his famous essay and book "The Cathedral and The Bazaar" states that "with enough eye balls, all bugs are shallow" or more formally: "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone." The rule was formulated and named in honor of Linus Torvalds as linus' law. This rule suggests that there exists a positive relationship between the number of people involved, bug numbers, and software quality. As there are more people involved in development, bug detection and fixing is fast. As more people having the source code available when bug becomes an issue many of these people quickly resolve the problem. "By sharing hypotheses and results with a community of peers, the scientist enables many eyes to see what one pair of eyes might miss [15]. The success of many OSS products has proven that software test productivity scale up as the number of developers helping to debug the software increases. [16] Many researches agrees that collaborative development is the best ways to identify the most possible bugs in a program it has been stated that, "if you post it, someone will fix it,". Presenting the code to multiple developers with the purpose of reaching the consensus about its acceptance is a simple form of the software reviewing. Researchers and practitioners have repeatedly shown the effectiveness of reviewing process in finding bugs and security issues [17] and also that reviews may be more efficient than testing. In OSS projects there is a lack of systematic testing or other planned, prescriptive approach but still the code quality is maintained largely by "massively parallel debugging". The efficiency of bug finding and fixing is higher in OSS as the OSS is supported by people all over the world as compare to the traditional software which is supported by the parent company only.

Ruben et el [18] in their research on open source community found a positive relationship between the reliability and the number of developers in open source projects. Following are the finding of their research:

- The open source software tend to more reliable if bigger the percentage of developers in an open source community who actually use the software. In OSS producers of the software are the users also and mostly people participate in OSS development to solve a personal need. Therefore the software works according to specified need.
- The OSS is more reliable if the flow of information in an open source community is more transparent. Because of the openness the work of each contributor is openly visible in OSS development. Therefore every contributor before submitting the code thoroughly check the every piece of source code.
- More popular open source software are more reliable. As the OSS become popular among the users it attracts external stakeholders. They perform various measurement through their tools and methods and helps to improves the development process.

Crowston et el [19] has made an study on the success of source software and reported that number of users, Downloads, Inclusion in distributions, Popularity or views of information page, Reuse of code are the measure of success of the project. OSS projects mainly dependent on volunteer developers. The ability of a project to attract and retain developers on an on-going basis is important for its success. Therefore the number of developers involved in a project is an indicator of success. Wang et el in their study on evaluation of Ubuntu found that The open source community and its members play essential role in OSS evaluation. The quality of the OSS system is comparable to their commercial counterparts despite the fact that they are developed by not following traditional software engineering principles. Lakhani et el [20] argued that the quality is attributed by the review process done by a large number of contributors and the expertise and passion of the developer on the work they choose to do.

Open Source Development is a Community-Driven Development in its natural and all these studies and discussion suggests that these members not only play essential role in OSS evaluation but also there is a positive relationship between the number of community members and reliability of open source software. Therefore the quality and reliability of the OSS code needs to be studied on the scale of number of contributors. We propose a derived or computed metric No. of contributors per thousand line of code to assess the source code quality and reliability. This is calculated as follows:

$$\frac{\text{Total number of contributors}}{\text{Source line of code}} \times 1000$$

SLOC (Source line of code) metric is one of very first metric in software engineering and used to measure the size of a computer program by counting the number of lines in the text of the program's source code. Several experiments have confirmed that effort is highly correlated with source line of code and more development time is required with the program of large SLOC. Thus SLOC is typically used to predict the amount of effort that will be required to develop a program as well as to estimate programming productivity. Number of contributor is a repository metric and both these measurement are publicly available in the repository site of the project or OSS database site Ohloh. These measurements are not publicly available in the case of proprietary software. The proposed metric is an indicator of

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 10, December 2012

source code reliability. According to the Software Engineering Institute, an experienced programmer produces approximately one defect per 100 lines of code, or an average defect rate of 1 percent [21]. More the contributors per thousand line of code, lesser the chance that defect will occur resulting more reliable source code as each contributor has to write or examine few codes. This metric is an indicator for the user and less experienced programmers of OSS to tentatively assess the source code reliability and to take the adoption decision of open source software.

No. of commits per thousand line of code:

Another measurement which is publicly available related to open source projects is number of commits. A commit is an act of committing; in open source development environment a commit is the action with which a developer contributes a piece of code to the project's repository through the revision control system such as CVS or subversion. These commits in are the latest changes of the source code to the repository and become part of the head revision of the repository. Commits are submitted for bugs fixing; providing new features, to clean up a project and sometimes add whole new libraries in one go. All these activities may encompass in any of the following three actions: the addition of code, the removal of code, and the changing of existing code. A commit is an individual code contribution of a developer. Lind et el [22] show that lines of code are a good proxy for work spent on that code. Hence Commit is a fundamental unit of work in programming that a developer makes to the code base of the project in work. Every single commit action of all the projects over their entire history is available. When other users do an update or a checkout from the repository, they will receive the latest committed version. A particular challenge for analyzing software developer behavior and better understanding the open source software development process is to understand the intent of these code contributions [14]. The number of commits in a project tells how frequent the changes like bug fixing, addition of new features, release of new version etc. occur in a project.

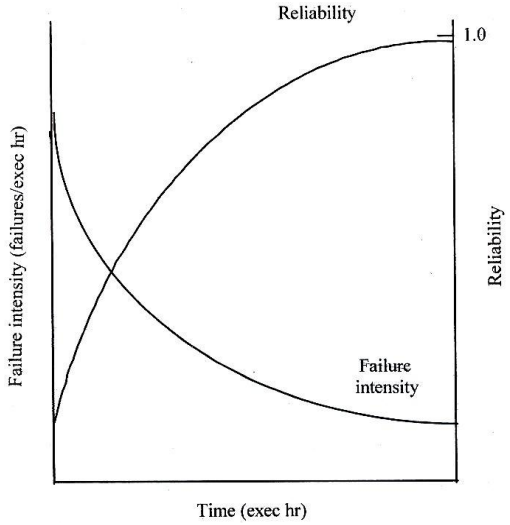

Figure 1: Reliability Curve:source [23]

Figure 1 shows the behavior of the software reliability. Reliability curve of software tells that how failure intensity and reliability typically vary as faults are removed. As the faults removes from the program failure intensity tends to drop and reliability tends to increase. Introduction of new features or design changes may introduce new faults causing step increase in the failure intensity. But quick issue of patches through frequent commits causes quick drop in failure intensity hence increase in the overall quality and reliability of the software. Based on this discussion we propose another derived metrics No. of commits per thousand line of code, which is calculated as follows:

$$\frac{\text{Total number of commits}}{\text{Source line of code}} \times 1000$$

In OSS there no formal testing is carried out, OSS users not only work as developers but as a test teams for the OSS projects. More commits on per thousand line of code implies that the source code is tested by many people. This also implies that how quickly OSS community responds to faults or adding new features in a project causing more reliable software in lesser time. Various researches has shows that software defects are unavoidable problem and open source community responds more rapidly than proprietary software vendors when a software flaw is discovered. Software reliability increases rapidly when response to software flaw is quick. Again this metric helps the user and less experienced programmers of OSS to tentatively assess the source code reliability and to take the adoption decision of open source software.

VIII. CONCLUSIONS

Software Reliability measurement is not an exact science and indirect measures are applied to measure the software reliability. Reliability metrics are used for quantitative

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 10, December 2012

measurement of software reliability. In this paper exploratory study is made on the software reliability metrics and discussion has also been made on the open source software perspective. Since the existing metrics are useful for experienced developers there is a need of simple metrics based on the publicly available data for the users of open source software which help them to take tentative adoption decision. Various studies show that open source community and their frequent commits play an important role in the development of open source software. On this perspective we have proposed two metrics. These metrics are simple enough for non experience developers and users of OSS and quantitative data is easily available to them from various repository sites. The metrics are preliminary in nature for the evaluation of reliability of open source software and need more quantitative investigation to verify our assumptions. These metrics does not help to take decision in the enhancement of the development process, but it simply helps potential adopters to assess the OSS and to take adoption decision.

REFERENCES

[1] Sommerville, I., Software Engineering, 7th edition, Addison Wesley, New York ,2004.
[2] Roger S. Pressman, Software Engineering A Practitioner's Approach, McGraw Hill, 4th Edition,1997.
[3] Futrell, Robert T.: Futrell ,Donald F. And Shafer, Linda I., "Quality Software Project Management", Pearson Education Asia 2002.
[4] N.E. Fenton and A.A. Kaposi, Metrics and software structure. Journal of Information and Software Technology, vol. 29, 1987, 301-320.
[5] Norman E. Fenton and Martin Neil, Software Metrics: Roadmap, International Conference on Software Engineering, Limerick, Ireland, pp 357-370, 2000
[6] Grady, R. B. and D. R. Caswell. Software Metrics: Establishing a Company-Wide Program. Engle- wood Cliffs, N. J.: Prentice-Hall, 1987.
[7] Mills, Everald E, "Software Metrics SEI Curriculum module SEI – CM – 12 – 1.1_, Carnegie Mellon University", Software engineering Institute, December, 1988.
[8] ANSI/IEEE, Standard Glossary of Software Engineering Terminology, STD-729- 199, ANSI/IEEE, 1991.
[9] Rosenberg Dr. Linda, Hammer Tad, Shaw Jak, (1999), Software Metrics and Reliability.
[10] P.Henry, O.Neill, J.Patrick , "Military Software Quality Metrics" ,Chief Equipment Department of China People Liberation Army, 2004.
[11] S. Xu, "Reconsideration of Software Reliability Measurements", 16th IEEE Asian Test Symposium, School of Computers, Shanghai University, CHINA, IEEE, 2007.
[12] Eric S. Raymond, (1999), The Cathedral and the Bazaar: Musingson Linux and Open Source by an Accidental Revolutionary, O'Reilly & Associates.
[13] J. D. Musa, A. Iannino, and K. Okumoto. Software reliability: measurement, prediction, application. McGraw-Hill, Inc., New York, NY, USA, 1987.
[14] Oliver Arafat, Dirk Riehle, The Commit Size Distribution of Open Source Software, In Proceedings of the 42nd Hawaiian International Conference on System Sciences (HICSS-42, track on Open Movements: FLOSS, Open Contents, and Open Communities). IEEE Press, 2009.
[15] Vixire, P. Software Engineering. In C. DiBona, S. Ockman & M. Stone(eds), Open Sources: Voices fi"om the Open Source Revolution, O'Reilly Sebastopol, 1999.
[16] Schmidt, D. C. and Porter, A. Leveraging Open-Source Communities to Improve the Quality & Performance of Open-Source Software. In Proceedings of the 23rd Intemational Conference on Software Engineering. (Toronto, Canada, May 15, 2001). ACM Press, New York, 2001, 52-56.
[17] Pfleeger, Charles P.; Pfleeger, Shari Lawrence (2003). Security in Computing, 4th Ed.. Prentice Hall PTR. pp. 154–157. ISBN 0-13-239077-9.
[18] Ruben van Wendel de Joode & Mark de Bruijne, The Organization of Open Source Communities: Towards a Framework to Analyze the Relationship between Openness and Reliability, Proceedings of the 39th Hawaii International Conference on System Sciences - 2006
[19] Crowston, K., Howison, J., and Annabi, H. (2006). Information systems success in free and open source software development: Theory and measures. Software Process:Improvement and Practice (Special Issue on Free/Open Source Software Processes.)
[20] Lakhani, K.R. and R.G. Wolf, —Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects, In Perspectives on Free and Open Source Software, edited by J. Feller, B. Fitzgerald, S. Hissam, and K. R. Lakhani, MIT Press, 2005
[21] W. S. Humphrey, The Quality Attitude, Software Engineering Institute (2004), http://www.sei.cmu.edu/news-at-sei/columns/watts_new/watts-new.htm.
[22] Lind, R., Vairavan, K.: An experimental investigation of software metrics and their relationship to software development e_ort. IEEE Transactions on Software Engineering 15, 649-653 (1989)
[23] John D. Musa, Software reliability Engineering: More reliable software faster and Cheaper, II edition, TMH, 2004, page 34.

Biography

Mr. Vinay Tiwari is qualified Computer professional having done PGDCA with distinction (1989) and MCA (2000). He is currently pursuing his Ph.D. in Computer Science. He has more than 20 years professional experience, 19 years of teaching experience at UG level and 12 years at P.G. level as counselor at IGNOU and R.D.University His Area of interests are Computer Programming, Web Designing and Software Engineering. In the last 5 years he has attended 5 International and 6 National conferences and published 5 research papers in the international journals of repute. His two books on computers have also been published.

Prof. R.K. Pandey is Ph.D. in Computer Science having more than 20 years of teaching experience at various level . Presently he is working as Director University Institute of Computer Science and Applications, R.D. University, Jabalpur. He has attended various International and National conference and presented research papers of repute. His current research area of interest: Software Engineering education and training, Software Architecture, Software design metrics.