INTERNATIONAL JOURNAL OF ENGINEERING AND MANAGEMENT SCIENCES

www.scienceandnature.org

SOFTWARE PERFORMANCE EVALUATION ALGORITHM EXPERIMENT FOR IN-HOUSE SOFTWARE USING INTER-FAILURE DATA

*Jimoh, R. G. & Abikoye, O. C.
Computer Science Department, University of Ilorin, P.M.B 1515 Ilorin, Nigeria

ABSTRACT
This work aims at testing software performance on three main yardsticks: Reliability, availability and maintainability using a package (designated IN HOUSE software) prepared by us as the case study. The inter-failure data was recorded during an experiment with the software and it provided information about the software failures; it is recorded in terms of execution time in (seconds) between successive failures using a simulation of the real operational environment. This is measured as a function of mean time to failure (MTTF), the availability is measured as a function of mean time between failure (MTBF) and Maintainability is measured as a function of mean time to Maintain (MTTM). The inter-failure data collected were analyzed to reflect the relationship among the three test parameters. It is proved that our IN HOUSE software can favorably compete with any existing model in its class.

KEYWORDS: Software Reliability, Software Availability , Software Maintainability, Mean time to failure (MTTF), Mean time between failure (MTBF) and Mean time to maintain (MTTM).

INTRODUCTION

Software performance evaluation has been of great concern to software engineers since it takes some degree of expertise to determine whether a software is of good performance regardless of whether it is operational or not. There have been many difficulties in determining the performance; the three nexus (reliability, availability and maintainability) have to be jointly considered for optimum result (Pfleeger, 1997). There are different ways of testing the system which include function testing, performance testing, acceptance testing and installation testing (Pfleeger, 1997).

The yardsticks are defined as followed from various scholarly points of view (IEEE, 1990; IAN, 1992; Musa, 1997; Pfleeger, 1997; Rankin, 2002; Land, 2003; Leach, 2003; Change tech Solutions Inc, 2003; Jawadekar, 2004)

- **Function Testing:-** This confirms the fact that the tasks to be performed by the system are correctly performed. It checks that the system performs its functions as specified in the requirement.
- **Performance Testing:-** This is a measure against the non-functional requirements of the system such as speed, reliability, availability, accuracy and so on. System performance is measured against the performance objectives set by the customers as expressed in the nonfunctional requirement.
- **Acceptance Testing:-** This test is conducted to validate that the requirements set by the customers have been implemented by the developed system. It helps in assuring the customers that there is no variation between the required system and the developed one.
- **Installation Testing:-** It allows the user to exercise system functions and document additional problems that are functions of difference in the operational environment.
- **Failure Data:-** This is the information captured when a particular software fails. Inherent in any set of failure data is a considerable amount of uncertainty. Even when we can ascertain all possible faults existing in the software, yet we cannot still state with certainty when it will fail next.

This work is mainly on performance evaluation (testing) using Reliability function, Availability function and Maintainability function in measuring software performance. The main work is to confirm the suitability of the metrics for the three measure of software performance having in mind that availability is not a function of execution time but rather the clock time.

TYPES OF PERFORMANCE TESTING

Performance test is based on the requirements, so that types of tests are determined by the kind of nonfunctional requirements specified.

- **Stress tests** evaluate the system when stressed to its limits over a short period of time. If the requirements state that a system is to handle up to a specified number of devices or users, a stress test evaluates system performance when all those devices or users are active simultaneously
- **Volume tests** address the handling of large amounts of data in the system. for example , we look at whether data structures (such as queue or stack) have been defined to be large enough to handle all possible situations. In addition we check fields, records, and files to see if their sizes can accommodate all expected data.
- **Configuration tests** analyses the various software and hardware configurations specified in the requirements.

Sometimes a system is built to serve a variety of audiences and the system is really a spectrum of configurations. For example we may define minimal system to serve a single user, and other configurations build on the minimal configurations to serve additional users. This test evaluates all possible configurations to make sure that each satisfies the requirements.

- **Compatibility tests** are needed when a system interfaces with other systems. We find out whether the interface functions perform according to the requirements. For instance, if the system is to communicate with a large database system to retrieve information, a compatibility test examines the speed and accuracy of data retrieval.
- **Regression tests** are required when the system being tested is replacing an existing system. The regression tests guarantee that the new system's performance is at least as good as that of the old. Regression test are always used during a phased development.
- **Security tests** ensure that the security requirements are met. We test system characteristics related to availability, integrity and confidentiality of data and services.
- **Timing tests** evaluate the requirements dealing with time to respond to a user and time to perform a function. If a transaction must take place within a specified time, the test performs that transaction and verifies that the requirements are met. Timing tests are usually done in concert with stress to see if the timing requirements are met even when the system is extremely active.
- **Environmental tests** look at the system's ability to perform at the installation site. If the requirements include tolerances for heat, humidity , motion , chemical presence, moisture, portability, electrical or magnetic fields, disruption of power or any other environmental characteristic of the site, then our tests guarantee the system 's proper performance under these conditions.
- **Quality tests** evaluate the system's reliability, maintainability and availability. These tests include calculation of meant time to failure and mean time to maintain, as well as average time to find and fix a fault. Quality tests are sometimes difficult to administer. For example, if a requirement specifies a long mean time between failures, it may be infeasible to let the system run long enough to verify the required mean.
- **Recovery tests** address response to the presence of faults or to the loss of data, power, devices or services. We subject the system to a loss of system resources and see if it recovers properly.
- **Maintenance tests** address the need for diagnostic tools and procedures to help in finding the source of problems. We may be required to supply diagnostic programs, memory maps, traces of transactions, circuit diagrams, and other aids. We verify that the aids exist and that the function properly.
- **Documentation tests** ensure that we have written the required documents. Thus, if user guides, maintenance guides and technical documents are needed, we verify that these materials exist and that the information they contain is consistent, accurate, and easy to read. Moreover, sometimes requirements specify the format and audience of the documentation; we evaluate the documents for compliance**.**
- **Human factor tests** investigate requirements dealing with the user interface to the system. We examine display screens, messages, report formats, and other aspects that may relate to ease of use. In addition, operator and user procedures are checked to see if they conform to ease of use requirements. These tests are sometimes called usability tests.

SOFTWARE PERFORMANCE MEASURES

Software Reliability (R_s)

The concept of software reliability can be defined as ability of software system to function consistently and correctly over long periods of time (Pfleeger, 1997). Software reliability can also be defined as the probability of a failure-free software operation for a certain period of time in a given operational environment (Immonen, 2006). Software reliability is an important factor affecting system reliability. It is quite different from hardware reliability since it reflects the design perfection rather than manufacturing perfection as in the case of hardware reliability. Software Reliability is a function of execution/operational time and not the real time (clock time) (Pan, 1999) so that it more accurately reflect system usage.

Software Reliability measures the system operation without failing over a long period of time. It operates on a scale between 0 and 1 that is $\mathbf{0 \leq R_s \leq 1}$. The software system is considered to be reliable if and only if it has reliability value close to 1. Consequently, the software system is considered not to be reliable if it has reliability value close to zero.

The measure of reliability reflects the system usage and it is always calculated based on execution time rather than the clock time.

Software Availability (A_s)

Availability of software can be guarantee when the system is operating successfully according to specification at a given point in time (Pfleeger, 1997). More formally, it is the probability that a system is functioning completely at a given point in time, assuming that the required external resources are also available. It is a function of clock time and not operational time. A system that is up and running has availability 1 while the one that is unusable has availability 0. In order to achieve an accurate measure of availability exactly experienced by the end user, there is need to fully understand the system configurations (Change Tech Solutions, 2003). Things to really understand include components and resources being used by the application in question (local and remote) and required hardware and software resources.

The availability is calculated by measuring the number of committed hours of availability (A) over a period of time as decided by the organiasation using the application (Pfleeger, 1997; Change Tech Solutions, 2003). Achieved Availability can rather be measured in terms of outage hours

(outage time measured in hours within committed hours of availability, both unplanned and predetermined outage must be consider in achieving continuous availability(Change Tech Solutions, 2003). The hours of outage can be used in measuring the mean time between failure. Table1 below explains the relationship between availability targets and hours of outage allowed for a continuous availability level to be achieved.

Table 1: Outage Hours and Continuous Availability (Change Tech Solutions, 2003)

Continuous availability target	Hours of outage allowed per month
99.99%	0.07 hours
99.9%	0.7 hours
99.5%	3.6 hours
99.0%	7.2 hours
98.6%	10.0 hours
98.0%	14.4 hours

Software Maintainability (M_s)

According to IEE standard computer dictionary (1990), Maintainability can be defined as the ease and speed with which any maintenance activity can be carried out on an item of equipment. It is also defined as the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.

Maintainability is analogous to Cumulative Failure time in reliability. Maintainability can be defined as the probability that a specified maintenance action on a specified item can be successfully performed (putting the item into a specified state) within specified characteristics using specific tools and procedures (Rosenberg, 2000).

Maintainability is the probability that for a given condition of use, a maintenance activity can be carried out within a stated time interval and using a stated procedures and resources. The scale is between 0 and 1 that is $0 \leq \mathbf{M_s} \leq \mathbf{1}$. This means that a system is maintainable if its maintainability is close to 1 and not maintainable if its maintainability is close to 0. Because Software reliability, availability and maintainability are defined in terms of failures they must be measured once the system is complete and working. The suitability of this approach is best explained by the calculated values of reliability, availability and maintainability.

MAIN RESULTS

In this work, we developed a software and it is designated an IN- HOUSE software for result computation of the Postgraduate Diploma Students in the Department of Computer Science, University of Ilorin for over six years now and has since then been put into effective usage. The most important factor in this work is the system failure which might be catastrophic, critical, marginal or minor. Information about the software failure is taken using two assumptions:

- That there is a possibility of causing another problem while solving a particular one.
- The inability to predict the next failure

Interfailure Data

Inter-failure data is a data of successive failures of the departmental result computation in house software in an operational environment over a particular period of time. The inter-failure data of an in house application, used in the computation of student's result (case study) are taken in terms of execution time in (seconds) between successive failures of a command-and-control system during in house testing using a simulation of the real operational environment (Musa, 1997). The data is presented in Table 2 below.

Table 2: Table of inter-failure time Read from Left to Right:

Kindly mmention the attribute in Table									
106	133	219	184	218	112	105	194	215	118
240	152	179	126	210	190	772	222	128	216
132	100	102	104	323	161	272	154	553	395
440	170	257	183	437	295	125	1080	715	130
175	580	852	190	115	900	618	925	702	1225
125	238	203	105	1033	712	402	275	292	185
1080	521	442	1315	2212	452	220	185	1020	800

Measuring Reliability, Availability and Maintainability

We want to measure reliability, availability and maintainability as attributes of the software developed measured as numbers between 0 (unreliable, unavailable and unmaintainable) and 1 (completely reliable, always available and completely maintainable). To derive these measures, we examine attributes of failure data. Assuming that we are capturing failure data and that we have seen $i-1$ failures. We can record the interfailure times, or times to failure as , , ,..., .The average of these values is the Mean Time To Failure (MTTF). After each underlying fault has been fixed and the system is again running. We now use T_t to denote yet-to-be observed next time to failure. T_t is a random variable.

There are several other time–related data important to calculating availability and maintainability. Once a failure occurs there is additional time lost as the faults causing the failure are located. The Mean Time to Maintain (MTTM) is the average time it takes to fix a faulty software component. The Mean Time to Failure (MTTF) can be combined with the Mean Time to Maintain (MTTM) to determine how long the system is unavailable. That means we measure availability by examining Mean Time between Failure (MTBF) = MTTF + MTTM.

As the system becomes more reliable, its MTTF should increase. We can use MTTF in a measure whose value is near zero when MTTF is small, and nears 1 as MTTF gets increased. Considering this relationship, a measure of reliability can be defined as:

R_s = MTTF/(1 + MTTF) ……..Reliability function
Similarly, we can measure availability as to maximize the MTBF

A_s = MTBF (1 – MTBF)……..Availability function
Also, maintainability can be measured to minimize the MTTM as

M_s = 1/(1 + MTTM)……...…….Maintainability function.
Note that MTTF, MTBF and MTTM are derived from the inter-failure data.

RESULTS AND DISCUSSION

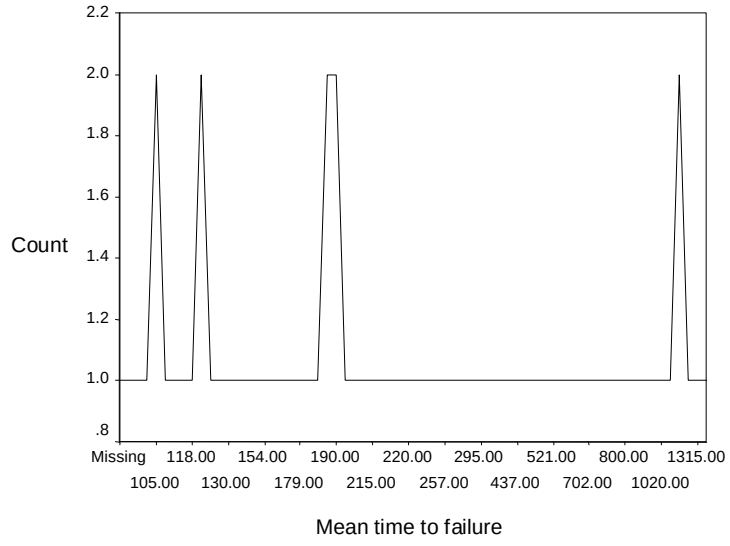
Table 2- Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
Mean time to failur	70	100.00	2212.00	403.8000	386.2255
Valid N (listwise)	70				

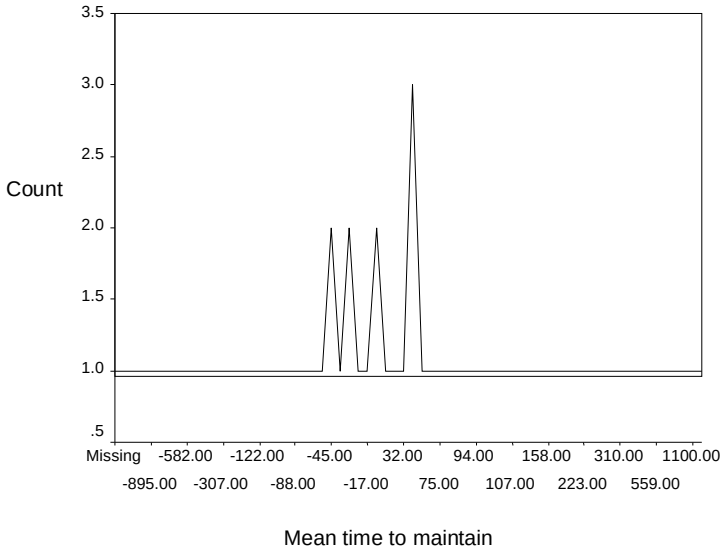
Table 3 : T-Test One Sample Test

	Test Value = 0					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Mean time to maintain	.028	69	.978	1.5143	-105.7221	108.7506

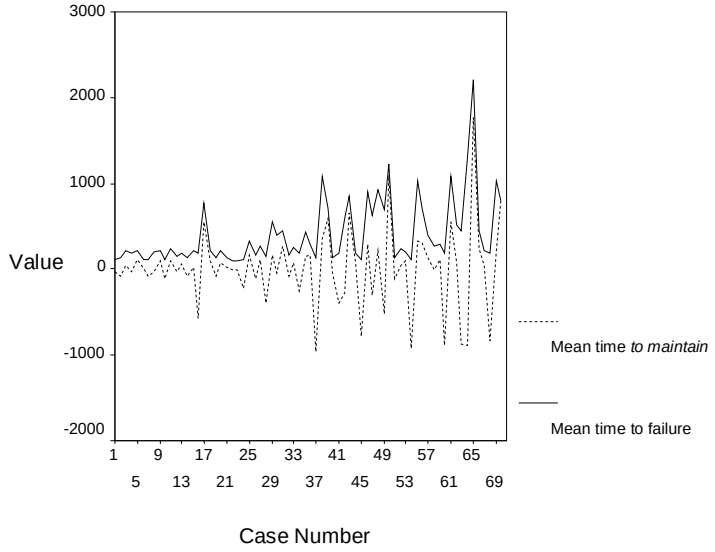
Graph 1: Mean Time to failure



Graph 2: Mean Time to maintain



Graph 3: Relationship between MTTF and MTTM



From the result of the analysis, we have the following values:

Mean Time to Failure (MTTF) = 403.800
Mean Time to Maintain (MTTM) = 1.5143
Mean Time Between Failure (MTBF)= MTTF + MTTM
MTBF = 403.8000 + 1.5143 = 405.3143.

The obtained values are then used in calculating the reliability, availability and maintainability as follow:

Reliability $\mathbf{R_s}$ = MTTF (1 + MTTF)
= 403.8000 /1 + 403.8000
= 403.8000/404.8000
= 0.9975

Availability $\mathbf{A_s}$ = MTBF (1-MTBF)
= 405.3143* (1- 405.3143)
= 405.3143 * -404.3143
= -163874.3675

Maintainability Ms = 1/(1 + MTTM)
= 1/(1+1.5143)
= ½.5143
= 0.4

CONCLUSION

As we can see from the result that the departmental result computation in house software is considered to be reliable with a reliability value 0.9975 which is very close to 1, also it is fairly maintainable since the maintainability value 0.4 is not very close to 0. But on the contrary, the availability value does not fall within the range; this can be attributed to the fact that we cannot use execution time to evaluate availability. Therefore such a metric cannot be used for availability.

From our result we are able to confirm that the metrics:

$\mathbf{A_s}$ = MTBF (1-MTBF)

cannot be used to measure software availability since mean time between failure MTBF is a function of execution time whereas availability of software has to do with clock time.

It can thus be recommended that another metrics should be used to measure software availability. Such revealed that using execution time cannot adequately capture the outage hours of the software system in question. This means it is practically impossible to measure reliability, maintainability and availability together using the same nexus. Such difficulty can be attributed to the fact that while one relies on execution time, the other relies on clock time.

REFERENCES

Change Tech. Solutions Inc. (2003) "How to measure system availability targets," The Harris Kern Enterprise Computing Institute, February, 2003, Accessed from http://articles.techrepublic.com.com/5100-10878_11-1060287.html on February,2008.

IAN C. (1992) *Software Engineering*, Addison-Wesley Publishing Company, New York, United State, pp 389 – 396.

IEEE (1990) IEEE Standard Glossary of Software Engineering Terminology, US. Accessed from http://ieeexplore.ieee.org/servlet/opac?punumber=2238 on July, 2007

Immonen, A. (2006) A method for predicting reliability and availability at the architectural level in Software Product Lines *Research Issues in Engineering and Management*, 2006 edition

Jawadekar, W. S. (2004) *Software Engineering, Principle & practice*, New Delhi: Tata McGraw-Hill Publishing Company Limited

Land, R. (2003) Measurement of software maintainability, A Publication of Malardalen University

Leach, R. J. (2000) *Introduction to software engineering*, Boca Raton, London New York Washington D.C: CRC Press

Musa, J. D. (1997) Introduction to Software engineering and testing, *8th international symposium on software reliability engineering*

Pan, J. (1999) *Software Reliability in Embedded Systems*, Carnegie Mellon http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/presentation.pdf

Pfleeger, S. L. (1997) *Software Engineering, theory & practice*, Washington D.C: Pearson Education

Rankin, C. (2002) The Software Testing Automation framework, *IBM systems journal*, 41(1), 126 -139

Rosenberg, J. (2000) *Can we measure maintainability*, 901 San Antonio Road, Palo Alto, California: Sun Microsystem,