C. Wohlin, P. Runeson and A. Wesslén, "Software Reliability Control using Statistical Usage Testing", Proceedings First Norwegian ENCRESS Seminar, Oslo, Norway, 1996.

Software Reliability Control using Statistical Usage Testing

Claes Wohlin, Per Runeson and Anders Wesslén

Department of Communication
Systems, Lund University
Box 118, S-221 00 LUND
SWEDEN
Fax: +46-46 14 58 23
e-mail: (claesw, perr, wesslen)@tts.lth.se

Abstract

Software reliability is a frequently used term, but very seldom the reliability is under control, at least not in quantitative terms, during a software development project. This paper outlines a method, Statistical Usage Testing (SUT), which provides the opportunity to estimate and predict, and hence control software reliability. SUT is the reliability certification method described as a part of Cleanroom software engineering. The main objective of SUT is to certify the software reliability and to find the faults with high influence on reliability. SUT provides statistically based stopping rules during test as well as an effective use of test resources, which is shown by practical application of this and similar methods. This paper presents the basic ideas behind SUT and some of the current research being conducted in this area at the department of Communication Systems.

1.0 Statistical Usage Testing

1.1 Introduction

Statistical Usage Testing (SUT) is the certification part of Cleanroom Software Engineering. Traditional testing is often concerned with the technical details in the implementation, for example branch coverage, path coverage and boundary-value testing. SUT on the contrary takes the view of the end user. The focus is not to test how the software is implemented, but how it fulfils its intended purpose from the users' perspective. SUT is hence a black box testing technique taking the actual operational behaviour into account. It treats the software as being a black box and is only concerned with the interfaces to the users.

SUT has two main objectives:

- To find the faults which have most influence on the reliability from the users' perspective.

- To produce data which makes it possible to certify and predict the software reliability and thus to know when to stop testing and to accept the product.

The latter implies that a usage profile is needed as it is not possible to certify and predict reliability in operation from other black box testing techniques.

1.2 Cost effectiveness

Studies show that usage based testing is an efficient way to find the faults which have most impact on the reliability [Adams84]. The referenced study shows a gain with a factor 20. From seven software development projects at IBM it is concluded that 1.6% percent of the faults caused 58% percent of the failures during operation, while 61% percent of the faults caused only 2.8% percent of the failures. Thus it is more efficient to remove the 1.6% of the faults.

Software reliability depends not only on the number of faults in the software, but also on how the software is used. A fault in a part of the software which is frequently used has larger impact on the reliability than a fault in a less frequently executed part.

As the study by Adams shows, the most efficient way to improve software reliability is to remove the faults causing most of the failures, and not those which occur very seldom. In SUT test cases are selected to test according to the operational usage and are hence effective in order to find the faults which affect software reliability.

1.3 Software reliability certification

The other objective of SUT is reliability certification, i.e. getting a reliability measure corresponding to the intended operational usage. To certify software reliability, there is a need for a reliability model, which based on failure data from testing can estimate and predict software reliability.

Most reliability growth models which can be used for reliability certification and prediction have a common prerequisite: usage based testing. This prerequisite has not been the major focus in software reliability research earlier, but during the last couple of years the interest has grown.

1.4 Software acceptance

The software reliability measure obtained in usage based testing can be used as a criterion for software acceptance as well as a stopping rule for the testing.

The contract between a supplier and a purchaser often includes a software reliability requirement to be fulfilled at delivery. Neither the supplier nor the purchaser however can prove that the requirement is fulfilled or not. SUT provides an opportunity for both parties to get objective measures which may be used for judgement about the requirement fulfilment.

From the supplier’s side a question of interest is when to stop testing. Large parts of a software development project costs are spent on testing. There is a need for saving testing costs. However it can cost money for a supplier to deliver bad products as well in

terms of damages or bad reputation. This emphasizes a need for controlling software reliability by using reliability measures as stopping criteria for software testing, which are provided by SUT.

2.0 SUT models and methods

2.1 Introduction

The presentation below focuses on the research results from the department of Communication Systems, Lund University. The research has been focused on developing a dynamic usage model, although the dynamics is not included in the brief presentation below, and another issue has been test case generation from a usage model. Finally, some on-going research questions are highlighted.

2.2 Usage specification

The usage specification is a model which describes how the software is intended to be used during operation.The primary purpose of a usage specification is to describe the usage to get a basis for how to select test cases for the usage based testing. It can however be used for analysis of the intended software usage as well, to plan the software development. Frequently used parts can be developed in earlier increments and thus be certified with higher confidence. In this paper the State Hierarchy (SHY) usage specification is briefly presented [Runeson92, Wohlin94]. It consists of a usage model, which is the structural part, and a usage profile, which is the statistical part.

2.3 Usage model

The SHY usage model is a hierarchical state model. The basic concept of the SHY model is shown in Figure 1. Examples below are taken from the telecommunications field.

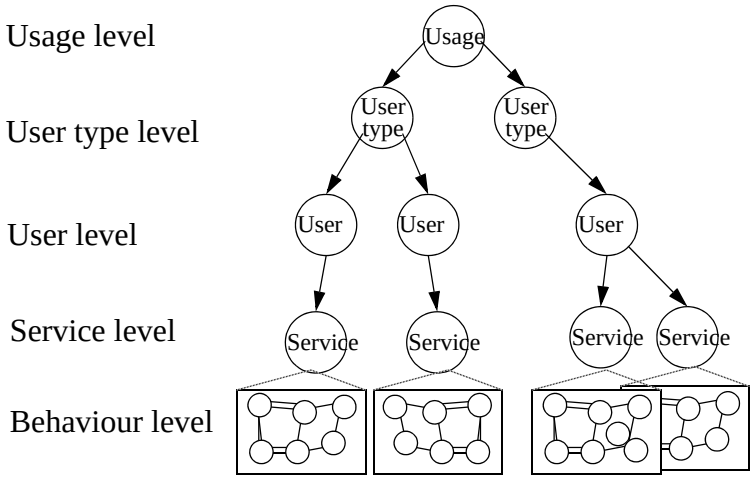

Figure 1. SHY model.

The usage is specified as a hierarchy. The state on the top represents all the usage. The users can be divided into different user types or categories, for example for a small business exchange, secretaries, other employees and modem connections. Note that this example shows that a user must not be human.

For each of the user types, a number of individuals are specified on the user level, for example one secretary, four other employees and one modem connection.

Each user individual can use a number of services, which are specified on the service level, for example “basic call” and “call forward”.

The usage of the services is then specified as plain Markov chains on the behaviour level.

The SHY model can be applied with different levels of detail depending on the application. The behaviour level can for example be excluded if less details are to be specified in the usage model.

2.4 Usage profile

The usage profile adds the probabilities for selection of the branches to the usage model. Probabilities are assigned to the transitions in the behaviour level Markov chains as well.

The probabilities are assigned based on measurement on usage of earlier releases or on expert knowledge. The SHY model makes it possible to analyse parts of the usage and assign probabilities for only that part of the model at a time, for example a user type.

The assignment does not necessarily have to be in absolute figures. Classes of usage frequency can be used, for example very frequently, frequently and seldom used. These classes can be assigned relative probabilities which may be an easier task than to assign every single probability.

2.5 Reliability model

To analyse the failure data collected during the statistical testing a reliability model is needed. Several models have been published over the last 20 years, see [Goel85] for an overview. Models of different complexity and possibility to estimate the software reliability have been presented.

One very simple model which is suitable for software certification is the hypothesis testing control chart model [Musa87]. It is based on a traditional quality control technique: sequential sampling [Grant88].

The model is based on a control chart with three regions, reject, continue and accept, see Figure 2. The control chart is constructed based on the required level of confidence in the estimation.

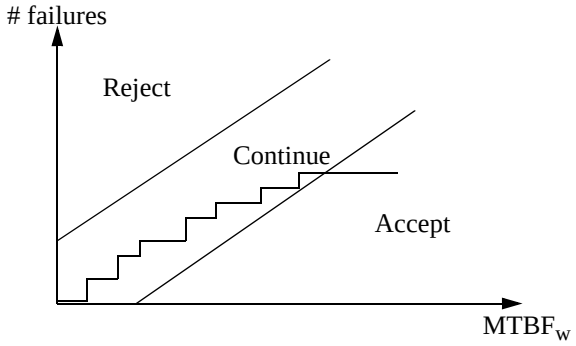

Figure 2. Hypothesis testing control chart.

The failure data are plotted in the chart, failure number towards weighted time between failures[1]. As long as the plots fall in the continue region, the testing has to continue. If the plots fall in the rejection region, the software reliability is so bad that the software has to be rejected and re-engineered. If the plots fall in the acceptance region, the software can be accepted based on the required MTBF with given confidence and the testing can be stopped.

Thus the hypothesis certification model provides a means for certifying the software and giving a reliability measure for the software as well as a means for controlling the testing effort.

2.6 SUT method outline

The models presented above can be applied according to the following method:

During specification:

1. Produce the usage model.
2. Assign the usage profile.

During test:

3. Select test cases from the usage specification.
4. Run the test cases and collect failure data.
5. Certify the software.

During step 5 a decision is made based on the certification model outcome. If the failure data plots fall in the continue region, the method is repeated from 3 to 5 again. If the software is rejected, it is put back for redesign and finally if the failure data fall in the acceptance region, the certification is terminated and the software is accepted.

1.Weighted time between failures means the measured time divided by the MTBF requirement.

3.0 Generation of test cases

3.1 Introduction

Test cases must be generated to verify the software system. Instead of developing a separate tool for the hierarchical usage model, the research has been focused on transforming the model into a representation for which tool support is available. It was decided to primarily work with translations into SDL (Specification and Description Language). To generate test cases, the SDL model must be executed on its own, i.e. separated from the system. When separating the SDL model from the system, the executer must play the roll of an oracle. The oracle analyses the stimuli from the usage specification and answers with the expected answer from the system. The stimuli from the users and the expected answers from the oracle form the test cases. A test case ends when the system reach a specific state or when the test case is of certain length. A more comprehensive treatment of test case generation in a usage-oriented context can be found in [Wesslén95].

3.2 Generation of test cases in any language

If the test cases are needed in a specific language, it can be generated automatically. A case study generating the test cases as Message Sequence Charts has been conducted successfully [Runeson95]. A process can be added in the SDL model between the environment and the root process, see Figure 3, and all signals between the environment and the root process passes through this process. The added process writes all stimuli and expected answers to a file in the wanted syntax.

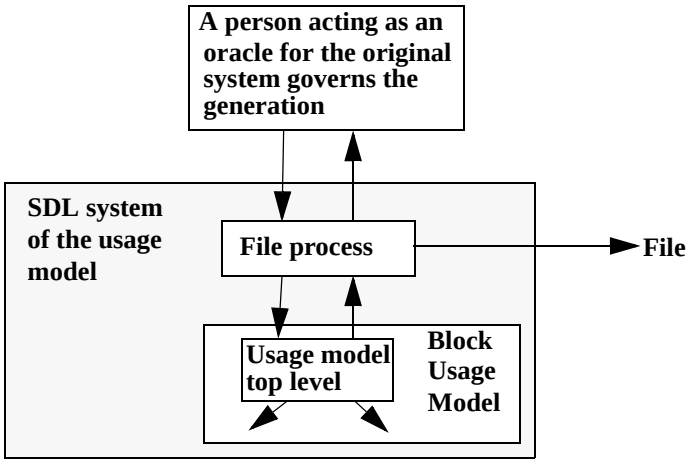

Figure 3. Generation of test cases.

When starting the generation, the process opens a new file, writes the program beginning and then waits for signals between the environment and the root process (top level in the usage model). When a signal comes from the environment the expected answer is written on the file and when a signal comes from the root process a stimulus is written. This procedure ends when the test case ends and then the process writes the ending of the program and close the file. In the beginning or end of the program, fault handling must be written to the file to handle faults in the test case or in the system.

4.0 Usage testing using an SDL model

The test cases that have been generated can be put together with the developed software system, hence providing a basis for dynamic analysis [Wohlin96], simulation and test. One of the major advantages with the approach is the opportunity to perform the analysis automatically as the system together with the usage model form a closed system. This is illustrated in Figure 4.

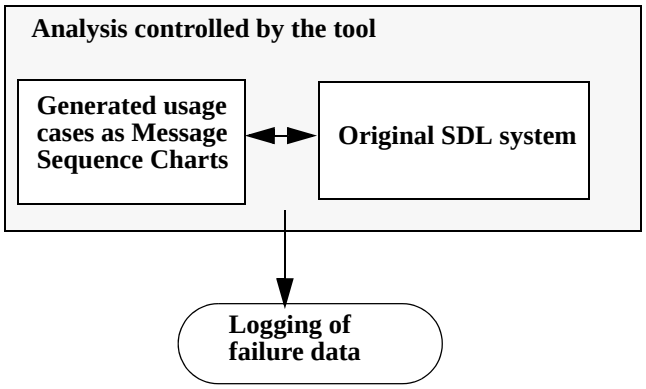

Figure 4. Usage analysis based on the generated test cases.

The execution results in a logging of the failures according to the fault handling procedure introduced in the usage model. The times between failures are recorded and these form the input to the certification procedure, for example, to the hypothesis testing control chart.

5.0 Some open research issues

Although, the area has advanced during the last couple of years, a number of open issues remains to be further investigated. Some examples are:

- estimation of software reliability during software specification and design,
- improved dynamic modelling,
- inclusion of time in the usage model,
- sensitivity to errors in the profile,
- automatic generation of test cases,
- early estimation of reliability model parameters.

6.0 References

[Adams84] Adams, E. N., “Optimizing Preventive Service of Software Products”, IBM Journal of Research and Development, January 1984.

[Goel85] Goel, A. L., “Software Reliability Models: Assumptions, Limitations and Applicability”, IEEE Transactions on Software Engineering, Vol. 11, No. 12, pp. 1411-1423, 1985.

[Grant88] Grant, E., and Leavenworth, R. S., “Statistically Quality Control” Sixth edition, McGraw-Hill Int., 1988.

[Musa87] Musa, J. D., Iannino, A. and Okumoto, K., “Software Reliability, Measurement, Prediction and Application”, McGraw-Hill Int. 1987.

[Musa93] Musa, J. D., “Operational Profiles in Software Reliability Engineering”, IEEE Software, pp. 14-32, March 1993.

[Runeson92] Runeson, P., and Wohlin, C., “Usage Modelling: The Basis for Statistical Quality Control”, Proceedings 10th Annual Software Reliability Symposium’, Denver, USA, pp.77–84, 1992.

[Runeson95] Runeson, P., Wesslén, A., Brantestam, J., and Sjöstedt, S., “Statistical Usage Testing using SDL”, Proceedings 7th SDL Forum, Oslo, Norway, September 1995.

[Wesslén95] Wesslén, A., and Wohlin, C., “Modelling and Generation of Software Usage”, Proceedings Fifth International Conference on Software Quality, pp. 147-159, Austin, Texas, USA, 1995.

[Wohlin94] Wohlin, C., and Runeson, P., “Certification of Software Components”, IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 494-499, 1994.

[Wohlin96] Wohlin, C., Runeson, P., and Wesslén, A., “Software Reliability Estimations through Usage Analysis of Software Specifications and Designs”, International Journal of Reliability, Quality and Safety Engineering, Vol. 3, No. 2, pp. 101-117, 1996.