C. Wohlin , B. Regnell, A. Wesslén and H. Cosmo, "User-Centred Software Engineering - A Comprehensive View of Software Development", Proceedings Nordic Seminar on Dependable Computing Systems, pp. 229-240, Lyngby, Denmark, August 1994.

User-Centred Software Engineering – A Comprehensive View of Software Development[1]

Claes Wohlin, Björn Regnell, Anders Wesslén and Henrik Cosmo
Department of Communication Systems, Lund Institute of Technology
Lund University, Box 118, S-221 00 LUND, Sweden
Phone: +46-46-103329, Fax: +46-46-145823, E-mail: claesw@tts.lth.se

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

Abstract

Dependability is foremost a user-centred quality attribute. It is in the interest of the user that the software system is dependable. Therefore, a user-centred approach to software development is argued as it allows for continuous visibility and traceability of requirements for the user. The objective is to work with a framework for user-centred software engineering with the aim to increase software dependability by having the user in focus throughout the life cycle. The concepts used in software development are not normally understandable by the user. Two user-centred concepts are focused upon, i.e Object-Oriented Software Engineering with a use case driven approach and Usage Testing. These techniques give the user concepts to relate to during development. Therefore, the user may influence the software product and its dependability during development, instead of being disappointed as the software is delivered. The framework forms the basis but more research is needed and the objective is to cover this need in the future work.

1. Introduction

User-Centred System Design has been proposed as a method to design user interfaces. The method is briefly described in [Sommerville92]. The objective of the method is to focus on the user needs to ensure that the interface is accepted by the user when the software product is delivered. This view can be enlarged to incorporate other types of software as well. A more user-centred approach to software development will increase user and customer satisfaction. In particular the software will be perceived as more dependable as the procurer, whether company internal or external, will not face any surprises at delivery. A *User-Centred Software Engineering* approach means that the procurer of the software has a better visibility of the software throughout the development than today.

The objective of this paper is to outline a more user-centred development approach and to present two methods that allow for this approach, namely *Object-Oriented Software Engineering* (OOSE) with a *use case driven approach* [Jacobson92] and *usage testing* [Mills88, Runeson93]. The latter approach provides an opportunity to evaluate for example reliability prior to release of the software. The two methods are presented briefly and then a vision of how a User-Centred Software Engineering approach can be achieved is introduced. This vision is the basis for the current research project, entitled “Engineering Reliable Software for Complex Systems”. Some improvements needed in use case driven development and usage testing are current areas of research to support the overall vision. These improvement areas are depicted and briefly described below.

The dependability of a software system is a composite quality attribute and it is a critical one for the procurer and of course for the end user of the software. Therefore, it is necessary to assure that dependability and its constituents are in accordance with the expectations of the users throughout the development. It is not possible to develop complex systems where the procurer signs a requirements specification and then does not see the software until the acceptance test. No silver bullet will be found to cope with dependability problems. Specific methods or techniques may improve

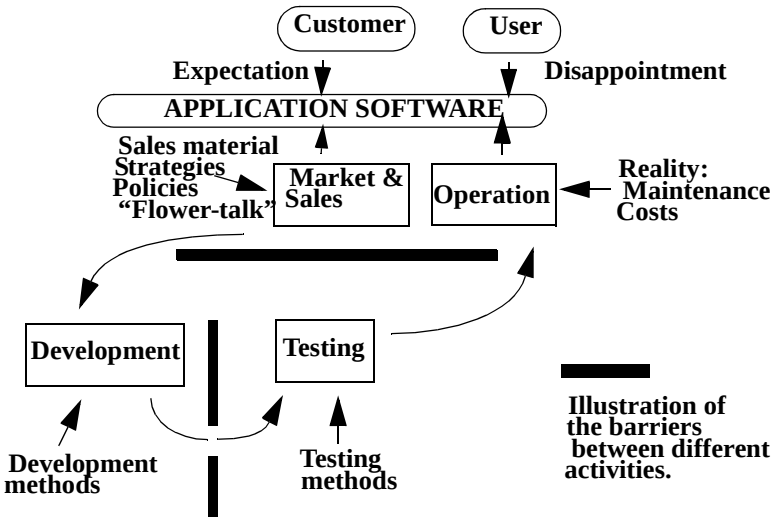

FIGURE 1. A view of the current situation with different roles in the software life cycle.

dependability, but (at least) equally important is to provide a suitable framework to support development of dependable software systems. This conviction is the basis for the work discussed in this paper.

A continuous visibility and evaluation of the software is needed throughout the software life cycle, i.e. a comprehensive view with the procurer and end user in focus must be enforced. Process assessment is a growing area of interest, but it is not enough. The software development methods, for example design and test, must be changed to include concepts that are visible and of interest for the procurer. The concepts must reflect the software system to be delivered to give the procurer a valuable insight in the software to be delivered. This will increase the reliance the procurer has for the software as it is delivered to acceptance test, since the procurer has had the opportunity to influence the implementation as it is developed. Procurer (or user) centred concepts are valuable in the communication with the customer. This type of concepts mean much more to the procurer than abstract data types and object classes. These are valuable concepts, but primarily for the implementor and not the procurer or the end user of the software.

A somewhat pessimistic view on how software development is performed today is discussed in section 2 and in section 3 two different external views of the software system are discussed, first the view of the procurer of the system and then the view of the marketing and sales persons at the software supplier. The external views are first discussed from the view in section 2 before outlining some needs that are not fulfilled in this particular view. In section 4 object-oriented development is discussed, with particular emphasis on a use case driven approach, while section 5 discusses testing with focus on usage testing. Use case driven development and usage testing are combined in section 6 where a vision of User-Centred Software Engineering is presented. In section 7 some research and improvement suggestions are introduced, which aim at achieving the overall objective of having a more user-centred development. Finally in section 8 some conclusions are presented.

2. Current situation

The situation in software development of today is well-known with frequent schedule and cost overruns and poor dependability, which of course is unacceptable. The customer expects a specific software as it is ordered, but too often gets disappointed as the software is delivered, due to poor visibility and traceability of the requirements throughout the development and testing of the software. The procurer of the software formulate requirements, but normally also has some expectation which has not been put into the requirement specification. The objective of the supplier must be to fulfil both the written requirements and the expectations of the procurer. Therefore it is not only positive for the procurer if visibility and traceability of the requirements are improved, but also for the software supplier who will be in better control and hence can be more confident that the procurer will accept the software as it is delivered.

In figure 1 the current situation in software development is illustrated from a slightly pessimistic view, but unfortunately not uncommon.

The customer wants some application software and is in contact with the market and sales department at the supplier. The market and sales department presents sales material and nice talk about the company and too often has a poor understanding of the technical problems with a software product. Therefore, the department may make promises which will be hard to fulfil. This is a result of the barriers between different departments and activities. The barriers may be a result of either poor communication or poor understanding between market and sales and the development department.

The department responsible for the development applies their methods for specification, design and implementation. These methods primarily supports the needs of the developers and they are not adjusted to the needs of the customer or not even the test department within the same company. The test department is often seen as an unwanted necessity and they are responsible for removing faults introduced during the development. The time too perform adequate testing is often very limited due too delays and time constraints. As a result, the software delivered to the operational phase is not dependable and it will be costly to maintain. The problem of the quality of the software is left to the maintenance department, which is obvious as the relative cost of maintenance is studied, see for example [Sommerville92].

The quality of the delivered software often leads to a disappointed procurer or user, which of course is unsatisfactory for all parts involved. The depicted situation must be improved, which is discussed further below.

3. External view

3.1 Procurer view

The procurer of a software system is mostly not an expert in software engineering hence the procurer tends to be incapable of understanding the development and testing of the software as the concepts used are unfamiliar. Many software procurers have observed this problem and assessments of suppliers are increasing as well as requiring that software is bought from certified suppliers. This is a step in the right direction, but the procurers of software systems must be supported throughout the software life cycle. Some work is performed to launch metrics programmes to support procurers, but this is not enough because it means an acceptance that the procurer can not understand the actual development process. New concepts must be introduced to support the communication between suppliers and procurers to obtain dependable computer systems in operation.

The concepts used in the software life cycle must mean something to the procurer. Therefore, it is necessary to introduce user-centred concepts throughout the life cycle. The concepts are available, but they have not been put together to form a comprehensive way of developing software and the formalization of the concepts must be improved as well to be useful. The introduction of user-centred concepts are beneficial both for suppliers and procurers because it removes some of the uncertainty as the software shall be delivered and it produces probably more dependable software. Thus a win-win situation with a satisfied procurer and supplier occurs.

3.2 Market and sales view

The software product is marketed and sold by persons mainly responsible for contacts with the customers in the initial stage of the product life cycle, even if they have contact with the customer until full payment has been obtained. The persons responsible for market and sales are mostly not experts in the system actually being developed. This situation often means that software systems are sold with unrealistic expectations set on them.

The type of concepts proposed here would improve not only the communication with procurers, but also between different departments within a company. This is particular the case when technical experts communicate with non-technical experts, for example development department with market and sales department. An improved internal communication is essential since it supports a better understanding of technical problems hence allowing for realistic promises to procurers. The latter will of course lead to satisfied customers.

4. Development view

4.1 General

The central problem in software development is how to transform an *informal* idea about the system to be built, represented by some requirements specification, to a *formal* description of the system that can be executed with the desired result. A development process model is a description of how this work is done. (The term *development* is used for denoting the entire fabrication of a software system, and *construction* is used for denoting the (sub)activities *design* and *implementation*.)

There exists a number of different approaches to development process modelling, e.g. Waterfall Model, Explorative Development, Spiral Model [Boehm88, Sommerville92]. In all such process models the following activities are represented in one way or another:

- Specification
- Analysis
- Construction (Design & Implementation)
- Verification & Validation

These activities may be carried out in sequence or in parallel. They are often controlled by some underlying principle or philosophy, e.g. structured programming, top-down or bottom up development, functional decomposition, or object orientation. These so called paradigms are not all exclusive and can in principle be used together in different activities during the development process. However, the following two sections concentrate on the object-oriented approach, as it is very well suited for the design of large and complex systems, giving understandable and changeable models of the system.

4.2 Object-Oriented Development

The main idea of object orientation, which is well-known and presented in for example [Jacobson92], is to group data and operations into objects in a way motivated by the problem domain. The system is viewed as a model of the problem domain, and object orientation provides a number of powerful concepts for modelling purposes.

An *object* is composed of operations and information saved in a state. The operations can change the state of the object. An operation is issued by sending *stimuli* to an object, and the sender of a stimulus may receive a *response* as a result of the operation. The behaviour and information are *encapsulated* in the object, thus supporting the concept of information hiding. *Aggregation* or composition is used to model parts-whole relations.

A *class* is a template that defines a set of objects in terms of their internal structure. An *instance* is an object created from a class. An instance is an object that belongs to a certain class, hence object and instance are used as synonyms. Classes can form a classification hierarchy for modelling generalisation and specialisation by the use of *inheritance*.

By the use of virtual operations an inheritance structure can offer *polymorphism*. This is used for dynamic binding of types to objects by specifying the specialization of the object at runtime. The operation actually performed when a stimulus is sent depends on the dynamic type of the object.

Object-oriented development has a number of necessary activities:

- **Object Identification** - The different objects in the system must be identified. This activity involves analysis of the requirements and the problem domain.
- **Object Organization** - The organization of the objects, with respect to for example inheritance and aggregation, must be determined.
- **Object Interaction** - How objects communicate must be defined in terms of how stimuli are sent between objects.
- **Object Operations** - The way objects respond to a stimulus is defined by specifying the operations.
- **Object Implementation** - The object is implemented using results from other activities.
- **Object Verification & Validation** - Finally, the objects must be verified and validated, to be sure that they are correct and correspond to the requirements.

4.3 Use-Case Driven Approach

In OOSE [Jacobson92] the concept of use cases is central in the development of software systems. A *use case* is a flow of events in the system connected to a demarcated operation issued by a user type, an *actor.* The Use Case Model created early in the analysis process serves as a basis for all subsequently devel-

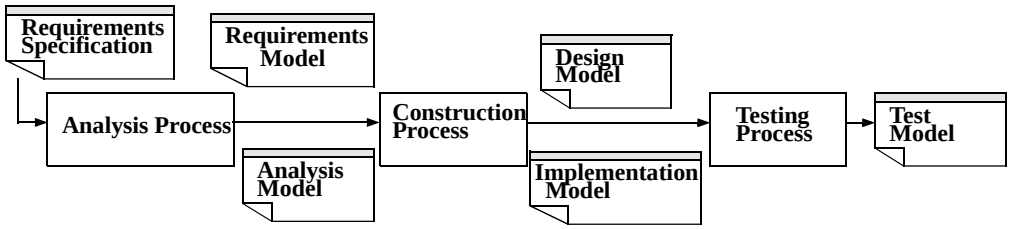

FIGURE 2. OOSE Processes and models.

oped models of the system. The use case driven approach provides *traceability* and *seamless transitions* between the different models.

OOSE consists of a number of processes which results in different system models, each capturing a specific aspect of the system. The processes and models are shown in figure 2. The different processes in OOSE are supposed to be carried out in parallel. The different models may consist of several (sub)models with their special syntax and semantics. For instance, the design model make use of so called interaction diagrams.

There is an important distinction between the analysis process and the construction process: During analysis the system is viewed from an ideal point of view, with no regard to a specific implementation environment, not taking into account the restrictions imposed by a non-ideal environment. The effect of the specific non-ideal implementation environment is not considered until construction.

The Use Case Model is part of the Requirements Model and is the basis of every other model of the system. The Use Case Model is used in different ways to aid the construction of the other models, as shown in figure 3.

The use cases are initially described in natural language. The use cases can be interrelated by the relations *uses* and *extends*. The uses-relation is used to describe common parts of many use cases. The extend-relation is used to describe exceptional events and error-handling.

OOSE and the use case driven approach is a good platform for object oriented development of large and complex software systems. Some areas in OOSE, however, needs improvement and further refinement. Examples of such areas are *formalization* of the use case concept and models, and combining OOSE with recent development in *verification and validation*. The research and improvement areas are discussed in more detail in section 7.

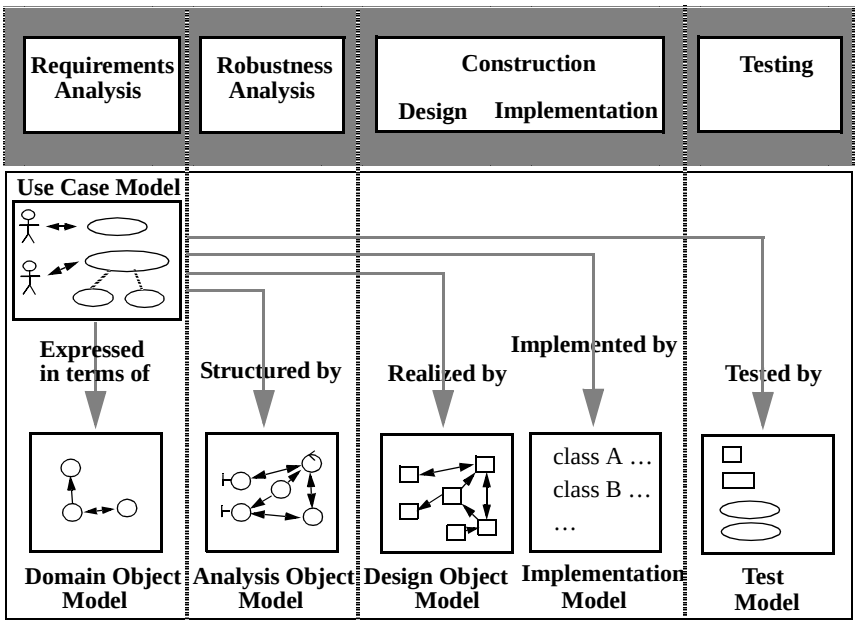

FIGURE 3. Use case driven development.

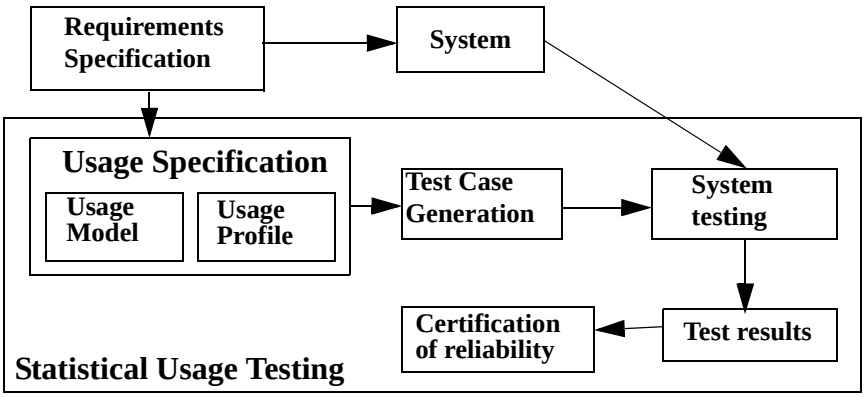

FIGURE 4. The procedure of Statistical Usage Testing.

5. Testing view

5.1 General

Before a software system is delivered to the procurer, the software must be tested. The testing includes both verification and validation. Testing of the software can be made at different times in the development of the system. In conventional software development many different types of tests are made. The different types of testing can be module, integration, system, and acceptance testing. Module testing is a process of testing the individual subprograms, subroutines or procedures in a program. Integration testing concerns the cooperation between functions, modules and subsystems. System testing is a test of how the system fulfils its intended objectives and acceptance testing is the test when the procurer tests the system to evaluate if it meets the initial requirements.

The objective of the testing of today is often to find as many faults in the design and implementation as possible and this is reflected in the methods that is used to derive test cases. To find faults, the test cases can cover all paths in the software, this method is called exhaustive path testing, or the test cases can cover the most used paths in the software. The most used paths are derived from how the software is used and by experience.

Another way of dividing the testing methods is to divide them in functional, or structural testing. Functional testing is when the specification of the component is used to derive test cases. How the system is designed or implemented are of no interest for the tester. The component is seen as a black box and the only thing that is of any interest is how the component communicates with its environment. Functional testing is to test the component against its specification. Structural testing is when the tester uses the detailed knowledge of the design and implementation to derive the test cases. Structural testing tests the component's code against its design.

Finally, random testing is another way of testing the system. One type of random testing is to try to generate test cases according to the anticipated usage of the system. This type of testing is often referred to as Statistical Usage Testing and it is described subsequently.

5.2 Statistical Usage Testing

5.2.1 Objective

Statistical Usage Testing, SUT, is the certification part of *Cleanroom Software Engineering*, [Mills87, Mills88]. The basis for the testing is the specification of the software and not its implementation and hence it is a black box technique. The objectives of SUT are to find the faults that influence the reliability the most and that the testing produces data which makes it possible to certify and predict the software's reliability. If the work is concentrated on the faults that influence the reliability the most it becomes more cost effectiveness than conventional testing, [Musa93]. The test cases are statistically generated from the expected usage and therefore the test results can be used to certify the reliability of the software with statistical confidence, whereas the conventional testing provides no such confidence, [Currit86].

5.2.2 Overview

The procedure of Statistical Usage Testing is described in figure 4. From the requirements specification a *usage specification* is produced, which consists of two

parts, a *usage model* and a *usage profile*. From the usage specification *test cases* are generated. The testing produces test results and those results are used to certify the reliability of the software. The construction of the system is made in parallel of the generation of the test cases.

5.2.3 Usage specification

In SUT the basis for the testing is the expected usage by the future users of the system. The expected usage must be analysed and form a part in the requirements specification. The usage is described in the usage specification, which is made from the analysis of the usage. The usage specification consists of two parts, the usage profile and the usage model. Markov chains have been proposed as a suitable technique to model the usage, [Whittaker92, Runeson92].

A usage model is the structural part of the usage specification and consists of states and arcs between the states. A usage profile is the statistical part of the usage specification and is the transition probabilities to the usage model. A *stimulus* is the event that the user produces to stimulate the system. A stimulus is connected to a transition in the usage specification.

A test case is a flow of user stimuli, together with the expected answers from the system. The user stimuli are used to stimulate the system and the expected answers are used to evaluate if the system responds correctly to the stimuli.

Test cases are generated from the usage specification. A test case is generated by starting in a predefined state in the usage model and then by making a transition by random from the usage profile. The stimulus connected to that transition is recorded and analysed and the expected answer from the system is also recorded. The procedure is repeated from the new state. The test case ends when it has a correct length or when a specified state is reached.

5.2.4 Certifying the reliability

In Statistical Usage Testing one objective is that it must be possible to certify the software's reliability. A software reliability model is needed to fulfil this objective. The reliability model can then estimate or predict the reliability of the software. One reliability model, which is based on the time between failures, is described in [Currit86]. Failure data, which is part of the test results, is the normal type of input data to a software reliability model.

5.2.5 Advantages

The following advantages are obtained when using Statistical Usage Testing:

- The software is tested against the expected use of the system.
- The work is concentrated on those faults that influence the reliability the most, i.e. the testing is more cost effective.
- No knowledge of design or implementation is required.
- Reliability models can be applied on test results from Statistical Usage Testing.
- Evaluation of software reliability requirements.
- The test cases can be developed in parallel with software development.
- The software can be tested with a usage that is different from the expected, i.e. the usage profile can be changed.
- The procurer can easily follow how the testing is performed.

6. Vision: User-Centred Software Engineering

The objective of the research project and the vision is to provide formalized concepts and solutions which support the user-centred approach hence providing techniques to develop dependable software systems. The basis for the research is:

- Object-Oriented Software Engineering, [Jacobson92]
- Statistical Usage Testing, [Mills88, Runeson93]
- Cleanroom Software Engineering, [Mills87, Cosmo93]
- Service Creation and Interaction with particular emphasis on complex systems, for example telecommunication systems, [Morgan91, Kimbler93]

In figure 5 the framework of the vision is illustrated.

The introduction of Object-Oriented Software Engineering with a use case driven approach means that a concept which relates to the user is introduced in the development phase. Usage testing is recommended for the testing phase since the objective of usage testing is to resemble the anticipated usage in operation hence certifying the reliability of the software prior to release. The use case driven approach and the usage

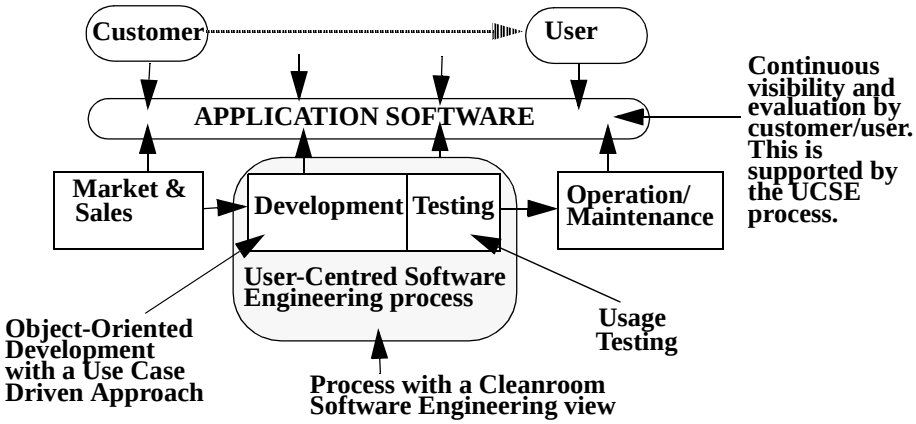

FIGURE 5. A vision of User-Centred Software Engineering

testing approach are to be based on the same major principle as in Cleanroom Software Engineering, i.e. “zero defect software can be developed”. The user-centred process is influenced by principles and methods packaged within Cleanroom.

The objective is to propose techniques and methods for a use case driven development approach and usage testing, hence increasing visibility and traceability of the software requirements. This also indicates that the User-Centred Software Engineering approach gives customers and users a continuous view of the software ordered from the development and testing departments, which is indicated in figure 5 by smoother transitions between different phases in the software life cycle. The customer may be either external or internal.

7. Research and improvement areas

7.1 Introduction

The objective is to propose techniques and methods to support the overall vision presented in the previous section. The aim is not to formulate a fully specified process or a new development method. The objective is that the proposed concepts can be implemented in existing processes, hence making them more beneficial for the industry. It is difficult to adopt completely new processes, but if the proposed solutions can be implemented in existing processes and improvements can be achieved, then the technology transfer from academia to industrial practice will be simplified considerably. The latter is an important issue as the objective is to make the research relevant for industrial use.

The way to achieve the objective is further outlined in figure 6 and some important areas for research and hence improvements are briefly described subsequently.

The informal requirements from the procurer must be formalized to provide an adequate basis for software development and testing. By introducing a *formal use case specification* this can be achieved. The objective must be to make the formal use case specification understandable to the procurer. The common basis, i.e. the formal use case specification, is a prerequisite in development and testing otherwise there is a substantial risk that the software and the test cases are developed based on different understanding of the informal requirements of the procurer. The formal use case specification is input to both the *use case driven development process* and the *usage testing process*. The division into two processes must not be confused with the representation of the test process and test model in figure 2, where the test process refers to the testing in the development process and the test model refers to the test cases with test results. Testing is treated very briefly in Object-Oriented Software Engineering as described in [Jacobson92], which motivates combining it with usage testing as proposed here.

The common basis also means that a continuous communication between the development process and the testing process is supported. The procurer is capable of understanding the processes and the results since the user focus is maintained throughout development and testing. Other concepts are of course not ruled out. It is not possible to develop and test a system based on only one concept, but the user-centred concept must be one of the concepts used.

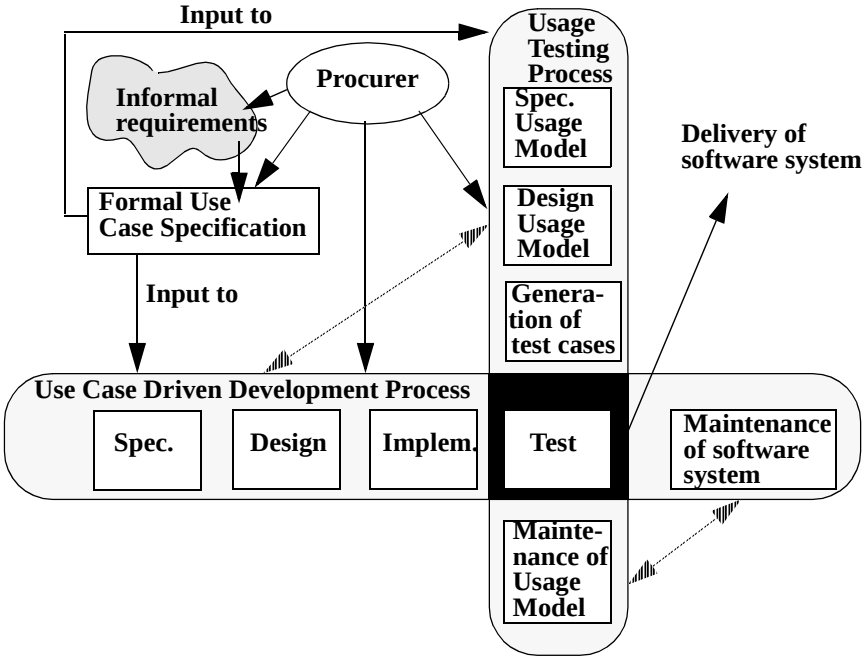

FIGURE 6. A User-Centred approach.

The development process and the testing process intercept at the test phase, where the developed software must be tested from a usage perspective to certify that the quality (in particular reliability) fulfils the requirements stated.

The software must after delivery be maintained for future releases, but it is also important to notice that the objective is to maintain the usage model developed for testing the software as well. The objective is to add new parts to the usage model in a similar way as new services must be addable to the software implementation.

This approach means that several areas must be improved, hence the research is directed toward these. The main areas of research identified so far are described in the following sections.

7.2 Formalization of use cases

One very important research area is formalization of the concept of use cases and the formalization of models of use cases. This regards both specifying use cases, and applying a use case driven approach to development.

The idea of a development process centred around a model of the system's functionality as viewed by the user, is very appealing, as such a process is possible to make visible to the user. However, if use cases are to be a solid basis for the entire development, the concept has to be formalized. Before an appropriate formalization can take place, the demands on such a formalization has to be defined.

In our vision, a *formal use case specifications model* serves as a basis for all other models, implying that there are many different requirements, possibly contradictory, on what this model should be able to supply. The testing process and the development process are likely to have different demands on the model. A formalization of the use case specifications model has to take these demands into account.

When concentrating on formalization of the use case concept and modelling, it is fruitful to start by identifying problem areas of use cases today. The use case concept is both problematic and insufficient. The following areas are included in the research:

- Formalization of the *definition* of the use case concept
- A more formal approach to the *identification* of use cases
- Investigation of use case *structures* and their relations
- Investigations on *granularity* of use cases with respect to size and complexity
- Extending use cases with an invocation *context*
- *Visualization* of use cases

- *Traceability* from requirements to use case specification and object implementation

7.3 Usage model development based on formal use case specifications

Object-Oriented Software Engineering and Usage Testing have each of them a concept for the usage of the system. Use cases are discussed in Object-Oriented Software Engineering while a usage model is depicted as important in Usage Testing. These two concepts are similar but are used in different ways. If the development of the system and the testing shall have the same basis, only one concept is preferred to make it easy for the procurer to understand the process.

If the use case concept is formalized, it can form the basis for both object-oriented development and statistical usage testing. To use the formal use case concept in statistical usage testing there has to be methods for building usage models from use cases.

7.4 Object-oriented development and usage testing of components

Formal development is one basis for the low defect software produced with the Cleanroom approach. Today OOSE, [Jacobson92], does not support formal specification of components. This is not satisfying and therefore the following research is needed:

- Identification of components from the Use Case Specification
- Specification and development of software system´s components

Object orientation is also depicted as a suitable development strategy to develop reusable components, but a system can not be composed of components from a repository without having a certified reliability. Therefore, it is important to develop methods to certify reliability of components and also to formulate methods to derive system reliability from the reliability of the system components. The research includes:

- Usage testing and reliability certification of software components
- Approaches to derive system reliability from the components of the system

Some results have been presented within this problem domain [Poore93, Wohlin94a], but many problems remain to be solved.

7.5 Usage evaluation from a life cycle perspective

Above, usage testing has been described as a suitable technique to allow for a better evaluation of the software prior to the operational phase, but the concept can be applied throughout the life cycle. A method proposal for usage analysis of a software design has been proposed in [Wohlin92] and the objective of the research is to develop this approach further. The aim is also to develop a simulation procedure based on the usage testing concept. Some preliminary results are presented in [Wohlin94b]. The usage testing process illustrated in figure 6 must hence be enlarged to incorporate more interception points with the development process.

The research must include:

- Formulation of a usage evaluation process, which enlarges the approach presented in figure 6. Some findings are presented in [Wesslén94].
- Further development of the method for usage analysis, which allows for a first reliability evaluation during analysis of the software design.
- Formulation of a usage simulation method, which can either be a functional simulation or it may include real time aspects which means that capacity issues can be evaluated as well.

These research areas must be combined with usage testing results to obtain a comprehensive usage evaluation method which covers most of the life cycle.

7.6 System re-certification without re-testing

A major problem in reliability certification is that the certified value is only valid for the applied usage profile. The reliability estimate may be considered trustworthy for minor deviations from the applied profile, but there is no guarantee, and it is not relevant for major changes in the profile. It is, however, not cost-effective to re-test the software as the usage profile changes or a new service is included in the system. Instead it is favourable if it is possible to re-calculate the system reliability based on a prior certification and the known changes.

Some preliminary findings have been presented in [Wohlin93], but more research is needed, as for example:

- Development of a method for sensitivity analysis of the reliability estimate as the usage profile changes
- Formulation of a method to re-certify the system reliability based on a calculation procedure instead of re-testing the software

The development of a calculation procedure to re-certify a software system can probably be enlarged to incorporate reliability certification from other test procedures than usage testing. The latter is still most cost-effective, but an organisation may choose to use a calculation procedure instead of applying usage testing, hence making the procedure more generally applicable. An outline of a method is presented in [Wohlin94c]

7.7 Cleanroom and process improvements

In figure 5, it was depicted that the user-centred process approach should be influenced by the principles in Cleanroom Software Engineering. Therefore, an important research area is to improve the proposals and methods in Cleanroom as well as to investigate the user-centred development process. Some suggestions concerning improvements to Cleanroom have been presented in [Wohlin93] and [Wohlin94d].

7.8 Incremental development

Incremental development is one of the basic concepts proposed within Cleanroom to produce low defect software. Incremental development as defined in [Mills87] requires that a specification of the total system and one specification for each increment is done. When using this approach, the specification and verification of the increments are very time consuming since each increment has to be specified and verified separately. In addition to this the specifications of each increment must when being appended together be equivalent to the specification of the total system.

Incremental development is very appealing when requirements change a lot, which is almost a rule in software development. When implementing a changed requirement, developers are supported by good traceability from requirement to specification down to design and implementation.

The following items have to be researched:

- A Use Case Specification that can act both as a specification of the total system and as a specification of each increment
- Identification of a closer relationship between requirements and its specification
- Identification of a closer relationship between specification and its components
- Service interaction and incremental development

8. Conclusions

Software dependability is a life cycle commitment and it can not be achieved with a single method or technique. Therefore, a comprehensive view of software development is needed. Furthermore, it must be focused upon customer requirements and expectation. The notion of User-Centred Software Engineering is introduced to cover this need, hence user-centred concepts are requested throughout the life cycle. The key issues are customer visibility of the software throughout the life cycle and traceability of requirements.

Two techniques are available which provide user-centred concepts, i.e. Object-Oriented Software Engineering with a use case driven approach and Usage Testing. The objective of the on-going research project is to combine these concepts and improve them to support the vision of User-Centred Software Engineering. The aim of the approach is to remove some of the barriers and misunderstanding between suppliers and procurers, both company internal and external, as well as between different departments in the development organisation.

The focus in the research is on improvements and suggestions that can be combined with existing development processes and not to formulate new processes and complete development methodologies. Several research areas where improvements are needed have been presented and it is believed that these improvements will enhance the dependability of software systems in the future.

9. References

[Boehm88] Boehm, B. “A Spiral Model of Software Development and Enhancement”, IEEE Computer, pp. 61-72, May 1988.

[Cosmo93] Cosmo, Henrik, Johansson, Erik, Runeson, Per, Sixtensson, Anders and Wohlin, Claes, “Cleanroom

Software Engineering in Telecommunication Applications", Proc. Software Engineering and its Applications, pp. 369-378, Paris, 1993.

[Currit86] Currit, P. Allen, Dyer, Michael and Mills, Harlan D., "Certifying the Reliability of Software", IEEE Trans. on Software Engineering, Vol. SE-12, No 1, pp 3-11, 1986.

[Jacobson92] Jacobson, Ivar and et al., "Object-Oriented Software Engineering – A Use Case Driven Approach", Addison-Wesley Publishing Company and ACM Press, ISBN 0-201-54435-0, 1992.

[Kimbler93] Kimbler, Kristofer and Söbirk, Daniel, "Use Case Driven Analysis of Feature Interactions", Feature Interaction Workshop, Amsterdam, 1994.

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

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

[Morgan91] Morgan, Michael J., Cosky, Michael J., Gruenenfelder, Thomas M., Curtis Holmes Jr., T., and Raack, Gerald A., "Service Creation Technologies for the Intelligent Network", AT & T Technical Journal, pp. 58-71, Summer 1991.

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

[Poore93] Poore, J. H., Mills, Harlan D., and Mutchler, David, "Planning and Certifying Software System Reliability", IEEE Software, pp. 88-99, January 1993.

[Runeson92] Runeson, Per and Wohlin, Claes, "Usage Modelling: The Basis for Statistical Quality Control", Proc. 10th Annual Software Reliability Symposium, pp. 77-84, Denver, Colorado, USA, 1992.

[Runeson93] Runeson, Per and Wohlin, Claes, "Statistical Usage Testing for Software Reliability Certification and Control", Proc. 1st European In. Conf. on Software Testing, Analysis and Review (EuroSTAR), pp. 309-232, London, UK, 1993.

[Sommerville92] Sommerville, Ian, "Software Engineering", Addison-Wesley Publishing Company, ISBN 0-201-56529-3, 1992.

[Wesslén94] Wesslén, Anders and Wohlin, Claes, "Usage Modelling and Generation of Validation and Verification Cases", Technical report, Dept. of Communication Systems, Lund, Sweden, 1994.

[Whittaker93] Whittaker, James A., and Poore, J. H., "Markov Analysis of Software Specifications", ACM Trans. on Software Engineering Methodology, Vol. 2, No. 1, pp. 93–106, 1993.

[Wohlin92] Wohlin, Claes, and Runeson, Per, "A Method Proposal for Early Software Reliability Estimations", Proc. 3rd Int. Symposium on Software Reliability Engineering, pp. 156-163, Raleigh, North Carolina, USA, 1992.

[Wohlin93] Wohlin, Claes, "Engineering Reliable Software", Proceedings 4th Int. Symposium on Software Reliability Engineering, pp. 36-44, Denver, USA, 1993.

[Wohlin94a] Wohlin, Claes and Runeson, Per, "Certification of Software Components", IEEE Trans. on Software Engineering, Vol. 20, No. 6, 1994.

[Wohlin94b] Wohlin, Claes, "Evaluation of Software Quality Attributes during Software Design", Informatica, Vol. 18, No. 1, pp. 55-70, 1994.

[Wohlin94c] Wohlin, Claes, "Re-Certification of Software Reliability without Re-Testing", Submitted to First IFIP/ SQI Int. Conf. on Software Quality and Productivity, Hongkong, 1994.

[Wohlin94d] Wohlin, Claes, "Managing Software Quality through Incremental Development and Certification", Proc. Software Quality Management, Edinburgh, Scotland, UK, July 1994.