C. Wohlin and P. Runeson, "A Method Proposal for Early Software Reliability Estimations", Proceedings 3rd International Symposium on Software Reliability Engineering, pp. 156-163, Raleigh, North Carolina, USA, 1992.

A Method Proposal for Early Software Reliability Estimation

Claes Wohlin and Per Runeson

E-P Telecom Q-Labs, IDEON Research Park, S-223 70 LUND,
SWEDEN, Phone: +46-46-182980, E-mail: cw@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 description 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 design descriptions. The proposed method will allow for software reliability estimations that can be used both as a quality indicator, but also for planning and controlling resources, development times etc. at an early stage in the development of software systems.

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, 3], 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 an idea of how software reliability can be estimated already before the coding phase. This is thought to be one step in mastering the reliability problems of software encountered today. The problems of estimating the reliability before testing is also indicated as a challenging research opportunity in [4]. The basis for this work is the ideas from Statistical Usage Testing within Cleanroom, a formal description technique and a suitable tool for analysing the descriptions of the system being developed. The presented work is part of a project being conducted for the Swedish Telecom. The objective of the project is to provide the Swedish Telecom with methods for certification of software reliability. In particular, in the role as a purchaser of software systems.

The method proposed shall be used to estimate the reliability of the software during dynamic analysis of the software described with a formal description technique. This will be exemplified with SDL (Specification and Description Language [5]) and the tool environment SDT (SDL Design Tool [6]), in particular the dynamic analysis tool within SDT, i.e. SBA (SDL Behaviour Analyser, [7]). It must be noted that the method is general even if some specific techniques are used to exemplify the method. It should also be observed that it is possible to generate the code completely from formal description techniques, for example tools exist for generating C from SDL.

The method can be applied either on (customer) specifications or during the design as a step in the development process. This provides an opportunity to estimate both the reliability of the specification on which the implementation is based as well as on the actual design of the implementation. This double usage of the proposed method must be noted. The presentation below will, however, concentrate on how the method can be used during analysis of the design as an early estimation compared with testing or operation, even if the analysis of specifications are equally important.

The paper will first give an introduction to the certification process as proposed in Cleanroom. The general idea will then be discussed before introducing SDL and the analysis tool. Use case modelling with SDL will be discussed before presenting how this modelling can be used in the tool environment. The use case modelling is then combined with the functional description of the system. The relevance of the estimations from the dynamic analysis in comparison with the failures that occur during operation is then investigated. Finally, some conclusions from the work is presented.

2. Cleanroom certification of reliability

The certification process is an important issue, since one use of it is as an interface between the developer and the purchaser, in many cases the manager of the software. This application of the process is the foundation for acceptance of a software product and a key issue in the quality control of software products. The objective is to certify during testing that the reliability requirements during operation are fulfilled. The basis for this is that the testing procedure resembles or models the operational profile. In Cleanroom this type of testing is referred to as Statistical Usage Testing, [8, 9].

The problem of certification involves two parts, both discussed in [10]:

- estimation of the reliability, i.e. software reliability models.
- modelling operation during testing, i.e. a usage model and the corresponding statistical usage profile

Numerous software reliability models can be found in the literature, some examples are presented in [11, 12, 13]. The model proposed within the Cleanroom concept is presented in [8]. Most of these models are based on the assumption of operational usage, but not much emphasis has been put into actually modelling and performing tests that fulfil this assumption.

The certification of software within the Cleanroom methodology is discussed in [8]. As stated above, the certification process consists of two equally important parts. The first part is the software reliability models. These shall model the behaviour of software failures and in particular predict the future behaviour and the reliability of the software. The models are based on several assumptions, where one of the most critical ones is the assumption that failures occur according to the operational usage. Thus the models can only be applied during operation or during testing where it is possible to generate test cases from an operational profile. The latter is the basis for Statistical Usage Testing.

The second item, i.e. modelling the usage, has been much less studied than the software reliability models. One approach is to model the behaviour and generate the test cases based on a plain Markov chain. This approach is discussed in [14]. The plain Markov approach is useful in many cases, but for some applications it is not suited. The problem of modelling the usage for complex multi-user systems has been overlooked in the past, but if it shall be possible to certify the reliability under testing conditions this problem has to be solved, [15]. This has been one of the key issues in the project conducted for the Swedish Telecom. The reason is, of course, that it has been shown that usage testing is superior to other types of testing in finding the faults that influence the reliability during operation, [16].

3. A method for estimation of reliability

The objective with this paper is to present a method from the on-going project whose aim is to apply the ideas presented in Statistical Usage Testing to telecommunication software. 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 [17], it was noted that it ought to be possible to make the estimations during analysis of the formal description. 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 doing dynamic analysis of formal descriptions, led to the conclusion that a method for doing reliability estimations from formal descriptions of the software ought to be possible to formulate.

The idea and possibility described in this paper is general. It does not depend on a particular description technique neither on a particular tool set. It does though depend on that a well-defined description 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 description technique will be used to exemplify the usability of the method. SDL, [5, 18], will be used throughout the paper. The reasons for choosing SDL as a suitable design method to be used are many, e.g. it is standardised and tools are available. The motives are further discussed in [17].

A brief introduction to SDL will be given below, as well as a brief description of the tool set, which provide an opportunity of doing dynamic analysis. The meaning of dynamic analysis will also be described briefly in connection with the presentation of the analysis tool.

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.

4. Brief introduction to SDL

The CCITT Specification and Description Language, [5], known as SDL, was first defined in 1976. It has been extended and reorganised in four study periods since this first definition. These have resulted in new recommendations for the language published in 1980, 1984 and 1988 respectively. A new recommendation will appear in 1992.

SDL is intended to be well-suited for all systems whose behaviour can be effectively modelled by extended finite-state-machines and where the focus is to be placed especially on interaction aspects. SDL is a unique language which has two different forms, both based on the same semantic model. One is called SDL/GR (graphical representation) and is based on a set of standardized graphical symbols. The other is called SDL/PR (phrase representation) and is based on program-like statements. SDL is further described in [5, 18].

The main concepts in SDL are system, blocks, channels, processes and signals. These concepts form the basis for SDL, where system, blocks and channels describes the static structure while the dynamic behaviour is modelled with the processes and its signals. The processes are described by several symbols.

System: Each system is composed of a number of blocks connected by channels. Each block in the system is independent from every other block. Each block may contain one or more processes which describe the behaviour of the block. The only means of communication between processes in two different blocks is by sending signals that are transported by channels. The criteria leading to a certain division of the system into blocks may be to define parts of a manageable size, to create a correspondence with actual software/hardware division, to follow natural functional subdivisions, to minimize interactions, and others.

Block: Within a block, processes can communicate with one another either by signals or shared values. Thus the block provides not only a convenient mechanism for grouping processes, but also, a boundary for the visibility of data. For this reason, care should be taken when defining blocks to ensure that the grouping of processes within a block is a reasonable functional grouping. In most cases it is useful to break the system (or block) into functional units first and then define the processes that go into the block.

Channel: Channels are the communication medium between different blocks of the system or between blocks and the environment.

Signal: Signals can be defined at system level, block level, or in the internal part of process definition. Signals defined at system level represent signals interchanged with the environment and between system blocks. Signals defined at block level represent signals interchanged between processes of the same block. Signals defined within a process definition can be interchanged between instances of the same process type or between services in the process. Signals are sent along signal routes between processes and on channels between blocks or when interchanged with the environment.

Process: A process is an extended finite-state-machine which defines the dynamic behaviour of a system. The extended finite-state-machine handles data within tasks and decisions. Processes are basically in a state awaiting signals. When a signal is received, the process responds by performing the specific actions that are specified for each type of signal that the process can receive. Processes contain many different states to allow the process to perform different actions when a signal is received. These states provide the memory of the actions that have occurred previously. After all the actions associated with the receipt of a particular signal have occurred, the next state is entered and the process waits for another signal.

Processes can either be created at the time the system is created or they can be created as a result of a create request from another process. In addition, processes can live forever or they can stop by performing a stop action. A process definition represents the specification of a type of process; several instances of the same type may be created and exists at the same time; they can execute independently and concurrently.

5. Tool support for SDL

SDT has been used as a suitable tool for our purposes, [6]. SDT is a tool environment supplied by TeleLogic AB, Sweden. The environment includes tools for editing SDL graphs and an analyser for both syntax and semantic analysis. It also includes tools for simulation, both functional and performance simulations, as well as code generators, e.g. SDL to C. The tool also has a browser and a report generator.

The most interesting tool within the environment for this work is the prototype tool SBA (SDL Behaviour Analyser) [7]. The objective of SBA is to support the specifier to avoid unwanted dynamic properties in the specified behaviour. This is done by automatic detection of some fault types e.g. deadlock, more than one possible receiver of a signal and the existence of queues that can grow forever. Two types of properties are detected: faults and warnings. The faults are violations of the rules in SDL, while the warnings are a result of situations that have an effect on the dynamic behaviour of the specified system, but they are not violations of the rules of SDL. The detected faults and situations that give warnings are

described in more detail in [7].

The analysis is made by using a tree expansion procedure. The signals in the description determines how the analysis is made. It is possible to have priorities on different types of signals, i.e. internal, external and timers. It is also possible to determine if a certain fault type shall be reported or not. The analysis is made as a closed system, i.e. the user of the tool does not generate any input to the tool in terms of signals. The analysis is halted when a failure is encountered. The failure type, the place of the failure and the number of passed states since the last failure are reported.

6. Use case modelling with SDL

It has been shown in [15], that it is possible to reduce the state explosion problem in statistical usage testing descriptions of telecommunication systems by introducing a state hierarchy model. The study introduces a hierarchical Markov chain description technique to describe the users and their usage of the system. The advantages of using SDL have been discussed briefly above. These advantages lead to that the objective of the current work, with this new method for early estimation of software reliability, is to formulate rules for modelling the hierarchical descriptions of the usage with SDL instead (State Hierarchy with SDL, SHY-SDL).

An algorithm for test case selection based on the usage profile shall be applied to the usage model of the system. The result from the algorithm shall be a test sequence. This can be described in text, by SDL or any other suitable description technique. The examples presented in [19] give a verbal description of the test cases. It is felt that it is equally important to study the possibility to describe the test cases in SDL. The advantages referenced above can be complemented with:

- If the usage is described with SDL, then it is easy to generate the test cases automatically in SDL.
- By describing the test cases in SDL, it is possible to use SBA.

7. Analysis and use case modelling

The description of the users and the usage of the system shall be used in SBA with the original SDL description of the system, i.e. the functional description of the system. There are three major ways of using SBA with SHY-SDL:

1 The original SDL system becomes a block and the SHY-SDL model becomes another block. These two together form the new SDL system, see figure 1. This solution can not be implemented directly, since it will mean that the SBA tool will make a full dynamic analysis of the SDL system according to the limitations of the tool, which contradicts the objective of the statistical quality control procedure. An advantage with this approach would be that a consistency check between the description of the system and the actual environment is obtained. This approach can, however, be valuable as a complement to another approach, for further discussion see below.

2 The solution discussed in item 1 can be used if the SBA tool is adapted to this particular usage of the tool. It must execute according to a random walk procedure, where the particular paths to execute are chosen by the SHY-SDL model according to the usage profile, i.e. by

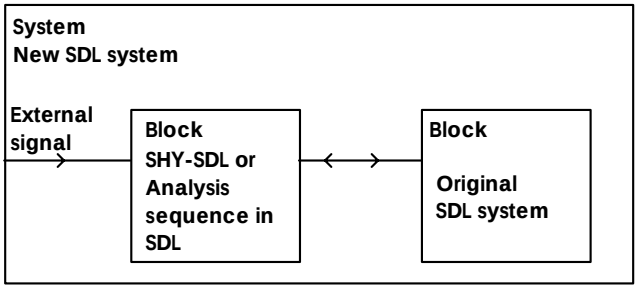

Figure 1: An SDL solution for the SBA

the probabilities modelling the usage of the system. The SBA tool has to be complemented in two aspects, i.e. the handling of decisions and a scheduler has to be implemented.

3 The last possibility is to generate the analysis (test) sequences from the SHY-SDL model and let the SDL description of the analysis cases become a block in the SDL system, in a similar manner as discussed for the SHY-SDL above, see figure 1. This solution means that only the chosen cases of the usage are analysed with the SBA. The main advantage with this solution is that the SBA of today can be used.

8. Reliability estimation from analysis

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

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

The first approach means that the analysis is made as one analysis sequence, while the second one requires that the description of the environment's behaviour is divided into several analysis cases. During analysis with the SBA, 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 will be

obtained. The latter case will only be possible if the execution of the analysis tool can continue without fault correction.

The main problem with the second analysis procedure is that it is difficult to perform due to the non-interactive work with the SBA. It is difficult to analyse one case at the time, if several are implemented in a block. A solution to the problem would be to only implement one analysis case at the time. This would, however, force the user to re-generate the code for each analysis case.

The approaches described in the previous section can be combined to obtain a second estimate of the software's reliability. The dynamic analysis with the usage profile can be combined with the full analysis (complete in terms of SBA). 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 figure 2.

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 description 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.

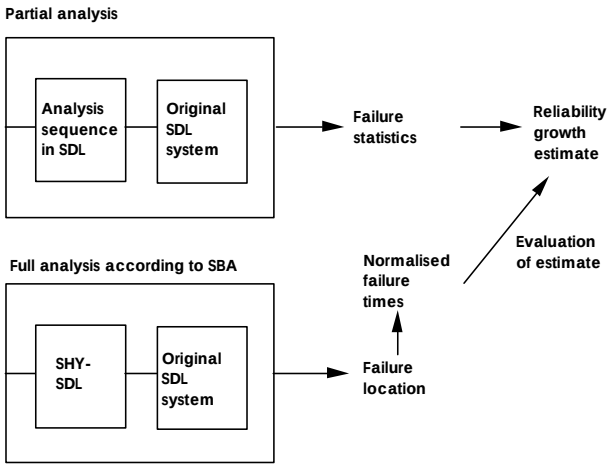

Figure 2: A procedure for evaluation of the software's reliability growth

The normalisation procedure can be summarised in the following steps:

1 Perform a full dynamic analysis based on the opportunities with SBA.
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 in order according to size.
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 figure 2. 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 below.

9. Usage profile

It shall be observed that it can be fruitful to change the usage profile, in particular during dynamic analysis since it is easy to make a second analysis. The need for different usage profiles is also stressed in [4]. The objective with changing the usage profile is to examine the reliability for another usage profile. This is valuable since it is probable that the usage of a system will change over time, which means that a highly reliable system can become less reliable due to changes in the usage.

An example of another usage profile is random testing, i.e. all events or signals into the system are equally probable. This would work as a test of the system's dependability in the future. It is also possible to formulate a usage profile that takes the critical parts into special account. This might be valuable if certain types of failures just not may occur.

Another possibility of getting a picture of the reliability of parts seldom executed during normal usage, is to use the full dynamic analysis to note special fault prone parts in the software. A list of these parts may give a picture of how changes in the usage profile, may alter the perceived reliability of the software system.

10. Relationship between failures types

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?

10.1 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 SBA

is a subset of all possible failures, see figure 3.

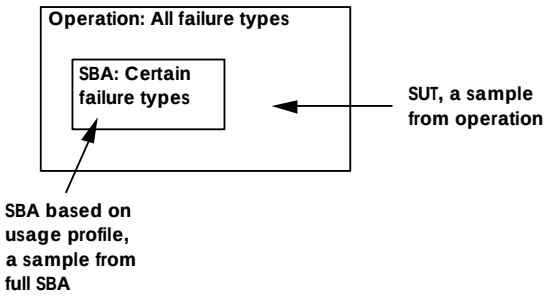

Figure 3: Dynamic failures found with SBA compared to all failure types

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 SBA, during a certain time, is a scaling factor here denoted c.

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 figure 4, i.e. Statistical Usage Testing (SUT) is a sample of the operation, see figure 3. This assumption is a central basis for SUT and a basis for most reliability estimation models as well.

4 The analysis with SBA based on the usage profile is a good picture of full analysis with SBA, see B in figure 4, i.e. analysis with SBA based on the usage profile is a sample from full SBA, see figure 3. A full dynamic analysis walks through all the states. When using the operational 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 SBA using the usage profile is comparable with SUT, see C in figure 4. The analysis 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.

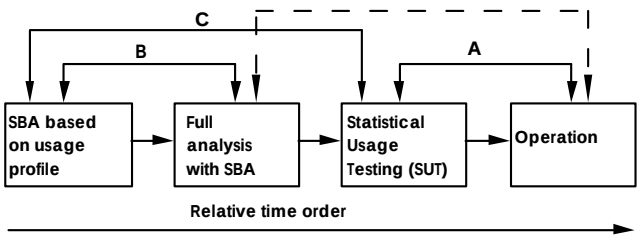

Figure 4: Relationships between different activities in the life cycle

The time axis in figure 4 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 figure 4 indicates the possibility to evaluate the prediction of the operational behaviour. Based on the mapping algorithm in section 8, 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 figure 4, lead to the conclusion that it ought to be possible to use analysis with SBA (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.

10.2 Derivation 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 which steps are related to figure 5:

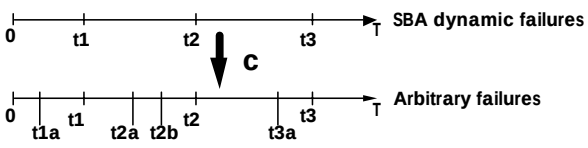

Figure 5: Failure data for arbitrary failures derived from dynamic analysis

1 Make dynamic analysis according to the operational profile. In figure 5, t1 to t3 are the failure times. The failure data can be used to estimate MTTF (Mean Time To Failure) for dynamic failures according to SBA by e.g. the Cleanroom reliability estimation model [8].

2 Determine c, i.e. the ratio between the total number of failures and the number of dynamic failures found by SBA. The c value is to 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 Select failure times within the interval to place the failures, denoted t1a, t2a, t2b etc. in figure 5. 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.

5 Estimate MTTF for the analysed and the calculated failure data, t1a, t1, t2a, t2b, t2 etc. shown in figure 5. This is now an estimation of the MTTF 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.

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.

11. 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 (e.g. dynamic analysis) has 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 implementation's reliability. The Swedish Telecom as 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 shall be made during dynamic analysis of the implementation. 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.

As a first preliminary recommendation it is suggested mainly based on the available tool that:

- the approach where the analysis cases are described in SDL ought to be used, i.e. not the whole SHY-SDL is used as a block in the dynamic analysis (see approach 3, section 7).
- the analysis shall be made as one sequence (see section 8).
- the times between failures shall be recorded and the faults shall be corrected, i.e. the reliability growth will be estimated (see section 8).
- finally it is recommended that a full dynamic analysis according to the tool is performed, which provides a check of the estimated reliability growth (see approach 1, section 7 and section 8).

This new method is not fully developed. It does need more work, but the idea in itself is very relevant and if the objective can be fulfilled and the method implemented it is believed to be an important step towards early estimations of software reliability. A practical study of the method is needed to evaluate the method, before it is put into actual use. The proposed method will work as a complement to Statistical Usage Testing. In particular, the new method will provide 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.

Acknowledgement

The project is being conducted for the Swedish Telecom, to whom we are grateful for specifying this project and in particular for letting us publish the results.

Many thanks to Erik Johansson and Bo Lennselius, E-P Telecom Q-Labs for interesting and fruitful discussions and comments throughout the project.

References

[1] Mills, Harlan D., Dyer, Michael and Linger, Richard C., "Cleanroom Software Engineering", IEEE Software, September 1987, pp. 19-24.

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

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

[4] Musa, John D., and Everett, William W., "Software-Reliability Engineering: Technology for the 1990s", IEEE Software, November 1990, pp. 36-43.

[5] CCITT, "Recommendation Z.100: Specification and Description Language, SDL", Blue book, Volume X.1, 1988.

[6] Nilsson, Gert, Ljungdahl, Ingemar and Madsen, Pär, "SDL Toolbox to Support Different SDL Environments", in SDL'89: The Language at Work, edited by O. Færgemand and M. M. Marques, Elsevier Science Publisher, North-Holland, 1989, pp. 87-93

[7] Ek, Anders and Ellsberger, Jan, "A Dynamic Analysis Tool for SDL", SDL '91: Evolving Methods, edited by R. Reed and O. Færgemand Elsevier Science Publisher B V (North Holland) 1991, pp. 119-134.

[8] Currit, P. Allen, Dyer, Michael and Mills, Harlan D., "Certifying the Reliability of Software", IEEE Transactions on Software Engineering, vol SE-12, no 1, January 1986, pp. 3-11.

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

[10] Musa, John D., Iannino, Anthony, and Okumoto, Kazuhira, ”Software Reliability: Measurement, Prediction, Application”, McGraw-Hill, New York, 1987.
[11] Goel, Amrit L., ”Software Reliability Models: The State of the Art”, IEEE Transactions on Software Engineering, Vol. SE-11, No. 12, 1985, pp. 1411-1423.
[12] Jelinski, Z., and Moranda, P., ”Software Reliability Research”, Statistical Computer Performance Evaluation, 1972, pp. 465-484.
[13] Goel, Amrit L., and Okumoto, Kazuhira, ”Time-dependent Error-detection Rate Model for Software Reliability and Other Performance Measures”, IEEE Transactions on Reliability, Vol. R-28, No. 3, 1979, pp. 206-211.
[14] Whittaker, James A., ”Markov Chain Techniques for Software Testing and Reliability Analysis”, Dept. of Computer Science, University of Tennessee, Knoxville, USA, 1992, Ph.D. Dissertation.
[15] Runeson, Per and Wohlin, Claes, ”Usage Modelling: The Basis for Statistical Quality Control”, Proceedings 'Software Reliability Symposium', Denver, USA, June 1992, pp. 77-84.
[16] Adams, E. N., ”Optimizing Preventive Service of Software Products”, IBM Journal of Research and Development, January 1984.
[17] Wohlin, Claes, "Software Reliability and Performance Modelling for Telecommunication Systems", Dept. of Communication Systems, Lund, Sweden, ISSN 1101-3931, Technical report - 106, 1991, Ph.D. Dissertation.
[18] Belina, Ference, Hogrefe, Dieter and Sarma, Amardeo, ”SDL with Applications from Protocol Specifications”, Prentice-Hall, UK, 1991.
[19] Runeson, Per, ”Statistical Usage Testing for Telecommunication Systems”, Dept. of Communication Systems, Lund, Sweden, Report no. CODEN:LUTEDX (TETS-5134)/1–49)/(1991)&Local 9, 1991, Master thesis.