Statistical Usage Testing for Software Reliability Control

Per Runeson and Claes Wohlin

Q-Labs
IDEON Research Park
S-223 70 LUND
SWEDEN
Phone: +46-46 18 29 97
Fax: +46-46 15 28 80
e-mail: pr@q-labs.se

Department of Communication Systems, Lund University
Box 118, S-221 00 LUND
SWEDEN
Phone: +46-46 10 33 29
Fax: +46-46 14 58 23
e-mail: claesw@tts.lth.se

Abstract

Software reliability is a frequently used term, but very seldom the reliability is under control during a software development project. This paper presents a method, Statistical Usage Testing (SUT), which gives the possibility 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 briefly discusses the theoretical basis as well as the application of the method.

Key Words: Statistical Usage Testing, Software Reliability, Usage Profile, Operational Profile, Statistical Quality Control.

1. Introduction

The software development community is not in control of the software reliability. This can be stated based on a quote from Tom DeMarco, [DeMarco82]: “You can’t control what you can’t measure”. It is not possible based on traditional development techniques to actually measure the software reliability hence the reliability is out of control.

Software reliability engineering is currently a fast growing area. Therefore the situation is not hopeless; the techniques are becoming available to control the reliability. The software process will become more and more controlled which means that methods to estimate, predict and certify the fault content and reliability will be introduced. This is the only way towards managing the process of actually engineering reliable software, instead of crafting unreliable software.

The testing techniques normally applied are aimed at finding faults. This is a devastating point of view since it implicitly accepts that errors are made and that the testers have to remove them. This reasoning may be philosophical, but it is believed to be one of the key issues in controlling software reliability. One of the most important aspects in the development is the motivation and belief in being able to do something, in this case develop software with few or no defects. Therefore it is essential to provide techniques so that the software developers can believe in developing zero-defect software, which also includes methods to certify the actual reliability level.

Cleanroom Software Engineering [Mills87, Cobb90, Dyer92, Linger94] emphasizes the intellectual control in software development. Cleanroom is a collection of several sound management and engineering techniques, in particular it is emphasized that it is possible to develop nearly zero-defect software. One of the engineering techniques emphasized in Cleanroom is Statistical Usage Testing, which is a method for statistical control of the software reliability during the system or acceptance testing phase. The objective of this paper is to propose a method for Statistical Usage Testing and to illustrate its use.

The requirement on the testing phase, when applying usage testing, is that it resembles the operational phase to be able to apply the techniques to actually remove failures most critical for the user and certify a particular reliability level. It may be difficult to forecast the exact usage, but it is important to try to understand how the software probably will be used. It is often, at least, possible to identify usage classes, and the exact probabilities are not that critical. One possibility is to certify the software for one or several scenarios, which then can be used in negotiations between developers and procurers.

Statistical Usage Testing and its opportunities to promote statistical control of the software reliability are discussed in this paper. The usage testing technique provides a basis to certify a reliability requirement, see section 2. The principles behind Statistical Usage Testing are discussed in section 3, while the techniques within usage testing are further described in section 4, i.e. usage specification and reliability certification. In section 5 a minor example is presented to illustrate the usage testing technique described. Section 6 presents some practical experiences with usage testing techniques. Finally in section 7 some conclusions are presented.

2. Reliability requirement

2.1 Requirements specification

The requirements specification contains normally both functional requirements and quality requirements; in particular reliability or availability requirements are put into the specification. The fulfilment of the functional requirements is evaluated through using the functions specified in the requirements, but the other requirements ought to be fulfilled as well. The methods for reliability certification have, however, not been available or the available techniques have not been applied. This must change, either it is no use formulating reliability requirements or methods to evaluate and certify the reliability must be applied.

Statistical Usage Testing is a method to actually certify the reliability requirement. This type of method must be applied, since it is not possible to keep applying traditional systematic testing techniques and then see the system fail in operation. This is a result of not certifying the reliability requirement. The society can not afford software system failures, neither in safety critical systems nor in other cost intensive systems. For some systems, certification of the required level is not possible, in these cases certification must be combined with correctness proofs of the software. Usage testing is thus not the solution to obtain high quality software, but it enables certification of a reliability level.

The reliability requirement aims at the reliability as perceived by the users when the software system goes into operation. Therefore usage testing must be applied, since reliability is not only the number of faults but also the actual location of them compared with usage of the software system.

The method being presented allows for certification of the reliability requirement, which means that we will be in control of the reliability before releasing the software product instead of being surprised as the system fails in operation. In particular, it provides an opportunity to formulate a stopping criterion for the testing, where the criterion is based on the fulfilment of the requirement.

2.2 Validity of certification

Usage testing as a means for reliability demonstrations is well-known and its superiority compared to coverage testing is discussed in [Musa87, Cobb90, Adams84]. An argument often raised concerning usage testing is: it is not possible to determine the usage profile, therefore usage testing is not applicable. This is, however, not completely true. It will probably be difficult to determine the usage profile exactly, but it ought still to be possible to determine usage probabilities relative to each other. It is often well known which functions will be used most frequently in a software system. Thus by making a clever guess or an estimate of the probabilities this method will still achieve preferential results over coverage testing. Successful applications of a usage profile based approach have been presented in [Juhlin92, Abramson92].

A simple example will show the gain in applying usage testing, even when the usage profile, estimated during testing, is not the profile observed as the system is put into operation.

Let us assume that we have a system consisting of three modules as illustrated in figure 1.

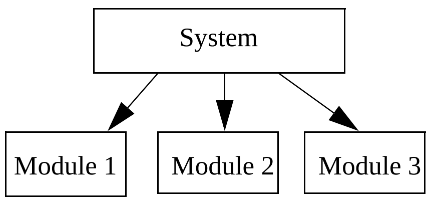

FIGURE 1. System architecture

For simplicity it is assumed that the MTBF values for the three modules are all equal to 100 when the system testing begins. All times are given in some undefined time unit. The system test can be performed either as a coverage test or a usage test. In coverage testing the modules are tested equally, see table 1, and the MTBF values for each module improve identically, e.g. to 1000, see table 2.

It is necessary to use a particular usage profile to be able to perform a usage test. The profile is identified from experience and knowledge of the probable usage of the system being developed. Test cases are generated according to this usage profile, see table 1. Usage testing gives different MTBF values for the modules. In [Adams84, Cobb90] it is shown in their particular case that based on a study of a number of projects that usage testing improved the perceived reliability during operation 21 times greater than that using coverage testing. Thus the MTBF for an arbitrary use in this example is assumed to be 21 000 (see table 3), i.e. 21 * 1000, where 1000 is the MTBF for an arbitrary use based on coverage testing. The MTBF values for the different modules are obtained based on the MTBF for an arbitrary use, and by assuming that the raise in MTBF is proportional to the probability of using a specific module. This reasoning leads to the figures presented in table 2.

The actual values are not particular interesting, but emphasis must be placed on the result after the system is put into operation and the increase obtained in perceived reliability using usage testing.

An actual usage profile is obtained as the software is put into operation. The usage profile actually being experienced during operation may differ from the one applied during usage testing. Two examples are shown in table 1. The first example (Actual operation 1) shows a minor change or error in the usage profile, whilst the second profile (Actual operation 2) is an example of a rather large difference between the profile used during testing and the profile in actual operation.

Table 1: Probabilities for using the three different modules

	Module 1	Module 2	Module 3
Coverage testing	1/3	1/3	1/3
Usage testing	0.001	0.01	0.989
Actual operation 1	0.002	0.05	0.948
Actual operation 2	0.10	0.15	0.75

Table 2: MTBF values per module

	Module 1	Module 2	Module 3
Before test	100	100	100
After coverage test	1 000	1 000	1 000
After usage test	121	314	21 230

The actual usage gives the perceived MTBF. The two testing techniques give different MTBF values in the two operation cases, see table 3. The values in table 3 are calculated by weighting the MTBF values for the three modules. The case “after usage testing (major difference)” is used to illustrate the calculations: MTBF = 0.1 * 121 + 0.15 * 314 + 0.75 * 21 230 = 15 982.

The MTBF estimates from testing are shown in the upper part of table 3. In the lower part of the table the perceived MTBF values during operation are shown.

Table 3: MTBF estimates from test and operation environments

ESTIMATION FROM TEST	MTBF
Coverage test	1 000
Usage test	21 000
PERCEIVED IN OPERATION	
After coverage test (independent of actual usage profile)	1 000
After usage test (minor change)	20 142
After usage test (major difference)	15 982

It can be concluded that usage testing gives an improved MTBF during operation. Usage testing is of course dependent on the accuracy of the usage profile. Even if the profile is erroneous, the perceived MTBF is an improvement on that obtained with coverage testing, provided the probabilities for usage are in the correct order. In this particular example the probability of usage of the modules (from highest to lowest) are ranked 3, 2 and finally module 1. The MTBF is, however, overestimated during usage testing if the profile is erroneous, but still the result is better than that of applying coverage testing.

The overall conclusion is that usage testing improves the probability (as opposed to coverage testing) of locating faults that influence the reliability during operation. This conclusion is also supported by other sources including [Adams84, Cobb90]. The example has also shown that even if the usage profile is erroneous during testing, the MTBF during operation will be higher than that obtained with coverage testing or other black box testing approaches.

3. Statistical Usage Testing

3.1 Cleanroom

Statistical Usage Testing (SUT) is the certification part of Cleanroom Software Engineering. Cleanroom is a methodology which consists of a set of software engineering and management principles as well as practices according to which software can be developed with very high quality and productivity [Mills87, Mills88, Cobb90, Linger94, Cosmo93]. The methodology has proven to be very successful when applied in software development companies in Europe as well as in the USA [Selby87, NASA90, Tann93].

Cleanroom aims at development of almost zero-defect software with measurable reliability. The basic idea is to do things right from the beginning instead of first introducing and then correcting errors. Management and engineering techniques in Cleanroom are:

- The software is developed by small teams (3–5 people) with clearly defined responsibilities. There are three types of teams, specification, development and certification teams. The teams are jointly responsible for the produced result.
- Very much emphasis is put on rigorous specifications which are the basis for the development.
- The development is done in increments, each of which is executable. By partitioning the software into increments each increment may be handled by different teams and developed in parallel. Furthermore the increments are small enough to be held under intellectual control.
- The software is developed in small steps from specification to design according to a step by step algorithm. Each step is a refinement of the prior one. For each step more details are added and finally it ends up in executable code.
- Each of the development steps is rigorously verified towards the previous steps. The verification is mainly performed by reviews, supported by a theoretically based method called “functional verification”.
- Traditional testing is replaced by certification of the software reliability by Statistical Usage Testing.

Most of the techniques are well-known but the combination of these and the management attitudes have given encouraging results concerning software quality as well as productivity and development time control. In this paper we concentrate on the certification part of Cleanroom, Statistical Usage Testing.

3.2 Usage based testing

Traditional testing is often concerned with the technical details in the implementation, for example branch coverage, path coverage and boundary-value testing, [Myers79]. 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.

3.3 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.

3.4 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 the software reliability, there is a need for a reliability model, which based on failure data from testing can estimate and predict the software reliability.

Most reliability growth models which can be used for reliability certification and prediction have a common prerequisite: usage based testing [Goel79, Jeliniski72, Musa87]. This prerequisite has been overlooked during the years, but has come into focus during the last years [Musa93, Runeson92].

3.5 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 is a possibility for both parts 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.

4. SUT models and methods

When applying SUT two kinds of models are needed, a model to specify the usage and a reliability model. In section 4.1 the usage specification is presented while the reliability models are treated in section 4.2. A method describing how to use the models during Statistical Usage Testing is presented in section 4.3.

4.1 Usage specification

The usage specification is a model which describes how the software is intended to be used during operation. Different types of models have been presented in the literature:

- Tree-structure models, which assign probabilities to sequences of events [Musa93].
- Markov based models, which can specify more complex usage and model single events [Whittaker93, Runeson92].

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. Development and certification of increments are further discussed in [Wohlin94a].

In this paper the State Hierarchy (SHY) usage specification is briefly presented [Runeson92]. It consists of a usage model, which is the structural part, and a usage profile, which is the statistical part.

4.1.1 Usage model

The SHY usage model is a hierarchical Markov chain which copes with specification of the usage of large multi-user software systems. The basic concept of the SHY model is shown in figure 2. Examples below are taken from the telecommunications field.

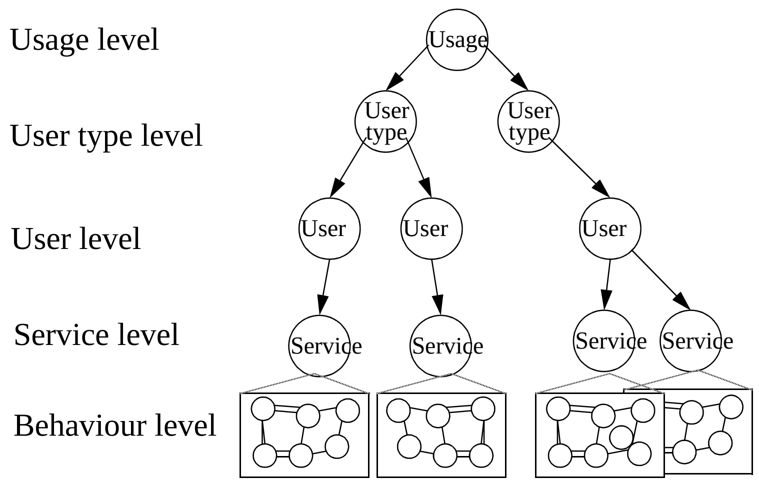

Figure 2. SHY model

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.

4.1.2 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 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.

4.2 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 3. The control chart is constructed based on the required level of confidence in the estimation.

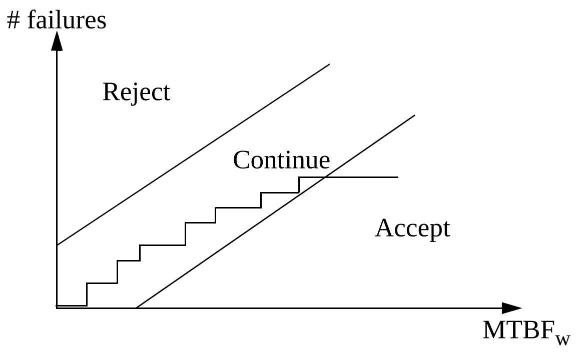

Figure 3. Hypothesis testing control chart

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.

4.3 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 stopped and the software is accepted.

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

5. Example

This section contains an example which purpose is to make the models and methods presented in section 4 easier to understand and to apply. The method followed is the one presented in section 4.3. The subsections below are numbered according to the method outline. The focus is on usage modelling, see section 5.1, while section 5.2 to section 5.5 are more briefly described since the techniques for usage specification are less known than the other techniques.

The example on which the test method is applied is a private branch exchange (PBX) for a small office, see figure 4. Five human users are connected to the PBX, one secretary and four other employees. Furthermore there is one modem line. The connection with the outer world is through two lines. More details about the example specifications are given throughout the example.

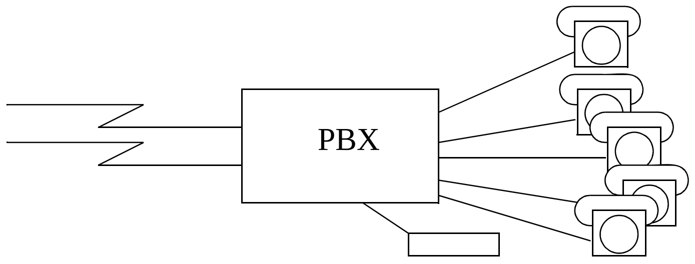

Figure 4. The example PBX system structure

5.1 Produce the usage model

The SHY usage model, see figure 2, is produced in a number of steps, each of which is small and rather easy to perform. The steps are:

1. Identify the services available for the users.
2. Identify the user types and determine which services are available for each type.
3. Determine the number of individuals for each type.
4. Specify the behaviour Markov chain for each service.
5. Instantiate the services for the users.

In the example the first three steps are fully shown, while the last is only partially performed in this paper.

The usage model is produced starting with identification of the services available. The services are internal and external basic call, internal call forward, internal call transfer and call from the network.

The different types of users are identified. In this example there are four: Secretary, employee, modem and in/out line. The employees have internal and external basic call and internal call forward. The secretary has furthermore internal call transfer. The modem line has only external basic call. The in/out lines can only handle calls.

The different individuals of the user types are identified. There is one secretary, four employees, one modem and two in/out lines in the example.

In table 4 the first three steps in preparing the usage model for the PBX are summarised.

Table 4: PBX usage

User type	Instances	Services
Secretary	1	Internal basic call (IBC) External basic call (EBC) Internal call transfer (ICT)
Employee	4	Internal basic call (IBC) External basic call (EBC) Internal call forward (ICF)
Modem	1	External basic call (EBC)
In/Out-line	2	Call (C)

The behaviour level Markov chains for the services are then specified, see figure 5 and figure 6. For the internal basic call the stimuli are selected to be: Off Hook (OfH) (lift the receiver), Dial Internal Number (DIN), On Hook (OnH). There is also an asterisk (*) stimulus which means that the transition is forced by another behaviour level Markov chain. The state with thicker line is the start state. Note that the IBC service referenced in the link table is another instantiation of the service (another user).

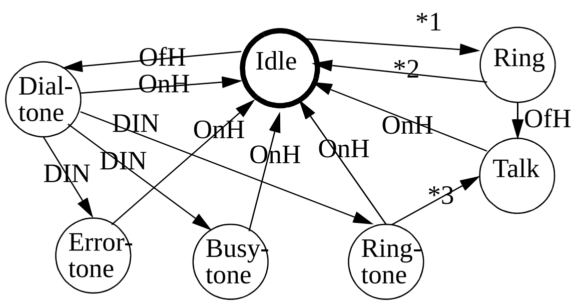

Table 5: Link table

Link	Forced by
1	IBC: Dialtone–DIN
2	IBC: Ringtone–OnH
3	IBC, ICT: Ring–OfH

Figure 5. Behaviour Markov chain for internal basic call (IBC)

The behaviour level Markov chain for the internal call transfer service is described in figure 6. The stimuli are: Activate Call Transfer (ACT), Dial Internal Number (DIN) and

Transfer The Call (TTC). The linked transitions are forced by an instance of the Internal Basic Call (IBC) service.

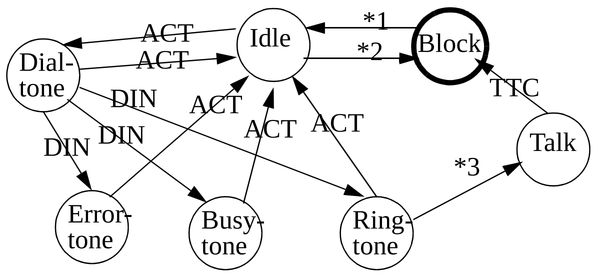

Table 6: Link table

Link	Forced by
1	IBC: Idle–OfH
2	IBC: OnH
3	IBC: Ring–OfH

Figure 6. Behaviour Markov chain for internal call transfer (ICT)

In this manner all the services are specified; in this example however only these two are specified. Finally the state hierarchy model is compounded by its parts. The usage model for the example as a whole is presented in figure 7.

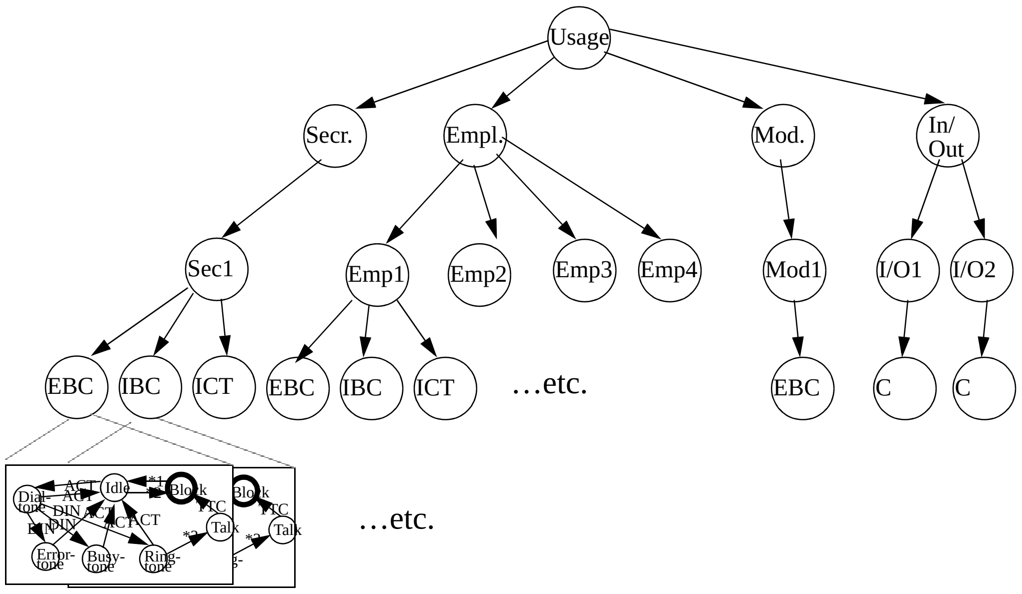

Figure 7. Usage model for the PBX example

5.2 Assign the usage profile

When the usage model is produced the probabilities for the arcs have to be assigned, i.e. the usage profile is assigned.

The assignment starts on the user level which corresponds to what is well-known, at least in terms of relations between the usage. In the example it is assumed that one out of five of the events origin from each of the in/out-lines. Among the other users it is assumed that an event from the secretary is three times as probable as events from three of the employees and equally probable as events from the fourth employee. Modem line events are equally probable as events from one of the three least probable employees. These assumptions must come from market surveys, knowledge of existing and similar systems and expert opinions.

Based on this information, it is possible to derive probabilities for the usage model hence providing a basis for generating test cases which resemble the anticipated behaviour in operation. An equation can be set up which gives the absolute probabilities for the users (P_a):

P_a(In/out1) = P_a(In/out2) = 0.2;

P_a(Secr) = 0.18;

P_a(Emp1) = P_a(Emp2) = P_a(Emp3) = 0.06; P_a(Emp4) = 0.18;

P_a(Modem) = 0.06.

To apply these figures on the SHY usage model, they have to be divided on the user types and the user individuals. The user level probabilities (P_{ul}) are given by the relations between the individuals. Since the sum of the probabilities equals one, the calculations for the employees are:

P_{ul}(Emp1) = P_{ul}(Emp2) = P_{ul}(Emp3) = 0.06/(0.06+0.06+0.06+0.18) = 0.166;

P_{ul}(Emp4) = 0.18/(0.06+0.06+0.06+0.18) = 0.5;

The sum of the absolute probabilities of a user type gives the user type level probabilities (P_{utl}):

P_{utl}(Emp) = 0.06+0.06+0.06+0.18 = 0.36;

When all of the calculations are performed the usage profile applied on the SHY model is according to figure 8:

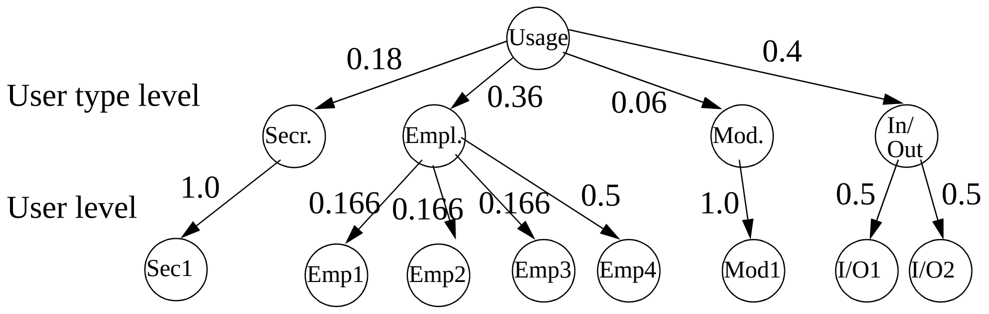

Figure 8. Usage profile for the upper levels of the example

The transitions in the behaviour Markov chains are assigned probabilities as well, except for the transitions forced by other services. This gives a complete usage specification from which test cases can be generated.

5.3 Select test cases

The test cases are selected from the usage specification by running through it beginning from the usage state and down to a single event. The actual path through the model is controlled by a random number sequence. For each event in a test case the specification is run through once. The beginning of an example test case is illustrated in table 7:

Table 7: Beginning of example test case

Event no.	Event
1	Emp3: Off Hook
2	Emp3: Dial Internal Number
3	In/Out2: Call Subscriber
…	…

5.4 Run test cases

The test cases are run as during any other type of testing. During testing, failure data are collected. The data form the basis for the certification, and are thus very important.

5.5 Certify the software

The failure data are input to the hypothesis testing model, see figure 3, which is used to certify a particular reliability level. The reliability is measured in terms of MTBF, and the certification is done with a given statistical confidence. The outcome from the certification is *reject*, *continue* or *accept*. In the case of reject, the software is sent back for redesign, in the case of continue, new test cases are selected and run, see section 5.3 and 5.4 above. If the outcome is accept the reliability level is certified and the testing can be terminated.

6. Practical experience

Statistical Usage Testing can be applied at different phases in the software life cycle. The testing can be applied during system testing or acceptance testing, but it may also be applied on software components, [Wohlin94b], which then can be put into a repository for future reuse. The reuse of components is one important aspect in the future to cope with the cost of software development. Reuse requires that reliability measures of the reusable components are stored with the component. Reliability measures must be stored together with the usage profile which has been used in the certification process. Based on the reliability of components it must be possible to calculate the system reliability. This issue is further discussed in [Poore93].

The new method presented is not fully developed, but it is beginning to be implemented and evaluated. Currently, a case study is conducted to evaluate the procedure. The study includes dynamic analysis, simulation and finally testing. The results from performing usage analysis during all these activities will form the basis for a thorough evaluation of usage analysis in the software life cycle. Furthermore, usage modelling with a hierarchical Markov chain (SHY), which then is transformed into a usage model in a high level design technique, has been used to generate usage test cases to the next release of a case tool, [Runeson95]. It is concluded from this application, that both the SHY model and the transformation into the high level design technique are useful concepts when generating usage cases. The main reason to transform the Markov model is to obtain tool support, which can be used to automatically generate test cases.

It can from this reasoning be concluded that usage testing is a useful technique, which can be applied at different phases in the life cycle with the common denominator that reliability certification is needed to stay in control of the reliability.

Application of Statistical Usage Testing or similar techniques have started at different companies, in particular in the USA. AT & T has reported that they have lowered the cost for system testing by 56% and the total cost in the project by 11.5% by applying Operational Profile Testing, [Musa92, Musa93, Abramson92, Juhlin92]. The objective with Operational Profile Testing is the same as for Statistical Usage Testing even if some of the techniques to specify the usage are different. The usage testing technique is starting to spread in Europe as well [Tann93, Cosmo93].

7. Conclusions

It is a fact that reliability or availability requirements are formulated as a part of the requirements specification, but it is also clear that neither the developer nor the procurer of the software is capable of evaluating these requirements. This is not satisfactory; the society depends so heavily on the systems that it is of outermost importance to be able to certify the software systems. A failure in operation may cause injuries either in terms of humans or at least in terms of financial losses.

A model to specify the usage has been presented and a reliability model based on hypothesis testing control chart has been described briefly. These techniques together have made it possible to formulate a method, which can be applied during the testing phase to actually evaluate the reliability requirements. The application of the proposed method has been illustrated in an example.

Some practical experiences reported in the literature as well as experience obtained while applying the proposed techniques have been presented. The overall conclusion is, that the only way towards control of the reliability before releasing a software product is through application of usage testing techniques. It is the only technique that has shown to be able to certify the reliability requirement in the same time as it is cost effective. The application of the testing technique facilitates the formulation of a stopping criterion for software testing, i.e. the testing can terminate as the required reliability level has been reached.

The time has come to change the way of testing software. The objective must not be to find faults in general, but to show that the reliability requirements have been met. Usage testing aims at finding the faults influencing the reliability the most, instead of just removing arbitrary faults. The technique is mature enough to be used and those managing the transition first will probably be the ones delivering the products with the right reliability, which not necessarily is the highest.

Acknowledgement

We would like to thank the anonymous referees for providing valuable comments which have improved the article.

References

[Abramson92] Abramson, S. R., Jensen, B. D., Juhlin, B. D. and Spudic, C. L., "International DEFINITY Quality Program", Proceedings International Switching Symposium, Yokohama, Japan, 1992.

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

[DeMarco82] DeMarco, T., "Controlling Software Projects", Yourdon Press, New York, USA, 1982.

[Cobb90] Cobb, R. H., and Mills, H. D., "Engineering Software Under Statistical Quality Control", IEEE Software, pp. 44-54, November 1990.

[Cosmo93] Cosmo, H., Johansson, E., Runeson, P. Sixtensson, A. and Wohlin, C. "Cleanroom Software Engineering in Telecommunication Applications", Proceedings 6th International Conference on Software Engineering and its Applications, Paris, France, pp. 369-378, November 1993.

[Dyer92] Dyer, M., "The Cleanroom Approach to Quality Software Development", John Wiley & Sons, 1992.

[Goel79] Goel, A. L., and Okumoto, K., "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures, IEEE Transactions on Reliability, Vol. 28, No. 3, pp. 206-211, 1979.

[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.

[Jelinski72] Jelinski, Z., and Moranda, P., "Software Reliability Research", Statistical Computer Performance Evaluation, pp.465-484, 1972.

[Juhlin92] Juhlin, B. D., "Implementing Operational Profiles to Measure System Reliability", Proceedings 3rd IEEE International Symposium on Software Reliability Engineering, Raleigh, North Carolina, USA, pp. 286-295, 1992.

[Linger94] Linger, R. C., "Cleanroom Process Model", IEEE Software, pp. 50–58, March 1994.

[Mills87] Mills, H. D., Dyer, M. and Linger, R. C., "Cleanroom Software Engineering", IEEE Software, pp. 19-24, September 1987.

[Mills88] Mills, H. D., and Poore, J. H., "Bringing Software Under Statistical Quality Control", Quality Progress, pp. 52-55, November 1988.

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

[Musa92] Musa, J. D., “Software Reliability Engineering: Determining the Operational Profile”, Technical Report AT & T Bell Laboratories, Murray Hill, NJ 07974, New Jersey, USA, 1992.

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

[Myers79] Myers, G. J., “The Art of Software Testing”, Wiley Interscience 1979.

[NASA90] “The Cleanroom Case Study in the Software Engineering Laboratory – SEL 90-002”, Software Engineering Laboratory, 1990.

[Poore93] Poore, J. H., Mills, H. D., and Mutchler, D., “Planning and Certifying Software System Reliability”, IEEE Software, pp. 88-99, January 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”, Submitted to 7th SDL Forum, Oslo, Norway, 25-29 September 1995.

[Selby87] Selby, R. W., Basili, V. R., and Baker, F. T., “Cleanroom Software Development: An Empirical Evaluation”, IEEE Transactions on Software Engineering, Vol. 13, No. 9, September 1987.

[Tann93] Tann, L-G, “OS-32 and Cleanroom”, Proceedings 1st European Industrial Symposium on Cleanroom Software Engineering, Copenhagen, Denmark, October 1993.

[Whittaker93] Whittaker, J. A., and Poore, J. H.,“Markov Analysis of Software Specifications”, ACM Transactions on Software Eng. Methodology, Vol. 2, pp. 93–106, January 1993.

[Wohlin94a] Wohlin, C., “Managing Software Quality through Incremental Development and Certification”, In Building Quality into Software, pp. 187-202, edited by: M. Ross, C. A. Brebbia, G. Staples and J. Stapleton, Computational Mechanics Publications, Southampton, United Kingdom, 1994.

[Wohlin94b] Wohlin, C., and Runeson, P., “Certification of Software Components”, IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 494-499, 1994.