H. Cosmo, G. Fagerhus, E. Johansson, E-A. Karlsson, P. Runeson, C. Sandahl, A. Sixtensson and C. Wohlin (authors in alphabetical order), "Cleanroom Software Engineering Applied to Telecommunications, Proceedings Nordic Seminar on Dependable Computing Systems, pp. 253-264, Trondheim, Norge, 1992.

Cleanroom Software Engineering Applied to Telecommunications

Q-Labs Cleanroom Competency Centre (QCCC) [1]
E-P Telecom Q-Labs
IDEON Research Park
S-223 70 LUND
Sweden
Phone: +46-46-182980
Fax: +46-46-152880
E-mail: qccc@q-labs.se

[1] Members of QCCC are: Henrik Cosmo, Geir Fagerhus (Manager Q-Labs), Erik Johansson, Even-André Karlsson, Per Runeson, Christer Sandahl, Anders Sixtensson (Project Leader QCCC) and Claes Wohlin.

Abstract

This paper presents a methodology for developing software systems denoted Cleanroom Software Engineering. The methodology has been developed at IBM and Software Engineering Technology (SET) in the USA, and is currently being adapted and applied to the field of telecommunications by E-P Telecom Q-Labs.

The paper gives a brief introduction to Cleanroom. The main objective of Cleanroom is to introduce a set of engineering techniques which shall form a sound basis for developing zero defect software. The emphasis in the presentation is on the work made to adapt the methodology to telecommunications.

The adaptations consist of two main areas, i.e. a development method and a certification method. The objective of the development method is to capture several different aspects of the system at an early stage by using different description techniques, while the objective of the certification method is to certify the reliability level, instead of as in traditional testing locate failures.

It is in particular emphasized how the Cleanroom methodology with its adaptations will provide a way to develop dependable computing systems in the future.

1. Introduction

A dependable computing system can be obtained in two major ways, either by making the system correct when implementing it or by introducing different sorts of fault tolerance in the system. In practice, the solution must be to combine the two alternatives. The first alternative is certainly the best approach, if it can be made without higher costs.

A methodology called Cleanroom Software Engineering, [Mills87, Mills88b, Dyer92], has shown that it is possible to improve the software quality and in the same time improve the productivity. This is obtained through a rigourous approach from the beginning. Cleanroom emphasizes:

- Organizational aspects, both through divisions into different teams and by team responsibility for the performed work within the team.
- Incremental development.
- Rigourous specification before design.
- Stepwise refinement in verifiable steps.
- Usage testing.
- Certification of the reliability.

Cleanroom has been developed at IBM and Software Engineering Technology (SET) in the USA and it is currently being adapted to telecommunication by Q-Labs. The paper gives a brief introduction to Cleanroom Software Engineering and in particular describes the work in adapting Cleanroom for telecommunication systems, i.e. a suitable development method as well as a certification method.

It is described how the proposed development method will give a sound basis for obtaining dependable systems, by employing different description techniques to capture the different aspects of system development at an early stage. The development is made through stepwise refinement. The method also includes a rigourous inspection strategy as well as the important aspect of team responsibility.

The presentation continues with a description of how the usage of telecommunication systems can be modelled and how the failure data can be used to make a statistical quality control of the software. The latter includes describing how a software reliability model can be applied to the failure data to certify the reliability. The certification method assures that the released product will be dependable during the operational phase.

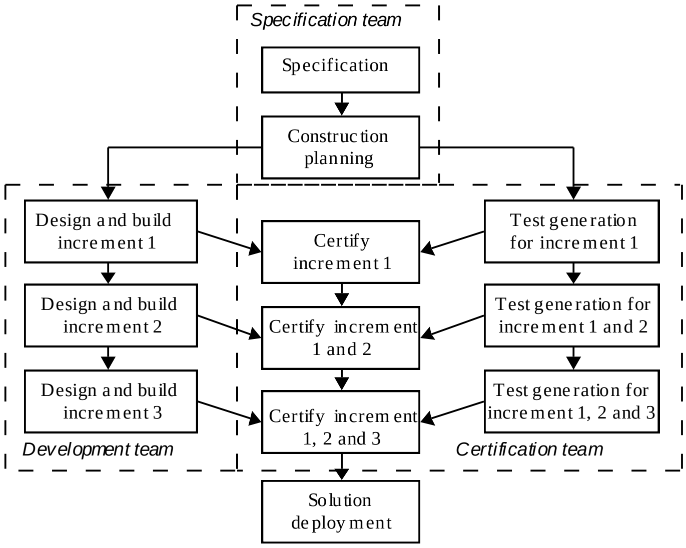


Figure 1. Outline over a three-increment Cleanroom project.

It will also be discussed how Cleanroom Software Engineering is believed to be one of the best ways towards dependable computing systems in the future. The on-going and future work with Cleanroom for telecommunications is described briefly. Finally, some conclusions from the work is presented.

It must in this context, however, be noted that Cleanroom as well as the adaptations made can be applied to other types of systems as well. The objective of the adaptations made is primarily to cope with some of the properties of telecommunication systems, e.g. large real-time multi-user systems, for which the original proposal of Cleanroom is not suited. Cleanroom can though be applied to all types of systems, even if some adaptations may have to be made to cope with some special properties of other types of system. It is believed that based on the original proposal in Cleanroom and the adaptations made, most systems can be developed with Cleanroom. Hence, software system development has not to be error-prone.

2. Cleanroom Software Engineering

The Cleanroom methodology is based on the philosophy that it is possible to develop zero defect software. 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. This avoidance of defect transfer through the consecutive development phases is the major reason for the high quality and high productivity in development using Cleanroom [NASA90, OS-32].

2.1 A Cleanroom environment

”The Cleanroom software development method has three main attributes: a set of attitudes, a series of carefully described processes, and a rigourous mathematical basis” [Mills88b]. Attitudes from the software engineers and managers to their job are very important parts in the development process. Cleanroom focuses on some points:

- The goal is producing zero defect software.
- All work is performed by teams. The team members have joint responsibility for their products.
- Every step in the development is verified towards the previous step.
- The manager must allow that time is spent on specification, design and verification, which leads to later coding.
- Certification of software reliability through statistical usage testing.

2.2 Organisation

Cleanroom assumes that software is developed by teams. Development tasks are managed by three different types of teams, specification, development and certification teams. The specification team produces the specifications and a construction plan. The development team implements the specified behaviour, there is no place for interpretations and additions of their own. Members of the development team neither compile nor make unit testing. They leave their constructed program parts as pure text, strictly verified to the certification team. The certification team generates test cases, compiles, tests and certifies the software reliability, without bothering about the implementation, see figure 1.

2.2.1 Specification team

The first step in a Cleanroom software project is to produce the specification which consist of four parts: external specification, internal specification, statistical usage profile and a construction plan. The external specification defines how the software shall behave and be apprehended from the user´s point of view. The internal specification must be implementation-independent and defines the responses in terms of stimuli history, i.e. the stream of stimuli to the program. The statistical usage profile describes how the software will be used, the stimuli generated to the program, and their distributions. The construction plan describes how the specification is decomposed into executable increments. Every increment is a partial set of the total functionality. This decomposition into executable increments, makes it possible to get an early indication on the software quality. The software quality is an indication on the development process quality as well. The possibility to certify the quality is given by the ability to run and test every increment.

2.2.2 Development team

The development team works with the internal specification for the actual increment as basis to implement it into program code. The implementation is done by stepwise refinement, following a strictly defined algorithm. Every step is verified back to the previous, before the next step in the process is started. The developers have neither compiler nor debugger. The common responsibility for the product forces the developers to take part of and verify each other´s code. It can be psychologically difficult in the beginning, but is very good in practice because, as known, four eyes can see more than two.

2.2.3 Certification team

In Cleanroom the certification always concerns the accumulation of produced increments, see figure 1. Unit testing is never used but is replaced by verifications. The idea is that, because of the strictly controlled development process, there is no need for such testing. Experience shows that debugging of smaller units correct smaller local faults, but introduce new global errors [Adams84].

In parallel with the construction work, the certification is started by generating test cases for the actual increment, see figure 1. The basis is the statistical usage profile, which is developed in the specification phase. The test cases are generated by some statistical selection method. The certification team compiles the code and then the testing activities start. The aim is to certify the software reliability. The results from executing the test cases are used to estimate the current reliability level as well as to predict the reliability of the next increment by applying a software reliability growth model. The software is always tested from the user´s point of view described in the specification. Any failures occurring and when they arise are documented, but it is a task for the development team to deal with them.

2.3 Methods

To reach the goals for software development, some methods are proposed in Cleanroom. Box Structures [Mills88a, Mills86a] is a method for specification and design. Stepwise Refinement and Functional Verification [Mills86b, Linger79] are methods for implementing code in small steps and verifying them mathematically. Statistical Usage Testing [Cobb90] describes how the certification is to be done in Cleanroom.

The Cleanroom methodology is being adapted to telecommunication systems by Q-Labs, i.e. to offer full support to large multi-user systems with high quality requirements. This has so far resulted in two major developments which will be described in the subsequent sections of the paper:

- a tailored development method for telecommunication systems.
- a method for statistical usage testing.

3. SMO development method

The SMO development method [Cosmo91] is a stepwise refinement and verification method in which several complementary description techniques are used to capture different aspects of the system, as well as providing different views on the system to capture more faults early. Thus, the main goal with the method is to support the production of correct specifications and to give the base for removing defects as they are introduced.

To cope with the problems specific for telecommunication systems, adaptations and extensions have been added to the Cleanroom methods. This section will after a brief description of the used Cleanroom concepts and methods describe the adaptations done.

The development of SMO has been made for and is currently used in a large development project at Ellemtel Telecommunication Laboratories, [OS-32].

3.1 Box Structures concept

The Box structures concept is based on three basic system structures that can be nested over and over again in a hierarchical system structure. The three system structures are called Black Box, State Box and Clear Box. They represent different abstractions and provide three aspects of a system or any of its sub-systems.

The *Black Box*, as the name implies, is a description of a system that omits all details of internal structure and operations. It deals solely with the behaviour that is visible to its users in terms of stimuli and responses. Any Black Box response is uniquely determined by the stimuli history of the system. Stimuli history is the order in which the stimuli have been received by the Black Box. The Black Box can be considered as a requirement statement for the (sub) system.

The *State Box* gives an intermediate system view by opening up the Black Box one step. The State Box (state-machine) consists of a state which is designed from an analysis of the required stimuli history and responses from the system and a machine, which performs some behaviour.

Finally, the *Clear Box*, opens up the State Box description of a system one more step and provides a view of the state and how stimuli are processed. The internal State Box is replaced with sequential or concurrent usage of other Black Box sub-systems. These new Black Boxes are expanded at the next level into State Box and Clear Box forms.

The expansion of Black Boxes into State and Clear Boxes produce a Box Structures hierarchy. The State and Clear boxes may use new Black Boxes, which then are expanded. A Box Structures hierarchy provides an effective means of management control in developing systems. By identifying Black Box sub-systems at higher levels of the system, only a manageable amount of state data and processing needs to be handled within each step. The Black Box sub-systems become well-defined and independent modules in the overall system.

3.2 Refinement and verification

Stepwise Refinement and Functional Verification (here abbreviated SRFV) [Mills86b, Linger79] defines a manner how to construct code from a defined program function and how to verify its correctness. SRFV produces an implementation, defined by a hierarchy of small implementation steps. It supports immediate verification of the correctness of the steps.

The basic idea in SRFV is implementing a program by decomposing it into subprograms down to the very lowest level where program constructs are used. A function is considered as being composed of smaller sub-functions. These sub-functions consist of other sub-functions, and the lowest level sub-function consists of pure program constructs. The verification of constructs are discussed in detail in [Linger79]. For verifying the whole program, the decomposition is reversed to a composition, where the correctness of each step is proved.

3.3 Box Structures method

The Box Structures method [Mills88a, Mills86a] is SRFV applied on the Box Structures concept. It is a structured method for system development. It provides a simple but rigourous framework for specification and design. There exists a twelve step algorithm called the Box Structures algorithm [Mills88a], which produces a hierarchy of small design steps that supports the immediate verification of their correctness.

3.4 SMO

The Box Structures algorithm supports development of a lot of different types of systems. But when applying Box Structures for large multi-user systems like telecommunication systems, adaptations are necessary. We have encountered the following major problems in the telecommunication domain:

- Box Structures does not give enough support in the analysis phases, when going from the mission domain to the software domain. That is when going from customer requirements to software specifications.
- Bos Structures is weak in describing parallel sub-systems communicating with each other in real time.
- It is difficult to express and completely specify the large system Black Box in stimuli history with the description techniques proposed today.
- More support is necessary in the stepwise refinement and verification procedure of Box Structures. This is especially true when not defining the Black Box completely, which is required by the Box Structures algorithm.

To cope with these problems the SMO method, besides Box Structures, includes two other description techniques: sequence charts and SDL (Specification and Description Language) descriptions. SDL is standardised by CCITT and described in [CCITT88, Belina91]. The two description techniques are integrated in different phases in Box Structures and help us to solve the problems discussed above. How and why will be described briefly in the following subsections, while a more detailed description is found in [Cosmo91].

The last phase of SMO when constructing code from lowest level Clear Boxes are based on SRFV. This work has meant adapting the SRFV-ideas to the used description techniques and to target languages of telecommunication systems.

3.4.1 Overview

The SMO method consists of five phases (figure 2):

- Analysis,
- Specification,
- High level design,
- Detailed design,
- Implementation.

3.4.2 Analysis

The first step is to identify the system boundary and the different users of the system. The second step is to identify the transactions. Transactions are the different ways in which users want to use the system. This is in high degree an iterative activity, which continues until no new transactions can be identified. The result is documented with sequence charts. The sequence charts, in this phase of SMO called Sequence Chart Specification (SCS), will also be a part of the specification, since they are requirements on different uses (functionality) of the system.

3.4.3 Specification

The system Black Box is completely defined based on the analysis results, i.e. the SCS. The Black Box is defined by identifying stimuli, responses, and the transitions, mapping stimuli histories into responses. The system Black Box is verified against the analysis results.
An SDL specification is made from the system Black Box and from the sequence charts. No new information, except user states, should be introduced at this stage. User states are the states of the system that the user can perceive. The SDL specification is verified against the system Black Box and the Sequence Chart Specification. The SDL specification should correspond to the usage model, see section 4.

Finally a textual specification of the functionality of the system is written. The document focuses on a functional view of the system and is useful for initial communication with the customer.

The specification of the system includes sequence charts, a system Black Box, and an SDL specification, in which the functionality is specified from *three different views*. Sequence charts use a function oriented view of the system, Box Structures uses a stimulus oriented view and SDL uses a state oriented view. The three views help specifying the mission correctly, to understand the system and to get an overview of different aspects of the functionality. They also make it possible to make an easy verification of the consistence of the specification.

The different views have different purposes later in the development as well. The stimuli history in the Box Structures gives us information of how to design our system and what stimuli history need to be stored as data. SDL and sequence charts can give us information of how to test our system.

3.4.4 High level design

In the high level design phase the top level architecture is designed and documented. Three main activities corresponding to the different description techniques are performed.

A Box Structures design is made from the system Black Box, by State Box expansion and Clear Box expansion. Decisions are taken whether to keep the data at this level in the Box Structures hierarchy or to migrate the data downwards to a lower level of Black Box sub-systems.

Next sequence charts, that describe the interaction between the Black Box sub-systems, are made. These are named sequence chart descriptions. The result is

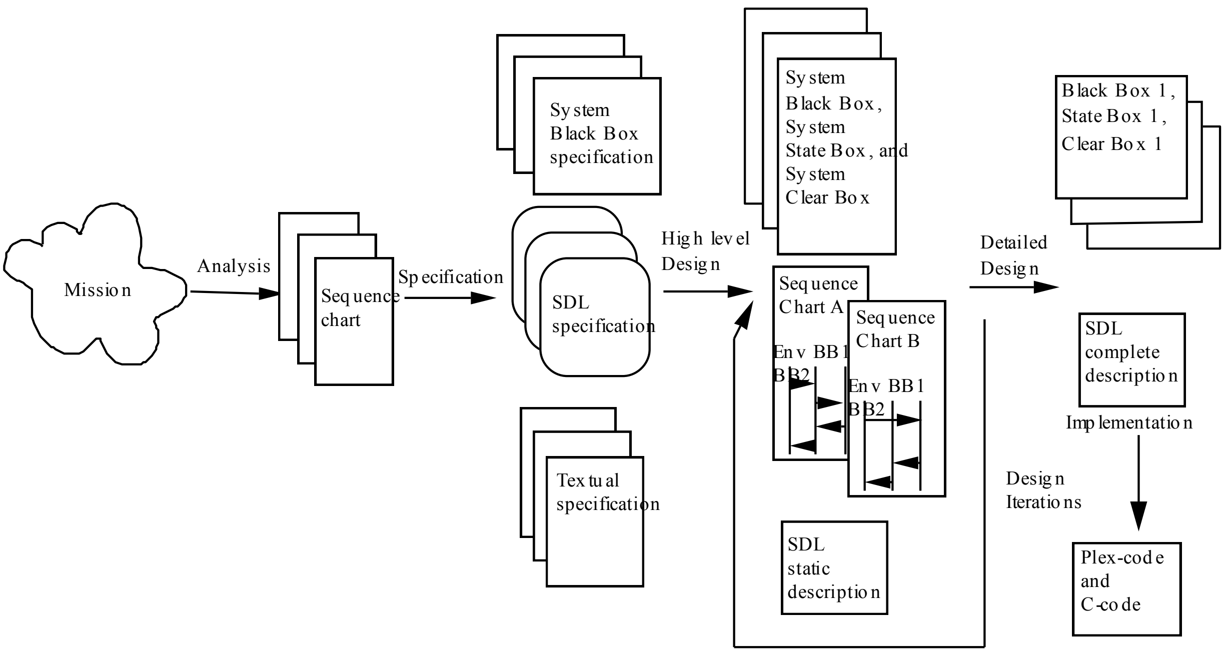


Figure 2. The SMO method

verified against the system Clear Box and the sequence chart specification.

Finally, a static SDL description, that describes the interfaces between the Black Box sub-systems, is made from the information kept in the sequence charts and the system Clear Box. A complete static SDL description is obtained by the CCITT method, ”Stepwise production of an SDL specification” [CCITT92]. The SDL description is verified against the sequence chart and against the Box Structures design.

All three activities include verifications against each other and previous phases. These verification activities will together with the different views used in the different activities give a base for zero defect design.

3.4.5 Detailed design

In the detailed design phase the same activities as in high level design are repeated but this time for the Black Box sub-systems. For each of the sub-systems even lower level sub-systems (sub-sub-systems) can be designed. Then detailed design is performed for them, etc.

When a complete Box Structures description of the whole system is completed, i.e. no more levels of Black Boxes exists, a dynamic SDL description is produced. The behaviour of each Clear Box is then described by an SDL process behaviour.

3.4.6 Implementation

In this phase the dynamic SDL descriptions of the lowest level Clear Boxes are refined to code. By using an algorithm based on SRFV the SDL description is stepwise transformed to target machine code.

3.4.7 Further adaptations

The development of SMO continues. Work has been and is being done in several areas. Important issues treated are: more support in the analysis phase, better description techniques for stimuli history and more rigourous verification algorithms.

3.5 Experiences

3.5.1 SMO experiences

The SMO method is used in a 100 man year project [OS-32] developing a new operating system for a telephone exchange. Since the project is running at the moment no formal results or metrics exist. However, clear improvements have been reported both in quality and productivity.

The use of SMO is one reason for the good results so far. But even more important is the organisational aspects. The project is in many parts organised as a Cleanroom project, which is a prerequisite for a successful SRFV-method like SMO, e.g.:

- More time and resources are allocated to the earlier phases of the project.
- The project is divided into teams according to Cleanroom. Team responsibility is an important factor for all teams.
- Verification procedures are performed in regular intervals by reviews. Each week consists of three days development, one day of preparation for review and one day of review.
- No unit testing is to be performed, time is instead spent in the earlier phases and in verifications.

3.5.2 Experiences with similar techniques

The current project using SMO is not finished, but experiences from other applications of some of the techniques indicate that the ideas of SMO are in the right direction for higher quality software. Some examples are:

- Experience from Ericsson in Norway indicates, two to three times increase in quality when using SDL [Rød]. They measure the amount of faults per line of code from integration and function tests.
- Russell at Northern Telecom showed 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” [Russel91]. The result is based on data collected from eight releases of totally 2,5 million lines of code. Fowler at AT&T have had similar experience [Fowler86].

4. Statistical Usage Testing

Statistical Usage Testing (SUT), [Cobb90, Dyer92, Mills87, Mills88b, Whitt92a, Whitt92b], 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 development made with the SMO method can be tested with Statistical Usage Testing.

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.

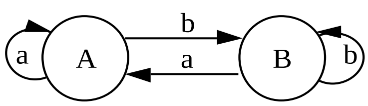
Statistical usage testing consists of two major parts, i.e. usage modelling, which includes construction of a usage profile, and reliability estimation. The adaptations to telecom concern both these parts. This project is being conducted for the Swedish Telecom to provide them with a certification method 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 tells 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. The inter-failure times are collected and applied on a reliability growth model.

4.1 Usage modelling

The original proposal in Cleanroom for modelling the usage is a plain Markov model, [Whitt92a, Whitt92b]. We have encountered that this type of model will soon become too large and complex for large multi-user systems. The problem has been solved by introducing a hierarchical Markov model, presented in [Rune91, Rune92].

4.1.1 Markov model

The usage model is an external description of usage events and the usage profile defines their probabilistic relationship. The usage is modelled as a finite state machine. A simple statistical usage profile example is shown in figure 3.



from/to	A	B
A	0.2	0.8
B	0.9	0.

Figure 3. Statistical usage profile.

The test cases are selected from the statistical usage profile. Starting in an initial user state a transition is chosen by e.g. the Monte Carlo method[2]. The stimulus needed for this transition is recorded. From the new state a new transition is chosen etc. A test case can be made up of multiple state transitions. It can be of random length, or be finished by ending in a termination state.

The test cases are randomly selected with respect to their probability of use and are then a representative subset of the use cases in operation. They are used to represent the operation and, like national polls, are the basis for the prediction of future results.

4.2 State Hierarchy model (SHY)

Based on the conclusion that the existing models are insufficient, a hierarchical Markov model is developed. This model copes with the problems encountered in telecommunication systems. Both the problems encountered and the possible solution is further discussed in [Rune91, Rune92]. Besides that this model is needed for test case generation, it is a mean for easy communication between users and developers and helps understanding the software functionality.

4.2.1 SHY structural model

An example SHY model is shown in figure 4. The levels in the figure can be described as follows:

- The upper level is the Usage Level. It contains one state, which is the main state for selecting the underlying user types.
- On the User Type Level (UTL), above the User Level, a choice between different types or categories of users can be done. This makes it easier to handle large systems.
- On the User Level (UL) the individuals of the user types are shown.
- Each user can use a number of services, which are described on the Service Level. This implies that the usage of each user is described as a set of different services, each of them describing a part of the usage. When adding new functionality it is easy to add new services to a user. This supports modularity and reuse of the usage model parts.
- The Behaviour Level describes the behaviour of the services. Each service is described by a BL state machine.
- A stimulus can be refined by using a Sub-Behaviour Level (SBL) state machine. E.g the stimulus ”digit” can be chosen on the BL and then an SBL choice selects the exact digit, 0 to 9.

[2] Monte Carlo is a method for random sampling. It is used for choosing a transition in the Markov chain. A random decimal between 0 and 1 is chosen from which the appropriate transition is determined.

4.2.2 Hierarchical usage profile

On the Behaviour Level the probabilities for the transitions are recorded like for the plain Markov chain, see figure 3. Every state is given a state weight as well, which reflects the probability for the service in the actual state to generate next stimulus. The upper level probabilities are calculated as the weighted sum of the state weights for the actual states of the underlying levels. This is expounded in [Rune92].

4.2.3 Test case selection

Test cases are selected by traversing the SHY model, see figure 4, controlled by random numbers as discussed in section 4.1.1. First the main state, Usage, is entered from which a selection of a User Type is done. If e.g. User Type 2 is selected, there is only one user and this will hence be drawn on the User Level. One of the services connected to User 4 is drawn, e.g. Service 2, and then a transition in its Behaviour Level state machine. The selected stimulus and its possible influence on other BL-state machines are added to the test script, or if there is a Sub-Behaviour Level connected to the stimulus, a refinement of the stimulus is drawn and it is added to the test script. Then the probabilities are updated and the model can be traversed again.

4.3 Certification

By certification means 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. We have studied the reliability model proposed in Cleanroom, section 4.3.1, and examined another type of criterion for determining whether a product can be accepted or not, the hypothesis testing, section 4.3.2.

4.3.1 Cleanroom software reliability model

The software reliability model in Cleanroom is as follows:

$$MTTF_k = A * B^k \text{, with } k = 0, 1, 2 \ldots.$$

This form is supposed to describe the change in MTTF (Mean Time To Failure) when faults are corrected. The model is discussed in more detail in [Currit86, Dyer92]. The parameters A and B are estimated from the collected failure data. This will make it possible from the equation above to predict future failure occurrences. From the value of MTTF it is possible to calculate the reliability of the software. Thus meaning that based on a required reliability, it is possible to evaluate the software

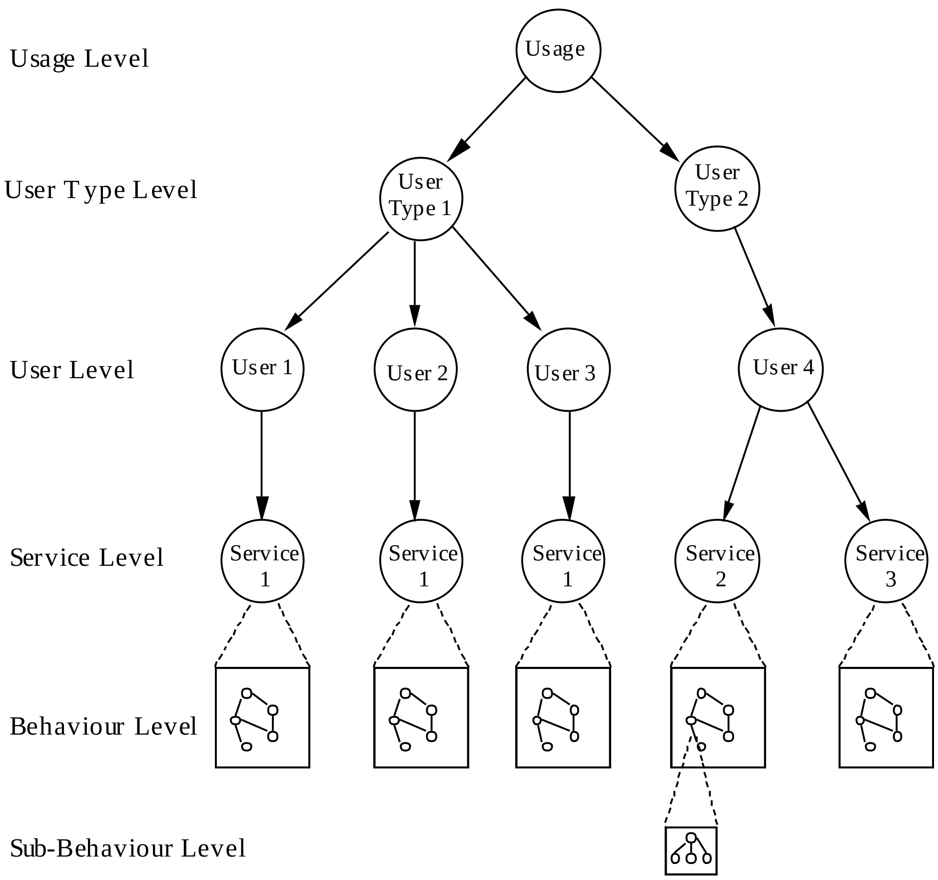


Figure 4. SHY model.

against this requirement.

The Cleanroom model is simple in theory but difficult to apply in practice. It is very sensitive to variations in failure data. If some data values in the beginning are much above the mean value, it clearly overestimates the quality. Another example of problems arises when some low data values occur after a while. Then it will take quite a long time for the estimates to recover. Our conclusion is that the model is not useful for acceptance of software but still it can be useful for prediction of future reliability.

4.3.2 Hypothesis model

As an alternative to the Cleanroom model we have tried a hypothesis testing model. In [Musa87] p. 201–203 a method for reliability demonstration testing is described which is a form of hypothesis testing. A hypothesis is raised and then the testing aims at giving a basis for acceptance or rejection of the hypothesis.
The hypothesis is a failure intensity objective ($1/t_{objective}$). The hypothesis is rejected if the objective is not met with the required probability and accepted if it is. In the interval between the both, the testing has to continue.

The reliability demonstration testing is performed by plotting the measure points in a control chart (see figure 5): failure number (r) towards normalized failure time (t_{norm}). The failure time is normalized by multiplying the failure time by the failure intensity objective.

If the measure points are in the continue region, the testing is continued. When the measure points are in the rejection or the acceptance region, the testing is interrupted and the software is rejected or accepted respectively.

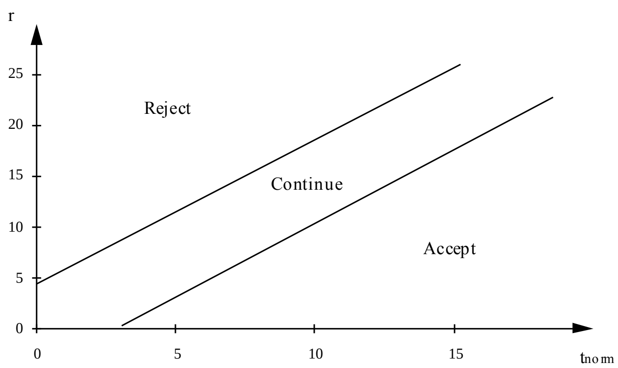


Figure 5. Control chart for reliability demonstration testing.

The control chart is constructed by drawing the acceptance and rejection lines. They are based on the requirements on the probability for acceptance and rejection of the tested product. How they are calculated is described in [Musa87].

As a conclusion can be said that the hypothesis testing model is easy to understand and to use. The hypothesis testing model gives support for decision on acceptance at specified levels of certainty.

4.4 Reliability predictions from analysis

The objective with statistical usage testing can be applied much earlier in the software life cycle. 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 objective of the proposed method is to perform the estimation during some form of analysis during which failures are detected. The approach is based on that the usage 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 [Ek91]. 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 method and its opportunities are discussed in more detail in [Wohlin92].

4.5 A method for SUT in telecom

Using SUT on telecom applications can be summed up in the following method:

- Model the software usage.
- Develop the usage profile.
- Generate test cases.

\+ Outside SUT: Execute test cases and collect inter-failure data.

- Certify the reliability.
- Predict the reliability growth.

5. Cleanroom and dependability

A dependable system is a system on which the user can trust and the basis for trust is the absence of failures. It must be better to design a software system with zero defects than introducing fault tolerant software or having redundancy in the system. In particular, this must be the case when it does not cost more than normal development to apply the Cleanroom Software Engineering methodology. Cleanroom is 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 not free from defects, it is always wise to combine Cleanroom with some form of fault tolerance. The fault tolerance technique to use must be a function of the requirements of the system and the actual fault content. Thus meaning that it may be possible to apply a less complex fault tolerance strategy when applying Cleanroom than otherwise.

Cleanroom produces dependable 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 and maintainable system.

The engineering approach, as in Cleanroom, includes several techniques and it is the sound application of the total concept that makes the software dependable. 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 dependable computing 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 turns software development into an engineering discipline, by the techniques presented above. Hence, Cleanroom will help in the development of dependable systems in the future.

6. Current work

Q-Labs has today a number of on-going Cleanroom activities:

6.1 QCCC at Q-Labs

Q-Labs Cleanroom Competency Centre (QCCC) has been established this year at E-P Telecom Q-Labs, Ideon Research Park in Lund. QCCC has at this date eight members from Q-Labs, including associates from the University of Lund. The goal with QCCC is to:

- collect experience and knowledge from our commercial Cleanroom projects and in that way incrementally increase the Cleanroom competency in QCCC.
- offer adaptations of the Cleanroom methodology to different environments and applications.
- offer education or seminars on different levels, both for technical personnel and management.
- make further development of the Cleanroom methodology.
- have an intense cooperation and communication with SET – one of the companies responsible for the development of Cleanroom.

6.2 Industrial projects

Q-Labs has a number of industrial projects active on adaptations of the different methods in Cleanroom. Two of them (SMO and SUT) are described in section 3 and 4 of this paper. The SMO method developed for Ellemtel is used in a 100 man year project [OS-32] that is reporting improvements both in quality and productivity. The SUT project, performed for the Swedish Telecom, is in its second phase. The first phase consisted mainly of adaptation of the method to the field of telecommunications. The second phase includes in particular a practical application of the proposed methods for acceptance of software products, but also a further refinement of the methods.

6.3 Case study

An internal Cleanroom case study has just been finished. A number of topics have been under study and evaluation, for example the new Cleanroom Process Manual, the use of all Cleanroom methods integrated in one single project and the team approach. We have no metrics at this date on the productivity and quality of the software produced but the indications and expectations are positive. The results from the case study has shown that it requires a lot of discipline to do it the ”right” way and that training is essential to be able to make efficient use of the Cleanroom methods.

7. Conclusions

Cleanroom Software Engineering with its adaptations to the problems encountered in telecommunication systems will be one way to provide dependable systems in the future. The engineering approach emphasized by the techniques within in Cleanroom is a necessity for software development. The most important ones are:

- The organizational aspects.
- Incremental, stepwise refinement with verification.
- Certification based on the usage of the software.

The paper has presented two major adaptations of Cleanroom to cope with problems encountered in large real-time multi-user systems, i.e. a development method and a method for software certification.

7.1 Conclusions development method

The ideas of stepwise refinement and verification are one of the bases for developing zero defect software. Cleanroom emphasises this by using the Box Structures and Stepwise Refinement and Functional Verification (SRFV). The SMO method has shown that it is possible to successfully adapt these ideas to the field of telecommunication.

SMO provides a true specification technique, with emphasis put on the external behaviour. The three different views in the specification are new in system specification. They give us a powerful tool to specify the system correctly.

The method supports stepwise refinement and verification from specification to code by integrating three different description techniques in a SRFV manner.

The possible gains in quality and productivity by Cleanroom is indicated by the project using SMO. The project also shows that the use of Cleanroom organisation ideas are as important as the development method.

7.2 Conclusions certification method

It can be concluded the statistical usage testing provides an opportunity to certify the reliability of the software. Statistical quality control of software is possible using SUT, i.e. an objective measure for acceptance of software products can be obtained.

The method includes a hierarchical Markov model to describe the usage of a telecommunication system. This model overcomes some of the problems encountered when using a plain Markov chain. The developed model is easy to understand and its division into levels lets the user concentrate on one aspect at the time. It is easy to add new parts to the model, which can be useful when a system is extended. The model is an important part in being able to apply SUT to the telecommunications field.

The estimation of software reliability through applying software reliability growth models is difficult. A study has shown that it is difficult to simply apply a model and get a reliable estimate. The solution to the problem is to apply the methods sensible, evaluate the result and possibly combine the objective estimates with the subjective judgements based on experience. The proposal in the method is to apply a hypothesis acceptance criterion and then to apply the software reliability model proposed in Cleanroom to get a prediction of the future reliability growth.

A method for applying SUT in telecommunication has been formulated. Some work remains to be done, but the results and the method can be used and ought to be used. The method ought to be applied to some real projects to be evaluated, improved and adapted to the practical needs. The method is believed to be mature enough to give valuable results already in the first real project.

8. References

[Adams84] Adams, E. N., ”Optimizing Preventive Service of Software Products”, IBM Journal of Research and Development, January 1984, pp. 2-14.

[Belina91] Belina, F., Hogrefe, D. and Sarma, A. ”SDL with Applications from Protocol Specifications”, Prentice-Hall, UK, 1991.

[CCITT88] CCITT, ”Recommendation Z.100: Specification and Description Language, SDL”, Blue book, Volume X.1, 1988.

[CCITT92] CCITT, ”SDL Methodology Guidelines”, Appendix I to Z.100, 1992.

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

[Cosmo91] 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.

[Currit86] Currit, P. A., Dyer, M. and Mills, H. D., ”Certifying the Reliability of Software”, IEEE Transactions on Software Engineering, Vol. SE-12, No. 1, 1986, pp. 3-11.

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

[Ek91]Ek, A. and Ellsberger, J., ”A Dynamic Analysis Tool for SDL”, In ”SDL '91: Evolving Methods”, Editors: O. Færgemand and Ri Reed, pp 119-134, North-Holland, The Netherlands, 1991.

[Fagan76] Fagan, M. E., ”Design and Code Inspections to Reduce Errors in Program Development,”, IBM Systems Journal, No. 3, 1976, pp 182-211.

[Fowler86] Fowler, P. J., ”In-process Inspections of Workproducts at AT&T.”, AT&T Technical Journal, March/April 1986, pp. 106.

[Linger79] Linger, R. C., Mills, H. D. and Witt, B. I., ”Structured Programming Theory and Practice”, Addison-Wesley Publishing Company, 1979.

[Mills86a] Mills, H. D., Linger, R. C. and Hevner, A. R. ”Principles of Information Systems Analysis and Design”, Academic Press Inc. 1986.

[Mills86b] Mills, H. D., ”Structured Programming: Retrospect and Prospect”, IEEE Software, Novemver 1986, pp. 58-66.

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

[Mills88a] Mills, H. D., ”Stepwise Refinement and Verification in Box-structured Systems.”, IEEE Computer, June 1988, pp. 23-36.

[Mills88b] Mills, H. D. and Poore, J. H., ”Bringing Software Under Statistical Quality Control”, Quality Progress, November 1988, pp. 52-55.

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

[Musa90] Musa, J. D. and Everett, W. W., ”Software Reliability Engineering: Technology for the 1990s”, IEEE Software, November 1990, pp. 36-43.

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

[OS-32] Presentation material from the OS-32 project, Ellemtel, Sweden, 1992.

[Rune91] 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.

[Rune92] Runeson, P. and Wohlin, C., ”Usage Modelling: The Basis for Statistical Quality Control”, Proceedings 10th Annual Software Reliability Symposium, Denver, Colorado, USA, pp. 77-84, 1992.

[Russel91] Russell, G. W., ”Experience with Inspection in Ultralarge-scale Developments.”, IEEE Software, Jan 1991, pp. 25-31.

[Rød90] Rød, T., ”Erfaringer - SDL og måltall”, EEN, 900829.

[Whitt92a] 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.

[Whitt92b] 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.

[Wohlin92] Wohlin, C. and Runeson, P., ”A Method Proposal for Early Software Reliability Estimations”, To appear in Proceedings 3rd International Symposium on Software Reliability Engineering, Raleigh, North Carolina, USA, October 1992.