

Software Reliability: Achievements and challenges

Monika, Puneet Jai Kaur

U.I.E.T, Panjab University
monikahsp@yahoo.com, puneetkaur79@yahoo.co.in

Abstract: Employing effective software reliability engineering techniques to improve product and process reliability would be the industry's best interests as well as major challenges. Evaluating the reliability of software is a major issue. In this paper we will discuss various metrics being applied by the Software Assurance Technology Center (SATC) at NASA to evaluate the reliability of software. We will also examine the process of optimizing the reliability of software.

1. Introduction:

The IEEE defines reliability as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time." To most project and software development managers, reliability is equated to correctness, that is, they look to testing and the number of "bugs" found and fixed. While finding and fixing bugs discovered in testing is necessary to assure reliability, a better way is to develop a robust, high quality product through all of the stages of the software lifecycle. That is, the reliability of the delivered code is related to the quality of all of the processes and products of software development; the requirements documentation, the code, test plans, and testing [1].

IEEE 610.12-1990 defines reliability as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time."

IEEE 982.1-1988 defines Software Reliability Management as "The process of optimizing the reliability of software through a program that emphasizes software error prevention, fault detection and removal, and the use of measurements to maximize reliability in light of project constraints such as resources, schedule and performance." Using these definitions, software reliability is comprised of three activities:

1. Error prevention
2. Fault detection and removal
3. The Measurements to maximize reliability specifically measures that support the first two activities.

The NASA Software Assurance Standard, NASA-STD-8739.8, defines software reliability as a discipline of software assurance that:

1. Defines the requirements for software controlled system fault/failure detection, isolation, and recovery.
2. Reviews the software development processes and products for software error prevention and/or reduced functionality states.
3. Defines the process for measuring and analyzing defects and defines/derives the reliability and maintainability factors.

Mathematically reliability $R(t)$ is the probability that a system will be successful in the interval from time 0 to time t :

$$R(t) = P(T > t), t \geq 0$$

where T is a random variable denoting the time-to-failure or failure time. Unreliability $F(t)$, a measure of failure, is defined as the probability that the system will fail by time t .

$$F(t) = P(T \leq t), t \geq 0.$$

In other words, $F(t)$ is the failure distribution function. The following relationship applies to reliability in general. The Reliability $R(t)$, is related to failure probability $F(t)$ by:

$$R(t) = 1 - F(t).$$

2. Fault lifecycle techniques

Achieving highly reliable software from the customer's perspective is a demanding job for all software engineers and reliability engineers[3]. We summarize the following four technical areas which are applicable to achieving reliable software systems, and they can also be regarded as four fault lifecycle techniques:

- 1) Fault prevention: to avoid, by construction, fault occurrences.
- 2) Fault removal: to detect, by verification and validation, the existence of faults and eliminate them.
- 3) Fault tolerance: to provide, by redundancy, service complying with the specification in spite of faults having occurred or occurring.
- 4) Fault/failure forecasting: to estimate, by evaluation, the presence of faults and the occurrences and consequences of failures. This has been the main focus of software reliability modeling.

Fault prevention is the initial defensive mechanism against unreliability. A fault which is never created costs nothing to fix. Fault prevention is therefore the inherent objective of every software engineering methodology[2]. General approaches include formal methods in requirement specifications and program verifications, early user interaction and refinement of the requirements, disciplined and tool-assisted software design methods, enforced programming principles and environments, and systematic techniques for software reuse.

Fault prevention is the initial defensive mechanism against unreliability. A fault which is never created costs nothing to fix. Fault prevention is therefore the inherent objective of every software engineering methodology[2]. General approaches include formal methods in requirement specifications and program verifications, early user interaction and refinement of the requirements, disciplined and tool-assisted software design methods, enforced programming principles and environments, and systematic techniques for software reuse.

Fault prevention mechanisms cannot guarantee avoidance of all software faults. When faults are injected into the software, fault removal is the next protective means. Two practical approaches for fault removal are software testing and software inspection, both of which have become standard industry practices in quality assurance.

Fig. I shows an software reliability engineering framework in current practice [3]. First, a reliability objective is determined quantitatively from the customer's viewpoint to maximize customer satisfaction, and customer usage is defined by developing an operational profile. The software is then tested according to the operational profile, failure data collected, and reliability tracked during testing to

determine the product release time.

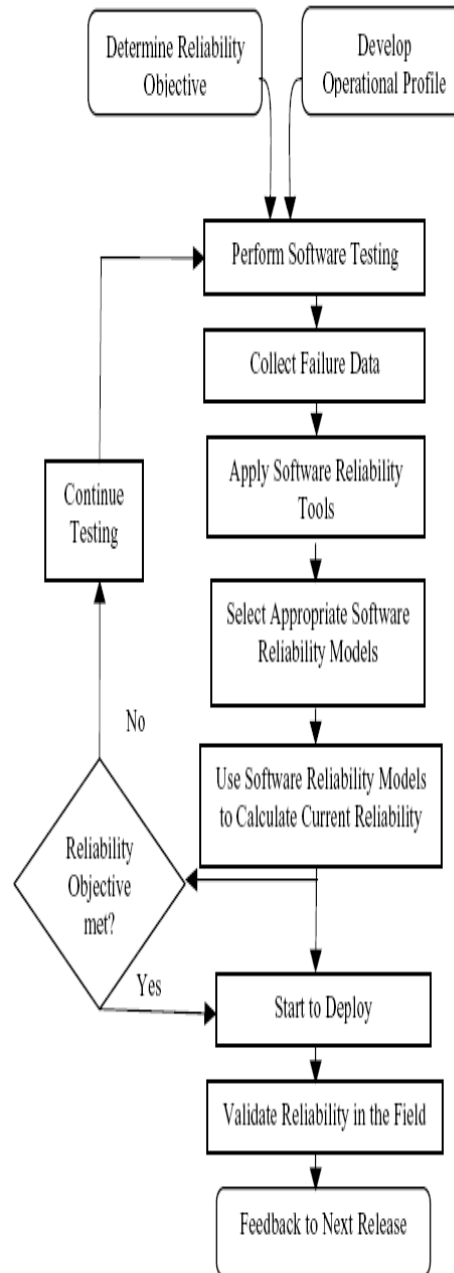


Fig. I

This activity may be repeated until a certain reliability level has been achieved. Reliability is also validated in the field to evaluate the reliability engineering efforts and to achieve future product and process improvements. It can be seen from Figure 1 that there are four major components in this SRE process, namely

- (1) Reliability objective
- (2) Operational profile
- (3) Reliability modeling and measurement
- (4) reliability validation

3. Software Reliability Techniques

Reliability techniques can be divided into two categories: Trending and Predictive

- Trending reliability tracks the failure data produced by the software system to develop a reliability operational profile of the system over a specified time.
- Predictive reliability assigns probabilities to the operational profile of a software system; for example, the system has a 5 percent chance of failure over the next 60 operational hours.

In practice, reliability trending is more appropriate for software, whereas predictive reliability is more suitable for hardware. Trending reliability can be further classified into four categories: Error Seeding, Failure Rate, Curve Fitting, and Reliability Growth.

- Error Seeding: Estimates the number of errors in a program by using multistage sampling. Errors are divided into indigenous and induced (seeded) errors. The unknown number of indigenous errors is estimated from the number of induced errors and the ratio of errors obtained from debugging data.
- Failure Rate: Is used to study the program failure rate per fault at the failure intervals. As the number of remaining faults change, the failure rate of the program changes accordingly.
- Curve Fitting: Uses statistical regression analysis to study the relationship between software complexity and the number of faults in a program, as well as the number of changes, or failure rate.
- Reliability Growth: Measures and predicts the improvement of reliability programs through the testing process. Reliability growth also represents the reliability or failure rate of a system as a function of time or the number of test cases.

4. Software Metrics for Reliability

Software metrics are being used by the Software Assurance Technology Center

(SATC) at NASA to help improve the reliability by identifying areas of the software requirements specification and code that can potentially cause errors. We will address three life cycle phases where error prevention techniques and software metrics can be applied to impact the reliability: requirements, coding, and testing.

4.1 Requirements Reliability Metrics

Requirements specify the functionality that must be included in the final software. It is critical that the requirements be written such that is no misunderstanding between the developer and the client. There are three primary formats for requirement specification structure, by IEEE, DOD and NASA[4,5,6].

Complete requirements are stable and thorough, specified in adequate detail to allow design and implementation to proceed. Requirement specifications should not contain phrases such as TBD (to be determined) or TBA (to be added) since the lack of specificity of these phrases may have a negative impact on the design, causing a disjointed architecture.

To increase the ease of using requirements, they are usually written in English [vice a specialized writing style (e.g., Z notation)], which can easily produce ambiguous terminology. In order to develop reliable software from the requirements phase forward, the requirements must not contain ambiguous terms, or contain any terminology that could be construed as an optional requirement. Ambiguous requirements are those that may have multiple meanings or those that seem to leave to the developer the decision whether or not to implement a requirement.

The SATC has developed a tool to parse requirement documents. The Automated Requirements Measurement (ARM) software was developed for scanning a file that contains the text of the requirement specification. During this scan process, it searches each line of text for specific words and phrases. These search arguments (specific words and phrases) are indicated by the SATC's studies to be an indicator of the document's quality as a specification of requirements. ARM also evaluates the structure of the document by identifying the number of requirements at each level of the hierarchical numbering structure. This information may serve as an indicator of a potential lack of structure[7].

4.2 Design and Code Reliability Metrics

Although there are design languages and formats, these do not lend themselves to an automated evaluation and metrics collection. The SATC analyzes the code for the structure and architecture to identify possible error prone modules based on complexity, size, and modularity.

It is generally accepted that more complex modules are more difficult to understand and have a higher probability of defects than less complex modules [8]. Thus complexity has a direct impact on overall quality and specifically on maintainability. While there are many different types of complexity measurements, the one used by the SATC is logical (Cyclomatic) complexity, which is computed as the number of linearly independent test paths.

Size is one of the oldest and most common forms of software measurement. Size of modules is itself a quality indicator. Size can be measured by: total lines of code, counting all lines; non-comment non-blank which decreases total lines by the number of blanks and comments; and executable statements as defined by a language dependent delimiter.

The SATC has found the most effective evaluation is a combination of size and complexity. The modules with both a high complexity and a large size tend to have the lowest reliability. Modules with low size and high complexity are also a reliability risk because they tend to be very terse code, which is difficult to change or modify [9].

4.3 Testing Reliability Metrics

Testing metrics must take two approaches to comprehensively evaluate the reliability:

The first approach is the evaluation of the test plan, ensuring that the system contains the functionality specified in the requirements. This activity should reduce the number of errors due to lack of expected functionality.

The second approach, one commonly associated with reliability, is the evaluation of the number of errors in the code and rate of finding/fixing them.

To ensure that the system contains the functionality specified, test plans are written that contain multiple test cases; each test case is based on one system state and tests some functions that are based on a related set of requirements. The objective of an effective verification program is to ensure that every

requirement is tested, the implication being that if the system passes the test, the requirement's functionality is included in the delivered system. An assessment of the traceability of the requirements to test cases is needed.

In the total set of test cases, each requirement must be tested at least once, and some requirements will be tested several times because they are involved in multiple system states in varying scenarios and in different ways. But as always, time and funding are issues; while each requirement must be comprehensively tested, limited time and limited budget are always constraints upon writing and running test cases[10]. It is important to ensure that each requirement is adequately, but not excessively, tested.

5. Conclusion:

Reliability is probably the most important factor to claim for any engineering discipline, as it quantitatively measures quality, and the quantity can be properly engineered. In this paper we have discussed various metrics being applied by the Software Assurance Technology Center (SATC) at NASA to evaluate the reliability of software. We have also examined the process of optimizing the reliability of software.

References:

- [1] Dr. Linda Rosenberg, Ted Hammer, Jack Shaw "Software Metrics and Reliability" presented at the 9th International Symposium, "BEST PAPER" Award, November 1998, Germany.
- [2] Michael R. Lyu, Software Reliability Engineering: A Roadmap Future of Software Engineering (FOSE'07), 2007 IEEE.
- [3] M.R. Lyu (ed.), Handbook of Software Reliability Engineering, IEEE Computer Society Press and McGraw- Hill, 1996.
- [4] IEEE Standard 982.2-1987 Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software.
- [5] NASA Software Assurance guidebook, NASA GSFC MD, Office of Safety and Mission Assurance, 1989.
- [6] Department of Defense. Military Standard Software Development and Documentation (December 1994), MIL-STD-498.

[7] Wilson, W., Rosenberg, L., Hyatt, L.,
Automated Quality Analysis of

Natural Language Requirement Specifications
in Proc. Fourteenth Annual Pacific Northwest
Software Quality Conference, Portland OR,
1996.

[8] Kitchenham, Barbara, Pfleeger, Shari
Lawrence, Software Quality: The Elusive
Target, IEEE Software 13, 1 (January 1996)
12-21.

[9] Rosenberg, L., and Hammer, T., Metrics
for Quality Assurance and Risk Assessment,
Proc. Eleventh International Software Quality
Week, San Francisco, CA, 1998.

[10] Hammer, T., Rosenberg, L., Huffman, L.,
Hyatt, L., Measuring Requirements Testing in
Proc. International Conference on Software
Engineering (Boston MA, May 1997) IEEE
Computer Society Press.