SOFTWARE RELIABILITY: FAILURES, CONSEQUENCES AND IMPROVEMENT

A. E. OKWONG AND E. E. UMHOH
(Received 16 July 2009; Revision Accepted 11 May 2010)

ABSTRACT

Software reliability is one of a number of aspects of computer software which can be taken into consideration when determining the quality of the software. Software reliability is an important factor affecting system performance. It differs from hardware reliability in that it reflects the design perfection rather than manufacturing perfection. The high complexity of software is the major contributing factor of software reliability processes. Software reliability is not a function of time, but it is believed that some modelling technique for software reliability is reaching propensity, by carefully selecting the appropriate model for a particular situation. Measurement of software reliability is still in its infancy. No good quantitative model has been developed to represent software reliability without excessive damage. With software embedded into many electronic devices, software failure has caused more inconveniences and losses. Software errors have caused human death. The causes are ranged from poorly designed user interfaces to direct programming errors. This work tends to draw attention on the standards to be adopted for software reliability.

KEYWORDS: Software Reliability, Hardware Reliability, Bathtub Curve, Model, Metrics

INTRODUCTION

With the advent of the computer age, computers as well as the software running in them are playing a vital role in our daily lives. But we may not have been familiar and noticed that appliances such as washing machines, televisions and watches are having their analog and mechanical parts being replaced by Central Processing Units (CPUs) and software. The computer industry has improved tremendously in software development, processes control, software control systems in terms of compact design, flexibility, handling risk feature thereby reducing cost (Osuagwu, 2008).

The idea of people about software is that software never breaks, unlike mechanical parts such as bolts and levers or electronic parts such as transistors, capacitors. Meaning that, software will stay as long as there is no problem in the hardware that changes the storage content or data path. Software does not age, rust, wear out, deform or crack. There is no environmental constraint for software to operate as long as the hardware processors it run can operate (Jianlao, 1996).

In a nutshell, software has no shape, colour or material mass. It can not be seen or touched, but it has a physical existence and is crucial to system functionality. Therefore, the functionalities affect environmental changes with series of tragedies (Ho-Won Jung et al 2004).

Software reliability unlike many other software quality factors, can be measured directly and estimated using historical and developmental data. Software reliability is defined in statistical terms as “the probability of failure-free operation of a computer program in a specified environment for a specific time”. Software reliability problems can always be traced to errors in design or implementation (Jianlao, 1996).

Significance of the Study

This paper intends to complete the following:

(i) To enable software developers have a full knowledge of project management in terms of software development.
(ii) To estimate cost, time and energy required in software development
(iii) To produce quality and reliable software.

Software Concepts

Definition: Software reliability is defined as the probability of failure during software operation for a specified period of time in a specified environment. Although software reliability is defined as a probabilistic function and comes with a notion of time, we must note that, it is different from traditional hardware reliability. Hardware components may rust or wear out with time and usage, but software will not rust or wear out during its life cycle. Software will not change over time unless intentionally changed or upgraded.

Software reliability is an important factor in software quality, performance, together with functionality, usage, maintainability and documentation (Stephen, 1998). Software reliability is hard to achieve, because the complexity becomes too high. While any system with a high degree of complexity, including software will be hard to reach a certain level of reliability. System developers tend to push complexity into the software layer with the rapid growth of software reliability. We see the complexity inversely related to software reliability; it is directly related to other factors in software quality, especially functionality, capability, etc. Therefore, this factor adds more to complexity of software.

Software reliability is one of a number of aspects of computer software which can be taken into
consideration when determining the quality of the software. Although the term ‘quality’ could connote a subjective as in qualitative-evaluation. Software reliability is generally meant to be measured using some objective criteria called metrics. With software embedded into many devices today, software failure has caused more than inconveniences and tragedies. (Keene, 2005).

The Goal of Software Reliability
The need for a means to objectively determine software quality comes from the desire to apply the technique of contemporary engineering principles to the development of software. That desire is a result of the common observation by lay person and specialists, computer software does not work the way it ought to. In other words, software is seen to exhibit undesirable behaviour up to and including outright failure.

Since software reliability is one of the most important aspects of software quality, reliability engineering approaches are practices in software field as well; Software Reliability Engineering (SRE) is the qualitative study of the operational behaviour of software based systems with respect to user requirements concerning reliability. (Keene, 2005).

Software Reliability Metrics
Measurement of quantities is common in other engineering fields, but not in software engineering. Although, it is frustrating to measured software reliability, the quest never ceased, until now, we still do not have a good way of measuring software reliability. Measuring software reliability remains a difficult problem because we do not have a good understanding of the nature of software. There is no clear definition to what aspects are related to software reliability. We can not find a suitable and convenience way to measure software reliability, and most of these aspects related to software reliability. Even the most obvious, such as software size have no uniform definition.

Although, software can not be measured directly, the current practice of software reliability measurement can be categorized into the following:

- Function Metrics
- Function Point Metrics
- Test Coverage metrics
- Project Management Metric
- Process Metrics
- Fault and Failure metrics

Product Metrics- software is thought to be reflective of complexity, development effort and reliability. Lines of Code (LOC), or LOC in Thousand (KLOC), are instuitive initial approach to measuring software size. But there is not a standard way of counting. Typically, source code is use (SLOC, KLOC) and comments and other non-executable statements are not counted. The advnets of new technologies of code reuse and code generation techniques also cast doubt on this simple method.
SOFTWARE RELIABILITY: FAILURES, CONSEQUENCES AND IMPROVEMENT

Function Point Metrics - This is a method of measuring the functionality of a proposed software development based upon a count of inputs, outputs, master files, inquiries and interfaces. This method can be used to estimate the size of a software system as soon as the functions can be identified. It is a measure of the functional complexity of the program. It measures functionality delivered to the user and is independent of the programming language.

Test Coverage Metrics - This is a way of estimating fault and reliability by performing tests on software products, based on the assumption that software reliability is a function of the portion of software that has been successfully verified or tested.

Project Management Metrics - It is of important to note that good management can result in better product. Research has demonstrated that a relationship exist between the development process and the ability to complete projects on time and within the desired quality objectives. Costs increase when a developer uses inadequate processes, risk management process, configuration management process etc.

Process Metrics - Based on the assumption that the quality of the product is a direct function of process, process metrics can be used to estimate, monitor and improve the reliability and quality of software, ISO - 9000 certification is a reference book for standard. (Michael, 1995).

Fault and Failure Metrics - The goal of collecting fault and failure metrics is to be able to determine when the software is approaching failure-free execution. Summary can be made, and observation made during testing (before delivery) and the failures (Problems).

The Effect of Software Failure and Consequences
Software failures may be due to errors, ambiguities, oversight or misrepresentation of the specification that the software is supposed to satisfy, carelessness or incompetence in writing codes, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems. (Musa et al, 1997). When comparing between software reliability and hardware reliability, hardware faults are mostly physical faults, while software faults are design faults which are difficult to visualize, classify, detect and correct (Michael, 1995). A partial list of the distinct characteristics of software compared to hardware is listed below:

* Failure Cause: Software defects are mainly design defects.
* Wear-Out: Software does not have energy related wear-out phase. Errors can occur without warning.
* Repairable System concept: Periodic results can help fire software problems.
* Time Dependency and Life Cycle: Software reliability is not a function of operational time.
* Environmental Factors: Do not affect software reliability, except it might affect program inputs.
* Reliability Prediction: Software reliability can not be predicted from any physical basis, since it depends completely on human factors in design.

* Redundancy: can not improve software reliability if the identical software components are used.
* Interference: Software interference are purely conceptual than visual.
* Failure Rate Motivators: Usually not predictable from analysis of separate statements.
* Built with Standard Components: Well understood and extensively tested standard parts will help improve maintainability and reliability. Strictly speaking, there are no standard parts for software except some standard logic structures.

As software permeates to every aspect of our daily life, software related problems and the quality of software can cause serious problems. The defects in software are significantly different than those in hardware and other components of the system. No matter how hard we try, defect-free software can not be guaranteed. The defects in software have lead to the following.

- The therac 25 accident - This event will always be remembered in history, a computer controlled radiation-therapy machine in the year 1986 caused by the software not being able to detect a race condition, alert that it is dangerous to abandon our old but well understood mechanical safety control and surrender our lives completely to software controlled safety mechanisms. (Nancy, 1993).
- Software can make decision, but can just be as unreliable as human beings. The British Destroyer Shellfield was sunk because the radar system identified an incoming missile as friendly. The defense system matured to the point that it was now mistaken the rising moon for incoming missile, but gas-field fire, descending space junk, etc, were also examples that can be misidentified as incoming missile by the defense system. (Lin, 1995).
- Software can also have small unnoticeable errors on drifts that can culminate into disaster. On February 25, 1991 during the Gulf war, the chopping error that missed 0000008 second in precision in every 10^6 of a second accumulating for 10 hours made the patriot missile fail to interpret a scud missile. 28 lives were lost. [http://www.math.psu.edu].
- Trying to fix problems may not make the software more reliable; on the contrary, new problems may arise. In 1991, after changing three lines of code in a signaling program which contains millions of lines of code, the local telephone system in California and along the Eastern Seaboard came to a stop. [Lions, 1996].
- After the success of Adriane 40 Welert, the maiden flight of Adriane 5 went into flame caused as a result of software failure. [ISO, 2001].

There are more disastrous stories to tell, with a very big question as to whether software is reliable at all. And also, whether we should use software in safety embedded applications.

The Bathtub Curve for Hardware Reliability
Over time, hardware exhibits the failure characteristics shown in figure 1, known as bathtub curve, Period A, B and C stands for burn-in phase, use life phase and end of life phase.
Software reliability however does not show the same characteristics similar as hardware. A possible curve shown in figure 2 is obtained if we project software reliability on the same axes. There are two major differences between hardware and software curves. One difference is that in the cast phase, software does not have an increasing failure rate as hardware does. In this phase software is approaching obsolence; there are motivation for any upgrade or changes to the software. Therefore, the failure rate will not change. The second difference is that in the useful life phase, software will experience a drastic increase in failure rate each time an upgrade is made. The failure rate levels off gradually, partly because of the defects found and period after upgrade.

The relationship between key concepts in hardware reliability and their applicability to software has been a topic of discourse. Although an irrefutable link is yet to be established (Rock, 1990). Let us consider a few key concepts that apply to both system elements.

If we consider a computer-based system, a simple measure of reliability is mean-time-between-failure (MTBF), where

\[ MTBF = MTTF + MTTR \]

Where MTTF and MTTR are mean-time-to-failure and mean-to-repair respectively

Many researchers agree that MTBF is a far more useful measure than defects/KHX or defects/FP. Stated simply an end user is more concerned and not with the total error count. Because each error contained within a program does not have the same failure rate, the total error count provides little indication of the reliability of the system (Keene, 2005).

Software engineering practitioners have often asked the question “when are we done testing? Musa and Ackerman suggest a response that is based on statistical criteria we cannot be absolutely certain that the software will never fail, but retire to a theoretically sound and experimentally validated statistical model.
We have done sufficient testing to say with 95 percent confidence that the probability of 1000 CPU hang of failure free operation in a probabilistically defined environment is at least 0.995" (Musa, 1989).

Using statistical modeling and software theory models of software failures (uncovered during testing) as a function of execution time can be developed. A version of the failure model, called Logarithmic Poisson Execution Time Model takes the form.

\[ F(t) = \frac{1}{p} \ln(\text{Io} pt + 1) - 1 \]

Where \( f(t) \) = Cumulative number of failures that are expected to occur once the software has been tested for a certain amount of execution time, \( t \).

\( \text{Io} \) = the initial software failure intensity (failure per time unit) at the beginning of testing.

\( P \) = the exponential reduction in failure intensity as errors are uncovered and repairs are made.

The instantaneous failure intensity \( l(t) \) can be derived by taking the derivation of \( f(t) \)

\[ l(t) = \frac{\text{Io}}{(\text{Io} pt + 1)} - 2 \]

Using the relationship noted in equation 2: testing can predict the drop-off of errors as testing progress. The actual error intensity can be plotted against the predicted cure (Fig 3). If the actual data gathered during testing and the Logarithmic Poisson Excursion time model can be used to predict total testing time required to achieve an acceptable low failure intensity.

**Prospects for Future improvements**
- Good engineering methods can largely improve software reliability
- Before the development and use of software products, testing, verification and validation are necessary steps to accomplish.
- Software testing is heavily used to trigger, locate and remove software defects.
- Various analysis tools such as trend analysis, fault-free analysis, orthogonal defects classification and formal methods, etc, can be used to reduce the possibility of defects before release.
- Filed data can be gathered and analyzed to study the behaviour of software defects.

**CONCLUSION**

Software reliability is a key part in software quality. Software reliability is hard to achieve. The difficulty of the problem stems from insufficient understanding of software reliability and in general characteristics of software. Until now, there is no way of determining defect-free software.

As more software is embedded into systems, we must be sure that they do not cause disaster. If care is not taken, software reliability can be reliability bottleneck of the entire system but is not an easy task. Software reliability can be measured using the following, modeling, measurement and improvement of using software engineering principles or field.

Software reliability modeling has matured to the point that meaningful result can be obtained by applying suitable models to the problem. There exist many models, but no single one can be adjudged as being right to solve the complexity problem. Software reliability is far from reality and can not be directly measured.

Therefore, ensuring software reliability is not an easy task, but as hard as the problem may be,
promising progresses are still being made toward more reliable software.

REFERENCES


Osuagwu, O., 2008. A Pragmatic and Technical Perspectives, 58p


