International Journal of Reliability, Quality and Safety Engineering
Vol. 13, No. 2 (2006) 149–156
© World Scientific Publishing Company

IMPACT OF SOFTWARE FAILURES ON THE RELIABILITY OF A MAN-COMPUTER SYSTEM

WOJCIECH ZAMOJSKI* and DARIUSZ CABAN[†]
Institute of Computer Engineering, Control and Robotics
Wroclaw University of Technology
ul. Janiszewskiego 11-17
50-370 Wroclaw, Poland
**wojciech.zamojski@pwr.wroc.pl*
[†]dariusz.caban@pwr.wroc.pl

Received 15 September 2005
Revised 15 December 2005

Software failures and human errors are the most common reasons of inoparability of computer systems. Computers are increasingly reliable, but the level of transcient faults, caused by errors hidden in the programs, remains the same. For this reason software is becoming the key factor in the synthesis of highly reliable systems. Software errors do not result from operation, they either exist from the start or are inserted when patching or upgrading it. Some errors result from incorrect human interaction or unexpected environmental changes. Assessment of software impact requires functional-reliability approach to reliability analysis: the software fault occurs when it causes incorrect operation and not when it is introduced to the system. It is proposed to use software reliability models in system analysis, to predict the intensity of software faults. The software recovery after a failure is realized by restarts of various extend (microrestarts, minirestarts and macrorestarts). The impact of software failures and restarts on system availability is assessed.

Keywords: Software reliability; system availability; recovery.

1. Introduction

Reliability of a computer system, defined in its broader sense as the ability to function correctly, is influenced by its hardware, software and usage patterns.[1,2] This is obvious, yet most effort is put into the improvement of the hardware. Even though a hardware failure is very improbable.

Software and human errors are often neglected in reliability considerations. They are much more numerous than hardware faults. Fortunately, these failures are usually simple to recover from. So, their consequences are not as grave as their number warrants.

The paper shows that the software and human errors are the dominating factors when predicting the operational availability of modern computers. Their consequences can be significantly reduced by widespread use of microrestarts: a technique for improving data recovery.[3] A unified approach to the analysis of such systems is proposed.

2. Model of a Man-Machine System

The presented model was developed for manned computer systems, used to realize numerous tasks at the same time. It can be generalized to describe other man-machine systems, where there are technical resources, controlled by fixed procedures (algorithms), operated by men.

The above statement determines all the interacting components of such a system:

- set of tasks J (jobs) that are processed by the system,
- set of technical resources H (hardware components) used for processing the tasks,
- set of software components (applications, systems programs, shared libraries, operating system functions) denoted as S_{soft},
- system users and operators S_{man}.

The interactions of these components are expressed as task configurations. Each active task in the system has its own configuration, i.e.,

$$S_{\text{conf}}(j) = \langle H(j), S_{\text{soft}}(j), S_{\text{man}}(j) \rangle, \quad j \in J, \tag{1}$$

where $H(j)$ is the subset of hardware components used in processing the task j, $S_{\text{soft}}(j)$ is the subset of software elements, $S_{\text{man}}(j)$ — the subset of human users/operators (often just a single user).

It is convenient to consider such a system to have the same set of tasks throughout its operation. This can be done using the notion of inactive tasks, which are not currently processed. The configuration of an inactive task consists of empty subsets of hardware, software and users.

Task configurations change in time as the tasks are being processed. The changes are determined by the software, reacting with the system users. Some of the changes may be the result of detecting system failures and reacting to them. This is called system reconfiguration.

The subsets of resources used by the tasks do not need to be disjoint. A resource that can be allocated to more than one configuration at the same time is called sharable, whereas one that cannot is nonsharable. Some resources, such as the central processor, are "time-sharable". This is a technique that allows sharing of resources that are essentially nonsharable, by very fast switching of the allocation in time. For the purposes of the presented considerations, these resources are treated as sharable (to avoid very numerous changes in task configurations, which would otherwise occur in the model).

The state of the system is expressed by the current configuration of all its tasks, i.e.,

$$S_{\text{conf}} = \langle S_{\text{conf}}(j) : j \in J \rangle. \tag{2}$$

The state changes in time during system operation, since both the task configurations change and the informational state of hardware and software changes. Furthermore, the above definition may be inadequate for State-Transition modeling, since the transitions depend on sequences of changes of the configurations. For this purpose, the notion of system chronicle may be introduced to the definition of task configurations, as proposed by Zamojski.[2] The chronicle is defined as a set of those moments in time, when the configuration has changed, and the change has impact on further system operation.

3. System Malfunctions and Renewal

In a man-computer system there are three sources of malfunctions: the hardware faults, the bugs in the software and human errors.

Permanent hardware failures cause some of the resources to become unusable. The tasks that use these resources are affected. They either change their configuration or become inoperational. Thus, the system performance degrades.

Transient hardware faults affect just the state of the software (its code and the data). Thus, they manifest themselves similarly to software faults.

Software bugs are caused by programming errors. They reside in the software since its manufacturing. Some are eliminated when the software is patched or upgraded, though new ones can be introduce in the process. Software-faults occur when erroneous code is used (executed). During normal operation, disclosed software errors are not eliminated: when exactly the same conditions occur the software will fail again. This is not very likely as the erroneous software state usually depends on the coincidence of a number of factors.

Human errors in a man-machine system are the result of misunderstandings during the interactions between the users and the software. Ideally, the software should detect all the errors in the input and take corrective actions (though this is virtually impossible). For the purpose of these considerations, human errors and software bugs are undistinguishable.

Figure 1 illustrates a typical situation when a hardware fault occurs during the realization of a task. The hardware needs to be repaired (technological renewal period τ_{TR}). Then, the data lost as the result of this failure must be recovered (information renewal period τ_{IR}). While the technological renewal time is the same for all the tasks sharing the broken resource, the informational recovery differs for each one. Of course, for every task j,

$$\tau_J^{(j)} = \tau_{TR} + \tau_{IR}^{(j)} + \tau_{J0}^{(j)}, \tag{3}$$

where τ_{J0} is the time required to complete the task if there are no failures.

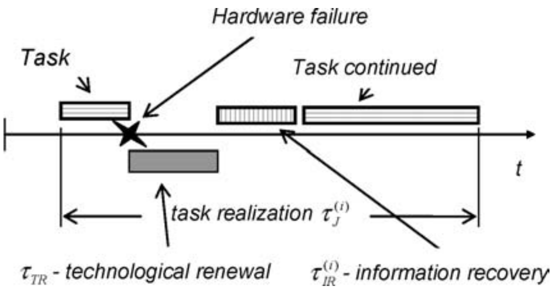
Hardware failure
Task
Task continued
t
task realization $\tau_J^{(i)}$
τ_{TR} - technological renewal
$\tau_{IR}^{(i)}$ - information recovery

Fig. 1. Task completion in presence of a single hardware failure.

If a software or human fault occurs during task realization the situation is similar. There is no technological renewal in these cases. Instead, just the informational recovery is performed. It is assumed that the reexecution of the task does not necessarily result in the repetition of a similar failure.

Modern computer systems have very reliable hardware; the hardware failure rate is measured in fractions of a single failure per annum. The technological renewal, on the other hand, is very efficient — the faulty modules are replaced. This results in downtimes of a few minutes (if the spare parts and qualified personnel are at hand) or few days at most.

On the other hand, software failures (including those caused by human errors) are very frequent. There can well be a few failures per day. The information recovery is usually completed in a few seconds, upto a few minutes. In some cases, especially if it requires manual entry of data by the user, this recovery may take hours! As such, the impact of software failures is much more noticeable than that of hardware faults. For this reason, in the following considerations, the effort is concentrated on predicting this impact.

4. Software and Human Faults

As already stated, the software contains errors that manifest themselves only in certain situations. Before release, programs are debugged to localize and remove majority of these errors. Still, some usually remain undetected. There are three classes of programming errors that are particularly hard to debug:

- synchronization errors, that is those that manifest themselves only when a number of independent, asynchronous events occur in a particular sequence;
- human interaction errors, caused by user actions that were not foreseen by the programmers or that were specifically disallowed (strictly speaking the last situation is not a software error but a human one);
- errors in the fallback procedures, that is in the software that is only activated as a result of detected hardware or software failures; these include errors in the software responsible for task reconfiguration and information recovery.

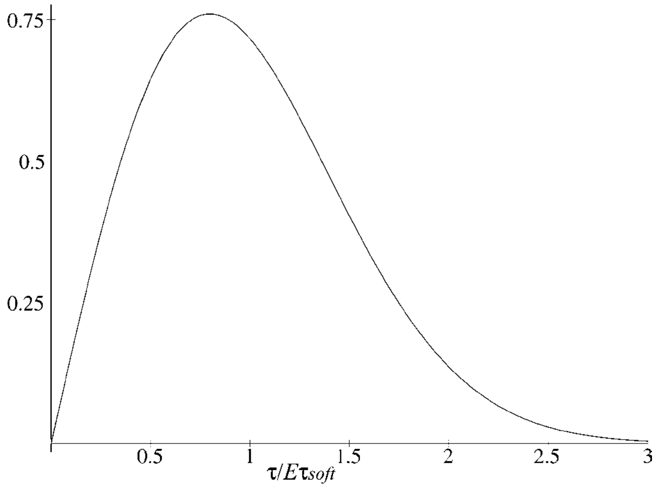

Fig. 2. Pdf function of software failure occurrence.

The existence of errors in the software does not necessarily lead to a failure. Software reliability models are used to predict the distribution of the time to software failure on the basis of the expected error content.[4,5] Empirical studies lead to the conclusion that the Schick-Wolverton (SW) model is most fitting. It assumes the Rayleigh distribution of the time to software failure, given by the formula:

$$F(\tau) = 1 - e^{-\varphi n \tau^2}, \tag{4}$$

where n is the number of bugs in the software; φ is a constant of proportionality. Both these parameters are determined statistically, based on the recorded software failure occurrences during debugging.

Figure 2 shows the pdf function of the software fault occurrence. The most likely time of failure occurrence is the average of the distribution:

$$E\tau_{\text{soft}} = \frac{1}{2}\sqrt{\frac{\pi}{\varphi n}} \tag{5}$$

The software model is fairly accurate, provided that its usage patterns coincide with those prevailing during debugging. If these differ significantly, then the predictions based on debugging records may be very inaccurate — they are still used since there are no better available. This may be particularly noticeable in case of the human interaction failures and hardware fault handling ones.

5. Factors Influencing Data Renewal

Informational recovery must follow any failure of a man-computer system. The term covers all the actions performed in order to obtain the same system state as it had just before failure. This is achieved in two stages: first, the system is brought to a state, recorded during operation, that it had prior to failure. Then, all processing realized since the last recorded state is repeated.

Data renewal may be a lengthy process, especially if the loss of data necessitates its manual reentering. The solution to this lies in frequently storing the system states and thus reducing the amount of data to be restored. The frequency of storing the restart data and its extend are the basis for classification of informational renewals:

- Microrestarts are performed if the diagnostic mechanisms allow immediate detection of a failure and there is sufficient data to perform a restart involving repetition of just a few instructions in the currently active task.[3] The restart duration is very short and it is limited to a single task. Unfortunately, this requires extensive data redundancy and some hardware support. In the model, microrestarts are characterized by the probability of occurrence $p_{\mu IR}$ and the percentage of a task to be reprocessed $\eta_{\mu IR}$.
- Minirestarts occur when the consequences of software failures are localized to single tasks or just a few tasks sharing some resources. It may involve repetition of the affected task at most. These are also modeled using the probability of occurrence p_{mIR} and reprocessed task percentage η_{mIR}.
- Macrorestarts are the global procedures leading to the recovery of system activity. These are performed when all the other methods fail. This may happen if a failure is not detected in time to localize it to a single task or if there is no data redundancy in the system, allowing a less extensive restart. Macrorestarts involve repetition of a significant number of tasks, active when the failure occurred (often all such tasks).

Considering the above classification, the mean recovery time can be modeled as:

$$E\tau_{IR} = \begin{pmatrix} p_{\mu IR}\eta_{\mu IR} + (1 - p_{\mu IR})p_{mIR}\eta_{mIR} \\ +(1 - p_{\mu IR})(1 - p_{mIR}) \end{pmatrix} \tau_{J0}. \tag{6}$$

This may be used to model the informational recovery in the man-computer model.

6. Reliability Measures of a Man-Computer System

The performance of a man-computer system can be considered from the point of view of its ability to perform a specific task in time. This is expressed as the system availability of the task, i.e.

$$A_J = \frac{E\tau_{\mathrm{soft}}}{E\tau_{\mathrm{soft}} + E\tau_{IR}}. \tag{7}$$

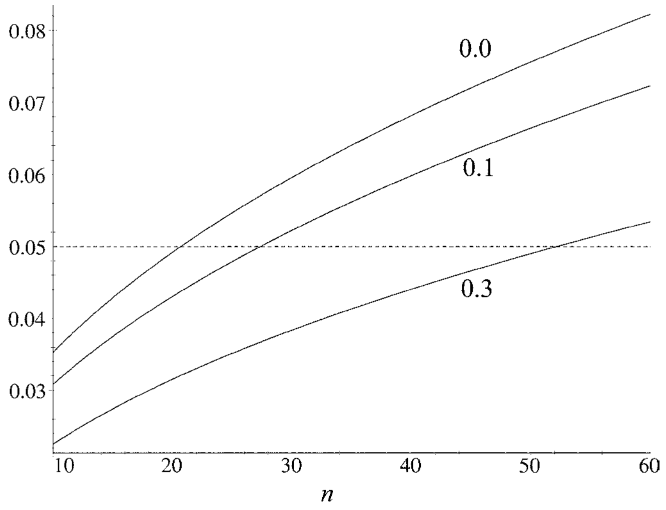

Fig. 3. Task unavailability as a function of software errors, drawn for probability of microrestart $p_{\mu IR}$ set to 0, 0.1 and 0.3.

Taking into account Eqs. (5) and (6), this gives a simple (but interesting) assessment of the task availability.

It was used to determine the task availability shown in Fig. 3 (actually the plots illustrate the task unavailability, defined as ones complement of availability). These were drawn for a man-computer system, in which each minirestart requires repetition of half a task and microrestart 0.01 τ_{J0}. Average task duration is 0.1 hours, error activation constant φ is 0.005 hours^{-2}. Probability of a macrorestart is modeled at 10% of all restarts. The effect of reducing minirestarts by microrecovery is analyzed. The results show that introduction of microrestarts can very significantly improve task availability.

The global approach is to consider the performance of the whole system. This is a lot more difficult, since the various tasks are realized in parallel. While some tasks are inoperational (being restarted) other might be processed correctly. For this reason availability is inadequate to measure the reliability of the system. Instead, the efficiency of the system is considered, i.e., how the system throughput (measured as the number of tasks processed in a fixed period of time) is affected.

The efficiency is determined by simulating the transitions between the reliability functional states of the man-machine model, discussed in Sec. 2. The probabilistic transitions model the changes in the task states (tasks becoming active and inactive, tasks reaching their restart points), changes in hardware configurations, hardware/software failures, renewal of hardware resources and restarts.

After accumulating the results of an adequate number of simulation runs, the system efficiency is determined using statistical approach. This yields both the average system efficiency and empirical distribution of the measure.

7. Conclusions

Simple considerations presented in the paper show that the reliability of man-computer systems depend mainly on their software and competence of users.

Surprisingly, the most effective method of improving the operational availability of computers is by improving the informational recovery. Figure 3 shows that the same level of availability is achieved by reducing the number of errors from 52 to 21 (i.e., by 60%) as by ensuring 30% microrestarts!

References

1. W. Zamojski and D. Caban, Trends in the theory and engineering of reliability applied to the NBIC technology. in *Proc. 3rd Safety and Reliability Int. Conf. KONBIN* (Gdynia, 2003), pp. 383–390.
2. W. Zamojski, Functional-reliability model of a computer-man system (in Polish) in *Collective Works: Computer Engineering*, W. Zamojski (ed.) (WKiL, Warsaw, 2005).
3. D. Patterson *et al.*, *Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies.* (Berkeley Comp. Sc. Technical Report, 2002).
4. S. H. Kan, Modeling and software development quality, *IBM Systems Journal* **30**(3) (1991).
5. G. J. Schick and R. W. Wolverton, An analysis of competing software reliability models, *IEEE Trans. on Software Engineering* **SE-4**(2) (1978).

About the Authors

Wojciech Zamojski is a Professor of Computer Science at Wrocław University of Technology. He holds MSc. (1965), Ph.D. (1969) and DSc (1980) in Computer Science. In 1985 he was granted full professorship by the Polish authorities. He coordinates numerous national and international research and education programmes on systems reliability, artificial intelligence and computer based training.

Dariusz Caban is a Lecturer at Wrocław University of Technology. He holds MSc. (1978) and Ph.D. (1982) in Computer Engineering. He actively participated in international educational and research programmes on computer design, systems simulation and reliability modeling.