Engineering Software Qualities in Telecommunications: Three Cases from Industry

Claes Wohlin
Dept. of Systems and Software Engineering
School of Engineering
Blekinge Institute of Technology
Box 520, SE-372 25 Ronneby
Sweden
E-mail: Claes.Wohlin@bth.se

Abstract

Software has become an integral part of telecommunication systems over the last 30 years. The relative cost for software in the systems has increased continuously over these years. This increase has meant that there has been a need to also focus on evaluation and prediction of different software qualities of the systems. This paper presents three studies evaluating software qualities in different ways. The firsts study focuses on performance and touches upon reliability. The reliability aspect is discussed in more detail in the second case study. Finally, the third study presents methods to estimate the fault content from software inspections. These studies are presented as representative examples of the work that resulted in that the author of this paper received the Telenor Nordic Research Award in 2004. Some ongoing research on engineering software qualities is also briefly presented.

1. Introduction

The importance of software and its development has grown over the last 30 years. A major shift (at least) in the Nordic countries occurred in 1976 when the first AXE system was run on trial basis in Södertälje in Sweden. AXE was one of the first digital public telephone switches on the market. The first digital AXE exchange was installed in Turku, Finland in 1978. The proportion of cost devoted to software research and development in telecommunications has grown considerably over these years. Telecommunication systems have become more complex as new technologies have been developed since mid 70ies, including analogue mobile telephony, digital mobile telephony (GSM), Internet and broadband services. At the same time the functions and services provided by the systems have exploded. A number of services that would have been regarded as science fiction 30 years ago are a reality today. Internet is 10 years old and it is already taken for granted as a part of the infrastructure on which a large part of the society depends on.

The speed of change is tremendous and software is in many cases becoming the competitive advantage in different types of products. Software was initially developed for specific customers, but today it is a natural part in many mass-market products too. More and more products have software embedded into them and the systems should be connected and hence being able to work together in a distributed environment. These changes have of course had a major impact on society and completely changed businesses.

The introduction of digital solutions and an increased use of software have transformed telecommunications. For example, Ericsson is the largest software development company today in Sweden, and a large proportion of their R&D is devoted to software. The introduction of software has led to new possibilities, but also to new challenges. Software is different than hardware. The difference manifests itself in terms of invisibility, changeability, conformity and complexity [Brooks87]. These inherit problems (or challenges) with software means among several other things that engineering of the qualities of the software becomes a challenge. It is easy to determine whether a specific delivery date or the budget has been met, but it is much more difficult to address product qualities such as performance, reliability and maintainability.

The challenge of engineering and managing different quality aspects (or qualities) of software has led to an increased research into these aspects of software. The objective of this paper is to briefly present some of the research that has been conducted by the author of this article and which formed the basis for receiving the Telenor Nordic Research Prize in 2004. The presentation is by no means exhaustive, and hence the aim is primarily to provide a glance of some of the research on software qualities in relation to telecommunication systems.

The research related to engineering of software qualities is illustrated with three studies conducted together with different industrial partners. The following studies are reported:

- Section 2: A study of estimation of performance and reliability from software design description using simulation is presented briefly. The study was conducted together with Telelogic AB.
- Section 3: The experience from simulating usage in Section 3 is used in the study reported here. The objective of this study is to certify software reliability using a statistical approach to testing, in particular in acceptance testing. The study was conducted together with Telia AB.
- Section 4: The work on reliability led to a study of software faults. The objective was to capture faults before testing. The focus was on software inspections and fault content estimations. This research project was conducted together with Ericsson AB.

Thus, the three studies reported show how experiences from one study help in forming the basis for new studies. These three studies are reported briefly here to give a flavor of the research. More details of the studies can be found in the references provided. The report from the studies is followed by a presentation of a current research project, in Section 5, where several qualities are addressed. Finally, some general conclusions and an outlook are given in Section 6.

2. Performance and reliability estimation

The main objective of this work was to formulate a general (independent of software description technique) method for functional, performance and reliability evaluation at an early stage of software development. The long term objective was to formulate a method that can be applied throughout the software life cycle to evaluate and assess the quality attributes of software systems. The objective was that the principles presented can be used throughout the software life cycle even if the actual level of detail in the models used may vary depending on available information.

The aim was to provide a method for evaluation of functional real time behavior, performance (in terms of capacity) and reliability of software design descriptions. The method is based on that the software design descriptions are specified with a well-defined language, for example SDL (Specification and Description Language), which can be transformed automatically into a simulation model of the software design. A tool prototype, performing the transformations of SDL descriptions into a simulation model of the software, was implemented at Telelogic. The underlying method is presented in [Wohlin91]. Some initial ideas were presented in [Wohlin89]. It must be stressed that SDL is used as an example assuming that SDL is the normal software development method at the company applying the proposed evaluation method.

Transformation rules were formulated for SDL, hence showing that it is possible to actually use the design in the evaluation of quality attributes instead of formulating a separate simulation model of the behavior of the software. The transformed model is then distributed on a simulation model of the architecture. The input to the system (transformed software design distributed on a simulation model of the architecture) is then modeled in a usage model, which is a simulation model of the anticipated usage of the system. The method consists hence of three separate models: software model, architecture model and usage model, as is shown in Figure 1.

The software model is a direct transformation of the actual design of the software to be used in the final system. The usage model and the architecture model are formulated in the same language as used in the software design, but these two models are supposed to be simulation models of the actual architecture and of the anticipated behavior of the users of the system. The three models are hence described with the same description technique which is the same technique as the software is being designed in. The strength of the method is its opportunity to combine the actual software design with simulation models.

The usage model is used as a traffic generator to the system, i.e. it sends signals to the system in a similar way as expected when the system is put into operation. The reliability of the software can be evaluated since failures occur as they would in operation, since the usage model operates with a usage profile which describes the anticipated usage in terms of expected events. This type of evaluation is further discussed in a testing context in Section 3. The capacity of the system is determined based on the inputs coming into the system and measurements on loads and throughputs. The analysis allows for identification of bottlenecks in the system as well as delays. The real time functional behavior is analyzed in terms of locating unexpected functional behavior. In particular, it is possible to find functional behavior that is a direct consequence of the delays in the system. The linkage between reality and the models as well as the relations between the models is depicted in Figure 1.

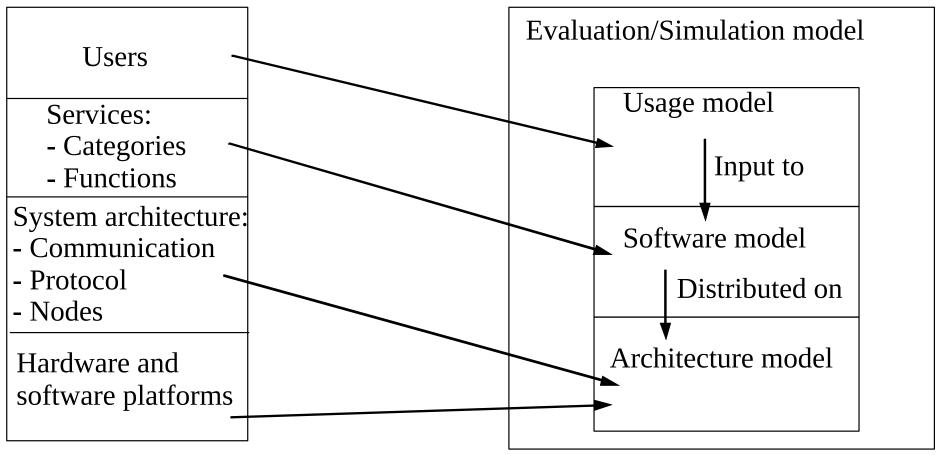

Figure 1: Mapping of the layers of uses and system concepts on the modeling concepts.

The difference between the work presented here and other approaches is the opportunity to combine the software design with simulation models described in the same description technique as the software design. The idea in itself is general and no direct limitations concerning for which design techniques this approach can be applied have been identified. The objective has neither been to formulate a tool set nor to advocate the use of SDL. The major difference with existing approaches is that a special notation has not been used and hence the method is believed to be general and the method aims at more than one quality attribute. This implies that it should be possible to adapt the general idea and formulate transformation rules etc. for other design techniques as well. The aim is to provide a framework and a method supporting early evaluation based on the actual software design as well as other description levels in the future.

The advantages with the proposed scheme can be summarized by:

- the evaluation of quality attributes can be performed at an early stage, i.e. during the design (cf. below with for example statistical usage testing),
- the concepts are general even though transformation rules have to be formulated for each specific design language,
- the actual software design is included in the evaluation method hence allowing for a good basis for decisions regarding the quality of the software,
- the method aims at analyzing performance, reliability and real time functional behavior hence no separate analysis has to be performed for each quality attribute.

Reliability estimation is also of major importance in testing. The next section highlights this further.

3. Reliability and statistical testing

Testing may be defined as any activity focusing on assessing an attribute of capability of a system or program, with the objective of determining whether it meets its required results. Another important aspect of testing is to make quality visible. Here, the attribute in focus is the reliability of the system and the purpose of the testing is to make the reliability visible. The reliability attribute is not directly measurable and must therefore be derived from other measurements. These other measurements must be collected during operation or during test that resembles the operation to be representative for the reliability. Reliability is often viewed as one important attribute of dependability. The latter is discussed in more detail in, for example, [Helvik04].

The difficulty of the reliability attribute is that it only has a meaning if it is related to a specific user of the system. Different users experience different reliability, because they use the system in different ways. If we are to estimate, predict or certify the reliability, we must relate this to the usage of the system.

The reliability attribute is complex. The reliability depends on the number of remaining faults that can cause a failure and how these faults are exposed during execution. This implies two problems:

- The product has to be executed in order to enable measurement of the reliability, although it may be estimated earlier. Furthermore the execution must be operational or resemble the conditions under which the software is operated.
- During execution, failures are detected and may be corrected. Generally, it is assumed that the faults causing the failures are removed.

In order to solve these problems, two different types of models have to be introduced:

- A usage specification. This specification, consisting of a usage model and a usage profile, specifies the intended software usage. The possible use of the system should be specified (usage model) and the usage quantities in terms of probabilities or frequencies (usage profile). Test cases to be run during software test are generated from the usage specification. The specification may be constructed based on data from real usage of similar systems or on application knowledge. If the reliability is measured during real operation, this specification is not needed.
- A reliability model. The sequence of failures is modeled as a stochastic process. This model specifies the failure behavior process. The model parameters are determined by fitting a curve to failure data. This implies also a need for an inference procedure to fit the curve to data. The reliability model can then be used to estimate or predict the reliability.

The principle flow of deriving a reliability estimate during testing is presented in Figure 2.

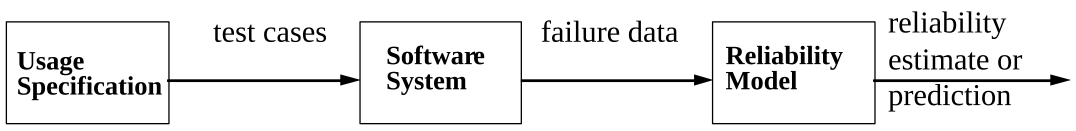

Figure 2: Reliability estimation from failure data.

Failure intensity is an easier quantity to understand than reliability. Failure intensity can in most cases be derived from the reliability estimate, but often the failure intensity is used as the parameter in the reliability model.

As indicated by Figure 2, measurement of reliability involves a series of activities.

The process related to software reliability consists of four major steps:

1. Create usage specification
   This step includes collecting information about the intended usage and creation of a usage specification.
2. Generate test cases and execute
   From the usage specification, test cases are generated and applied to the system under test.
3. Evaluate outcome and collect failure data
   For each test case, the outcome is evaluated to identify whether a failure occurred or not. Failure data is collected as required by the reliability model.
4. Calculate reliability
   An inference procedure is applied on the failure data and the reliability model. Thus a reliability estimate is produced.

If the process is applied during testing, then process steps 2-4 are iterated until the software reliability requirement is met.

Additionally, it is possible to use attribute models to estimate or predict software reliability. This means that software reliability is predicted from other attributes than failure data. For example, it may be estimated from different complexity metrics, in particular in the early phases of a project. Then the estimates are based on experience from earlier projects, collected in a reliability reference model as outlined in Figure 3.

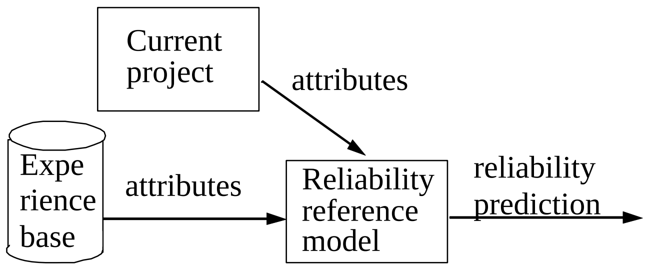

Figure 3: Reliability prediction from failure data.

The reliability measurement can be used for different purposes in software project management. First of all we differentiate between reliability estimation and reliability prediction:

- Reliability estimation means assessment of the current value of the reliability attribute.
- Reliability prediction means forecasting the value of the reliability attribute at a future stage or point of time.

Reliability measurements can be used for different purposes. One of the most important is certification:

- Certification means to formally demonstrate system acceptability to obtain authorization to use the system operationally. In terms of software reliability it means to evaluate whether the reliability requirement is met or not.

The certification object can be either a complete product or components in a product or in a component library. Component certification is discussed in [Wohlin94]. The certification can be used for internal development purposes such as controlling the test process, by relating the test stopping criteria to a specific reliability level as well as externally as a basis for acceptance (as in the case with Telia).

Reliability predictions can be used for planning purposes. The prediction can be used to judge how long time is remaining until the required reliability requirement is met.

Predictions and estimations can both be used for reliability allocation purposes. A reliability requirement can be allocated over different components of the system, which means that the reliability requirement is broken down and different requirements are set on different system components.

From the usage specification, test cases are generated according to the usage profile as mentioned above. If the profile has the same distribution of probabilities as if the system is used during operation, we can get a reliability estimate that is related to the way the system is used, see Figure 4.

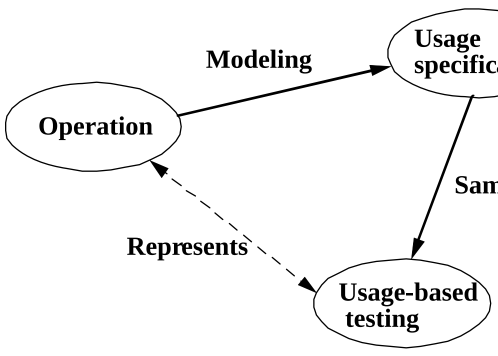

Figure 4: Relationship between operation, usage specification and usage-based testing.

To evaluate whether the system responses from the system for a test case are right or wrong, an oracle is used. The oracle uses the requirements specification to determine the right responses. A failure is defined as a deviation of the system responses from its requirements. During the test, failure data is collected and used in the reliability model for the estimation, prediction or certification of the system's reliability.

The generation of test cases and the decision, whether the system responses are right or wrong, is not simple matters. The generation is done by "running through" the model and every decision is made as a random choice according to the profile. The matter of determining the correct system responses is to examine the sequence of user input and from the requirements determine what the responses should be.

Statistical usage testing or operational profile testing is one of the few methods to actually predict the expected reliability during testing. Statistical usage testing was initially proposed as part of Cleanroom Software Engineering [Linger95]. Some extensions to the original approach are presented in, for example, [Runeson92]. In parallel to this work, operational profile testing was developed as a best practice at AT&T. More information about the approach used at AT&T can be found in [Musa93]. An introduction to software reliability can be found in, for example, [Wohlin01].

Reliability is clearly related to the faults in the software, although the actual relationship depends on the usage of the software. Thus, one way to get in control of reliability is to start managing software faults early. In particular, it is important to find ways of estimating fault content early. The next section addresses this challenge.

4. Fault content estimation

Inspections are used as a way of controlling quality, in terms of faults, but it is mostly done rather informally. The procedure is informal in the sense that we do not normally have any method or model to objectively judge the quality of the document or code being inspected. A capture-recapture method has been proposed to overcome this problem [Eick92], which is a method that is also used in biology to estimate animal populations. However, it was suggested as a method in software engineering in [Eick92]. The method is based on the inspection information from the individual reviewers and through statistical inference conclusions are drawn about the remaining number of faults after the inspection. Informally, the estimate is obtained based on the amount of overlap between findings of individual reviewers. More overlap means that fewer faults are probably remaining and vice versa. This is illustrated in Figure 5 with the helps of bugs. In Figure 5, it is shown how three reviewers find different sets of faults. Moreover, the overlap between their findings is shown together with some faults that have not been found.

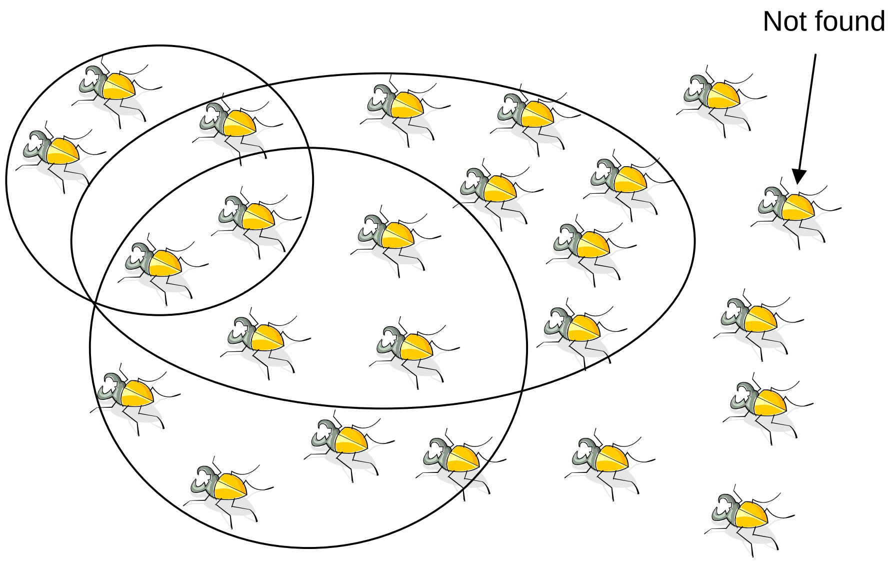

Figure 5. An illustration of the principle behind capture-recapture.

This would allow us to take informed and objective decisions regarding whether to continue, do rework or inspect some more. The capture-recapture approach is based on applying a statistical method to the collected data. Three methods have been applied for this purpose: the maximum-likelihood estimator, the jackknife estimator and the Chao estimator. The maximum-likelihood method has been applied in, for example, [Eick92, Runeson98], and the jackknife method has been compared with the maximum-likelihood method in [Briand97]. The Chao estimator has also been examined and some results are presented in [Briand97].

In [Eick92, Wohlin95], it is reported that both the maximum-likelihood method and the jackknife method seemed to consistently underestimate the number of faults. This result is contradicted by [Runeson98], where the maximum-likelihood method overestimates the number of faults with approximately 10% in average. An experience-based approach is also presented in [Runeson98]. This method is based on historical data, and it does in average as good as the maximum-likelihood method, but it seems a little less sensitive to variations in the data. Based on the difficulty to achieve good and consistent estimates, two new methods were proposed in [Wohlin98].

The novel idea behind the new approaches is that we start from a plot of the actual data (inspired by software reliability models), and through the plot we are able to better understand the data. Thus, we are also able to understand how the actual data deviate from the assumptions in using, for example, a capture-recapture approach.

The two methods identified were:

- Detection profile method
  In this method, the data are plotted with fault number on the x-axis and the number of reviewers that found a particular fault on the y-axis. The ordering of the faults on the x-axis is done based on the number of reviewers that found a specific fault. The data are plotted in a bar graph, and it is assumed that the data can be approximated with a decreasing exponential function which then is used to estimate the total number of faults, and hence the number of remaining faults as the inspections is done.
- Cumulative method
  This approach is based on the cumulative plot of all faults found by the reviewers. The data are plotted with the faults on the x-axis, and with the cumulative number of faults found by the reviewers on the y-axis. This means that the first bar gives the number of reviewers that found the fault found by most reviewers, the second bar adds the number of reviewers that found the next fault to the first bar and so on. The y-axis is simply the cumulative number of faults found. It is assumed that the bars can be approximated with an increasing exponential curve. The exponential curve is then used to estimate the total number of faults, and hence the number of remaining faults.

The objective of the detection profile estimation method is to estimate the number of faults using the information of how many reviewers that found a specific fault. The method is based on sorting and plotting the number of reviewers that found different faults, and then estimating the total number of faults approximating the data with a mathematical function.

Based on the studies we have conducted [Runeson98, Wohlin95], we have here found it suitable to use an exponentially decreasing function. It should, however, be noted that we would in particular recommend to sort and plot the data, and then based on the plot choose an appropriate function.

A fictitious data set resembling the form we have observed from real inspection data is given in Figure 6. The objective is to illustrate the form of the plot underlying the detection profile estimation method. It should be noted that we have assumed 8 reviewers (J=8) and 10 faults (K=10), which is rather unrealistic but convenient for illustration purpose. In Figure 6, we can see that the first fault is found by all eight reviewers, the second fault is found by six reviewers and so forth. An alternative interpretation is to view it as fault number one occurred eight times. The basic idea behind this method is that the data should be sorted according to the number of occurrences.

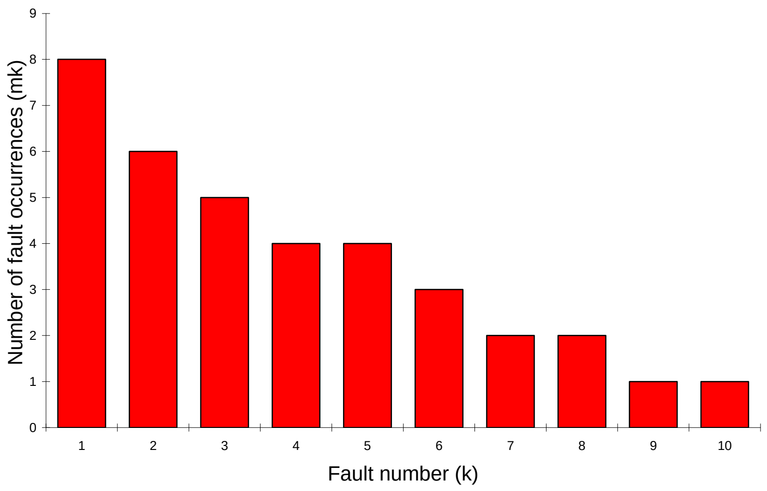

Figure 6: Fault content estimations from fault data in inspections.

In plotting the data according to Figure 6, it can be noted that it should be feasible to approximate the plot with an exponentially decreasing function. This can be used to estimate the total number of faults, if assuming:

- Adding more reviewers means that more faults will be discovered and finally all faults will be found.
- The data resemble an exponential distribution when plotted. This fulfillment of this assumption is probably dependent on the applied inspection method.
- The total number of faults is estimated from the function, assuming there is an additional fault if the function has a value greater than 0.5 for integers above the number of faults already found. Thus, the total number of faults is estimated as the last integer value which results in that the function is greater than 0.5.
- Furthermore, it is assumed that transforming the exponential function and the data into a linear model does not considerably affect the estimate. Basically, it is assumed that this transformation does not affect the estimate more than the actual uncertainty in the data.

This method has no explicit assumption about independent reviewers, although it is assumed implicitly through the assumption of an exponentially decreasing function, where it is expected that at least some faults are found by a single reviewer.

In conclusion, these types of methods, capture-recapture estimations or fault content profile methods provide an opportunity to get an estimate of the remaining fault content after an inspection. Normally, the found faults are noted, but the number of remaining faults is unknown, which may not be the best situation to base the further development on. A more in-depth introduction and overview of these types of methods are provided in [Petersson04]. An overview of software inspections in general can be found in [Aurum02].

The three studies have shown some of the work conducted with respect to software qualities. The next section gives a brief introduction to some ongoing research.

5. Current main project

Software products should fulfill several quality requirements that may be divided into three main areas (related to user, developer and manager). These three areas are: operational qualities (for example performance, reliability and usability), development or evolution qualities (for example maintainability) and business qualities (for example lead time, effort and cost). Any software project needs to trade-off between different qualities. Based on this need a major research project called “Blekinge - Engineering Software Qualities (BESQ)” was launched. The vision of the project can be summarized as “Quality-balanced software”. The overall objective of the research is to provide methods, techniques and tools to engineer the different qualities and the trade-off between them through the software life cycle from requirements to acceptance testing via design and implementation.

To achieve the above objective expertise in the following area are joined:

- Processes and Methods for Development and Management
  The research in this area is concerned with methods and techniques to improve the work process of developing software. The research includes several subprojects related to requirements engineering. One of these is focused on improving impact analysis from a requirements level. This includes supporting both managers and developers in predicting system impact. Another study is aimed at requirements prioritization of dependent software requirements. The objective is to develop a method to allow for the best possible subset of requirements to include in, for example, the next software release. Other studies in this area include software faults, methods to control faults and the faults relation to software reliability.
- Software Architecture
  Research in this area is aimed at design and use of software architecture. The focus is also on software architecture in industrial use, including both software product lines and evaluation of software architecture. Issues of particular interest are methods, techniques and mechanisms for addressing: variability of a software product line architecture, evolution of a software product line architecture and its derived software product architectures. Software architecture evaluation methods and techniques are of importance since they can be used for a number of objectives: predict quality levels of quality attributes, deciding about trade-off between different qualities and assess the architecture’s suitability in a software product line. The objective is to develop and validate usable software architecture evaluation methods.
- Performance and availability aspects
  This area includes investigating techniques, methods and guidelines for obtaining acceptable performance and availability without sacrificing other qualities, e.g. maintainability and usability. Design techniques which minimize maintainability may result in unacceptable performance due to excessive dynamic memory management. Software architectures for high availability often contain redundant components, which may increase the maintenance cost. The objective is to identify where such conflicts occur and then attack them. This can, for example, be done by providing guidelines on how to obtain reasonable trade-offs, or by providing new technical solutions in the form of resource allocation algorithms

etc. (e.g. dynamic memory algorithms for object-oriented programs) that will remove the performance and availability penalties of some popular design styles.

- Use Orientation in Software Development
  The research in this area is concerned with methods and techniques supporting the development of socially embedded systems. The focus is both on the software development process - evolutionary co-development of work practice and the supporting software - as on product qualities - changeability, maintainability, and user tailoring. For example, in an ongoing project the focus is on techniques to allow for user tailoring and the implication of the development of such techniques on software development, maintenance and "Design in Use". An empirical approach is used that facilitates, on the one hand, to explore the use and situated adaptation of methods and techniques in an industrial context. On the other hand, the objective is to (further) develop these methods and techniques in participation with practitioners.

The BESQ research environment was launched in July 2002. The Knowledge Foundation (KK-Stiftelsen) is the main initial sponsor of the initiative together with matching funds (in kind) from industrial partners. The initiative is initially run as a research project with a total funding exceeding 85 Million Swedish Crowns for the first six years. The objective is to create a strong research environment within Software Engineering during these six years to enable a successful continuation of the research beyond 2008.

There are six companies actively involved at the moment. The project is conducted together with Ericsson, UIQ Technology, Danaher-Motion Särö, ABB, Sigma Exallon and Vodafone. During 2004 two subprojects ended, one with Volvo IT as industrial partner and one with Swedish Nuclear Power Inspectorate, a government authority, as co-operation partner. There are eleven subprojects running and as many Ph.D. students involved. Among these eleven Ph.D. students, two doctoral students are industrial Ph.D. students that are located at their respective company. In total around twenty people work in BESQ.

6. Summary and outlook

The importance of understanding software qualities in telecommunications has become more and more important over the years. Software is and will be for a foreseeable future a core component and asset in telecommunications. The challenge of managing qualities of software to handle trade-off and prioritization of different aspects is an important aspect of developing software for telecommunications.

This paper has briefly presented three different studies conducted together with industrial partners from the telecommunication domain. The studies have illustrated that some methods and techniques are available today and the research is continuing. A key problem in research is that in many cases, also in the reported cases, the focus is on individual quality aspects. This is good, but not sufficient. In an industrial situation, these qualities have to be traded against each other. The paper has reported on a major research project trying to address some of these challenges.

In addition, different initiatives try to address how software should be developed. This implies improvement of software development processes, including different methods, techniques and tools. This challenge relates to all aspects of software development, including how to handle distributed software development and outsourcing. The leading software research organization world-wide, Software Engineering Institute, has launched an initiative to formulate a roadmap for the future research into the software process [SEI04]. Important aspects that will be discussed and most likely included in the roadmap are issues related to: component-based processes, flexible processes, process verification and validation, process simulation, and empirical validation of changes. Bottom-line is that the development process has to be adapted based on the needs, for example, in terms of which software qualities are important for specific types of products. One size processes do not fit all projects due to different requirements and expectations on the qualities of the software.

Acknowledgment

First of all, I would like to express my sincere thanks to Telenor for awarding their Nordic Research Prize to me in 2004. It underlines the importance of software engineering as a crucial aspect of the development and evolution of telecommunication systems. I would also like to thank former and current colleagues at companies and universities for many interesting and fruitful discussions and collaborative projects over the years.

References

[Aurum02] A. Aurum, H. Petersson and C. Wohlin, “State-of-the-Art: Software Inspections after 25 Years”, Software Testing Verification and Reliability, Vol. 12, No. 3, pp. 133-154, 2002.

[Briand97] L. Briand, K. El Emam, B. Freimut, B. and O. Laitenberger, “Quantitative Evaluation of Capture-Recapture Models to Control Software Inspections”, In Proc. 8th International Symposium on Software Reliability Engineering, Albuquerque, New Mexico, USA, 1997), pp. 234-244.

[Brooks87] F. P. Brooks, “No Silver Bullet: Essence and Accidents of Software Engineering”, IEEE Computer, Vol. 20, No. 4, pp. 10-19, 1987.

[Eick92] S. G. Eick, C. R. Loader, M. D. Long, L. G. Votta and S. A. Vander Wiel, “Estimating Software Fault Content Before Coding”, In Proc. 14th International Conference on Software Engineering, Melbourne, Australia, 1992, pp. 59-65.

[Helvik04] B. Helvik, “Perspectives on the Dependability of Networks and Services”, Telektronikk, No. 3, pp, 27-44, 2004.

[Linger94] R. Linger, “Cleanroom Process Model”, IEEE Software, Vol. 11, No. 2, pp. 50-58, 1994.

[Musa93] J. D. Musa, “Operational Profiles in Software-Reliability Engineering”, IEEE Software, Vol. 10, No. 2, pp. 14-32, 1993.

[Petersson04] H. Petersson, T. Thelin, P. Runeson and C. Wohlin, “Capture-Recapture in Software Inspections after 10 Years Research – Theory, Evaluation and Application”, Journal of Software and Systems, Vol. 72, No. 2, pp. 249-264, 2004.

[Runeson92] P. Runeson and C. Wohlin, “Usage Modelling: The Basis for Statistical Quality Control”, Proceedings 10th Annual Software Reliability Symposium, pp. 77-84, Denver, Colorado, USA, 1992.

[Runeson98] P. Runeson and C. Wohlin, “An Experimental Evaluation of an Experience-Based Capture-Recapture Method in Software Code Inspections”, Empirical Software Engineering: An International Journal, Vol. 3, No. 4, pp. 381-406, 1998.

[SEI04] Software Engineering Institute, “International Process Research Consortium”, http://www.sei.cmu.edu/iprc/, last accessed 1 December 2004.

[Wohlin89] C. Wohlin and D. Rapp, “Performance Analysis in the Early Design of Software”, Proceedings 7th International Conference on Software Engineering for Telecommunication Switching Systems, pp. 114-121, Bournemouth, United Kingdom, 1989.

[Wohlin91] C. Wohlin, “Software Reliability and Performance Modelling for Telecommunication Systems”, Doctoral thesis, ISBN 1101-3931, KF Sigma, 1991.

[Wohlin94] C. Wohlin and P. Runeson, “Certification of Software Components”, IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 494-499, 1994.

[Wohlin95] C. Wohlin, P. Runeson and J. Brantestam, “An Experimental Evaluation of Capture-Recapture in Software Inspections” Software Testing, Verification and Reliability 5, 4 (1995), pp. 213-232.

[Wohlin98] C. Wohlin and P. Runeson, “Defect Estimations from Review Data”, Proceedings 20th International Conference on Software Engineering, pp. 400-409, Kyoto, Japan, April 1998.

[Wohlin01] C. Wohlin, M. Höst, P. Runeson and A. Wesslén, “Software Reliability”, in Encyclopedia of Physical Sciences and Technology (third edition), Vol. 15, Academic Press, 2001.

Author CV

Dr. Wohlin is a professor of software engineering at the School of Engineering at Blekinge Institute of Technology in Sweden and he is also pro vice chancellor for the Institute. Prior to this, he has held professor chairs in software engineering at Lund University and Linköping University. He has a Ph.D. in Communication Systems from Lund University. Dr. Wohlin has five years of industrial experience as software developer, quality manager and project manager. He is co-editor-in-chief of the journal of Information and Software Technology published by Elsevier. Dr. Wohlin is on the editorial boards of Empirical Software Engineering: An International Journal, and Software Quality Journal.