Reliability Certification of Software Components

Claes Wohlin and Björn Regnell
Department of Communication Systems
Lund Institute of Technology, Lund University
Box 118, SE-221 00 Lund, Sweden
E-mail: (claesw, bjornr)@tts.lth.se

Abstract

Reuse is pin-pointed as a key factor to improve productivity and reliability of software systems. Verification and validation of software components and the resulting system is important for reuse to be beneficial on a broad industrial basis. This paper suggests a modelling approach which is suitable for reliability certification of modular systems. It discusses a general reliability certification procedure and provides guidelines and opportunities for how to certify software components and also presents some alternatives for certification of modular systems. We conclude that to create reliable systems, we must start certifying the individual constituents of the systems.

1: Introduction

Reuse, modularization, and certification of software are three areas given a lot of attention in the software engineering community today [1]. This paper focuses on the importance of combining these three areas. Object-oriented techniques, or at least modular techniques, are pin-pointed as being suitable for reuse, but it must be remembered that components often are not reused if their reliability cannot be guaranteed. Therefore it is essential to realize that reliability certification is a must when discussing reuse. We use the term certification to distinguish reliability certification from verification and validation in general.

The term component is used throughout this paper as a generic entity which can be certified and reused. Components to be retrieved from a repository must have a quality stamp in terms of what level of reliability can be expected from them as they are put into a system. The reliability must reflect the intended usage of the component, i.e. a component may be viewed as being reliable for one user and unreliable for another depending on the intended usage of the component.

This paper focuses on the certification of software reliability for a modular system. Reuse in general and object orientation in particular are discussed frequently, but few of these discussions highlight the problem of actually assigning a reliability measure to the components being put into the repository. This paper focuses on techniques available for certifying both components as well as systems. The certification process includes two major activities: usage specification (consisting of a usage model and usage profiles) and certification procedure using a reliability model.

This paper provides insight into the reliability certification domain to support others wishing to evaluate and validate the models and methods presented in this paper. At this stage of the research, no validation project has been run, but we hope that through presenting the ideas, we can encourage others to experiment and thus evaluate and validate the proposals. The main objective is to structure the area and highlight some of the problems related to certification of software components.

Section 2 gives an introduction to usage-based testing in general, and Section 3 provides a structuring of the area in terms of key concepts and their relations. In Section 4, some possible ways of addressing component and system certification are presented. The section includes a classification of components and identifies different ways to certify a system. Section 5 outlines some further research needed, and finally some conclusions are presented in Section 6.

2: Usage-Based Testing

No specific test methods are prescribed by current object-oriented/based development methods. The problem domain of testing object-oriented systems [2] has just recently become of major interest, as it has been realized that object orientation in itself is not sufficient to create high quality software [3]. Two different major alternatives exist: black-box testing and white-box testing. Black-box testing techniques take an external view of the system, and test cases are generated without knowledge of the interior of the system. White-box testing techniques aim at covering paths in the code or all lines in the code or maximising some other coverage measure. The main objective of most testing techniques is to validate that the system fulfills the requirements, the focus is mostly on functional requirements. But, test cases can also address quality issues: for example, they can either be derived to locate as many faults as possible or to certify the reliability level of the software.

Reliability certification of software focuses on detecting the faults that cause the most frequent failures, hence maximising the growth in reliability. The focus in this paper is on black box testing and, in particular, on statistical usage-based testing, which can be used to certify a particular reliability level and, of course, to validate the functional requirements as well.

The ability to certify software during testing is based on a user-oriented approach. This requires a model of the anticipated usage of the software and quantification of the expected usage as the software is released. Several approaches have been investigated and used in this area, for example, Musa advocates operational profile testing [4], several authors discuss random testing based on the operational profile as part of Cleanroom Software Engineering [5, 6, 7]; and Runeson and Wohlin present an approach with user-state dependent random testing based on the operational profile [8, 9]. Musa reports on considerable gains in reducing project cost by using an operational profile to control testing [4] and Mills et. al. also provide figures indicating major improvements [5].

A usage model for a system should in some way include modularization. The model described by Runeson and Wohlin [9] is introduced to cope with a large modularized system, which in particular means it is suitable for components and reuse. Some preliminary findings concerning certification of software components are discussed by Wohlin and Runeson [10]. They are also included in [1], where in particular the mapping between system components and usage specification parts is identified as a key aspect to allow for component certification. This finding is the key to successful certification of systems, see Section 4.2. Subsequently the term **certification component** is used to denote the entity to be certified, which can be either an object in a system (sub-system, class, a collection of classes) or an entire system.

3: Reliability Requirements Specification with Usage Profiles

3.1: Introduction

Non-functional requirements are an essential part of requirements specifications. In particular, the reliability requirements are often regarded as one of the most important non-functional requirements. These cannot be formulated as a single figure (e.g. probability for failure or mean time between failures), since more information is needed. It is necessary to take the anticipated usage into account as the reliability of the system is dependent on the usage; the usage for which the requirement is valid must be stated at the time of formulating the requirements of the system.

In the requirements specification, it should be stated how the requirement is to be validated during system testing. This must be written clearly in the specification to avoid misunderstandings when the reliability requirements are validated. Usage-based testing with reliability certification as a means for validating reliability requirements is a quite new technique and it may not be accepted by all suppliers of software systems unless clearly stated in the requirements specification. Furthermore, usage-based testing allows for functional validation and reliability certification at the same time.

3.2: Concepts in usage-based testing

A system consists of a number of services provided to the system users. These services are implemented by components. To enable certification of these components a framework for specifying usage and for enabling reliability certification of components that are parts of a system, as well as certification of an entire system, is needed. In Figure 1 parts of the framework are introduced. Figure 1 illustrates the relations between target system and environment concepts. These concepts are the basis for creating a specification of the usage of the certification component and also for quantifying the anticipated usage. The specification is further discussed below and shown in Figure 2.

The concepts in Figure 1 can be defined as follows. The software to be certified is referred to as the **certification component**. A certification component has a **reliability**, which is the probability that the component works as intended for a specified time and specified

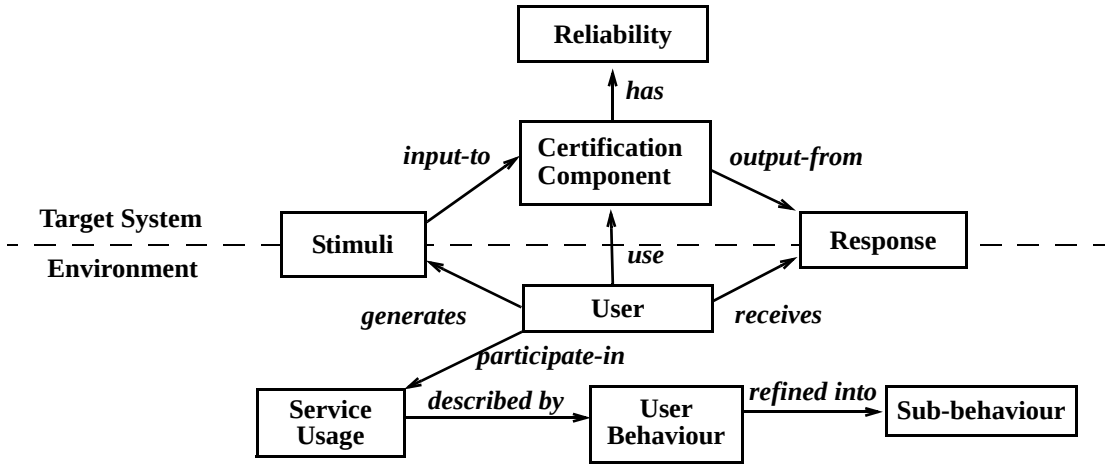

Figure 1. Certification concepts and their relationships.

environment. The certification component is used by one or many users, which can be either human users or other systems. The communication between the user in the **environment** and the **target system** is made through **stimuli** generated by the user and **responses** sent by the system. The user of the certification component participates in **service usage**, which is described by the **behaviour** of the user when using a specific service. The behaviour can, when needed, be refined into a **sub-behaviour**.

To enable certification, the environment must be modelled to allow for generation of test cases, which resemble the anticipated behaviour in the operational phase. Thus, modelling concepts capturing the environment are needed. Depending on the type of testing being applied, different test models have to be derived. The focus here is solely on usage-based testing, which leads to the following key definitions:

- The **test model** is a **usage specification**, although other aspects, such as criticality, may be of interest, see Figure 2.
- The usage specification consists of a **usage model**, which describes the possible behaviour of the users, and the **usage profile**, which quantifies the actual usage in terms of the probability of different user behaviour.
- The usage model is described through user **states** and **transitions** between user states.
- The usage profile is divided into a **hierarchical profile**, which describes the probabilities for choosing one specific user in the environment, and the **individual profile**, which models the behaviour of a single user, while using the available services.

The usage specification may in part be input to the development as well, although the usage profile is mainly aimed at the testing (usage-based testing).

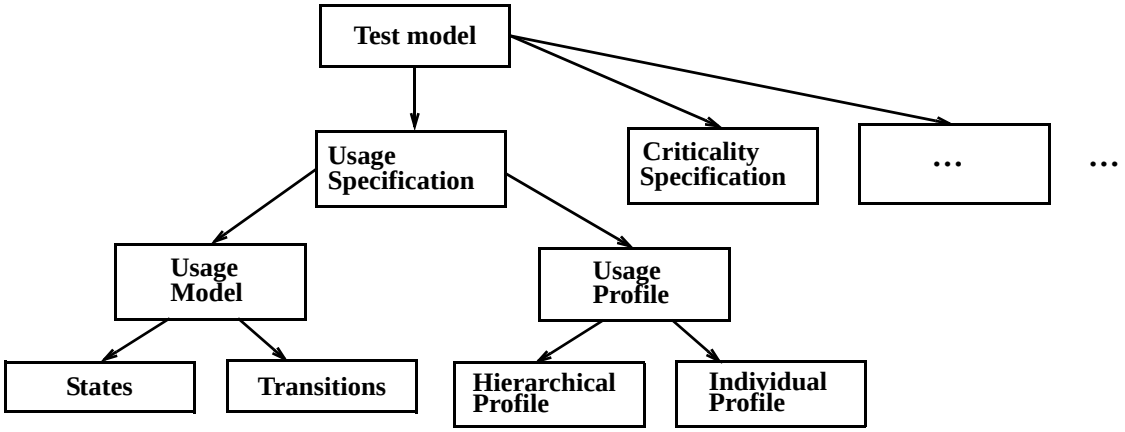

Figure 2. The test model and its usage-oriented modelling concepts.

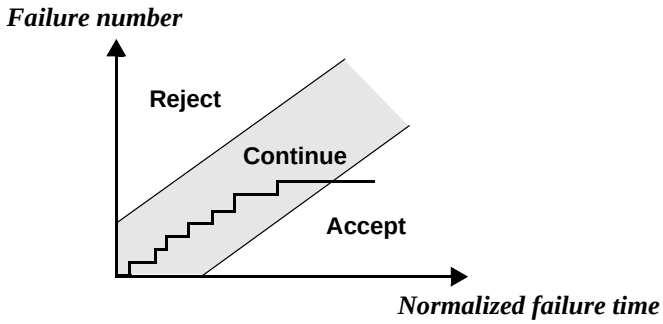

Figure 3. Reliability demonstration chart.

3.3: State hierarchy model

By combining the concepts in Figure 1 and Figure 2, it is possible to find a suitable mapping between them. It is also suitable to introduce some new modelling concepts, which allow for a grouping of the users, based on their usage models and usage profiles. A **user type** is defined as the collection of users having the same possible behaviour (normally equivalent to users having the same goal), i.e., they have exactly the same individual usage models. A **user subtype** is a further division of the users into a group where all users also have the same usage profile, hence having a similar statistical behaviour.

The usage specification and its model and profile allows for generation of test cases, which resemble the anticipated usage in the operational phase. Thus, a model is needed that is easy to run through, and to generate the next event to be put into the test script. As stated in Section 2, some different approaches exist. The focus here is on the state hierarchy model. This model can be divided into two parts, one hierarchical model part and one service model part. The hierarchy is introduced to limit the size of the model, since a non-hierarchical model would be unmanageable for large systems [8]. The hierarchical part allows us to find the next service part in which the next event will occur. The service part thus models, for example, a specific service available to a user.

The hierarchical model part is of particular interest in the context of reusing software components, since it allows for a mapping between model parts and system components. This is a key issue to enable reliability certification systems in an effective way.

3.4: Reliability certification

Certification, in general, does not assume that faults have been corrected; it is only proven that a specific reliability level has been reached. This is not the normal case for software, where a correction of a fault means that the fault has disappeared once and for all. This is not completely true due to erroneous corrections, but it is at least the objective. Thus, models taking reliability growth into account are needed, and the model to use in certification should be included in the requirements specification. These reliability models are referred to as software reliability growth models [11, 12], and they are based on a number of assumptions (more or less realistic). Another opportunity is to take a non-correction procedure and adapt it to allow for fault correction, for example, the reliability demonstration chart [11]. The chart is illustrated in Figure 3.

The failure number indicates simply the number of failures found and the normalized failure time is equal to the actual time between failures and the required time between failures. For each failure that occurs, a new point is added in the diagram (start at the origin) and a decision is taken whether to reject or accept the software or to continue testing. Based on the definition above of normalized failure time, it is obvious that a failure time shorter than the required, results in a normalized failure time less than one. This means that the plot comes closer to the reject area, which seems reasonable as the software is worse than required. In Figure 3, the software can be accepted after 7 failures have occurred. The two lines in the figure are derived based on the requirement and the confidence in taking the appropriate decision [11].

4: Usage Based Testing for Reliability Certification

This section provides the main contribution of this paper, through the certification process, the division of components into classes for certification and by identification of three different ways for certifying software systems.

The objective here is to discuss and illustrate how the modelling approach, see Section 3.3, can be used to enable usage-oriented verification, validation and certification of components. Furthermore, the objective is to describe how components can be stored together with information about

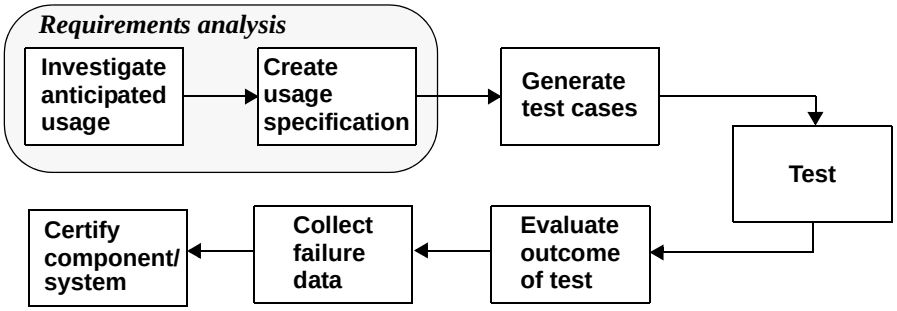

Figure 4. The certification process

their reliability for different usage profiles, and also how software certification can be done based on information about the components and their usage models.

4.1: Certification process

The functional requirements implemented in the certification component are validated during usage-based testing in the same way as in any other testing technique. Thus, the functional requirements form the basis for determining whether a failure has occurred or not. Obtaining a reliability measure requires a certification process. This process can be divided into a number of distinct steps, as shown in Figure 4.

The steps of the certification process can briefly be described as follows.

1. Investigate anticipated usage

   The anticipated usage is investigated through usage analysis of similar systems in the field and surveys of the market. The objective should not necessarily be to find very accurate probabilities, but to identify how often different users use a certain service relative to another service. This should be performed as part of requirements analysis.

2. Create usage specification of "certification component"

   The usage specification is created through a number of steps. The services of the certification component are first identified and its externally visible states and transitions are determined. The users are then studied and divided into types and subtypes. Based on this information, the hierarchical model is derived. The usage profile is derived from the investigation of the anticipated usage. Finally, the usage specification is analysed to ensure that it is a valid description of the anticipated usage of the certification component. The usage specification, or at least the information to be included in it, should be determined in the requirements specification phase. The usage model may be derived in several steps when an incremental development approach is used. This is the approach advocated in Cleanroom [5].

3. Generate test cases

   The usage specification is run through using random numbers. To generate test cases, the tester acts as the system and provides the expected responses of the certification component using the requirements specification as a basis. The stimuli generated from the usage specification and the responses from the tester are stored on a test file, i.e. the tester acts as an oracle for the test cases during the generation. The test generation procedure is further described by Wesslén and Wohlin in [13]. It should be noted that the test cases may be generated in parallel with the software development as the answers provided by the tester is based on the expected answer from the system, not the actual implementation. Further research in this area includes work on automatic checking versus the requirements specification. Use cases or scenarios would be particularly interesting since they have the same view as the statistical testing, i.e. the user perspective.

4. Test

   The tests are performed automatically by running the test files against the certification component. The test files contain both stimuli, expected responses, and a failure handling routine. This means that the test file can be run against the certification component as long as the component responds correctly, i.e. in correspondence with the expected response, no action has to be taken. The results from the test are stored in a test log.

5. Evaluate outcome of test

   The evaluation means examining the test log. It is essential to determine the cause of any unexpected

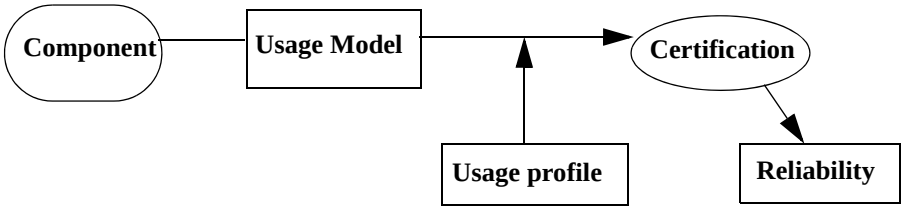

Figure 5. Certification of software components.

behaviour. A discrepancy is not necessarily a fault in the program, it may be the result of an error done by the tester when generating the test case. A test oracle is used for evaluation of the test cases, see item 3.

6. Collect failure data

   After the examination of the test log, it is possible to collect failure data in terms of time between failures. To enable this, it is necessary to log the execution time, or any other suitable time measure for keeping track of the time between failure occurrences, when the unexpected behaviour occurred.

7. Certify component

   Certification of the component is made by applying one or several of the models discussed for reliability certification. Thus, components can be reused on the basis of their quantified reliability level.

The above certification process is subsequently discussed for components and systems separately. The intention is in particular to try to define a number of different types of components to certify and also to look at different alternatives when certifying software systems.

4.2: Reliability certification of components and systems

4.2.1: Component certification. The derivation of a realistic usage model and corresponding profiles is the key issue in enabling certification. For a system, the users are mostly well-known, but it is not obvious when looking at components as parts of a system. On the other hand, the usage model is bound to be smaller for a smaller part of the system, hence making it easier. Thus, the important question is: How can a usage specification be derived for components?

It is reasonable to try to have a correspondence between components in the system and the service usage model. Thus, a service provided to a user should be packaged in a component and a component should be modelled as a usage model part. This way, it is easier to reuse components and at the same time reuse usage model parts. This is one of the main benefits of the state hierarchy model, i.e. its support for reuse. Thus, to each component a usage model should be connected and by adding a specific usage profile, it is possible to certify the reliability of the component for this particular profile, see Figure 5.

The components should then be stored for possible future reuse. They should be stored and maintained together with their usage model, which may be a part of a larger usage model, and the usage profiles for which it has been certified. To each profile, a specific reliability is connected, since the validity of the reliability measure is closely related to the usage profile, see Figure 6.

In a reuse situation, it is important to examine the profiles for which the component has been certified. If the expected profile is different from the ones under which it has been certified, it is necessary to consider how far the new profile is from the ones used for certification. Distance measures between profiles are an important area of research. It is not reasonable to expect certification of

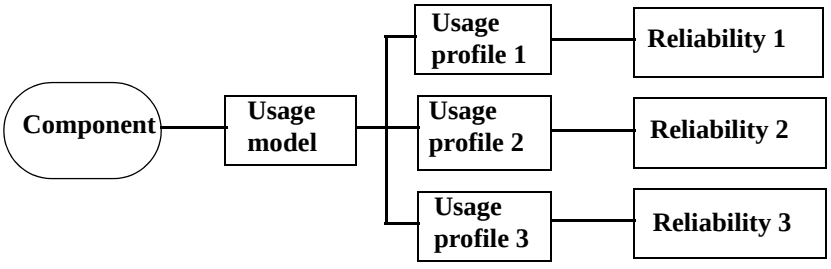

Figure 6. A component and information to be stored with it.

components for all possible usage scenarios. Instead, the focus should be on the expected profiles, and if the intention is to reuse the component for a different profile, it is necessary to certify it for the new profile and store the new information together with the component, see Figure 6. Independent of the ability to predict future usage profiles, it is important to at least provide certification information for the profiles for which it has been certified. In this way, the potential reuser at least knows for which profiles the component has been certified, and of course also for which profiles the component provider is unable to state any reliability information. Consolidating all information about a component is standard procedure in configuration management, but it is worth emphasizing from a reliability certification perspective.

The problem, mentioned briefly above, concerning determining a suitable usage model and profiles for components can partly be overcome by identifying three specific types of components. The three types are:

1. Random components

   A random type component is characterized by the usage of a number of different users, and these users, together, seem to generate an almost random profile. This is a simplification, but if it is judged that it is difficult to determine the usage profile and the component is used by a number of users, then a random assumption may be a good approximation. It is not possible to state exactly what is a suitable criterion for random; it has to be based on experience.

2. External component

   An external component is related to the interface of the system. The usage model and profile for this type of component are assumed to be derivable from the externally observable usage of the system, i.e. the system users should be examined.

3. Thread component

   A service is often implemented as a sequence of actions relating to different components (sometimes referred to as use cases or scenarios). This fact is used to define a thread component as a component participating in a limited number of services. Thus, the usage model and profile should be derived by examining how services map to the thread component in the design and implementation models.

If it is not possible to determine which of the above types a component belongs to. The profile has to be derived based on explicit knowledge of the usage of the components. The objective is to formulate as good model and profile as possible.

After having classified the components, and derived the usage profile, the system certification method in Section 4.1 is used.

4.2.2: Certify or derive system reliability. Another essential question for the systems is: How can the system reliability be determined based on knowledge about the components? Three major approaches have been identified, see also Figure 7. These approaches take advantage of the opportunities to certify components, and in particular the different opportunities available with the identification of the different component types, which provide some insight into how to certify components. One approach is to derive the usage model from scratch, and two different approaches can be identified for certifying the system,

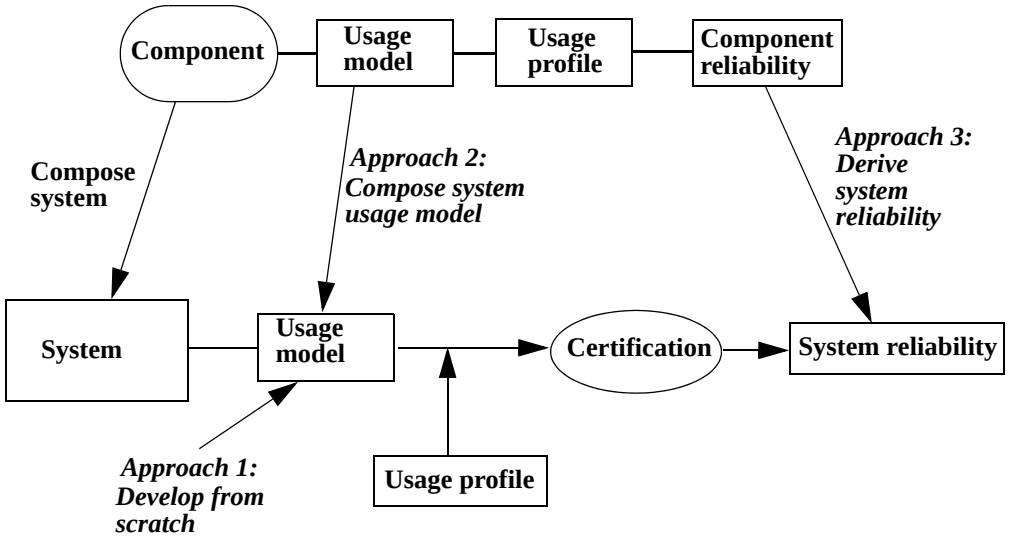

Figure 7. System certification alternatives.

when some knowledge about the components in the system is available. The three approaches are, see also Figure 7:

1. Develop from scratch

   The usage model of the system can be derived from scratch independent of any certified component. This approach is the same as described when certifying a single component, i.e. the system is viewed as the component. This is always possible, but it may be very costly to derive a new usage model and re-certify the complete system just because one new service has been added to the system. For large software systems, this is certainly not a useful approach. If this approach is used, then any usage modelling technique can be applied as long as the model is possible to handle.

2. Compose system usage model

   Instead of deriving the usage model from scratch, it should be possible to compose the usage model of the system from the usage models of each component, similar to the way a system is composed out of its components. This approach requires a correspondence between system components and usage model parts. The approach still requires re-testing to certify the system, but at least it is not necessary to derive a completely new usage model.

3. Derive system reliability

   This is the preferred approach, i.e. to be able to derive system reliability from component reliability in a similar manner as it is done in hardware. The accuracy of the certification will, of course, suffer, but the cost for re-testing large systems means that this approach should be further developed. Some work has been done in the area [14], but more research is needed, see Section 5. The objective of the classification of components in Section 4.2.1 is to provide support regarding component certification, which is the basis for deriving system reliability.

The three alternative ways of deriving system reliability are only outlined here. Further research is needed to both propose and evaluate the different alternatives. In particular, some empirical studies are needed to evaluate the third alternative, and address the challenge of handling dependencies between components.

5: Further Research

The usage-based testing approach raises a number of questions. It is not always easy to model usage and find the correct profile. Thus, the obvious questions is: How critical is the model and profile? No general answer can, of course, be given, but by trying, a better understanding of the product and its potential use is gained. Therefore, it seems reasonable that it is always better to try to quantify the usage than ignore it. As stated in Section 2, positive figures concerning its cost-effectiveness have been published.

An important issue, which must be further researched, is the sensitivity of the reliability estimate based on a changing profile. Sensitivity analysis is discussed by, for example, Chen et. al. [15]. The problem with re-certification when the usage profile changes is also discussed by Wohlin [16].

An area for further research is methods for derivation of system reliability based on knowledge of the components. This is particularly important for a large evolving system, where it is infeasible to re-certify the complete system as new services are added. This area includes the need for case studies to evaluate the opportunities to derive system reliability from the components of the system.

Another interesting area of research involves consideration of systems which are truly object-oriented. This means that objects are specialized and behaviour is inherited. A major certification problem may arise: is it

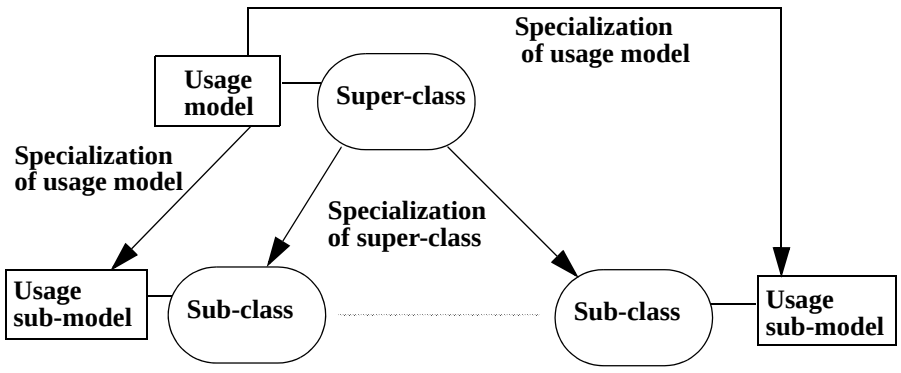

Figure 8. Specialization of classes in object orientation and their usage models.

possible to certify a super-class and what is meant by certification when the object is subclassed? Are we able to subclass a usage model too? How does a reliability measure of a super-class relate to the reliability of subclasses? The problem is partly illustrated in Figure 8.

It is also important: to study different classes of faults. Some fault classes may be more critical and it is possible that the requirements specification states different reliability requirements based on the fault class, hence different fault classes may have to be certified separately. A potential problem may be that if the faults are divided into too many classes, too few failures will occur in each class and we are unable to derive the reliability. Thus, there is a trade-off between fault classes and the ability to determine the reliability. This is an area for further study.

One area of particular interest for further work is of course industrial application and empirical studies in the area. The studies have to investigate both the classification scheme of components (random, external and thread), and the ability to certify a system based on component reliability. Furthermore, empirical studies are needed to evaluate how changes in the usage profile affects the perceived reliability. Both experiments in a laboratory environment and industrial case studies are needed to facilitate the understanding of component reliability and its relationship to system reliability

6: Conclusions

Testing of object-oriented systems has to be improved [3], and we believe that the need includes verification and validation of components. This statement can be enlarged to incorporate methods for reliability certification, as this is probably a prerequisite to enable reuse on a large industrial scale. Companies cannot reuse software components without knowing how reliable they are.

This paper provides some methods and guidelines for suitable directions to support reuse of certified software components. This is an important area of research and its industrial relevance must be hard to question. The four major contributions of the paper in this area are:

- a structure of terms in the area of certification of software reliability and their relationship,
- the need to store the usage model and profile together with the component,
- a classification of components to simplify the derivation of a usage profile, and
- three ways of certifying software systems built from components

This paper does not provide all the answers, but it provides a starting point for further research and for introducing the methods in an industrial setting to obtain experiences from certification of software components and systems.

Acknowledgement

We would like to thank Per Runeson and Anders Wesslén, Department of Communication Systems, Lund University for inspiring joint work in the area of usage modelling and reliability certification. This work is partly supported by National Board for Industrial and Technical Development (NUTEK), Sweden, Project number: 1K1P-97-09690. We would also like to thank the anonymous referees for valuable comments. Finally, we would like to express our gratitude to Dr. Sholom Cohen, SEI and Dr. Kathy Yglesias, IBM for valuable input during the update of the paper. In particular, we are grateful to Dr. Cohen for helping us improving the English.

References

[1] E-A. Karlsson (editor), *Software Reuse: A Holistic Approach*, Wiley, New York, 1995.

[2] R. V. Binder, "Testing Object-Oriented Software: A Survey," *Software Testing Verification & Reliability*, Vol. 6, No. 3/4, 1996, pp. 125-252.

[3] R. V. Binder, "Trends in Testing Object-Oriented Software," *IEEE Computer*, October 1995, pp. 68-69.

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

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

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

[7] J. A. Whittaker and J. H. Poore, "Markov Analysis of Software Specifications", *ACM Transactions on Software Eng. Methodology*, Vol. 2, January 1993, pp. 93–106.

[8] P. Runeson and C. Wohlin, "Usage Modelling: The Basis for Statistical Quality Control," *Proc. 10th Annual Software Reliability Symposium*, 1992, pp. 77–84.

[9] P. Runeson and C. Wohlin, "Statistical Usage Testing for Software Reliability Control," *Informatica*, Vol. 19, No. 2, 1995, pp. 195-207.

[10] C. Wohlin and P. Runeson, P., "Certification of Software Components," Trans. on Software Engineering, Vol. 20, No. 6, 1994, pp. 494-499.

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

[12] M. Lyu, *Handbook of Software Reliability Engineering*, McGraw-Hill, New York, 1996.

[13] A. Wesslén and C. Wohlin, "Modelling and Generation of Software Usage," *Proc. 5th International Conference on Software Quality*, 1995, pp. 147-159.

[14] J. H. Poore, H. D. Mills and D. Mutchler," Planning and Certifying Software System Reliability," *IEEE Software*,

January 1993, pp. 88-99.

[15] M-H. Chen, A. P. Mathur and V. Rego, “A Case Study of Reliability Estimates to Errors in the Operational Profile,” *Proc. 5th International Symposium on Software Reliability Engineering*, 1994, pp. 276-281.

[16] C. Wohlin, “Re-Certification of Software Reliability without Re-Testing,” in: M. Lee, B-Z Barta and P. Juliff, ed., *Software Quality and Productivity: Theory, Practice, Education and Training*, pp. 219-226, Chapman & Hall, London, UK, 1995.