



A stochastic model of dependability of computer systems in Zimbabwe

G. Kabanda

*Central Computing Services, P.O. Box CY704,
Causeway, Harare, Zimbabwe*

ABSTRACT

A theoretical approach to evaluation leads to the hypothesis that human-computer interaction is affected by personality, influencing types of dialogue, preferred methods for providing help, etc. A practical framework suggests that evaluation is an integral part of the design and that the iterative use of the evaluation techniques throughout the conceptualisation, design, implementation, and post-implementation stages would satisfy user needs and fulfil usability goals. Evaluation is crucial to the improvement of the design and performance of a computer system, ascertaining the value of the software, legalising its use and promotion of such systems.

The aim of the paper was to determine and utilize design evaluation techniques in the software development process. In this study, computer system evaluation measures used a stochastic model which represents the dynamics of the system's structure, internal states, and environment during each period of the system's operation and interruptions. The operational periods begin with validation tests. Validation of a computer system looks at the utility of the system as opposed to verification which involves semantic and structure checking. Verification during development is planned to include modelling and simulation for performance, design reviews of user specifications, design team reviews, and use of design metrics for quality. It is plausible to produce a convincingly validated testbed which is of proven effectiveness and low risk. Convincing validation is envisaged to involve measurement of software project personnel performance indicators, and software development and maintenance costs.

Dependability is a concept encompassing reliability, availability, maintainability and safety of a computer system.

INTRODUCTION

Design evaluation requires design optimization through a comprehensive set of design quality metrics and a highly interactive and responsive user interface. Hence, there is a need to determine and utilize design evaluation techniques in the software development process. This could lead to an improvement in software productivity, reliability, manageability and

524 Artificial Intelligence in Engineering

maintainability.

There are several issues intricately intertwined in program verification that need to be disentangled. Programs are verifiable when they are subject to deductive procedures. In particular, absolutely verifiable programs have program axioms allowing inference to be drawn concerning the performance that a machine will display when such a program is executed. Both computer programs and mathematical proofs are syntactical entities consisting of sequences of lines of completely formalized entities with completely appropriate formal procedures. An analogy discussed by [3] suggests that proofs are to theorems as verifications are to programs.

The commonly used software quality metrics consist of correctness, maintainability, usability, reliability, portability, security, efficiency, reusability and interoperability. In order to determine the utility of a system, the evaluation needs to be done through the assistance of users. Computer system evaluation measures use a stochastic model which represents the dynamics of the system's structure, internal states, and environment during each period of the system's operation and interruptions. The operational periods begin with validation tests. All software and hardware design faults are permanent, present, and not detected by these tests [4]. Other untested hardware faults may be present. Thus system operational periods begin with a set of hard to find latent errors. Operational tasks continue until interrupted by a dependability task. This depends upon the error occurrence rate, measured by the number of detected errors plus the number of restarts and failures. Both recovery rates, the fault removal, restart, and failure rates are also directly measurable. The existence or occurrence of a fault cannot be directly measured as it does not imply an error occurrence. Thus the fault occurrence rate is a poor parameter to use to control operational dependability evaluations. Stochastic Petri nets can use error occurrence as the basic parameter. Such descriptions can ensure that all parameters are realistic and measurable.

The 11 quality factors used at Central Computing Services are Correctness, Reliability, Efficiency, Integrity, Maintainability, Testability, Flexibility, Portability, Reusability, and Interoperability. The software quality requirements are affected by basic characteristics e.g. if human lives are affected. In identifying the important quality factors one should consider life cycle implications, perform trade-offs among the tentative list of quality factors, identify most important quality factors, and provide explanations for choices.

DEPENDABILITY MODELLING

Modelling and evaluation of computer systems, composed of hardware and software, are increasingly becoming important in the detection of physical and design faults. Software failure is a random event relative to the observer, and so "reliability"

cannot be an issue of concern when applied to a system that includes software, but rather model the process as stochastic. Reliability is then simply the probability of successful operation for a given period under given conditions. Historically, hardware evaluation has concentrated on operational life, thus focusing on the influence on dependability of the system structure. Software evaluation has mainly concentrated on the development-validation phase, focusing on the reliability growth of single-component systems. Many models are aimed at predicting the future reliability from the failure data accumulated in the past.

A system failure may be due to either the activation of an internal fault that was previously dormant (which could have been physical or design faults) or an external fault originating from the human or physical environment of the system. The variation of the failure domain could be a result of accumulation of physical faults which remain dormant under low load conditions and are progressively activated as the load increases. Variation of the failure domain might also be a consequence of the creation of temporary faults resulting from the presence of rarely occurring combinations of conditions, e.g. change in parameters of a hardware component, synchronisation and marginal timing problems, etc. The failure probability at execution time may be considered as a random variable. The deployment of computerized safety-critical applications is a "bottleneck" problem since experimental checks of whether a system complies with its requirements in terms of catastrophic failure occurrence are impractical. We thus have to rely on predictive evaluations, whose confidence has to be determined via checks performed on less critical systems.

Let R_d be the probability of no system failure during an execution run comprising k input point selections. We then have:

$$R_d = \prod_{j=1}^k [1-p(j)]$$

R_d is the discrete-time system reliability.

Let $t_e(j)$ be the execution duration associated to the j th input selection, and t denote the time elapsed since the execution start:

$$t = \sum_{j=1}^k t_e(j)$$

The notion of (isolated) input point is not very well suited for a number of situations, such as control systems, executive software, hardware. Thus, let us assume that there exists a finite limit for $p(j)/t_e(j)$ when $t_e(j)$ becomes vanishingly small. Let $\lambda(j)$ be this limit:

526 Artificial Intelligence in Engineering

$$\lambda(j) = \lim_{\forall j, t_e(j) \rightarrow 0} \frac{p(j)}{t_e(j)}$$

The expression of R_d may be rewritten as follows, where $O(x)$ denotes a quantity tending toward zero faster than x :

$$R_d = \prod_{j=1}^k (1 - \lambda(j) t_e(j) + O[t_e(j)])$$

$$R_c = \lim_{\forall j, t_e(j) \rightarrow 0} R_d = \exp\left(-\int_0^t \lambda(\tau) d\tau\right)$$

R_c is the continuous-time system reliability, and $\lambda(t)$ its failure rate.

When $p(j)$ and $t_e(j)$ are assumed to be constant over the executions, $\forall j, p(j) = p, t_e(j) = t_e$, the above relations can be rewritten as:

$$R_d = (1-p)^k \quad t = kt_e \quad \lambda = \lim_{t_e \rightarrow 0} \frac{p}{t_e}$$

$$R_d = [1 - \lambda t_e + O(t_e)]^{t/t_e} \quad \text{and} \quad R_c = \lim_{t_e \rightarrow 0} R_d = \exp(-\lambda t)$$

This relation simply recalls that the exponential distribution may be seen as the limit of a geometric distribution. It could be argued that the above formulations are in fact better adapted to design faults or to external faults than to physical faults, since they are based on the existence of a failure domain, which may be felt as not existing with respect to physical faults as long as no hardware component fails. It should be remembered that, from a physics reliability viewpoint, there is no "sudden, unpredictable" failure. In fact, a hardware failure is due to anomalies (errors) at the electronic level, and these anomalies are caused by physico-chemical defects (faults). Stated in other terms, there are no fault-free systems - either hardware or software, there are only systems which have not yet failed [4]. However, the notion of operational fault, i.e. which develops during system life is - although arbitrary - a convenient notion.

The relationship between the $X_{j,i}$'s and the Y_j 's is determined by the policies of maintenance and of service restoration. A policy which accepts as special cases the policies generally considered in dependability evaluation is as follows: the j th system modification is undertaken after a_j failures have occurred since the j -1th system modification. We then have:

$$t_j = s_{j,a_j} \quad \text{and} \quad Y_j = \sum_{i=1}^{a_j} X_{j,i}$$

Two extreme special cases of this general policy are:

- (a) $a_j = 1 \forall j$, thus $Y_j = X_{j,1}$: service is restored only after a system modification has been performed; this case relates to:
- a usual hypothesis for software reliability (growth) models,
 - the case of critical systems, after a (potentially) dangerous failure occurrence;
- (b) $a_1 = \infty$, thus $Y_1 = \sum_{i=1}^{\infty} X_{1,i}$: service is restored without any system modification ever being performed; this case relates to:
- hardware, when maintenance consists in replacing a failed part by an identical new one;
 - software, when no maintenance is performed, service restoration corresponding always to a restart.

Although this policy is more general than the usually considered policies, it is noteworthy that it is however a simplification of real life, for (at least) the phenomena of (a) interweaving of failures and corrections, and (b) failure rediscoveries.

Let us consider the case where the $X_{j,i}$'s are exponentially distributed: $f_j(t) = \lambda \exp(-\lambda_j t)$ '. The inter-failure occurrence times constitute in such a case a piecewise Poisson process. Space limitation prevents us from giving the expression of $h(t)$ the failure/renewal density as a function of the failure rates $\lambda_1, \lambda_2, \dots$.

The results, may be summarized as follows:

- the failure intensity $h(t)$ is a continuous function of time;
- when $a_1 = \infty$, we are in the case of a classical renewal

process; then $h(t) = \lambda_1$, which is the formulation of a stable reliability;

- when the a_j are finite,

a condition for $h(t)$ to be non increasing function of time, i.e. a condition for reliability growth, is:

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_j \geq \dots$$

the smaller the a_j 's, the faster the reliability growth.

528 Artificial Intelligence in Engineering

METRICS

The three classes of metrics quality measurement are anomaly-detecting metrics (for deficiencies in documentation or source code), predictive metrics (measurements of logic of the design and implementations) and acceptance metrics (measurement applied to the end product to assess its final compliance with requirements). The relationship of the quality factors to the activities in a system life-cycle are shown on Table 1, where X denotes quality factors that should be measured and Y indicates areas where input of poor quality is realized. A trade-off between the cost to develop high quality software and to measure quality, and benefits of life cycle savings due to quality must be made.

The basic types of system organizations that were evaluated at Central Computing Services (CCS) were:

1. Processing Mode (Batch/Interactive)
2. Concurrency (Uniprogramming/Multiprogramming)
3. Memory Management (Manual/Virtual Memory)

A performance index was used as a descriptor to represent a system's performance of some of its aspects. For CCS interactive systems, the indices on Table 2 were used. Below a certain threshold, human users cannot even appreciate the differences among response times. Beyond another threshold, the level of satisfaction drops sharply, and feeling of frustration and impotence prevails. The characteristics of human users should always be given a great deal of attention by system designers and evaluators. In general, the response depends on the particular message on the system, and on the rest of the work load that the system is running. The main characteristics of measurement tools on educational computer systems at CCS were:

- | | |
|--------------------------------|--------------------------|
| (a) Interference | (b) Compatibility |
| (c) Accuracy | (d) Cost |
| (e) Resolution | (f) Scope |
| (g) Ease of use | (h) Ease of installation |
| (i) Prereduction capabilities. | |

A modelling approach to the evaluation of computer systems performance is said to be analytic if the model is solved by a technique which differs from simulation. Probabilistic models are more widely used at CCS than deterministic ones. For educational computer systems, program performance indices used were time and space complexity, and indices of program behaviour.

DISCUSSION AND CONCLUSION

A unified formulation of the system behaviour was derived from a stochastic Petri Net Model, encompassing hardware and software. At the initial instant, the system was assumed non-failed, and instants of service restoration after system modification were recorded, as well as instances of failure occurrences. The system was assumed to be the same after service restoration as it was before failure occurrence. If the system failure was due to a software fault, service restoration was made a pure restart, consisting in supplying the system with an input pattern

different from the one which led to failure. If the system failure was due to a hardware failure, the failed part was replaced by an identical non-failed one. When services were restored only after a system modification had been performed, this case related to a usual hypothesis for software reliability (growth) models and the case of critical systems after a potentially dangerous failure occurrence. When the services were restored without any system modification ever being performed, the case related to hardware when maintenance consisted in replacing a failed part by an identical new one, and to software when no maintenance was performed with service restoration corresponding always to a restart.

Educational computer system at CCS were run on VRX/E-processor based machines. The System Performance Measurement (SPM) utility on the VRX/E computers at CCS provided a method for checking the performance of the system. During the data collection phase, data samples were collected and the data was output to a data log file in each host processor in which a data collection job was running. The data log file was either the system log file associated with the host processor or a file that was defined by the user. SPM collected performance data for the system bus interface (SBI) and system memory (SM), which is a memory organization that contains one or more data storage processors (DSP) with many disks accessible from each DSP. Both sets of statistics contained data related to I/O operations. The SBI statistics gave information about I/O operations through the system bus, while the SM statistics gave information on I/O operations to each DSP. Each host processor maintained its own SBI statistics, while the system memory statistics were maintained by the master host processor. The SPM routine permitted one to specify both the length of the collection period and the length of the sample period.

For the Memory Utilization printout, samples were taken at the sample times. For the other printouts, data was collected during the sample periods, and the accumulated totals for each sample period were shown for the sample time that ended the secondary data collection jobs which frequently checked the status of the primary data collection job. During the data conversion phase, one can start the data conversion job, in any host processor. The primary data conversion job then causes secondary data conversion jobs to be started in other host processors if the host processor has output to a data log file. The function of the secondary data conversion job was to copy data from its associated data log file to an inter-host shared file that was accessible to that primary data conversion job. The primary data conversion then caused printouts to be created for each host processor that was running a data conversion job; it also created two printouts that showed information about the system. Below is a full list of the reports that were provided by the SPM:

(a) CPU UTILIZATION

- Percentage CPU Utilization
- Levels of CPU Utilization
- CPU Utilization Per Job
- Task CPU Utilization
- Percentage Wait Time
- DSP CPU Utilization



530 Artificial Intelligence in Engineering

- DSP CPU Utilization Histogram.
- (b) MEMORY UTILIZATION
 - Percentage Memory Utilization
 - Levels of Memory Utilization
 - Average Time/Memory Per Job
 - Task memory Utilization.
- (c) PAGING ACTIVITY
 - System Page Faults
 - Percentage Paging Activity
 - Percentage Paging Per Job
 - Task Paging.
- (d) CASHING ACTIVITY
 - System Bus I/Os
 - Percentage Cache Hit Rates
 - DSP Cache Scans
 - Cash Hit Rates Histogram
 - Percentage Cache Utilization
 - Cache Utilization Histogram.
- (e) I/O DEVICE UTILIZATION
 - Device Utilization
 - Inter Host Device Utilization
 - System Memory Devices.

The SPM produced a number of graphs to aid in analyzing the performance of the system. The graphs could be used to get information about CPU usage, memory usage, paging activity, caching activity, and peripheral device usage. It should be remembered that SPM was sample based and peaks could slip in between samples, and was a tool used for identifying performance problems but not present solutions to these problems. The areas to be investigated could be identified. In conclusion, the failure intensity was found to be a continuous function of time.

Design evaluation techniques were determined and utilized in the software development process. This led to an improvement in software productivity, reliability, manageability, and maintainability. A stochastic model dependability of educational computer systems has facilitated the standardisation of software and curriculum. The benefits of the model to CCS, though not yet complete, has been primarily in the evaluation of the computer systems, especially the system efficiency, the quality of the human-computer interface, and in maintenance of computer systems in general. If an expert system shell was used in the stochastic model of dependability, then the suitability and correctness of the reasoning techniques employed in the application software as well as the standard and quality of educational computer systems running at CCS could be investigated in a similar way to that done by [1].

REFERENCES

1. BERRY D.C. and HART A.E., 1990, Evaluating expert systems, *Expert Systems*, November 1990, Vol.7, No.4.
2. TUCKER R.N., 1988, Comparative study on criteria and procedures for the evaluation of educational software, *A study by the ICEM Information Technology Sub-Committee under UNESCO Contract No. 106.534.6 of 1988*.
3. FETZER J.H., 1988, Program verification: the very idea, *Communications of ACM*, September 1988, Volume 31, No. 9, p1048-1063.
4. LAPRIE J.C., 1989, Hardware and Software Dependability Evaluation, *Information Processing 89*, p109-114, IFIP World Congress Series, Volume 11, 1989.



TABLE 1: Quality Factors used at CCS and their relationships

DEVELOPMENT		EVALUATION		POST DEVELOPMENT			
Factors	Requ irem ents Anal ysis	Desi gn	Code and Debu g	Syst em Test	Opera tion	Revis ion	Tran siti on
Correctness	X	X	X	Y	Y	Y	Y
Reliability	X	X	X	Y	Y	Y	
Efficiency	X	X	X		Y		
Integrity	X	X	X		Y		
Usability	X	X		Y		Y	
Maintainability		X	X			Y	Y
Testability		X	X	Y		Y	Y
Flexibility		X		X			Y
Portability		X	X				Y
Reusability		X	X				Y
Interoperability	X	X		Y			Y

X - Where quality factors should be measured.

Y - Where input of poor quality is realized.

**TABLE 2: Indices of Computer Performance**

INDEX CLASS	EXAMPLES OF INDICES	GENERAL DEFINITION
Productivity	1. Throughput rate 2. Production rate 3. Capacity (maximum throughput rate) 4. Instruction execution 5. Data-processing rate	The volume of information processed by the system in the unit time.
Responsiveness	1. Responsive time 2. Turnaround time 3. Reaction time	The time between the presentation of an input to the system and the appearance of the corresponding output.
Utilization	1. Hardware module (CPU, memory, I/O channel, I/O device) utilization 2. Operating system module utilization 3. Public software module (e.g. compiler) utilization 4. Data base utilization	The ratio between the time a specified part of the system is used (or used for some specified purposes) during a given interval of time and the duration of that interval.