C. Wohlin, "Engineering Reliable Software", Proceedings 4th International Symposium on Software Reliability Engineering, pp. 36-44, Denver, Colorado, USA, 1993.

Engineering Reliable Software

Claes Wohlin[1]

Dept. of Communication Systems, Lund Inst. of Tech., Lund University,
Box 118, S-221 00 LUND, Sweden, E-mail: claesw@tts.lth.se

Abstract

Software reliability engineering is not only the use of software reliability models and similar techniques, it is the use of sensible engineering principles with cost/benefit analysis throughout the software life cycle to obtain reliable software. The need to have a comprehensive view on software development to engineer reliable software will be emphasized. Cleanroom Software Engineering is proposed as being the basis for developing reliable software. The paper will in particular discuss some extensions to Cleanroom, both in terms of adaptations and additions. Particular emphasis will be on high-level design techniques and methods for reliability certification of the software. The comprehensive view of software is supported by several success stories, both with references to results presented in literature as well as experiences from projects conducted by Q-Labs. The results obtained are encouraging. The methods proposed are shown to give a substantial gain in the development of reliable software.

1. Introduction

Software Reliability Engineering must be a life cycle commitment. The ultimate challenge of reliability engineering must be to provide software which does not fail in the operational phase, while the objective must be to develop software with the required reliability within schedule and budget. This goal is high for some types of system, for example safety-critical software.

The objective of this paper is to argue that the way towards engineering reliable software is a combination of several techniques. The reliability problems in software will not be solved with one single technique. The best techniques available today must be put together to obtain a substantial improvement in reliability of software systems. In particular some techniques that have been successfully introduced during projects conducted by Q-Labs will be emphasized. Experience gained is supported by other studies presented in the literature.

The introduction of high-level specification and design techniques is one step in the right direction, along with a rigorous and comprehensive approach in development similar to the one presented in Cleanroom Software Engineering, thus emphasizing early verification and inspection as well as early certification and prediction of reliability. The focus in this paper will be on extensions and additions to Cleanroom, particular emphasis will be on certification through statistical usage testing, which is the fishing net supposed to stop poor products from being put into operation. The introduction of these techniques will be supported by a presentation of some success stories, both performed by Q-Labs and found in the literature.

As the maturity of software development has grown, it is necessary to develop fault tolerant techniques to cope with the faults introduced in the software despite high-level design techniques and a rigorous approach. The introduction of fault tolerance is of course based on the assumption that the required reliability can not be achieved solely by a rigorous development. Fault tolerance will not be discussed in this paper. Neither will the necessity to continuously monitor the work being performed to be able to evaluate it and improve it in future projects to come be addressed.

The paper will not give any deep insight into any of these techniques. The goal is rather to outline a possible concept towards engineering of reliable software in the future.

An illustration of what is believed to be a concept towards engineering reliable software instead of crafting unreliable software is illustrated in figure 1. The concept is based on Cleanroom Software Engineering, but additions, extensions and adaptations will be proposed.

---

1 The practical experience of Cleanroom has been obtained when working as a consultant for Q-Labs, Lund, Sweden.

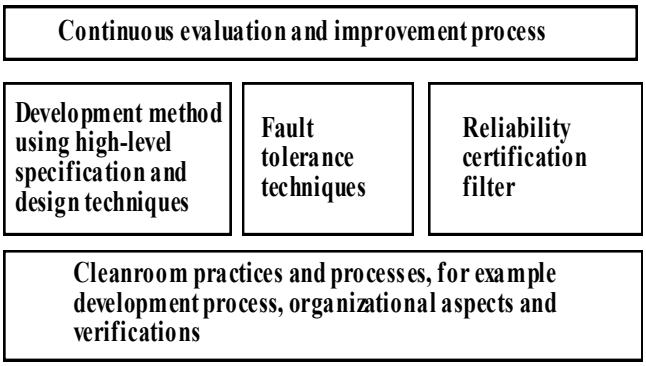

Figure 1. The context of the paper.

The concept for more reliable software is thought to be based on Cleanroom:

- Cleanroom Software Engineering provides a good basis for engineering reliable software. It contains a set of good engineering and management practices. In particular, the philosophy in Cleanroom including rigorous verifications as well as organizational and responsibility aspects forms a good basis. The Cleanroom practices are of course based on a well-documented and functioning development process. A brief introduction to Cleanroom is given in section 2.

Cleanroom contains methods for high-level specification and design as well as a method for certification, but these methods are not always sufficient for all types of applications. This motivates some adaptations of Cleanroom, then some additions are needed as well. The following items will be discussed as being suitable improvements of Cleanroom:

1 The use of other high-level specification and design techniques than the one presented in Cleanroom is argued. In particular, it is stated that the high-level design techniques are mature enough to be applied broadly in the industry, see section 3.

2 Fault tolerant techniques ought to be used, if the required reliability level can not be proven based on the rigorous development. Fault tolerance will though not be discussed in this paper.

3 The certification method proposed in Cleanroom is not particularly suitable for large system, since a state explosion in terms of user states occurs. Adaptations and additions to the certification method in Cleanroom are discussed from different views in section 4, 5 and 6.
   - Section 4 contains a brief presentation of a method which aims at certification before the testing phase.
   - Section 5 discusses an alternative certification method during testing than the one proposed within Cleanroom.
   - Section 6 presents some preliminary results concerning certification and problems with change in usage. This aspect is not discussed at all within Cleanroom.

4 A continuous evaluation and improvement process ought to be included as part of the Cleanroom development process. The evaluation and improvement process includes for example methods for early estimation of reliability or fault content. The general process will not be discussed, but a method for early reliability certification is discussed in section 4.

2. Cleanroom software engineering

2.1 Introduction

Cleanroom Software Engineering, [1, 2, 3], has shown that it is possible to improve the software quality and in the same time improve the productivity.

Cleanroom has been developed at IBM and Software Engineering Technology (SET) in the USA and it is currently being adapted to the telecommunications field by Q-Labs and the department of Communication Systems.

The Cleanroom methodology is based on the philosophy that it is possible to develop zero defect software, though it may be hard to prove. The overall principle in developing software systems using Cleanroom is to remove defects in the same development phase as they are introduced, instead of waiting for an executable code representation of the system to perform tests and defect removal on.

Two abstraction levels of Cleanroom can be identified, first the philosophical level and secondly the level containing the specific techniques. If interpreting Cleanroom primarily on the philosophical level then the actual methods can be changed as long as the overall philosophy is supported and its objectives achieved. The objective of this section is to present these two abstraction levels based on presentations of Cleanroom found in the literature.

2.2 Cleanroom philosophy

”The Cleanroom software development method has three main attributes: a set of attitudes, a series of carefully described processes, and a rigorous mathematical basis” [2]. Attitudes from the software engineers and managers to their job are very important parts in the development process. This is emphasized through the following viewpoints stressed in Cleanroom:

- Zero defect software is possible.
- Team responsibility of the work.

- Intellectual control of the software development.
- Process driven development.
- Incremental development.
- Stepwise refinement and rigorous verification.
- High level specification and design.
- Certification of software reliability.

These items are a collection of aspects stressed in the Cleanroom literature, but they still allow for application of different techniques. If these views are considered as Cleanroom, then several different techniques may be applied still fulfilling the objectives of Cleanroom, but some more specific techniques can also be found in the literature.

2.3 Cleanroom methods

To reach the philosophical objectives of Cleanroom for software development, some specific methods are proposed:

- Box Structures [4, 5] is the method proposed for specification and design of the software.
- Stepwise Refinement and Functional Verification [6, 7] are methods for implementing code in small steps and verifying them mathematically.
- Statistical Usage Testing [8] describes how the certification is to be done in Cleanroom. It is proposed that the usage shall be modelled with a plain Markov chain, [9, 10], hence allowing for generation of test cases according to the anticipated usage. The reliability model proposed is presented in [11].

These specific methods are not always possible to use, for example due to application domain, tool support or prior use of a specific development technique. It may be difficult for an organization to adopt several new methods and techniques, due to for example education and investments done in the past. This does, however, not mean that the Cleanroom philosophy can not be adopted. The philosophy in Cleanroom is believed to be more important than the actual technical methods. Therefore it will be argued that other methods may as well be applied as long as the philosophy and overall objectives of Cleanroom are fulfilled.

2.4 Adaptations to Cleanroom

The Cleanroom methodology is being adapted to telecommunication systems by Q-Labs and the department of Communication Systems. The objective is to offer full support to large multi-user systems with high quality requirements. The work has so far resulted in:

- a tailored development method for telecommunication systems, which includes application of another design technique than the one proposed within Cleanroom. The method has been applied to one of the projects discussed among the success stories, see section 3, 7 and [12, 13, 14].
- a method for statistical usage testing has been developed, see section 5, 7 and [15].
- a certification method for early reliability evaluation based on the usage has been proposed, see section 4 and [16].
- a method to evaluate the change in reliability based on a change in the usage is currently being researched. Some preliminary results are presented in section 6.

3. High-level specification and design techniques

3.1 Evolution of program development

The programming trade started with assembler and soon went on to unstructured programming. The disadvantages with programming using gotos and other non-structured concepts lead to the introduction of the structured programming concept, thus implicating that prior programming had been unstructured. It must though be noted that nobody thought of programming prior to structured programming as being unstructured. The objective of this paper is not to give an introduction to the history of programming, but to emphasize that these transitions between different types of programming were made based on belief and time. It always takes time to change a behaviour and even if computer scientists are performing quite a new trade compared to building houses etc. it is obvious that it is a conservative group of craftsmen.

Not much proof has been presented that the transition to structured programming increased the quality and in particular the reliability of the software, all the same today everybody (almost) agrees that structured programming is a necessity to cope with the systems of today. Some indications showing that structured programming actually improved quality does however exist.

A study has been presented in [17] indicating that from 35 complexity measures including McCabe´s cyclomatic complexity, the measure that had the highest correlation with software faults was the number of “gotos”. The study included about 30000 lines of code for a large telecommunication switch. Two types of gotos were used in the code of the product, namely:

- if .... then goto
- unconditional goto, i.e. only goto

The highest correlation obtained was for the number of unconditional gotos, thus emphasizing that the transition to structured programming and banning unconditional gotos possibly increased the reliability of the software being developed. This statement is supported with the

subjective feeling of many developers.

The conclusion from history is that new techniques evolve, but it takes time to introduce them. This ought in particular to be the case when the transition to new techniques can not be supported by success stories made by others, more courageous companies or project leaders.

The next step in programming evolution is the high-level design techniques, for example SDL (Specification and Description Language) standardised by CCITT [18, 19, 20] and Box Structures [4, 5]. The software community is probably not ready for formal description techniques as for example Z [21]. A step between structured programming and formal description techniques seems to be the right way. The problem encountered, when talking about high-level design techniques, is the conservative opinion that the way we do it today is the best way to develop reliable software. This is not the case. The evolution of software development will continue and this will give us more reliable software in the future.

History has shown, as pointed out above, that new more structured techniques will give an increase in reliability. Thus the introduction of high-level design techniques will probably give an increase in reliability compared with the reliability obtained in software development today.

3.2 Advantages of high-level design techniques

It has been argued that high-level design techniques ought to be introduced, hence some other benefits with the techniques will be emphasized in this section. The advantages with using a high-level design technique will be outlined to try to illustrate that these type of descriptions are mature enough to be used in an industrial environment today. In general the following advantages can be identified for high-level design techniques:

- High-level design techniques can be analysed both by tools and more easily by humans than code. The readability means that the description becomes inspectable and thus more faults are found in the early phases.
- A common description technique early in the life cycle also means that metrics can be collected early. Thus helping in planning, controlling and risk management in the early phases.
- Metrics from different types of analysis can also be used to estimate different software qualities, for example reliability and fault content. It may be possible to identify fault prone modules at an early stage.
- The outcome of the previous items will form a basis for decision making. It is hence possible to decide to re-design instead of implementing and going into test with a poor product.
- Finally, the tool support for different high-level design techniques is increasing. It is possible to find tools for editing, syntax and semantic analysis, dynamic analysis/behaviour analysis, simulation and code generation.

The list of advantages can probably be made longer if a particular high-level design technique is considered. As an example we will use SDL [18, 19, 20], since this is the high-level design technique used in our environment today.

SDL is based on dividing the system into blocks and then into processes. The processes describe the dynamic behaviour of the system. A system described with SDL consists of a number of inter-communicating processes, where the communication is made with signals. The signals contain a number of parameters. An introduction to SDL can be found in the references, unfortunately it is not possible to go into more details here. The major advantages with SDL are:

- SDL has a formal representation and it is standardized by the CCITT, hence being standardised primarily for real time systems which means that the concepts in the description technique have been adapted to the needs in real time systems.
- SDL is easy to teach and to learn. Students have found that SDL provides a good basis for describing real time systems. This results in that the industry easily can employ well-educated technicians with good knowledge in a high-level design technique, hence promoting the transition to application of high-level design techniques.
- SDL is human-oriented and it comes from the need to have a suitable description technique to describe the software at a higher level than the code. This means that SDL (or at least a subset of SDL) is widely accepted in the environments developing the software to the telecommunication systems of tomorrow.
- SDL has shown to be a suitable level for analysis of the software structure, which has been used as complexity metrics to correlate with the fault content. These metrics are applied prior to the code which make them important indicators of reliability. Some of the results have been presented in [22].
- Tools and methods for dynamic analysis of SDL [23] and simulations based on SDL have been developed, see for example [24, 25]. A method for evaluation of prediction of software reliability based on dynamic analysis is presented in [16] and the evaluation of software qualities based on simulation of the design in SDL is presented in [26]. The prediction technique from dynamic analysis is briefly introduced in section 4. The tools allow for earlier analysis than is possible otherwise, hence the tools help improving the quality since problems may be identified earlier.

Similar advantages can probably be found for other high-level design techniques as well. High-level design techniques are mature. The transition into using them on a

daily basis in the industry may take some time, but the companies managing the transition first will lead the evolution into engineering more reliable software in the future.

SDL together with message sequence charts have been used instead of Box Structures in the development method tailored for telecommunication systems. The method has been applied to a large project, see section 7.

4. Early certification of HLDT

A method for early estimation of software reliability is presented in [16]. The main advantage of the method which makes it different from other methods is the use of the operational profile for reliability certification before the testing phase and with full tool support. The method hence provides an opportunity to obtain an early and relevant indication of the expected reliability.

The certification can be made from failure statistics from for example dynamic analysis of formal descriptions. This analysis can be made either on the specification of the software or of the design of the software during development.

The approach is based on that the operational profile can be input to an analysis tool which detects certain types of probable dynamic failures. An example of a tool is SDL Behaviour Analyser (SBA) presented in [23]. From the failure statistics of the analysis tool, it will be possible to make a first prediction of the software reliability when in operation. This prediction can either be based on that the dynamic failures are supposed to be representative of the failures in the product, or a relationship between the dynamic failures and ”normal” failures has to be determined.

The proposed method can be summarized in the following procedure:

1 The usage is modelled with the same technique as the system is being described.
2 Test cases are generated from the usage model and the analysis cases are described with the description technique applied for system description.
3 The usage model is put together with the system description in the tool supporting the description technique.
4 The dynamic analysis tool is capable of locating certain types of faults. The analysis of the system is made based on analysis cases generated from the usage model. This means that a partial dynamic analysis is done, which shall be compared with the full analysis which would have been done if not controlling the analysis with the generated analysis cases.
5 The failure statistics from the analysis tool is put into a reliability growth model to allow for reliability estimation and prediction of future failure behaviour.
6 A full dynamic analysis is then performed using the analysis tool, hence locating all faults that the analysis tool can find.
7 The failure times are recorded and a normalization procedure is applied to get failure statistics which can be compared with the predicted failure behaviour. The goodness of the prediction can hence be calculated. This information is important to be able to compare with the usage testing, which is assumed to be carried out. The hypothesis is that if the prediction from dynamic analysis is good, then the prediction of the reliability growth from usage testing ought to be reliable.
8 A transformation from faults found by the analysis tool to an arbitrary fault in the software must be applied. This requires a re-calculation metric describing the number of faults remaining in the software in comparison with the number of faults found in the dynamic analysis.

This procedure will allow for estimation and prediction of the software reliability at an early stage in the development based on the operational profile, which make the method more realistic than most methods proposed for early estimation of software reliability. Most methods merely estimates the fault content and uses this as an indication of reliability.

The method and its opportunities are discussed in more detail in [16].

5. Certification during testing

5.1 Statistical Usage Testing according to Cleanroom

Statistical Usage Testing (SUT), [1, 2, 3, 8, 9, 10], is the certification method described as a part of the Cleanroom software development method. The goal for SUT in Cleanroom is not, as in traditional software development, to find as many faults as possible but to certify the software reliability. The planning and certification of software system reliability is also discussed in [27].

Software reliability depends not only on how correct the software is, but also on how it is used. If there is a failure for a certain state and stimulus, its effect on reliability will depend on how often this event arises. This depends on how often the state is reached and how often the certain stimulus is selected. This reality is considered by the Statistical Usage Testing and that is why it can be the basis for certification.

The original proposal in Cleanroom for modelling the usage is a plain Markov model, [9, 10]. We have encountered that this type of model will soon become too large and complex for large multi-user systems. The number of usage states soon becomes cumbersome, often referred to as the state explosion problem. The problem has been solved by introducing a hierarchical Markov model, presented in [28, 29], see also section 5.2 and 5.3.

Certification is the control of the quality fulfilment, e.g. to certify that a specific reliability has been obtained. Based on the fact that tests are carried out from the test cases compiled, it should be possible to predict the software reliability that can be expected in actual operation. The reliability model referred to in the Cleanroom literature is presented in [11]. The main disadvantage is that the application of a reliability model mostly requires a number of failures to occur, which contradicts the objective of Cleanroom in general. Therefore another type of certification method is needed, see section 5.2.

5.2 Adaptations of Statistical Usage Testing

Statistical usage testing consists of two major parts, i.e. usage modelling, which includes construction of a usage profile, and reliability estimation. The adaptations to telecom concern both these parts. Q-Labs has conducted a project for Swedish Telecom to provide them with a certification method to be used in acceptance testing when purchasing software systems. The usage model is a description of how the software is used in operation, which stimuli are sent in different cases. The usage profile illustrates the probabilities for the different events. The test cases are generated from the usage profile by random selection according to the software usage. The certification is performed by analysis of the failure data collected during testing. It must be possible to, with a certain confidence, certify a particular reliability level or predict the reliability at some point of time in the future.

The following adaptations have been made:

- Introduction of a new model to describe usage, i.e. a hierarchical Markov chain, [28, 29]. The model allows for dynamic changes in the probabilities based on the states of the users. This is not supported by the original method proposed in Cleanroom, neither by the method proposed in [30, 31, 32]. The hierarchical model aims at generating one event at the time and not complete functions. The objective is that the generation shall support a mixing of events which resembles the actual operational phase. The hierarchical Markov usage model is briefly described in section 5.3 and it is further discussed in [28, 29].
- Instead of applying reliability growth models as the one presented in [11], it is proposed that the hypothesis testing technique presented in [33] shall be applied. The main advantage of the hypothesis testing technique is that it does not require a number of failures to occur before it can be applied, but on the other hand it does not provide a prediction opportunity. Therefore it is recommended to use the hypothesis testing technique to accept or reject the software and then optionally it is possible to apply a reliability growth model for prediction.

  The hypothesis testing technique means that a control chart shall be used. The diagram shows the normalized failure time versus the failure number. The control chart contains two lines, which divides the diagram into three parts, one continue to test region and one region depicting if the reliability requirement has been fulfilled and finally one region showing if the software shall be rejected. The lines are drawn based on the reliability requirement and the needed confidence in the decision. This method allows decision making concerning the reliability based on the failure data available and a predefined confidence. The method does not need a certain number of failures to occur, therefore the method is better to use in Cleanroom projects where the expected number of failures found is supposed to be low.

These two adaptations to Cleanroom are the basis in a method provided to Swedish Telecom, which shall be used during acceptance testing of software products.

5.3 Hierarchical Markov usage model

The hierarchical usage model is developed based on the inability to apply a plain model on a large system with numerous users. The plain Markov model clearly grows too large, since it depends on the number of users in a non-linear way. This observation made it necessary to model the usage of the system in another way.

It was decided to model the usage in a hierarchical Markov model, because it gave a possibility to divide the problem domain in a natural way. The first level in the hierarchy is a common usage level, the next level is supposed to describe the different user types of the system, the third level shall model the actual users, while the fourth and last level models the services provided to the users.

The hierarchy means that a service used by several users is only modelled once and then instantiated for all users using that particular service. The generation of test cases are made through traversing the Markov hierarchy. The next event to be added to the test case is generated by first choosing a particular user type, then a specific user of the chosen type and finally based on the state of the chosen user a transition (event) is added to the test case. The last level will be referred to as the service level since this level models the services available to the users. Events are thus added to the test case, based on the operational (or usage) profile assigned to the usage model. The operational profile is hence taken into account in every event added to the test case.

The model also allows for dynamic probabilities, which better capture the actual behaviour in operation. It is obvious that it is more probable that a subscriber who has recently lifted his/her receiver dials a digit, than that a specific user lifts his/her receiver. This means that the choice of a specific user to generate the next event depends on the actual state of the user. This is handle by

introducing state weights which model the relative probability of generating the next event compared with the other states of the service. The dynamic probabilities are easily included in the test case generation procedure. This opportunity is not handled in any of the other models describing the usage. Therefore the hierarchical Markov model is believed to be a valuable addition to model usage both within Cleanroom and in other environments.

The structure of the usage model supports reuse as well with its object oriented approach. A service is modelled with a plain Markov chain and hence as a service is added to the software system, it is easy to add the usage model of the component implementing the service to the usage model of the system. The plain Markov model has to be re-developed as changes to the system are made. The hierarchical model is reusable to the same extent as the components. The certification of components and the reliability of systems based on component certification is further discussed in [27, 34].

The hierarchical Markov usage model is further discussed in [28, 29].

5.4 Some problems related to change in usage

The application of statistical usage testing has one major problem; the usage profile applied during testing is not correct due to changes in the usage. Some different reasons to experience a change in usage compared to the usage profile applied during certification can be identified:

- An erroneous profile was applied during the certification. The change will be experienced as the software is being released.
- A change will occur with the time, either slowly or perhaps quickly due to for example marketing of some specific services.

This calls for methods for predicting the reliability based on change in usage. This work is currently being done, but some ideas can be presented, see section 6.

6. Future certification techniques

6.1 Introduction

Two different approaches for estimating the reliability (or MTBF) of a probable future change in the usage can be identified. This implies that it is not suitable to wait until the change has occurred. The first opportunity is to change the usage profile and make a new certification, while the second possible solution is to make a re-calculation of the reliability based on the failure data obtained from the old usage profile and the new usage profile. Both of these approaches have their pros and cons. The two approaches have been examined in a minor pre-study from which the results are summarized below.

6.2 Conclusions: new profile

It can be concluded that it will be costly to execute several usage profiles, but it may be worthwhile to try to estimate the future reliability when the usage changes. It is advantageous if this type of investigation can be made in parallel with the operational phase, instead of having to examine several different possibilities during system test. This type of investigation shall be made to capture a possible reduction in reliability before it is experienced by the users. This does, however, not cover the case when the usage profile is slightly wrong during the original testing phase, because an erroneous profile during testing has already been experienced in terms of a decrease in reliability by the users. The application of a new usage profile may be very important to be able to continue to have the same software reliability, even if the usage changes over time.

6.3 Conclusions: re-calculation

It can be concluded that a re-calculation procedure is advantageous compared to applying a new profile. The main reason is of course that the re-calculation can be more easily performed, than by performing re-testing.

The major problem is, however, that it is questionable if the reliability estimate is trustworthy. The change in usage may result in failures during operation that have not been found before, due to another usage profile. These aspects have to be examined further or a procedure for estimating the number of faults in specific parts of the software must be identified. Some work has been done in the area of complexity metrics, but no general method being able to predict the fault content has been found [35].

A re-calculation procedure is still a tractable method, hence indicating that it ought to be used because of its simplicity and to gain experience from it. Research has however to be conducted in the area to make it really useful.

7. Success stories

Some success stories concerning high-level design techniques, verification techniques, usage based testing and Cleanroom approaches exist in the literature:

1 Two to three times increase in quality when using SDL has been experienced at Ericsson in Norway [36]. They have measured the number of faults per line of code from integration and function tests.
2 The introduction of SDL at AT & T has lead to that the expected number of faults found during the entire testing interval was reduced to 25%. This indicates a considerable increase in quality in terms of fault content when introducing high-level design

techniques. The results are presented in [37].

3 At Northern Telecom it has been shown that, ”Inspections were two to four times more efficient at finding errors than either formal designer testing or system testing. If non-execution errors such as code optimization and non-compliance to standards are included, the difference is even larger” [38]. The result is based on data collected from 2,5 million lines of code in eight system releases. Similar experiences are reported at AT & T [39].

4 Operational Profile Testing is currently used and being developed at AT & T. It is concluded from the projects at AT & T, [30, 31, 32, 40, 41], that the cost for system test and the overall project cost are reduced considerably. In [31], it is stated that the cost reduction for system test for a ”typical” project is 56%, which is 11.5% of the total project cost.

5 The major reason for the high quality and high productivity in development using Cleanroom is in [42] explained with the avoidance of defect transfer through consecutive development phases.

The experiences presented in the literature support the projects results obtained in two projects conducted by Q-Labs for two separate customers:

6 A method based on Cleanroom and high-level design techniques (message sequence charts and SDL) has been developed and it is used in a 100 man year project [43]. The project develops a new operating system for a telephone exchange. The method is described in [12, 13, 14]. The project can be characterized by:

- More resources and time are allocated to the earlier phases of the project.
- The project is divided into teams according to Cleanroom. The team responsibility is emphasized and it is an important factor for all teams.
- Reviews are conducted regularly and they are the basis in the verification procedure. Each week is divided into three days of development, one day of preparation for review and one day of review.
- The unit testing is omitted, instead the time is spent in the earlier phases and in the reviews

The project is running at the moment hence no formal results or metrics exist. However, clear improvements have been achieved both in quality and productivity. Some indications can however be presented:

- The project is still on the original schedule, i.e. the time schedule of the project has not been revised in 18 months. The functional content of the project has not been changed either.
- The effort used to locate a fault has been lowered considerably. The efficiency has increased by 20 times.
- The productivity has been doubled, i.e. in terms of lines of code per hour.

The objective is to apply usage testing as the product goes into the testing phase.

7 For the Swedish Telecom a method for acceptance testing has been developed. This method is currently put into requirements specifications to enforce that this method shall be used in the acceptance procedure. The method includes both a rejection criterion as well as an acceptance criterion. Thus giving the Swedish Telecom in its role as a large purchaser of software systems an opportunity to evaluate the software reliability requirements. An evaluation and improvement of the method will be done as the method has been in use in a couple of purchase situations.

8. Conclusions

A reliable software system is a system on which the user can rely and the basis for reliance is the absence of failures. The methods discussed: Cleanroom Software Engineering with high-level design techniques, verifications, inspections and certification as well as other techniques as fault tolerant techniques are not a guarantee for zero defect software, but it will increase the quality. Since it is not possible to actually prove that the software is free from defects, it is necessary to combine many methods to engineer reliable software, instead of producing software with too many faults as the product goes into operation. The latter is unfortunately more of a rule than an exception.

The proposed methods produces reliable software by turning software development into an engineering practice instead of looking at software development as a private art form for hackers. A large software system with ”smart” local solutions will never became a dependable, reliable and maintainable system.

The engineering approach, as in for example Cleanroom, includes several techniques and it is the sound application of the total concept that makes the software reliable. The problems of software failures in operations will not be solved with one technique, e.g. object-orientation, or by applying more sophisticated software tools. The only way to reliable software systems is to stay in intellectual control by applying sound engineering disciplines throughout the life-time of the software.

The application of sound engineering disciplines is accepted in almost all other fields of engineering. Who would drive across a bridge which was constructed based on ad hoc techniques similar to the ones applied in software development? Bridge building has, however, been around for quite a long time and it took a long time to get to where bridge building is today. This can, however, not be an excuse for not applying engineering techniques in software development. The society today depends heavily on the software, which makes us extremely vulnerable to the failures. Thus, the private art

of software development must be abandoned and turned into an engineering activity.

Cleanroom or similar concepts turn software development into an engineering discipline. Hence, these methods and techniques will help in the development of reliable software systems in the future.

Acknowledgement

I would like to acknowledge the reviewers of the paper, who provided many valuable comments which have improved the quality of the paper.

References

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

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

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

[4] Mills, H. D., ”Stepwise Refinement and Verification in Box-structured Systems.”, IEEE Computer, pp. 23-36, June 1988.

[5] Mills, H. D., Linger, R. C. and Hevner, A. R. ”Principles of Information Systems Analysis and Design”, Academic Press Inc. 1986.

[6] Mills, H. D., ”Structured Programming: Retrospect and Prospect”, IEEE Software, pp. 58-66, November 1986.

[7] Linger, R. C., Mills, H. D. and Witt, B. I., ”Structured Programming Theory and Practice”, Addison-Wesley Publishing Company, 1979.

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

[9] Whittaker, J. A., ”Markov Chain Techniques for Software Testing and Reliability Analysis”, Dept. of Computer Science, University of Tennessee, Knoxville, USA, 1992, Ph.D. Dissertation.

[10] Whittaker, J. A. and Poore, J. H., ”Statistical Testing for Cleanroom Software Engineering”, Proceedings 25th Annual Hawaii International Conference on System Sciences, pp. 428-436, 1992.

[11] Currit, P. A., Dyer, M., and Mills, H. D., ”Certifying the Reliability of Software”, IEEE Transactions on Software Engineering, Vol. SE-12, No. 1, pp. 3-11, January 1986.

[12] Cosmo, H., Sixtensson, A. and Johansson, E., ”SMO – A Stepwise Refinement and Verification Method for Software Systems”, In ”SDL '91: Evolving Methods”, Editors: O. Færgemand and R. Reed, pp 137-147, North-Holland, The Netherlands, 1991.

[13] QCCC (Q-Labs Cleanroom Competency Center), ”Cleanroom Software Engineering Applied to Telecommunications”, Proceedings Nordic Seminar on Dependable Computing Systems, pp. 253-264, Trondheim, Norway, 1992.

[14] QCCC (Q-Labs Cleanroom Competency Center), ”Cleanroom Software Engineering in Telecommunication Applications”, Submitted to Software Engineering and Its Application, Paris, France, November 1993.

[15] Runeson, P., and Wohlin, C., ”Statistical Usage Testing for Software Reliability Certification and Control”, Accepted for publication at EuroSTAR'93, London, UK, October 1993.

[16] Wohlin, C., and Runeson, P. ”A Method Proposal for Early Software Reliability Estimations”, Proceedings 3rd Int. Symposium on Software Reliability Engineering, pp. 156-163, Raleigh, North Carolina, 1992.

[17] Magnusson, C., ”A Study of Software Complexity”, Department of Communication Systems, Lund Institute of Technology, Sweden, Master thesis, 1983, (in Swedish).

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

[19] CCITT, ”SDL Methodology Guidelines”, Appendix I to Z.100, 1992.

[20] Belina, F., Hogrefe, D. and Sarma, A., ”SDL with Applications from Protocol Specifications”, Prentice-Hall, UK, 1991.

[21] Spivey, J., ”The Z Notion - A Reference Manual” Prentice-hall, Englewood Cliffs, N.J., 1989.

[22] Lennselius, B., ”Software complexity and its impact on different software handling processes”, Proceedings 6th International Conference on Software Engineering for Telecommunication Switching Systems, pp. 148-153, 1986.

[23] Ek, A. and Ellsberger, J., ”A Dynamic Analysis Tool for SDL”, In ”SDL '91: Evolving Methods”, Editors: O. Færgemand and R. Reed, pp 119-134, North-Holland, The Netherlands, 1991.

[24] Karlsson, J. and Ek, A., ”SSI - an SDL simulation tool”, In: SDL ´89 - The language at work, Editors O. Faergemand and M.M. Marques, Elsevier Science Publisher, North-Holland, pp. 211-218, 1989.

[25] Sredniawa, M., Kakol, B. and Gumulinski, G., ”SDL in performance evaluation”, Proc. 3rd SDL Forum, Eindhoven, Netherlands, pp. 21.1-21.11, 1987.

[26] Wohlin, C., ”Evaluation of Software Qualities during Software Design”, Submitted to IEEE Software, special issue on Safety-Critical Software, 1993.

[27] Poore, J. H., Mills, Harlan D., and Mutchler, David, ”Planning and Certifying Software System Reliability”, IEEE Software, pp. 88-99, January 1993.

[28] Runeson, P., ”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.

[29] Runeson, P., and Wohlin, C., ”Usage Modelling: The Basis for Statistical Quality Control”, Proceedings 10th Annual Software Reliability Symposium, pp. 77-84, Denver, Colorado, 1992.

[30] Musa, J. D., ”The Operational Profile in Software Reliability Engineering: An Overview”, Proceedings 3rd International Symposium on Software Reliability Engineering, pp. 140-154, 1992.

[31] Musa, J. D., ”Software Reliability Engineering: Determining the Operational Profile”, Technical Report AT & T Bell Laboratories, Murray Hill, NJ 07974, New Jersey, USA, 1992.
[32] Musa, J. D., ”Operational Profiles in Software Reliability Engineering”, IEEE Software, pp. 14-32, March 1993.
[33] Musa, J. D., Iannino, A. and Okumoto, K., ”Software Reliability: Measurement, Prediction, Application”, McGraw-Hill, New York, 1987.
[34] Wohlin, C., and Runeson, P., ”Certification of Software Components”, Submitted to IEEE Transaction on Software Engineering, special issue on Software Reliability, 1993.
[35] Lennselius, Bo and Rydström, Lars, ”Software Fault Content and Reliability Estimations from Telecommunication Systems”, IEEE Journal on Selected Areas in Communications, Vol. 8, No. 2, pp. 262-272, 1990.
[36] Rød, T., ”Experiences - SDL and measures”, EEN, 900829, (in Norwegian).
[37] Klick, Vickie, B., Patti, Joanna and Todd, Marie, L., ”Experiences in the Use of SDL/GR in the Software Development Process”, In: SDL ´91 - Evolving Methods, Editors O. Faergemand and R. Reed, Elsevier Science Publisher, North-Holland, pp. 449-457, 1991.
[38] Russell, G. W., ”Experience with Inspection in Ultralarge-scale Developments.”, IEEE Software, pp. 25-31, Jan 1991.
[39] Fowler, P. J., ”In-process Inspections of Workproducts at AT&T.”, AT&T Technical Journal, pp. 106, March/April 1986.
[40] Abramson, S. R., Jensen, B. D., Juhlin, B. D. and Spudic, C. L., ”International DEFINITY Quality Program”, Proceedings International Switching Symposium, Yokohama, Japan, 1992.
[41] Juhlin, B. D., ”Implementing Operational Profiles to Measure System Reliability”, Proceedings 3rd International Symposium on Software Reliability Engineering, pp. 286-295, 1992.
[42] ”The Cleanroom Case Study in the Software Engineering Laboratory – SEL 90-002”, Software Engineering Laboratory, 1990.
[43] Presentation material from the OS-32 project, Ellemtel, Sweden, 1992, (in Swedish).