Literature Review on Software Reliable Metrics

Kushal Patel
Computer Department
GIDC Degree Engineering CollegeNavsari,
Gujarat, India

***Abstract*---To getameliorate software quality ;many software processes have been flourished to comprehensive assesses of software reliableness. The main aim to evaluating software reliability is to predict possible software failures during the design phase prior to publishing the software. This paper will discuss about the requirement for assesses of software reliability and will discuss various methods applied to measure software reliability. In addition, discuss about relationship of reliability with Software Development Life Cycle (SDLC) phases. Furthermore, reliability metrics at different SDLC phases discussed**.

Keywords--- Software Reliability, Testing, Software development Life Cycle, Software Failures

I. INTRODUCTION

Software reliability is the probability that the software will work without failure for a specified period of time [1]. Software reliability refers itself with how considerably the software functions to satisfy the requirements of the clients. Reliability symbolizes a user oriented perspective of software quality. Initially, Software quality was evaluated by calculating the faults in the program. Software quality is developer oriented whereas reliability is user of the software oriented, because it concern to operation instead of the software design.

Developing software, that is reliable is invaluable. Reliability is the ability of a system to continue functional throughout its life time. Reliability is appraised as the probability that a system will not fail to execute its intended services throughout a defined time interval [2]. Software failures occur frequently, most of the time however, the failures are not that much costly. Software faults revealed after deployment can be diluted with the assistance of software reliability models.

II. WHY RELIABILITY?

Ascertaining the reliableness of software,it is significant to involve altogether stakeholder like Managers, Marketing, Designers, Programmers, and Customers [1]. If software is not reliable, then the failure of software can cause software developers and customers by simply being an annoyance, by costing time and money, or in a worst case scenario, by costing single or multiple lives. Everybody involved in the software process has reasons for desiring reliable software.

Reliability is crucial for following:

- Very important for safety-critical system
- System failure costs may be very high
- Unreliable system may causes information loss

Software reliability consists of Error prevention, Fault detection, removal, and Measurements to maximize reliability [3]. An error is usually a programmer action which results in faults, or known as bugs. A faults causes software to departs from its specified behavior i.e. failure [4].

2.1Relationship of reliability with SoftwareDevelopment Life Cycle (SDLC) phases

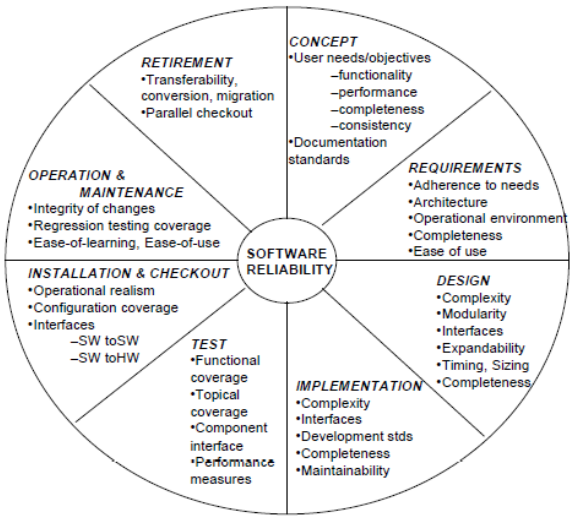


Fig. 1. Quality factors Impacting Reliability [3]

Establishing eminent reliability software depends on the application of quality properties at each phase of the development life cycle of the software with the emphasis on error prevention, especially in the early phases of the software life cycle. Metrics are required at each growing phase of the software to assess relevant quality properties. The main focusing is on error prevention for reliability. For that we required to identify and assess the quality properties applicable at various phases of software life cycle. Requirement Reliability Metrics, Design and Code Reliability Metrics, Testing Reliability Metrics are themajor properties for the Reliability Metrics as shown in figure 1. Each phase has its own Quality factors.

2.1.1 Requirement Reliability Metrics

Requirements for the particular software describe the functionality that must be included in the final software product. Software requirement specification should be

- Clear
- Unambiguous
- Easy to understand
- Complete
- Consistent

The prominent of correctly analyzed requirements induced to develop a significant number of assistances to the creation and management of the requirements specification documents and individual specification statements [3]. Software Assurance Technology Center (SATC) has developed a tool named Automated Requirement Measurement (ARM) to parse the software requirement document. In addition, ARM tool assess the structure of the software requirement document by identifying customer requirements at each phases. ARM has been applied to 56 National Aeronautics and Space Administration (NASA) requirement documents [3]. ARM tool assesses single specification statements and the vocabulary employed to state the essential requirements. In addition, it has the potentiality to evaluate the structure of the requirement document [5]. ARM has seven measures, which are describe in table 1. By employ, this seven-measure one can develop reliable software requirement documents.

Table 1: ARM requirement analysis metrics [3]

Measure	Description
Lines of Text	It is the count for the physical lines of text in the requirement documents.
Imperatives	Requirement documents words like “Shall, must, will, should, is required to, are applicable, responsible for”
Continuances	Requirement documents words like “as follows, following, listed, in particular, support”
Directives	figure, table, for example, note
Weak Phrases	adequate, as applicable, as appropriate, as a minimum, be able to, be capable, easy, effective, not
Incomplete	To be Determined (TBD), To be Supplied (TBS), To be Added (TBA)
Options	Requirement documents words like “can, may, optionally”

2.1.2 Design & Code Reliability metrics

When we design and code for the final software product, there may be many modules in the final product. More composite modules are harder to comprehend. Complexity of software has direct impact on the quality of software. Design and code reliability of the software depends on Complexity of the software, program size in Lines of Code (LOCs), Number of error message that are generated during execution of the software modules [6].

2.1.2.1 Complexity measure

There are various methods like Halstead’s theory of software science [7], CK Metrics suite [3][6][7], Cyclomatic complexity etc. available to decide complexity measure of the software [6][7].

Halstead’s software science

A computer program is an execution of statements formulated with the programing language tokens which can be classified as either operators or operands. Halstead theory uses to measure Halstead’s metrics. In the following we discuss Halstead’s metrics.

Halstead Program length

The total number of operator occurrences and the total number of operand occurrences.

$N = N1 + N2$

Where,

$N1$ = total no. of operator

$N2$ = total no. of operands

Minimum volume for algorithm

It is proportional to size of particular module or the algorithm of the program. It represents the size, in bits, of space necessary for storing the particular module or the algorithm of the whole program.

Actual volume

It is proportional to program size. It represents the size, in bits, of space necessary for storing the whole program.

Program level

It demonstrates the algorithm implementation of program language level. The same algorithm demands additional effort if it is written in a low level program language.

Development Time of the module/software

It shows time (in minutes) necessitated to convert the existing algorithm into implementation in the specified program language.

Development Effort

It measures the amount of mental activity required to translate the existing algorithm into implementation in the specified program language.

Projected number of faults

Basic primitive measure that may be derived after code generation will be given by following equations. We using code snippet to measure above metrics.

Estimated length

$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$

n_1= no. of distinct operators in program

n_2 = no. of distinct operands in program

$N1$ = total no. of operator

$N2$ = total no. of operands

Program volume,

$V = N \log_2 (n_1 + n_2)$

Volume ratio,

$L = 2/n_1 \times n_2/N_2$

Effort,

$E = V / L$

Number of delivered bugs,

$B = E^{2/3} / 3000$

Code Snippet

```
1. inti, j, k, l, m;
2. for ( i=0; i<10; i++)
3. {
4.      if ( i>= 5)
5.      {
6.          printf(“%d”, i);
7.          break;
8.       }
9. }
```

Operators

```
1. int ,  ,  ,  ,  ;
2. for (  = ; < ; ++ )
3. {
4.      if (  >= )
5.      {
6.       printf(“ %d ”,  );
7.          break ;
8.       }
9. }
```

Operands
1. i j k l m
2. i 0i 10 i
3.
4. i 5
5.
6. i
7.
8.
9.

From the above equation and code Snippet,
No. of distinct operators in program n1 = 18,
No. of distinct operands in program n2 = 8,
Total no. of operator N1 = 32,
Total no. of operands N2 = 13
Estimated length N = 99.05,
Program volume V = 465.58,
Volume ratio L = 0.07
Effort E = 6651.14,
Number of delivered bugs B = 0.1178

CK Metrics suite [7]

CK metrics suite design for Object Oriented design paradigm. Chaidamber and Kemerer propose CK Metrics. Object oriented paradigm formed of objects. Object is an instance of a class, which is a combination of variables, methods, and data structures.

Weighted Methods per Class (WMC)

This metric is determines as the sum of the complexities of all methods in a given class. It is the measure of amount of effort required to implement and test a particular class.

Coupling Between Objects (CBO)

It counts the number of classes whose attributes or methods are used by the given class and the number of classes, which use the attributes, or the methods of the given class. In a software engineering, coupling is the degree of interdependence between software modules. If CBO value is high, than reusability of particular software will be decrease. In addition, testing of the software will be complicated as the coupling is high.

Depth of Inheritance Tree (DIT)

It is define as the length of the longest path from a given class to the root class in the inheritance hierarchy. In software, it may be possible that classes will inherit many methods.

Lack of Cohesion of Methods (LCOM)

It counts the sets of methods in a class that are not relate through the sharing of some local variables of the class. In software engineering, cohesion is the degree to which the elements within a module belong together. It is desirable to keep cohesion high and keep LCOM low. It is suggested that if LCOM is high than class might be break into two or more separate class.

Number of Children (NOC)

It is the count of the total number of immediate child classes of a given class. If the NOC value is high than testing of the particular software, become difficult.

Response for Class (RFC)

It measures the number of methods (NOM) and constructors that can be invoked as a result of a message sent to an object of the class. If the value of RFC is high then testing and overall design complexity will increase.

Cyclomatic Complexity

Quality of program depends on the complexity of its control flow, rather than on operators and operands. Cyclomatic complexity is a source code complexity measurement that is being correlated to a number of coding errors [7]. It is calculated by developing a Control Flow Graph (CFG) of the code that measures the number of linearly-independent paths through a program module. Cyclomatic Complexity of software is given by below formulas,

V (G) = E – N + 2 (1)

Where, E = number of edge
N = number of node

V (G) = P + 1 (2)

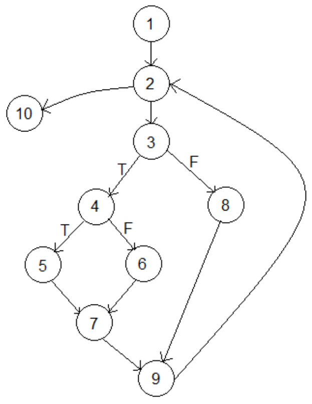
Where, P = number of predicate node

V (G) = R + 1 (3)

Where, R = number of closed region

From the below code snippet we calculated Cyclomatic Complexity using CFG.

```
1.      Start
2.      while (cond)
3.      if (cond) then
4.      if (cond) then
5.      true block
else
6.      false block
7.      end if
else
8.      false block
9.      end if
10.     end while
```



Path to tested
First 1-2-10
Second 1-2-3-8-9-2-10
Third 1-2-3-4-5-7-9-2-10
Fourth 1-2-3-4-6-7-9-2-10

From Equation 1
V (G) = E – N + 2
= 12 – 10 + 2
= 4

From Equation 2
V (G) = P + 1
= 3 + 1
= 4

From Equation 3

V (G) = R + 1
= 3 + 1
= 4

2.1.3 Testing Reliability Metrics

Software testing is a process, to assess the operation of a software application with the purpose to determine whether the developed software satisfied the specified requirements or not [8]. In addition, Software testing recognizes the shortcoming which ensures that the software product is defect free. Testing metrics must take two approaches to comprehensively assess the reliability [3].

Evolution of software testing plan which ensure that the software system comprises the operations defined in the software requirement documents.

Such testing should dilute the number of possible faults in the software product. It should also ensure that software product accomplish expected functionality. Test plans should be prepared such that it contains multiple test cases. Such test cases should be tested each requirement at least once, and some requirements will be tested several times for reduces run time errors.

2. Evolution of the number of error in the code and rate of finding/fixing them.

Costing and timing are consequences with test cases. Determined time and restricted budget are always constraints upon writing and executing test cases [3].

Time Interval between Failures [9]

A small time interval between successive failures tells us that the software system is failing frequently, and hence the reliability level is too low. If the time interval between successive failures is long, the reliability is perceived to be high, in spite of the occasional system failure. The consequences of failure are depended on duration of failures. For example, software failure for fun using application about once in a year can still beconsidered as system reliable but if a failure of critical software like pressure/temperature, measurement in boiler can be considered as extremely unreliable. Such critical software failure costs may be very high.

The three commonly used metrics based on time intervals are the:

• Mean Time To Failure (MTTF)

We assume that a system is not in operation while it is being repaired. Thus, the mean of all the time intervals between the completion of a repair task and the occurrence of the next failure is the MTTF.

• Mean Time To Repair (MTTR)

When a failure occurs, it takes a certain amount of time to repair the system. The mean of all the repair times is the MTTR.

• Mean Time Between Failures (MTBF)

The mean of the time intervals between successive failures is the (MTBF). A useful relationship between the three metrics can be stated as

MTBF = MTTF+MTTR

At the beginning of system-level testing, usually a large number of failures are observed with small time intervals between successive failures. As system testing continues and faults are actually fixed, the time interval between successive failures increases, thereby giving evidence that the reliability of the product is increasing. By monitoring the time interval between successive failures, one can get an idea about the reliability of the software system.

CONCLUSION

Software reliability measures starts from requirement phase At each phase of SDLC, metrics can identify the area of problem that may lead to error or faults. Finding faults at early phase decrease the cost, as well as effort required on later phases

REFERENCES

[1] J. Wadzinski, “Software Reliability Metrics Abstract : Introduction : Software Reliability Basics :,” 1999.
[2] Microsoft, “Chapter 16: Quality Attributes,” Microsoft, vol. 658094, pp. 1–10, 2009.
[3] L. Rosenberg, T. Hammer, and J. Shaw, “Software metrics and reliability,” Proc. 9th Int. Symp. Softw. Reliab. Eng., pp. 1–8, 1998.
[4] G. J. Pai, “A Survey of Software Reliability Models,” 2013.
[5] W. M. Wilson, D. L. Rosenberg, and L. E. Hyatt, “Automated Quality Analysis Of Natural Language Requirement Specifications,” in Fourteenth Annual Pacific Northwest Software Quality Conference, 1996.
[6] I. Sommerville, Ninth Edition. .
[7] R. S. Pressman, Software Engineering. .
[8] Rajkumar, “What Is Software Testing – Definition, Types, Methods, Approaches,” 2019. [Online]. Available: https://www.softwaretestingmaterial.com/software-testing/. [Accessed: 03-Apr-2019].
[9] K. Naik, AND QUALITY Theory and Practice