C. Wohlin, P. Runeson and A. Wesslén, "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.

Software Reliability Estimations through Usage Analysis of Specifications and Designs[1]

Claes Wohlin[*], Per Runeson[**] and Anders Wesslén[*]

[*]Department of Communication Systems, Lund University
Box 118, S-221 00 LUND
SWEDEN
Phone: +46-46 222 33 29
Fax: +46-46 14 58 23
e-mail: (claesw, wesslen)@tts.lth.se

[**]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

Abstract

This paper presents a method proposal for estimation of software reliability before the implementation phase. The method is based upon that a formal specification technique is used and that it is possible to develop a tool performing dynamic analysis, i.e. locating semantic faults in the design. The analysis is performed with both applying a usage profile as input as well as doing a full analysis, i.e. locate all faults that the tool can find. The tool must provide failure data in terms of time since the last failure was detected. The mapping of the dynamic failures to the failures encountered during statistical usage testing and operation is discussed. The method can be applied either on the software specification or as a step in the development process by applying it on the software design. The proposed method allows for software reliability estimations that can be used both as a quality indicator, and for planning and controlling resources, development times etc. at an early stage in the development of software systems.

Keywords:

Software quality, software reliability, operational profile, usage profile, software metrics.

1.This work is supported by National Board for Industrial and Technical Development (NUTEK), Sweden, reference Dnr: 93-2850.

1. Introduction

The reliability problem in software systems of today is a well-known fact. No silver bullet will solve this problem, instead the solution will be the combination of several approaches. That is improvements throughout the whole life cycle. These improvements include for example specification and design, verification and validation, certification as well as maintenance. This is the approach taken in the Cleanroom methodology, [1, 2], which includes methods for specification and design, verification and validation, as well as certification. In particular, Cleanroom supports the idea and philosophy that it is possible to develop zero-defect software.

The objective of this paper is to present a method for usage analysis during dynamic analysis of a software design. The method is based on the same principles as statistical usage testing in Cleanroom. The goal with statistical usage testing is to certify the software reliability during testing procedures. This does, however, seem too late if the product has to be re-designed due to poor reliability. It is also clearly not particularly useful if the result in the certification process shall be used for planning and controlling quality, resources, development time and release time of the software. The information from the certification process is really needed much earlier to cope with the management of the risks involved in the development of software systems.

Thus new methods have to be found for performing early reliability estimations. Based on the experience from applying formal specification techniques and tools supporting these techniques [3], it was noted that it ought to be possible to make the estimations during analysis of the formal specification. The estimations are consequently made before the coding phase. This implies that the result from the estimations can be used to plan and control the forthcoming phases in the development as well as the quality of the software.

It is a well-known fact that most problems encountered in the operational phase are due to semantic faults, [4]. Some types of semantic faults can be detected during dynamic analysis. This observation, in combination with that tools are available for performing dynamic analysis of formal specifications, led to the conclusion that a method for performing reliability estimations from formal specifications of the software ought to be possible to formulate. This idea was first presented in [5], but it has since been further enlarged and improved, and the major objective here is to present these improvements as well as the initial ideas to provide a comprehensive presentation of the method.

The idea and possibility described in this paper is general. It does neither depend on a particular specification technique nor on a particular tool set. It does though depend on that a well-defined specification technique with appropriate tool support is used. It is, however, difficult to describe the idea in general terms all the time and in particular it is hard to show the opportunities with the approach. This means that a formal specification technique will be used to exemplify the usability of the method. SDL, [6, 7], will be used throughout the paper. The reasons for choosing SDL as a suitable design method are many, for example, it is standardised and tools are available. The motives are further discussed in [3].

The main idea of the proposed method is to use the usage profile as input to an analysis tool which detects certain types of probable dynamic failures. The tool can detect all failures of the types it is designed to locate, but it is not certain that these situations

occur during the actual operation of the software. The tool is not capable of knowing this. Thus the user of the tool must either correct the failure assuming it is a real failure, i.e. it may occur in operation, or the user should verify that the encountered failure situation will never occur. From the failure statistics of the analysis tool, it will be possible to make a first estimation of the software reliability when in operation. This will be described in more detail below.

The paper is structured as follows, in Sec. 2, usage modelling is discussed, and Sec. 3 presents the generation of usage cases. The actual usage analysis is briefly described in Sec. 4 and in Sec. 5 the estimation and prediction of software reliability is discussed. Finally, in Sec. 6 a procedure, for mapping the failures found during dynamic analysis to arbitrary failures, is presented. The five method sections are then illustrated through a minor example in Sec. 7, and finally some conclusions are presented and the method is summarized in Sec. 8.

2. Usage Modelling for Dynamic Analysis

2.1 Introduction

Usage modelling is an essential part in usage testing [8, 9]. The specification of the users and their usage of the system shall be used in the tool with the original SDL specification or design of the system, i.e. the functional description of the system, subsequently referred to as the original SDL system. Different methods have been proposed to model usage, for example a tree structure has been suggested in [10], Markov chains are discussed in [11] and a state hierarchy (SHY) model is proposed in [12]. The latter type of model is used here. This type of model is also discussed in the context of reuse in [13].

The objective here is to describe the usage with a SHY model which then easily is translated to a usage model in SDL, i.e. the state hierarchy is represented using SDL. From the latter model, it is possible to semi-automatically generate analysis sequences which are documented in message sequence charts [14], see Sec. 3. These form the input to controlling how the analysis is carried out by the tool and how the analysis is performed, see Sec. 4. The failure data obtained can then be used for estimation and prediction of software reliability, see Sec. 5 and Sec. 6.

2.2 Modelling

Usage modelling means deriving a usage specification, which can be viewed as consisting of two parts, namely usage model and usage profile. The model describes the usage from a structural point of view, while the profile contains the usage probabilities hence providing a statistical view of the usage.

A SHY usage model is formulated according to the procedure described in [9]. The SHY model is easy to translate into an SDL model, when being familiar with the two specification techniques and the translations can also easily be implemented in a tool prototype if adopting the technique. The SHY model can be viewed as a tree, with the top level as the root. Every node in the tree is represented in the SDL model with a

process. The leaves in the tree are the services and each service forms a process. The service process is designed according to the behaviour level in the SHY model.

Links are used to describe dependencies between users in the environment. The links in the SHY model are converted into signals in the SDL model. The links between users are modelled through the system's response, the signals from the system to the SDL model go from the root process down in the hierarchy to a leaf process. The stimulus generated by a transition in a leaf process go from the leaf process up in the hierarchy to the root process. The links between a user's services are expressed as signals between the leaf processes describing the services.

The usage model is complemented with a usage profile, which describes the anticipated usage of the system as it is put into operation. The SHY model assumes a division of the usage profile into two parts, namely individual profile and hierarchical profile.

The individual profile describes the usage for a single user, i.e. how a user behaves when using the available services. The hierarchical profile is one of the major advantages with the SHY model as it allows for dynamic probabilities. The profiles are further discussed in [9].

Usage modelling and generation of usage cases from an SDL model are discussed in more detail in [15].

3. Generation of Usage Cases

3.1 Introduction

Usage cases must be generated to verify the software system. To generate usage 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 usage cases. A usage case ends when the system reach a specific state or when the usage case is of certain length.

3.2 Generation of usage cases in any language

If the usage cases are needed in a specific language, it can be generated automatically. A case study generating the usage cases as Message Sequence Charts [14] has been conducted successfully [16]. A process can be added in the SDL model between the environment and the root process, see Fig. 1, 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.

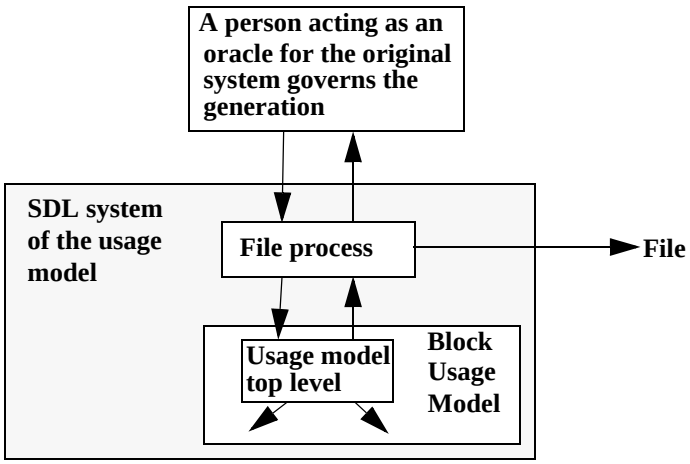

FIGURE 1. Generation of usage cases.

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 usage 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 usage case or in the system.

4. Usage Analysis

The usage cases that have been generated can be put together with the developed software system, hence providing a basis for dynamic analysis, 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 Fig. 2.

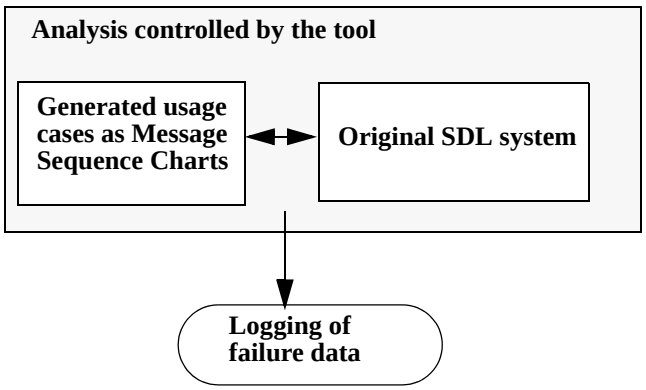

FIGURE 2. Usage analysis based on the generated usage cases.

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 discussed below. In particular, the certification procedure for the dynamic analysis is discussed. The analysis of failures from testing is more straightforward and discussed elsewhere, see for example [17, 18].

5. Reliability Estimation and Prediction

The first estimation of the reliability can be made in two different ways:

- by using the times between failures and relevant models.
- by counting the number of successfully executed usage cases compared to the total number of usage cases.

The first approach means that the analysis is made as one analysis sequence, while the second one requires that the specification of the environment's behaviour is divided into several usage cases. During analysis with the tool, the number of states analysed between two consecutive failures is reported. Thus there is a simple support for the first approach. If the failures are corrected we will observe a reliability growth which ought to work as an estimate of the reliability growth that will be obtained during testing and operation. An early estimate of this growth means that the test time to achieve the quality goals can be better planned. In case of no correction of failures an estimate of the actual reliability is obtained. The latter case is only possible if the execution of the analysis tool can continue without fault correction.

The approach described in Sec. 2 can be used to evaluate the estimate of the software's reliability. The dynamic analysis with the usage profile can be combined with the full analysis (complete in terms of the tool). They can be combined as follows:

1. do the analysis based on the usage profile and obtain an estimate of the reliability growth,
2. do the full dynamic analysis,
3. compare the normalised failure times with the estimates of the reliability growth, see Fig. 3.

The normalisation has to be done to be able to compare the times from a full analysis with the ones that should have been obtained if the analysis had continued to follow the usage profile. The times are normalised by recording where in the usage specification the failure occurred. Then we calculate the mean time to when the failure ought to have

occurred if the analysis was made according to the usage profile. This time is considered to be the actual failure time.

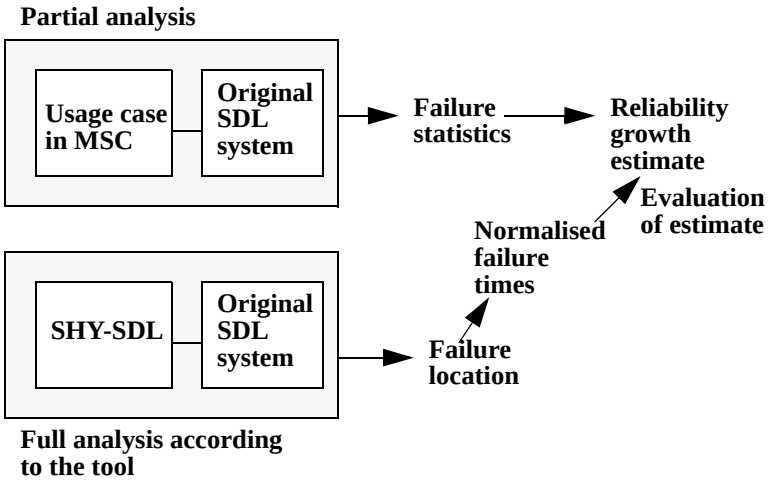

FIGURE 3. A procedure for evaluation of the software's reliability growth.

The normalisation procedure can be summarised in the following steps:

1. Perform a full dynamic analysis based on the opportunities with the tool.
2. The locations of the failures are recorded in the same time as the faults are corrected.
3. Based on the usage model, calculate the mean time when the located failures ought to have occurred.
4. Place the times on an ascending scale.
5. The obtained times are considered to be the real failure times. They are then compared with the prior estimated curve from the partial analysis, i.e. the one based on the usage profile.
6. The goodness of the estimate is judged in comparison with the backwards calculated expected mean times to failures.

These new times can be used to evaluate the estimate of the reliability growth from the partial dynamic analysis, see Fig. 3. This evaluation can be used to estimate the probable behaviour of the reliability and its growth during testing and operation. It is, however, necessary to relate the time axis during dynamic analysis to the real time experienced during testing as well as operation. This will be further discussed in the next section.

6. Relationship between Failures Types

6.1 Introduction

One problem encountered is the relevance of the dynamic failures found by the analysis tool compared to failures in operation. The question that has to be answered is: Are the dynamic failures detected by the analysis tool representative of the failures found in operation?

6.2 Assumptions

The reasoning above is based on five assumptions, of which two concern the failures:

1. The set of failures found in dynamic analysis by the tool is a subset of all possible failures, see Fig. 4.

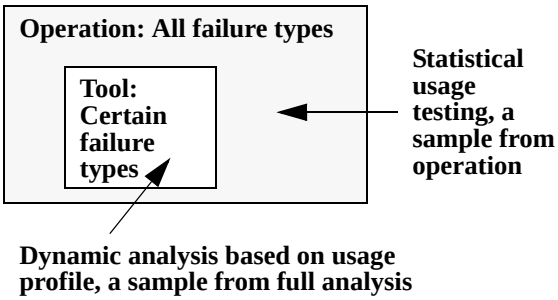

FIGURE 4. Dynamic failures found with the tool compared to all failure types.

2. The failures found during dynamic analysis are randomly spread among all failures, i.e. the ratio between the number of arbitrary failures and the number of dynamic failures found by the tool, during a certain time, is a scaling factor here denoted c. The actual time also has to be scaled, since the time between failures in the tool is logged in terms of number of states, while the time during testing and operation is real time.

Three assumptions concern the activities in the life cycle:

3. Testing according to a usage profile is a good approximation of the operation, see A in Fig. 5, i.e. Statistical Usage Testing (SUT) is a sample of the operation, see Fig. 4. This assumption is a central basis for SUT and a basis for most reliability prediction models as well.
4. The analysis, with the tool based on the usage profile, is a good picture of full analysis with the tool, see B in Fig. 5. Moreover the analysis with the tool based on the usage profile is a sample from full usage of the tool, see Fig. 4. A full dynamic analysis runs through all the states. When using the usage profile for a selective dynamic analysis, the selection of states to enter is made from the possible set of all states. The selection is not a random sample but a sample according to a specific usage profile.
5. The dynamic analysis with the tool using the usage profile is comparable with SUT, see C in Fig. 5.þThe usage cases selected for the dynamic analysis are chosen from the same usage profile model as the test cases for SUT are selected. The differences between the selections are only due to random variation.

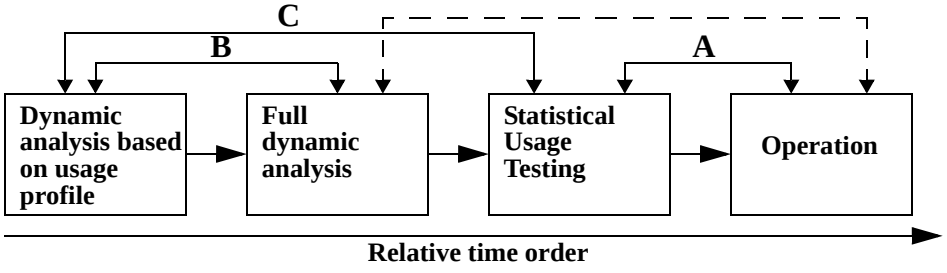

FIGURE 5. Relationships between different activities in the life cycle.

The time axis in Fig. 5 shows the relative order of the activities, it does not say that the activities do not overlap or that there is no other activities between the ones in the figure.

The dashed line in Fig. 5 indicates the possibility to evaluate the prediction of the operational behaviour. If the three separate relations (A, B and C) are accepted, then the dashed line must be accepted. This implies that by combining the different lines, it is possible to obtain an early estimate of the reliability in the operational phase. Based on a mapping algorithm, see below and in Fig. 6, results from the full analysis can be used to show some aspects of the operational behaviour. This behaviour can be compared and used to evaluate the prediction.

The relationships, indicated in Fig. 5, lead to the conclusion that it ought to be possible to use analysis with the tool (partial and full in combination) to obtain a first picture of the statistical usage testing and the operation. In particular, an earlier and better picture of the operation can be obtained than by using only statistical usage testing. Some relationships and possibilities of how to use the dynamic analysis to predict future failure behaviour and calculate the reliability will be discussed in the next section.

6.3 Derivation of failure times

To make the reliability growth, estimated from dynamic analysis, applicable on the reliability growth with respect to arbitrary failures, there must be a mapping of the dynamic analysis failure data to represent all failures. It can be performed by the following algorithm, where the steps are related to Fig. 6:

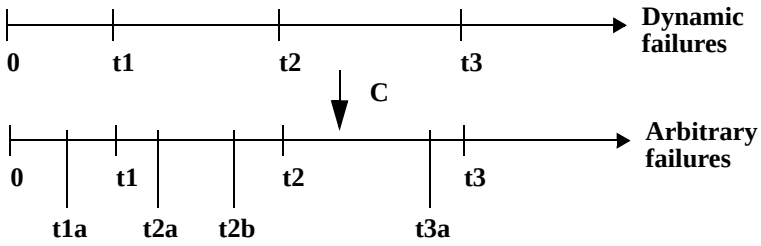

FIGURE 6. Failure data for arbitrary failures derived from dynamic analysis.

1. Make dynamic analysis according to the operational profile. In Fig. 6, t1 to t3 are the failure times. The failure data can be used to estimate MTBF (Mean Time Between Failures) for dynamic failures according to the tool by, for example, the Cleanroom reliability estimation model [19].
2. Determine c, i.e. the ratio between the total number of failures and the number of dynamic failures found by the tool. The c value must be based on metrics from earlier projects. The value can differ within programs with heterogeneous characteristics. These parts have to be analysed separately.
3. Determine the number of failures to occur in every interval. If c is not an integer, the number of failures in an interval is selected from a two-point distribution with the possible values trunc(c–1) and trunc(c), and a mean value c–1. If c is an integer, c–1 failures occur in each interval.

4. Re-scale the time axis to transform number of states to real time. Failure times within the interval are then selected. The times are denoted t1a, t2a, t2b etc. in Fig. 6. These failure times are chosen randomly within the interval. Further work has to be done to find a more realistic way to resemble the failure behaviour. The time scaling factor has to be based on experience in a similar way as c.
5. Estimate MTBF for the analysed and the calculated failure data, t1a, t1, t2a, t2b, t2 etc. shown in Fig. 6. This is now an estimation of the MTBF for all failure types.

The actual value of the analysis and its potential ought to be further investigated both theoretically as well as through practical application. It can however be concluded that as a first estimate it is relevant to consider the dynamic failures found during dynamic analysis of a software specification or design.

This estimation ought to be possible to use for planning and controlling the statistical usage testing phase, as well as necessary actions to take to achieve the quality required in operation.

7. An Example

Unfortunately, it is not possible to provide an extensive example of the method including a complete usage model of different services. Therefore, only a minor illustration is given through an example, where the objective is to illustrate the application of the different steps in the method, that is usage modelling, generation of usage cases, usage analysis, analysis of the failure data including estimation and prediction of software reliability. The example is based on a very simple model of a telephone exchange, which only has one user type and up to N users. All of the users can make ordinary telephone calls and they can also order and cancel call forwarding (CF).

Usage modelling

The first step is to model the usage and this must be a completely external view of the system. The model is first created using the state hierarchy (SHY) model as it provides an extremely simple notation. This model is then easily translated to a model in SDL, where unnecessary levels in the hierarchy is removed, due to that the system only consists of one user type. The SDL hierarchy is shown in Fig. 7. The SDL specifications of the usage of the services are not shown in the figure, but they are easily derived from

the SHY model. The descriptions are not included as they lead to far into the application domain of telecommunications, which is outside the scope of the paper.

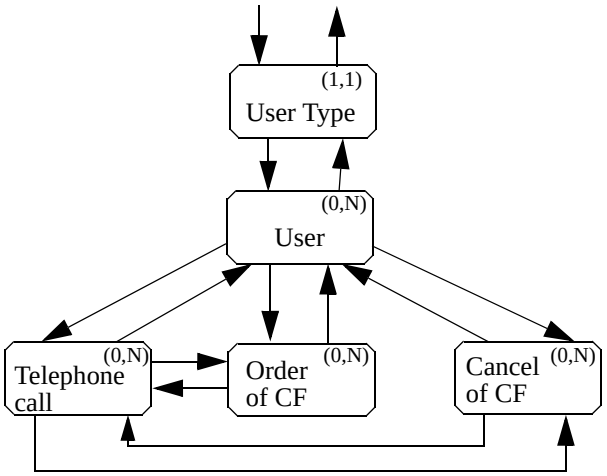

FIGURE 7. SDL hierarchy of the example.

Generation of usage cases

Usage cases are generated by executing the usage model in Fig. 7. The person performing the analysis must execute the model and act as the software system to be analysed. The different inputs, given by the person, and the outputs from the usage model are logged on a file in Message Sequence Charts format. The resulting file can then be used when performing usage analysis. The important aspect in this step in the method is that the person acts as the system, while the usage model is executed.

Usage analysis

The generated Message Sequence Charts can now be put together with the software system being developed to perform the analysis and then also obtain relevant failure data to perform the reliability estimation and prediction. The output data is received in terms of number of states since the last failure encountered. The tool locates certain types of dynamic failures as pointed out above. An example of failure data is shown in Tab. 1.

TABLE 1. Failure data.

Failure number	1	2	3	4	5	6	7	8	9	10
Time between failures	320	241	847	732	138	475	851	923	1664	1160

Reliability estimation and prediction of dynamic failures

The failure data from Tab. 1 are now input to the estimation and prediction procedure described in Sec. 5. First the failure data are fed into a software reliability growth model to get an estimation of the current reliability as well as a prediction of the future reliability growth. The model used is presented in [19] and it is based on the following formula:

$$MTBF(k) = A \times B^{(k-1)}$$

where $A = MTBF(1)$ and k is the failure number. The parameters A and B can be estimated from the failure data by taking the logarithm on the equation, and hence obtaining a linear equation with parameters A and B. Thus, A and B can be estimated from the failure data using the least square method. The estimation of the parameters is discussed in detail in [19].

Based on the available data, A becomes 278.7 and B is equal to 1.179. The outcome of the model is presented in Tab. 2, where both the estimated current level and a prediction of the five next failure occurrences are shown. The reliability growth model can, however, be used to predict an infinite number of forthcoming failure occurrences.

TABLE 2. Predictions of future failure occurrences.

Failure number	Current level	11	12	13	14	15
Time between failures	1270	1446	1705	2010	2370	2795

The software system being developed is now analysed fully by the tool, hence locating all faults that the tool is capable of finding. The faults are transformed into normalized failure times according to the procedure described in Sec. 5. That is, the dynamic analysis is done with the tool without controlling the analysis with the usage model. This is actually the normal usage of the tool. The locations of the faults in the software are noted as the faults are corrected. After having corrected the faults, the locations of the faults are compared with the usage model. It is determined where in the usage model the faults would have occurred, if we would have continued to analyse the software based on the usage model. Thus, the states and inputs in the usage model that would have caused the failures are identified. Since the usage model complemented with the profile is a state machine with probabilities governing the transitions, we are able to calculate the mean time until a certain state and a certain transition. The normalized times are the mean times, hence it is possible to determine the expected time to failure. The results of the procedure, i.e. the normalized times, are shown in Tab. 3.

TABLE 3. Normalized failure times.

Failure number	11	12	13	14	15
Time between failures	1534	1712	1944	2290	2864

The normalized failure data are now compared with the predictions presented in Tab. 2. The objective is to be able to perform an evaluation of how good the prediction from the failure data is. This can then form the basis for determining how reliable the predictions obtained from usage testing are in comparison from what can be expected when entering the operational phase.

It can be seen, by comparing the relative mean errors between Tab. 2 and Tab. 3, that the predictions from the dynamic usage analysis are good predictors of the actual outcome when doing a full analysis. Therefore, it is expected that we will also obtain a good prediction of the operational phase based on failure data from usage testing. The failure data obtained during the dynamic analysis can also be used to plan the test phase in terms of resources, that is number of test sites and test personnel. The failure data from dynamic analysis must, however, be related to failures that can occur during testing and the operational phase.

Relating the dynamic failures to arbitrary failures

The above procedure gives a view of how to estimate the time between failures during dynamic analysis, but this is not enough for the testing and operational phases. The failures from dynamic analysis must be mapped onto arbitrary failures, and the time scale has to be changed to take real time into consideration and not just counting the time between failures in terms of number of states. Therefore, two different measures are needed, first the mapping factor c, see Sec. 6 and Fig. 6, and secondly a factor describing how time between failures in terms of states are mapped into real time. The c factor is assumed to be known based on experience and in this particular case it is assigned the value 3.1, which means that for each dynamic failure found during dynamic analysis, it is expected that 3.1 other failures are found on average during testing and operation. The scaling factor between times is assumed to be 3, that is the times resulting from applying the c factor, see Fig. 6, must be multiplied by 3 to obtain the real time between failures.

Applying the procedure discussed in the Sec. 6.3 and then using the values discussed here together with random numbers lead to the results presented in Tab. 4. More specifically, the results in Tab. 4 are obtained as follows. First, random numbers are generated in order to put in some more failures among the failure data from the dynamic analysis controlled by the usage model, see Tab. 1. The failures should be put in to capture failure types that the tool can not find. It is assumed that we know from experience that 1 dynamic fault means 3.1 failures later on in the software life cycle. Thus, in average 2.1 failures should be put in between the failures logged in Tab. 1.

Second, random numbers are generated to give the failure times where the failures are assumed to happen. This means, for example, that in average 2.1 failures are put in prior to the first failure in Tab. 1. This changes the times between failures, hence the time between failures in Tab. 1 are not relevant at this stage. After the failures have been put into the failure history, new times between failures can be calculated. These new times between failures are then scaled with the time factor, i.e. in this particular case the figures are multiplied with 3. This results in the failure times presented in Tab. 4, which then are input to the software reliability model used above. The failure data, in Tab. 4, is assumed to be one possible representation of failures during usage testing and operation. It must, however, be noted that this particular outcome is of little interest, but it is interesting to use the data to estimate the mean expected behaviour, i.e. to apply a software reliability growth model.

TABLE 4. Prediction of failure times during usage testing and the operational phase (the table should be read row by row and from left to right).

597	87	276	246	147	330	3	1284	1254	1674	324	198
204	180	30	63	639	723	171	1980	402	2328	3	438
768	3381	843	780	453	1260	487	-	-	-	-	-

The parameters in the model are estimated from the available data and we get:

$$MTBF(k) = 161{,}9 \times 1{,}047^{(k-1)}$$

This formula can work as a predictor of the expected failures to find during usage testing and then also in the operational phase. The limit between testing and the opera-

tional phase is determined from the available test time and in particular of the reliability requirement. If it is assumed that the reliability requirement is 2000 time units, then it is just to keep track of the total test time until the formula estimates that MTBF is equal or greater than 2000 time units. This may be of great help when planning the work and taking further development decisions.

It can based on this example be concluded that, the procedure provides a valuable input to the planning of the testing phase and it also indicates the current reliability of the software and through the prediction also the expected future reliability growth. The current level is equal to MTBF(1), which is the estimation when we enter the testing phase. The early prediction means that, a poor software product can be detected at an early stage which allows for re-design instead of letting a poor design be handed over to the implementation phase.

8. Conclusions

It can be concluded that the statistical quality control of software products is an important issue. The certification process is central in this effort, in particular the earlier it is applied. This process is highly dependent on relevant software reliability models and a sound basis for estimation. The basis includes relevant failure data, i.e. data that is obtained under circumstances fulfilling the assumptions of the reliability models. In particular, this means that the failure data during testing and other type of analysis (for example dynamic analysis) have to be similar to the failure data encountered during operation.

A reliability estimation from dynamic analysis can be used either to estimate the reliability of a software specification in SDL or as a first estimate of the design's reliability. A specifier and purchaser of software systems can use both of these approaches, i.e. estimating the reliability of its own specifications and requiring that a first estimate of the reliability must be made during dynamic analysis of the design. The latter estimation can be made either by the supplier or as a part of a programme for quality control of suppliers made by the purchaser/customer.

Based on the presentation, a method for software certification can be formulated:

1. A usage model and profile must be formulated, first as a SHY model which then is easily translated to an SDL model, see Sec. 2.
2. Usage cases are then generated based on the usage model in SDL and a person acting as the system to be analysed, see Sec. 3.
3. The usage analysis can then be performed and the failure data logged, see Sec. 4.
4. The collected failure data can be used to estimate and predict software reliability at the early stage of dynamic analysis of the software design or even the software specification, see Sec. 5.
5. Finally, the data from dynamic analysis must be transformed to get an estimation of the behaviour for arbitrary failures during usage testing and operation, see Sec. 6.

The failure from the dynamic analysis and the prediction must be continuously monitored and compared with the actual outcome during software testing and then ultimately in the operational phase.

This new method 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 state hierarchy (SHY) model, which then is transformed into an SDL usage model (SHY-SDL) has been used to generate usage test cases to the next release of the tool used here [16]. It is concluded from this application, that both the SHY model and the transformation into SDL are useful concepts when generating usage cases.

The objective is that the proposed method shall work as a complement to Statistical Usage Testing. In particular, the new method provides a basis for planning and controlling the forthcoming testing phase, the release of the product and finally the operational phase. Thus the method is an important step in the risk management process, since it gives early estimates and consequently early warnings, which leads to that the risks can be managed and planned for.

References

1. H. D. Mills, M. Dyer and R. C. Linger, “Cleanroom Software Engineering”, *IEEE Software*, **September**, 19 (1987).

2. R. C. Linger, “Cleanroom Process Model”, *IEEE Software*, **March**, 50 (1994).

3. C. Wohlin, “Software Reliability and Performance Modelling for Telecommunication Systems”, Dept. of Communication Systems, Lund, Sweden, ISSN 1101-3931, Technical report - 106, 1991, Ph.D. Dissertation (unpublished).

4. J. D. Musa and W. W. Everett, “Software-Reliability Engineering: Technology for the 1990s”, *IEEE Software*, **November** 36 (1990).

5. C. Wohlin and P. Runeson, “A Method Proposal for Early Software Reliability Estimations”, in *Proceedings of the 3rd International Symposium on Software Reliability Engineering* (1992), pp. 156-163.

6. ITU-T, *Recommendation Z.100: Specification and Description Language, SDL*, Blue book, Volume X.1, (1988).

7. F. Belina, D. Hogrefe and A. Sarma, *SDL with Applications from Protocol Specifications* (Prentice-Hall, UK, 1991).

8. H. D. Mills and J. H. Poore, “Bringing Software Under Statistical Quality Control”, *Quality Progress*, **November** 52 (1988).

9. P. Runeson and C. Wohlin, “Statistical Usage Testing for Software Reliability Control”, *Informatica*, **19** 2 (1995).

10. J. D. Musa, “Operational Profiles in Software Reliability Engineering”, *IEEE Software*, **March** 14 (1993).

11. J. A. Whittaker and J. H. Poore, “Markov Analysis of Software Specifications”, *ACM Trans. on Software Eng. Methodology*, **1** 93 (1993).

12. P. Runeson and C. Wohlin, “Usage Modelling: The Basis for Statistical Quality Control”, in *Proceedings Software Reliability Symposium* (1992) pp. 77-84.

13. C. Wohlin and P. Runeson, “Certification of Software Components”, *IEEE Trans. on Software Eng.*, **6** 494 (1994).

14. ITU-T, *Recommendation Z.120: Message Sequence Charts* (1993).

15. A. Wesslén and C. Wohlin, “Modelling and Generation of Software Usage”, in *Proceedings Fifth International Conference on Software Quality* (1995) pp. 147-159.

16. P. Runeson, A. Wesslén, J. Brantestam, and S. Sjöstedt, “Statistical Usage Testing using SDL”, in *SDL´95 with MSC in CASE* (Elsevier Science B. V., 1995) pp. 323-336.

17. J. D. Musa, A. Iannino and K. Okumoto, *Software Reliability: Measurement, Prediction, Application* (McGraw-Hill, New York, 1987).

18. M. Xie, *Software Reliability Modelling*, (World Scientific Publishing Co. Pte. Ltd., 1991).

19. P. A. Currit, M. Dyer and H. D. Mills, “Certifying the Reliability of Software”, *IEEE Trans. on Software Eng.*, **1** 3 (1986).