Blekinge Institute of Technology
Research Report 2001:10

First Swedish Conference on Software Engineering Research and Practise : Proceedings

Editor:
PerOlof Bengtsson

Department of Software Engineering and Computer Science
Blekinge Institute of Technology

Proceedings
First Swedish Conference on
Software Engineering Research and Practise

SERP'01

October 25-26, 2001
Blekinge Institute of Technology, Ronneby, Sweden

Edited by:
PerOlof Bengtsson

Sponsored by:

MED STÖD AV
STIFTELSEN FÖR KUNSKAPS-
OCH KOMPETENSUTVECKLING

Foreword

It is a sincere pleasure to welcome you all to the first Swedish Conference on Software Engineering Research and Practise in Ronneby, Sweden. The purpose with **SERP'01** is to establish an informal networking conference. Although this copy of the proceedings is the tangible memento you bring home from this conference, we are confident that you will also make new acquaintances and get new ideas for exiting studies or cooperation.

Organizing a conference, even a small one like **SERP'01**, takes time, effort and dedication. First, I would like to express my appreciation to the *Program Committee*, that performed their tasks in time with excellent quality. Further, I thank all the *Authors* that submitted papers to this conference for having confidence in us. The programme also comprises research group presentations, thesis presentations, and a panel. The *Presenters* and panelists have our gratitude for contributing to the varied programme.

A very important contribution to this conference is that of our sponsors; *Vinnova* and the *KK-foundation*. Their sponsoring helped in making the **SERP'01** conference an attractive and pleasant event.

Professor Claes Wohlin, General Chair, and *Madeleine Pettersson*, Conference Secretary, have contributed their experience and valuable time. Without them there would be no conference.

There are many long term Software Engineering research efforts started or starting in Sweden. At BTH we have been granted a six years profile project in software engineering, only to mention one. It is clear that many have committed to the software engineering research field for some time and the foundation for a software engineering research community should be sufficient. It is our vision that SERP may serve this community as an annual informal meeting place for Swedish software engineering researchers and practitioners.

PerOlof Bengtsson
SERP'01 Program Chair

Conference Organization

General Chair

Prof. Claes Wohlin, IPD/SERL, BTH

Program Chair

PerOlof Bengtsson, IPD/SERL, BTH

Conference Secretary

Madeleine Pettersson, IPD, BTH

Program Committee:

Anna Brunstrom, Karlstads Universitet
Jürgen Börstler, Umeå Universitet
Ivica Crnkovic, Mälardalens Högskola
Even-André Karlsson, Q-Labs
Michael Mattsson, BTH
Per Runeson, LTH
Kristian Sandahl, Linköpings Universitet
Johan Schubert, Ericsson
Claes Wohlin, BTH

Other Reviewers

Carina Andersson, LTH
Lena Karlsson, LTH
Daniel Karlström, LTH

Table of Contents

Paper Session I: Process

Paper Session II: Architecture and Systems

Paper Session III: Quality Aspects

Understanding Software Processes through System Dynamics Simulation: A Case Study

Carina Andersson[1], Lena Karlsson[1], Josef Nedstam[1], Martin Höst[1], Bertil I Nilsson[2]

[1]*Department of Communications Systems*
Lund University, P.O. Box 118
SE-221 00 Lund, Sweden
e-mail: (carina.andersson, lena.karlsson, josef.nedstam, martin.host) @telecom.lth.se

[2]*Department of Industrial Management and Logistics*
Lund University, P.O. Box 118
SE-221 00 Lund, Sweden
e-mail: bertil.nilsson@iml.lth.se

Abstract

This paper presents a study with the intent to examine the opportunities provided by creating and using simulation models of software development processes. A model of one software development project was created through means of system dynamics, with data collected from documents, interviews and observations. The model was simulated in a commercial simulation tool. The simulation runs indicate that increasing the effort spent on the requirements phase, to a certain extent, will decrease the lead-time and increase the quality in similar projects. The simulation model visualizes relations in the software process, and can be used by project managers when planning future projects. The study indicates that this type of simulation is a feasible way of modelling the process dynamically although the study calls for further investigations as to how project or process managers can benefit the most from using system dynamics simulations.

1. Introduction

This study was performed in cooperation with Ericsson Mobile Communications AB and is based on a development project carried out in 1999.

As a step in the constantly ongoing work with quality improvements at Ericsson this study was made to show if simulation can be used for visualizing how different factors affect the lead-time and product quality, i.e. number of faults. One of the most important factors that affect the lead-time of the projects and the product quality is the allocation of human resources to the different process phases. Thus, the focus of this simulation study is on resource allocation.

Simulation is commonly used in many research fields, such as engineering, social science and economics. That is, simulation is a general research methodology that may be applied in many different areas. Software process modelling and improvement is, of course, no exception and simulation has started to gain interest also in this area. For example, in [4] a high-maturity organization is simulated with system dynamics models, and in [6] a requirements management process is simulated with a discrete event simulation model. In [8] an overview of simulation approaches is given.

There are several advantages of building and simulating models of software processes. By simulation new knowledge can be gained that can help to improve current processes. Simulation can also be used for training and to enforce motivation for changes.

The objectives of the study that is presented here are to investigate if it is possible to develop a simulation model that can be used to visualize the behaviour of selected parts of a software process, and to evaluate the usefulness of this type of models in this area.

The outline of the paper is as follows: In Section 2 the method used in this study is described. Section 3 describes the execution of the simulation study. Section 4 presents the results of the simulation and Section 5 discusses and summarizes the results of the study.

2. Method

This project was designed as a case study. Case studies are most suitable when data is collected for a specific purpose and when a subgoal of the study is establishing relationships between different attributes. A main activity in case studies is observational efforts.

With support from existing results in literature [3, 16], the research approach was created in three consecutive steps: problem definition, simulation planning and simulation operation. This methodology is based on the process chain concept, but due to lack of enough available, reliable

data, the process in practice went into an interactive pattern.

In the first phase, problem definition, the problem was mapped. Then through deeper definition and delimitation, an agreement was created around the study's purpose.

The main part in the second phase of the study, simulation planning, was to identify factors influencing the product quality. The assigner of this study wished to test the idea of using simulation models and this was governing in the details of the study. This was natural as most of the ideas to the quality factors were picked up from the organization's project, through interviewing the project staff and through documents. To add a broader perspective, results and ideas were taken from software literature. Influence diagrams were built including the different quality factors' relation to each other, but primary their effects on lead-time and product quality.

The third phase, operating the simulation model, started with translating a small part of the theoretical model into the simulation tool. A short test showed that the simulation tool worked properly. More features were added from the theoretical model into the simulation tool and more test runs were performed. The verification and validation of the model was made stepwise through the input of the whole model into the simulation tool, and the yardstick to compare with was given by documents and discussions with the assigner.

3. Developing the simulation model

In the simulation domain there are two main strategies: continuous and discrete modelling. The continuous simulation technique is based on system dynamics [1], and is mostly used to model the project environment. This is useful when controlling systems containing dynamic variables that change over time.

The continuous model represents the interactions between key project factors as a set of differential equations, where time is increased step by step. In the standard system dynamics tools, these interconnected differential equations are built up graphically. A system of interconnected tanks filled with fluid is used as a metaphor. Between these tanks or levels there are pipes or flows through which the variables under study are transported. The flows are limited by valves that can be controlled by virtually any other variable in the model. Both this mechanism and the level-and-flow mechanism can be used to create feedback loops. This layout makes it possible to study continuous changes in process variables such as productivity and quality over the course of one or several projects. It is however more problematic to model discrete events such as deadlines and milestones within a project [9, 10].

In the discrete model, time advances when a discrete event occurs. Discrete event modelling is for example preferred when modelling queuing networks. In its simplest form, one queue receives time-stamped events. The event with the lowest time-stamp is selected for execution, and that time-stamp indicates the current system time. When an event occurs an associated action will take place, which most often will involve placing a new event in the queue. Since time always is advanced to the next event, it is difficult to integrate continually changing variables. This might result in instability in any continuous feedback loops [9, 10].

To suit the purpose of this study, which is to visualize process mechanisms, continuous modelling was used. The continuous model was chosen in order to include systems thinking [13] and because it is better than the discrete event model at showing qualitative relationships.

3.1. Problem definition

The study is based on a process that is similar to the waterfall model [14]. The whole process is shown in Figure 1, but the simulation model was focused on the requirements phase and the test phase. The other phases, with broken lines in Figure 1, were excluded to get a less complex model. The requirements phase includes the pre-

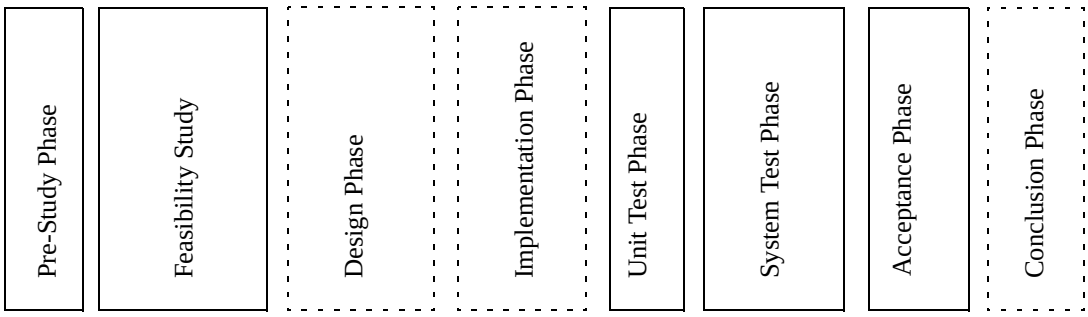


Figure 1. Process description

study phase and the feasibility study phase. The test phase involves the unit, system and acceptance tests. All these types of tests are included, since the data available did not separate between test types and they overlapped in terms of time.

3.2. Simulation planning

This step included identifying factors that affect the quality of the developed software and the lead-time of the project. This was made through interviews with project staff and based on information in literature [5, 7]. Among the factors discovered during interviews, only those considered relevant to software development processes were selected. The identified factors are listed in Table 1.

Discussions with concerned personnel pointed out the most important factors in respect to both quality and lead-time. The factors considered to affect quality and lead-time the most were chosen to be included in the influence diagrams, see Figure 2.

Table 1. Factors that affect quality and lead-time

Number of personnel in the project
Level of personnel education
Level of personnel experience
Level of personnel salary
Level of personnel turnover
Communication complexity
Geographical separation of the project
Software and hardware resources
Environment, e.g. temperature, light, ergonomics
Amount of overtime and workload
Level of schedule pressure
Level of budget pressure
Amount of new market requirements
Amount of requirements changes
Level of inadequate requirements
Amount of review
Amount of rework
Level of structure in the project organization
Standards that will be adhered to e.g. ISO and IEEE
Amount of software functionality
Testing and correcting environment and tools
Productivity
Amount of program documentation
Level of reusable artefacts, e.g. code and documentation

Influence diagrams [12] for the requirements and the test phase were built to show how the chosen factors affect the lead-time and the software quality. Each factor's importance for each phase was considered together with the relationships between the factors. The influence diagram for the requirements phase is shown in Figure 2. The factors in the influence diagram are further explained below.

- *Amount of functionality* is the estimated software functionality to be developed.
- *Amount of new market requirements* is a measure of the change in market expectations.
- *Amount of requirements changes* is a measure of the changes made in the requirements specifications.
- *Amount of review* involves reviewing requirements specifications.
- *Amount of rework* is the effort spent on reworking both new and inadequate requirements.
- *Communication complexity* is an effect in large project groups where an increasing number of participants increases the number of communication paths.
- *Level of inadequate requirements* is a measure of the requirements specification quality.
- *Level of personnel experience* is a measure of knowledge of the current domain.
- *Level of schedule pressure* is the effect of the project falling behind the time schedule.
- *Number of personnel* is the number of persons working with requirements specifications in the project.
- *Productivity* is a measure of produced specifications per hour and person.
- *Time in requirements phase* is the lead-time required to produce the requirements specifications in this project.

It is beyond the scope for this paper to present all details of the simulation model. In this paper the simulation model and related models, such as influence diagrams, are presented in some detail for the requirements phase. The requirements phase is by its nature more intuitive and easy to understand than the test phase. For a presentation of details of the complete simulation model with all related models refer to [2]. For example, the influence diagram for the test phase is presented in [2] and not here.

At the same time as the influence diagrams were constructed, causal-loop diagrams were built to get a basic understanding of the feedback concepts. Causal-loop diagrams are often used in system dynamics to illustrate cause and effect relationships [1]. When examining these relationships isolated, they are usually very easy to understand.

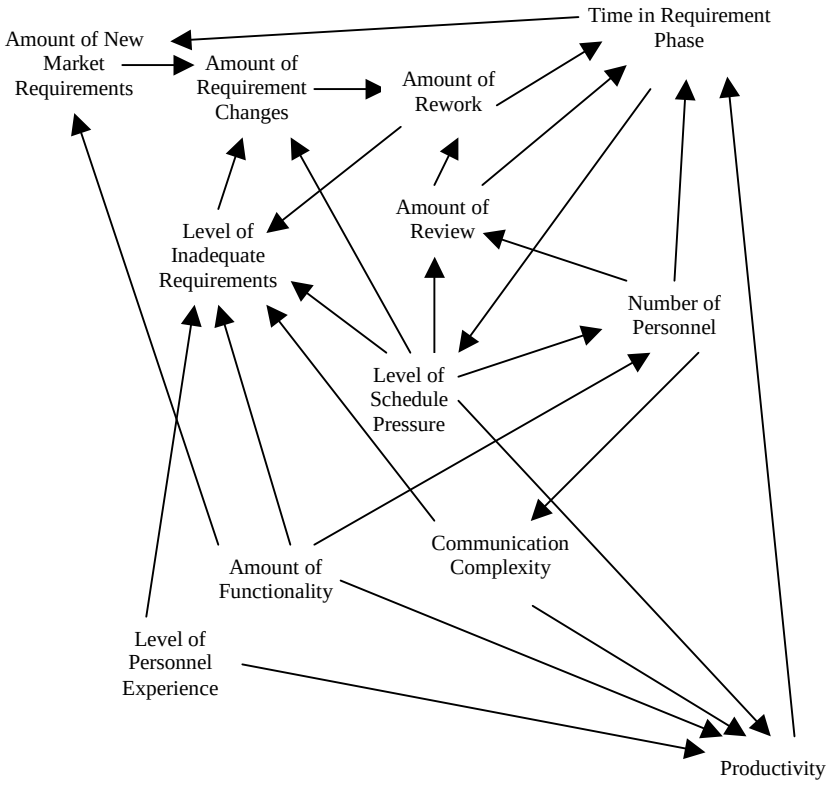


Figure 2. Influence diagram for the requirements phase

However, when they are combined into long chains of cause and effect, they can become complex. The causal-loop diagrams increase the understanding of these complex relations. Figure 3 illustrates how the schedule pressure affects the time spent in the requirements phase. An increased schedule pressure increases the error generation, due to a higher stress level. A high error density increases the amount of necessary rework and thereby increases the time in the requirements phase, which in turn increases the schedule pressure. At the same time, high schedule pressure increases the productivity because of its motivational role. Increased productivity decreases the time spent in the requirements phase, which in turn decreases the schedule pressure.

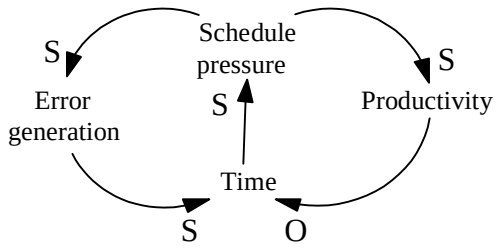


Figure 3. Causal-loop diagram

Information about the relationships between the factors in the causal-loop diagram is shown by adding an “O” or an “S” to the arrows. An “O” implies a change in the opposite direction, while an “S” implies a change in the same direction.

3.3. Simulation operation

The simulation model was built based on the knowledge gained from creating influence diagrams and causal-loop diagrams. The idea behind the model of the requirements phase is based on a flow of tasks, from customer requirements to finished specifications. In the requirements phase there is a transformation from uncompleted to completed tasks by the production of specifications. A fraction of the specifications are not acceptable and needs to be taken care of in the rework loop, see Figure 4.

The test phase in the model is based on the same idea as the requirements phase and is built in a similar way. A flow of test cases is performed, a certain percentage of the func-

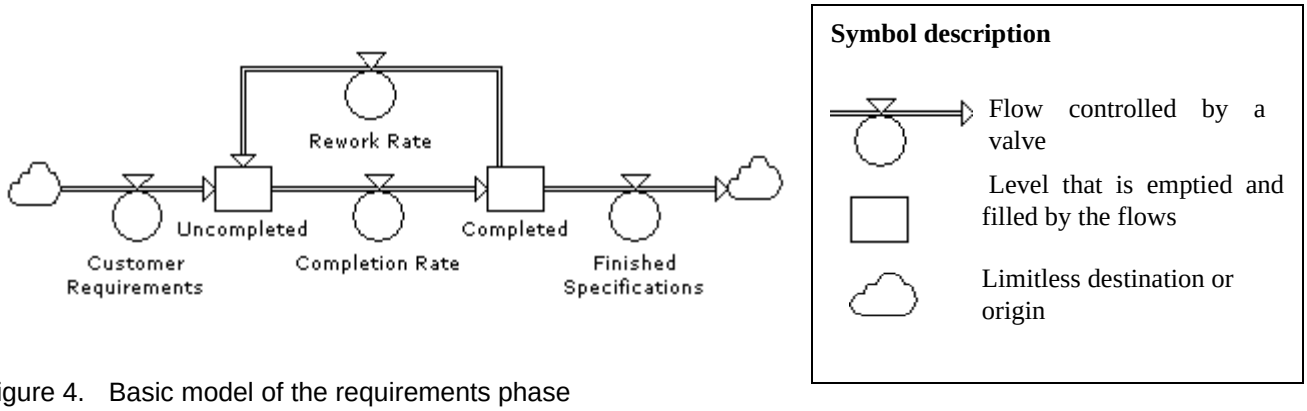


Figure 4. Basic model of the requirements phase

tionality has to be corrected and retested, and the rest is supposed to be acceptable.

This basic model was built in the Powersim simulation tool [11] and further developed with help from the factors in the influence diagrams. Factors from the influence diagrams were added to the model in order to affect the levels and flows. The causal-loop diagrams were also considered during the development, to ensure that the model was adapted to systems thinking.

To avoid getting a too complex model, all of the factors in the influence diagrams were not included in the simulation model. Some factors were included indirectly in the parameters in the model. These can be extracted from the parameters and are thereby possible to affect from the user interface, for example the communication complexity which is included in the productivity. The construction was made step by step, by adding a few factors at a time and then running the simulation. The values of the parameters were taken from project documentation except one that was taken from [7], *Amount of new market requirements*. This parameter was not available in project documentation but the value from [7] is an average from several software projects and was considered to be valid also for this project. Some values were estimated by iteration and verified by discussions with concerned personnel at the organization. The verification of the simulation model was made through checking that the amount of code that is used as an input to the model is the same as the output amount of code. The verification also included comparing the time in the simulation to the time according to the project documentation to ensure that the estimations were correct.

The final model for the requirements phase is seen in Appendix A. The flows in Figure 4 is the base of the final model, which is then further developed. To get a measure of the quality of the specifications, another flow was included, which counts the inadequate specifications. This measure affects the amount of defect code that is produced in the design and implementation phases which in turn affects the test phase. The design and implementation phases are in the simulation model modelled as a delay. A second flow is added to the basic model to terminate this phase and start the following phases.

The rest of the additions to the basic model can be described in four groups, where each group originates from the influence diagram.

- The first group, *Lines of code* and *Functionality*, describes the functionality of the code to be developed. This group controls the inflow to the phase.
- The second group is *Percentage*, *Effort* and *Duration.* The *Percentage* allocates a percentage of the planned total effort to the requirements phase and is controlled from the user interface. This makes it possible to study how the amount of resources in the requirements phase affects the lead-time and quality.
- The third group, *Productivity* and *Duration*, controls the completion rate of the specifications. The *Duration* also affects the amount of inadequate specifications because of the schedule pressure that might increase during the project's duration.
- The fourth group, *Amount of rework* and *Functionality*, decides how much of the specifications that needs to be reworked after the reviews.

Note that some factors are part of more than one group. This is because some factors affect more that one other factor.

4. Results from the simulation

The final model was simulated to show how a relocation of resources to the different process phases affects the quality of the software products and the lead-time of the project. This model included both the requirements phase and the test phase. The model was run several times with different values of the percentage of the planned project effort, spent on the requirements phase. The results are given in precise figures but since there are a number of uncertainties they should be broadly interpreted. For example, the results are uncertain because of the difficulty in measuring the values of the included factors. It is the ten-

dencies in the results that are important and not the exact figures.

The simulation runs indicate that the effort spent on the requirements phase has a noticeable effect on the lead-time of the project. The decrease in days, when increasing the effort in the requirements phase, arises from the increased specification accuracy. A more accurate specification facilitates the implementation and decreases the error generation and will result in a higher product quality from the start. This decreases the amount of necessary correction work and thereby shortens the time spent in the test phase. At a certain point the total lead-time will start to increase again because the time in the test phase stops decreasing while the time in the requirements phase continues to increase. The time in the test phase stops decreasing because there is always a certain amount of functionality that needs to be tested at a predetermined productivity. The number of days in Figure 5 is the total lead-time for the whole project.

In the same manner, the quality increases when increasing the effort in the requirements phase to a certain extent. The simulation runs indicate that the quality optimum appears in the same area as the lead-time optimum. The increase in quality originates from a higher specification accuracy, which is explained above. However, if too much effort is spent in the requirements phase, the quality will start to decrease again because there is less effort left for design, implementation and test tasks.

As a step in the verification of the results, they were compared to results in the software literature [7, 15]. This literature points at the same magnitude of effort in the requirements phase for a successful project as the simulation results.

To summarize, the simulations indicate that there is an optimum for both the quality and the lead-time. If the effort in the requirements phase is lower than the optimal value, increasing it towards the optimum will result in increased quality of the developed software and decreased lead-time.

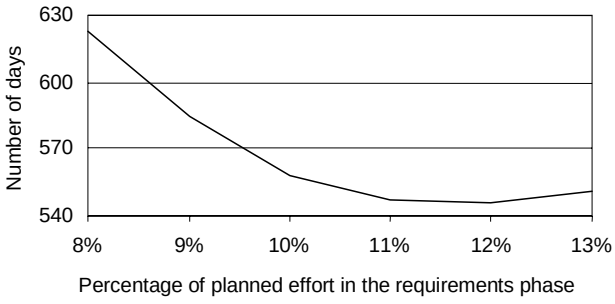


Figure 5. Simulation results for the total lead-time

5. Discussion

One result of this study is a simulation model that visualizes different relations in a software development process. A simulation of this kind can contribute to enhancing the systems thinking in an organization. Thereby it is easier for the members of the organization to understand the relationships between the quality factors in the process.

The results from this kind of simulation shall not be interpreted precisely since there, of course, are a number of uncertainties. It is the tendencies and the behaviour in the results that are important and by changing the parameters in the model it is possible to get a picture of how the process mechanisms interact. This is a simplified model of the reality and therefore there are a number of sources of uncertainty. The included factors might not be the ones that affect the model the most, the assumed relations between the factors might not be correct and the values of the factors can be incorrectly estimated. However, the results, that there is an optimum for the effort that is spent in the requirements phase, can be intuitively expected for many projects in software organizations.

A simulation of this kind can also be used to increase the motivation of the organization to work with quality issues and to increase the product quality early in the project.

One part of the knowledge gained from simulations is received in the model building process. The procedure to build the model forces the participants to communicate their mental models and to create a common image of the organization's direction.

To summarize, it seems to be feasible to build and use this kind of model for this kind of process. There are, however, a number of uncertainties which are important to take into account when the results are interpreted. The impression after developing and getting feedback on the model is that it is uncertain whether most knowledge is gained by developing the model or using it. This is one of a number of issues that need to be further investigated in the area of software process simulation. The models could either be used, for example by a project manager, by only changing the parameters, or they could be used by changing also the structure of the model, for example by adding or deleting factors and adding or deleting relationships between factors. It may be that users of the models need to understand the internal structure of the model and not only the interface to it. This would limit the choice of modelling techniques, and it would for example mean that models with an internal design, that is not easy to understand for the users of the models, would not be suitable in all cases.

Acknowledgement

The authors would like to thank Wladyslaw Bolanowski and Susanne S. Nilsson at Ericsson Mobile Communication AB for all their help with this study. This work is partly funded by the Swedish Agency for Innovation Systems (VINNOVA) under grant for Centre for Applied Software Research at Lund University (LUCAS).

References

1 Abdel-Hamid, T., Madnick, S.E., *Software Project Dynamics: An Integrated Approach*, Prentice Hall, 1991

2 Andersson, C., Karlsson, L., "A System Dynamics Simulation Study of a Software Development Process", CODEN:LUTEDX(TETS-5419)/1-83/(2001)&local 3, Department of Communication Systems, Lund Institute of Technology, 2001

3 Banks, J., Carson, J.S., Nelson B.L., *Discrete-Event System Simulation*, Prentice Hall, 1996

4 Burke, S., "Radical Improvements Require Radical Actions: Simulating a High-Maturity Software Organization", Technical Report CMU/SEI-96-TR-024, Software Engineering Institute, Carnegie Mellon University, Pittsburg, USA, 1996.

5 Fenton, N.E., Pfleeger, S., *Software Metrics: A Rigorous & Practical Approach*, International Thomson Computer Press, 1996

6 Höst, M., Regnell, B., Natt och Dag, J., Nedstam, J, Nyberg, C., "Exploring Bottlenecks in Market-Driven Requirements Management Processes with Discrete Event Simulation", Accepted for publication in Journal of Systems and Software, 2001.

7 Jones, T.C., *Estimating Software Cost*, McGraw-Hill, 1998

8 Kellner, M.I., Madachy, R.J., Raffo, D.M., "Software Process Simulation Modelling, Why? What? How?", Journal of Systems and Software, Vol. 46, No. 2-3, pp. 91-105, 1999.

9 Martin, R., Raffo, D., "A Model of the Software Development Process Using both Continuous and Discrete Models", *International Journal of Software Process Improvement and Practice*, 5:2/3, June/September, pp. 147-157, 2000.

10 Martin, R., Raffo, D., "Application of a Hybrid Process Simulation Model to a Software Development Project", proceedings of *PROSIM 2000*, July 12-14, London, UK

11 Powersim corporation, www.powersim.com, 010903

12 Rus, I., Collofello, J.S., "Assessing the Impact of Defect Reduction Practices on Quality, Cost and Schedule", proceedings of *PROSIM 2000*, July 12-14, London, UK

13 Senge, P.M., *The fifth discipline*, Random House Business Books, 1990

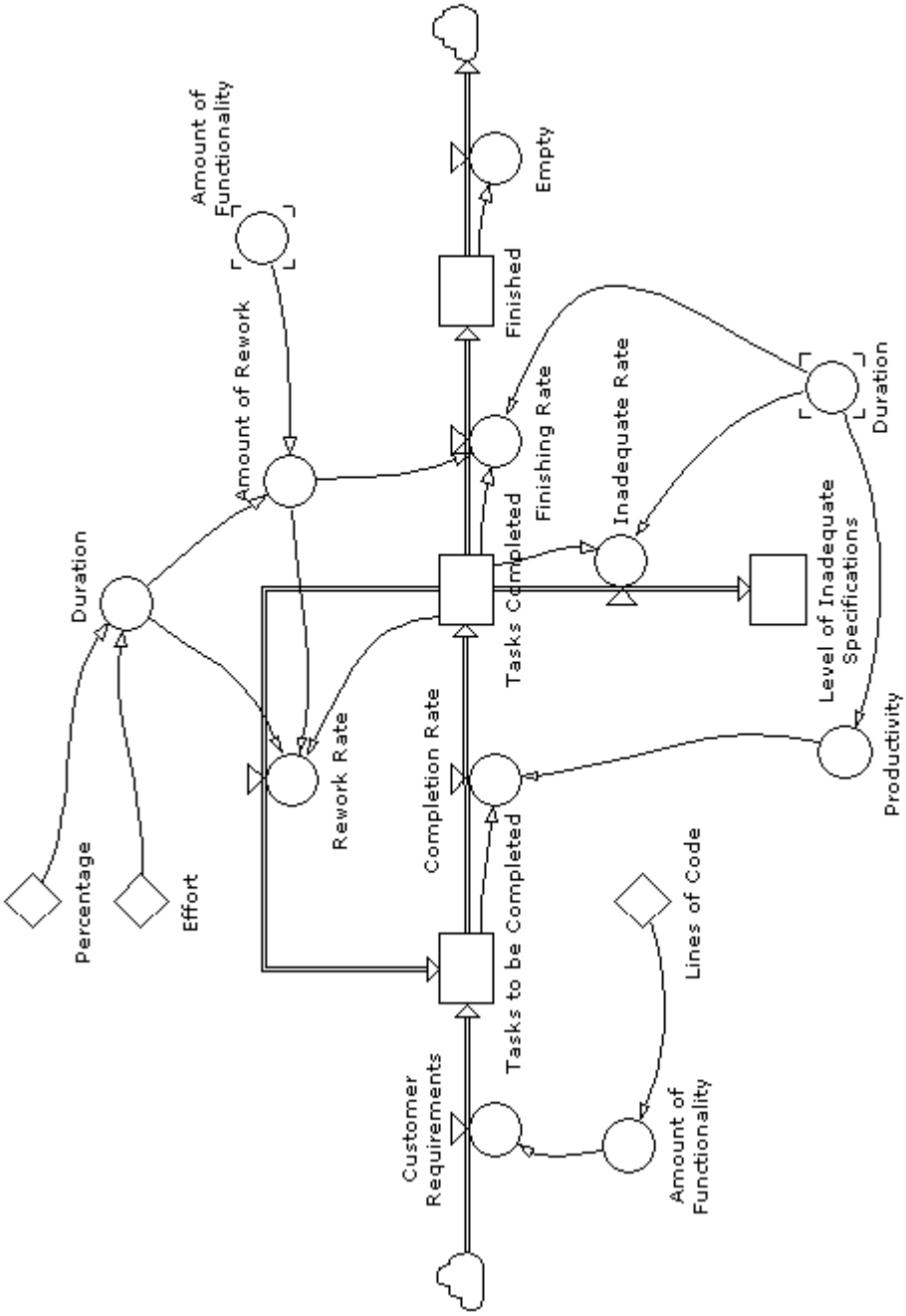
14 Sommerville, I., *Software Engineering*, Addison-Wesley, 1996

15 Stewart, R.D., Wyskida, R.M., Johannes, J.D., *Cost Estimator's Reference Manual*, John Wiley & Sons Inc, 1995

16 Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publisher, 2000

Appendix A

The simulation model of the requirements phase



Baselining Software Processes as a Starting Point for Research and Improvement

Thomas Olsson and Per Runeson
Dept. of Communication Systems
Lund University
Box 118,
SE-221 00 Lund, Sweden
[thomas.olsson|per.runeson]@telecom.lth.se

Abstract

Establishing a baseline of the current situation is an important starting point for software process improvement as well as for empirical research in software engineering. In practice, however, practical problems hinder the baselining process. This paper presents a case study to illustrate such problems. The case study aims at investigating the current status of the verification and validation process of a larger software development company. It is concluded that the quantitative data available is not sufficient for quantitative research, but qualitative issues are better covered through interviews. Finally, proposals are raised which could improve the situation on a longer term.

1 Introduction

Software process improvement has been in focus during the last decade for software development companies to get and stay competitive in their product development and service providing. Various methods for improvement have been developed, for example the IDEAL model [6] and the QIP model [2]. Most models have in common that an initial activity in the improvement process is some kind of an assessment or baselining of the current status of the organization to be improved. The assessment can take the form of, for example, a CMM assessment [6] or a baseline using the Goal/Question/Metric (GQM) method [1,10].

Empirical research within the domain of software engineering also takes a baseline as a starting point. This is particularly true for engineering type of research [5], which takes the current status of the research object as a starting point and base improvement suggestions on that baseline. The baseline can be established in a specific environment, which then is the research object. Alternatively, a broader survey can be performed, aiming at covering a domain or at least a set of different companies.

Establishing this type of baselines is easier said than done. Performing qualitative research [9], using for example surveys or interviews is one possible way of establishing such a baseline. However, this requires access to staff in different roles within the organization. Quantitative research [9], on the other hand, requires data to be collected on past projects in order to baseline the current situation. This data is seldom available in the format or granularity required for a useful baseline, if available at all.

This paper presents a case study conducted in a large Swedish company that aims at establishing a baseline of its verification and validation processes. The case study illustrates problems that may be encountered when trying to establish a baseline for research purposes. Finally, it is analysed what actually can be useful as a baseline for research, and what can be conducted on a longer term to improve the situation.

2 Case study

This section presents a case study performed at Swedish software developing company. The study is performed with a Goal/Question/Metric (GQM) approach [1]. The case study was conducted in June and July 2000. Parts of the case study have been presented in [8] as an illustration to an extended GQM method, the V-GQM (Validating Goal/Question/Metric).

The reason for performing the case study was to establish a baseline for a research co-operation between the academia and company in the case study.

2.1 Goal

The goal of the case study is defined in a format suggested in [10].

> Analyze the verification and validation process
> for the purpose of characterization
> with respect to effectiveness
> from the viewpoint of developers, testers and researchers
> in the context of one specific project

The goal takes a broad perspective, as a baseline of the current practice is the goal of the study. To fulfil the goal of characterizing the effectiveness, a quantitative approach is desired. However, qualitative aspects are also considered, but are not the focus in the goal.

2.2 Questions

The questions are divided into three categories: Process conformance, Process domain understanding and Quality focus. The questions were derived through brainstorming jointly with researchers and company representatives. The questions are summarized in table 1.

2.3 Data collection

In general, data can either be quantitative or qualitative [9]. Quantitative data implies that a measurement of an object can be attained on some scale [4]. For example, number of defects or time spent. Qualitative data cannot be measured on a scale. Feelings and opinions are examples on qualitative data. Question 12 and 13 in table 1 are examples on questions where the answer is of qualitative nature, and question 14 and 19 are quantitative examples. There is also a different set of questions in table 1, namely descriptive. Examples on descriptive questions are question 1 and 8.

The data collection has taken three different forms:

- Interviews - Several of the questions are of a subjective or individual nature. These questions are answered through interviews. The interviews were conducted during one day by one the authors. Eight persons were interviewed, varying from developers to testers and managers. Certain data is of a descriptive, unambiguous nature. These are extracted from the interviews. For example, how is the company organized?
- Reporting systems - Data concerning time consumption and defect distribution are collected through quantitative measures from existing reporting facilities, e.g. the number of defects in a specific part of the software.
- Experience - By observing at the work place, certain aspects of the procedures are documented. As the terminology and background differs between the company personnel and the researcher, this method is used to observe aspects in a direct way rather than by indirect methods such as interviews or equivalent.

The interviews are mostly qualitative, but not purely. Some quantifiable data as well as some descriptive data is also obtained in the interviews. The data gathered from the reporting systems is mostly quantitative. Information from, for example, defect reports might generate some qualitative data. The data from the direct observations is purely qualitative.

It should be noted that all collected information is treated anonymously.

3 Presentation and analysis of data

This section presents the baseline derived in the study. The presentation is divided into three parts:

- Organization and personnel (section 3.1) - Organization and Personnel characterizes the organization of the company and the background and training of the personnel.
- Process and metrics (section 3.2) - The analysis on the process metrics describes the current practice, concerning the development process in general and the V&V process specifically.
- Document management and use (section 3.3) - In this last part, the document use is elaborated on.

The analysis of the collected data is found in section 3.4.

3.1 Organization and personnel

3.1.1 Project and departments. The company is organized in a matrix organization. There are six departments, five development departments and one product department. The projects are usually large, involving virtually all development personnel. Several sub projects are usually defined and run in parallel. In total, about 100 persons are involved in the development projects.

The projects generally run for around one year. It is not uncommon that the projects are more than 100% late compared to the initial plans. These findings are not answered by the questions as such but were discovered in the interviews as a result of the qualitative nature of the interviews.

3.1.2 Training and experience. Most people working in the development organization have a master's degree or equivalent, though not always in computer science or corresponding areas. None of the interviewed persons reported that they had any formal training in verification and validation.

The level of education degree varied more among the people involved in testing. Except for the managers in the V&V department, the test personnel usually do not have any experience with development of the tool developed by the company. The tests performed by the verification and validation department focus on user interface issues. As these kinds of tests are suppose to be similar to actual use, too much experience with developing the tool being tested is usually ill advised.

3.2 Process and metrics

3.2.1 Process. The development process used is basically a waterfall model. The model defines deliverables at different stages in the process. The development is divided into several smaller projects, working in parallel with the product. However, all the groups still are bound to the overall milestones, as they are in common for all sub projects.

Each sub project defines several activities. An activity might be to implement a new communication protocol or to port the graphical user interface to a new platform. An activity can vary in size from one developer doing several activities during the development, to several developers working on the same activity. Each activity is developed more or less independently of the other activities. If there are dependencies between activities,

Table 1. Question definition

Process definition		Quality focus
Process Conformance	Process Domain Understanding	
1. Which test methods are used by the developers and testers? 2. Which inspection methods are used? Which documents are inspected? 3. Which group of people are responsible for producing the following test specifications: • Unit test cases • Integration test cases • System test cases • Acceptance test cases 4. Which group of people are responsible for performing the following activities: • Document reviews • Unit test • Integration test • System test • Acceptance test 5. Which design methods are used? 6. What is the relationship between the following documents: • Requirements • Design • Test cases 7. What kind of CASE tools are used, if any? 8. How is the company organized?	9. How well do the testers and developers understand the requirements? 10. How well do the testers and developers understand the overall function? 11. How much experience do the testers and developers have with the product? Within the domain? 12. How is the general atmosphere towards changes and improvements to the development and testing process?	13. What is the quality of the requirements? 14. How much time is spent in each phase by the different people involved in the project? 15. What is the lead-time of the development phases? 16. How does the distribution of faults look like over the different phases? 17. How many failures (defects still in the product) is observed in the delivered product? 18. How is the internal delivery quality perceived? 19. How many people are involved in the following activities: • Requirement specification • Development • Testing One person might very well be involved in several activities 20. When are the different documents produced?

then scheduling takes place to prevent some activity from having to wait for someone else to finish.

The process is mainly a management process. Overall milestones are defined as well as deliverables. However, little support is given to the individual developer or tester in their day-to-day work.

3.2.2 Time reporting. The time reporting is virtually non-existent. It is impossible to derive which effort is spent on the different activities. This is largely due to major organizational changes that took place about a year before the case study. As a result a new reporting system that was considered too clumsy to handle by the personnel was introduced. This lead to that time reporting in practice was not done.

3.2.3 Defect reporting. The procedures for reporting defects include logs from reviews and tests as well as a defect reporting system for individual defects. The latter is only used late in the development process. The project management determines the exact point in time when formal defect reporting is introduced.

Basically, when enough of the sub projects have been implemented and delivered to the V&V department, formal defect reporting is introduced. Formal defect reporting in this context means that all defects must be reported through the defect report system. It should be noted that the report system can be used before it is required. Some defects, for example from customer support and the V&V department, are reported formally even before it is required.

A problem with the defect reports is lack of information and granularity. The reports are often incomplete and lack information that can be used quantitatively. Traceability is present to persons handling the report, but not to the developed system. Also, no forms of characterization of the defects are used, for example severity or type.

3.2.4 Other process issues. Maintenance work is done in parallel with new development. Defects are reported continuously on both older and newly developed parts of the system, not just by developers and testers but also from support and customers.

No configuration management plan [7] exists. Basically, only one branch exists per project. A daily build strategy is used.

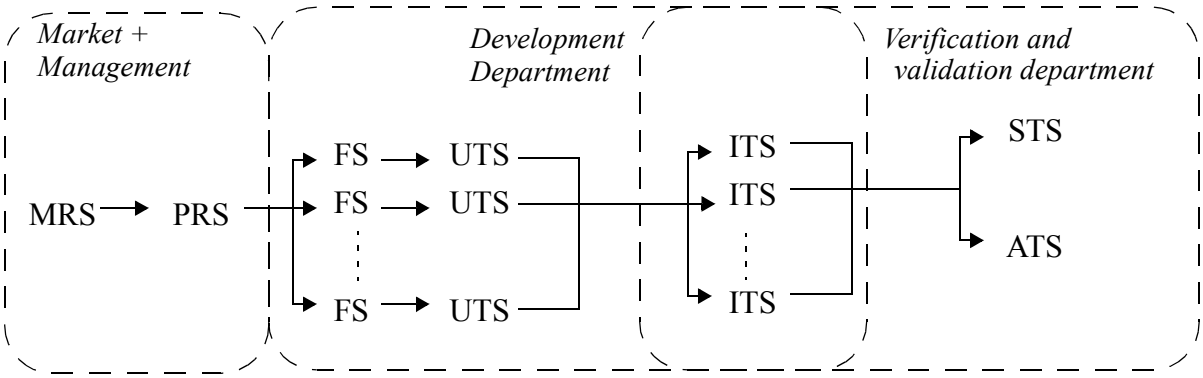


Fig. 1. Documents and dependencies

3.3 Document use

The development process is artefact driven [3]. This means that the process phases and activities are centred on the documents (artefacts) developed. The following documents are defined:

- Market Requirement Specification, MRS
- Product Requirement Specification, PRS
- Functional Specification, FS
- Unit Test Specification, UTS
- Integration Test Specification, ITS
- System Test Specification, STS
- Acceptance Test Specification, ATS

Figure 1 depicts the dependencies between the documents.

The MRS and PRS are produced by the market department and the project management jointly. From the PRS, sub projects and activities are defined and the development is continued in the different sub projects. The FS and UTS are produced by the development departments. Some ITS are produced by the development departments, some of the V&V department. ITS does not contain traditional integration testing. Rather, the ITS is various tests, determined by test managers and senior developers.

For each activity in each sub project, one FS and one UTS should be written. Hence, there exist several instances of these documents. There also exists several instances of the ITS, but these are not dependent on a certain activity or sub project. Final tests, the STS and ATS are performed by the test department and an acceptance test department respectively. Design documents are largely missing.

The dashed lines in figure 1 depict which department that produces which documents and the relationship among the documents.

3.3.1 Specification. In the first phase, the documents MRS and PRS are produced. This is done by market people and key personnel from the development organization. The MRS specifically but also the PRS are generally considered to be too abstract and ambiguous to use as they are. Each project therefore has to interpret these documents and make their own compromises. The MRS and PRS, interpreted by developers, are the basis for the FS. Verification and validation is not done with the help of the MRS or the PRS. Little or no traceability exists between the test documents and the MRS and the PRS.

3.3.2 Development. Based on the PRS, smaller sub projects are defined. For each project, several FS are written, for each identified activity. When the implementation is done, functional tests are defined and performed. The steps are: Write FS, implement the function, define and perform unit tests (UTS). This is the way the work is done in practice most of the time. According to the development process, the UTS is supposed to be written before or at least in parallel to the development, but this is often not done. Also, the work on the implementation and the functional specification (FS) does not always follow the steps mentioned above. Instead the work is performed in parallel or iterative manner, even though FS is supposed to be written in advance.

The FS and UTS have to be approved through a review before the subsequent step is initiated.

Even though the process is well-defined and baselines are clear, they are not always followed. It is mainly the distinction between implementation and test that tends to be less strict.

The steps in the process are well-defined, but the entry and exit criteria are not. Due to the fact that little or no training is received on how to write a functional requirements specification or what a test specification should look like, the quality of these document vary. Even though the FS and UTS are reviewed, they depend largely on senior developers to find flaws rather than a clear methodology or criteria for approval. Also, the instructions about the content of the FS and UTS are poor, adding to the varied character and quality.

From interviews it is also clear that the purpose and content of the FS and UTS is ambiguous. Often, design issues are found in the functional specifications, even though management generally agrees that it should be more of a requirements document.

3.3.3 Integration tests. The integration test specification is written by the developers or senior personnel, and

occasionally by staff from the V&V department. The purpose is to "test various things". The identification of which tests to perform is done by senior developers and test personnel. The areas that have had a lot of defects in previous releases are used to identify test cases for the coming release. Again, a clear purpose and criteria on content is lacking. The tests defined in the ITS range from low-level structural (white-box) testing to high-level user-centred (black-box) testing. The ITS are reviewed as all other documents.

It is not uncommon that the ITS tests the same things as the UTS. This is motivated by that the issues that are tested more than once is known trouble areas or new functionality that should be more thoroughly tested. However, the traceability between the different test specifications is poor and which leads to the exact same test sometimes being performed more than once.

3.3.4 Verification and validation. After the system has passed all ITS, the product goes through an internal release to the V&V department for system test (STS). They perform various tests, purely from a user perspective, with tests ranging from simple interactions tests to big configuration stress tests.

No means of statistical testing or quality measurements are used. Rather, based on experience of key test personnel, decisions are made on courses of action. The test cases are defined fairly abstract, as they are intended to be real user situations rather than detailed instructions. The tests are logged and all defects are reported using formal defect reporting.

The tests are of sampling type. That is, the system is tested only partly. Again, the test is based on experience of key personnel, not historic data or usage profile etc. The main difference from ITS is that only interface aspects are considered, never details in the implementation.

At no stage in the verification and validation process are any specific tools used or any form of automatic testing.

3.4 Analysis

Two major problems are identified: Lack of detailed process description and unclear document management.

A lot of information on the system that is being developed is not documented. Also, development procedures are implicit and not clearly defined. Tests are generally ad hoc and somewhat undirected. Little or no distinction is made between specification and design. These issues indicate that there are problems with the process in general. As many other software developing companies one of the problems here is that the personnel often lack a formal training in software engineering in general and verification and validation in particular. Specific problems relate to the overall waterfall model, which is more a management process rather than a development process. It is abstract and high-level and does not go into details on the development. As a result, the practice varies a lot among the developers and testers and experiences are difficult to capture.

The other issue of document management manifests itself by a varying content and quality of the document, as well as a lot of re-work of the same information at several places. Many developers and even testers reported that, for example, the functional specifications (FS) are not very useful and the information that was supposed to be in the documents is found elsewhere. The inconsistencies among documents are large and the level of traceability is low, which also indicate problems. Documents are generally written in natural language and without any tool support for creating or maintaining quality and dependencies among the documents.

It is difficult to make any statement about the efficiency of reviews and tests. The time recording does not contain sufficient details. It is also difficult to relate to the lead-time as a lot of tests are performed informally and parallel to development of code or defect removal. Added to this is the fact that a lot of maintenance work is done in parallel with new development, confusing the picture even more.

As the amount of quantitative data is very low, no conclusions can be drawn from it. It is clear, however, that there is a need to improve the data collection, both from a researchers perspective as well as from the company's perspective.

4 Experiences from the case study

The baselining conducted in the case study was aimed at constituting a starting point for a research project. The intention was to characterize the organization quantitatively, as much as possible to enable continued quantitative research. As presented above, most of the information collected was of qualitative character. Some data collection procedures existed, but they were not followed strictly enough for the research purposes. Neither were they designed with the research questions in mind. Instead, most information was collected through interviews.

4.1 Analysis of the data collection

As stated in the goals, the purpose of the case study is to characterize the verification and validation process as a basis for research co-operation. However, very little quantitative data could be collected. The time reporting is to a large extent not performed. Neither information on lead-time nor person-hours is available. As a result, any change introduced to reduce lead-time or person-hours won't be measurable.

Some data is possible to attain from the existing reporting systems regarding defects. However, the granularity is coarse. Also, the lack of time information, as mentioned, make the analysis of the defect reporting weak. Some analysis of effectiveness, however, is possible. The number of defects and on a coarse level where in the developed system the defects are found is available. However, efficiency questions cannot be answered.

As a result, mostly qualitative data could be collected. This information is valuable and provides a good in-

sight into the processes at the company. However, measuring the impact of changes is not possible solely based on qualitative data. Also, the lack of quantitative data has implications on monitoring and control of the development work. For example, from the case study it is discovered that the knowledge that a deadline will not be meet is attained very late. With more quantitative data this can, in some cases, be avoided.

It should be noted that much of the qualitative information collected is very useful. However, collecting qualitative data is costly, as this is typically done through interviews or questionnaires. If more quantitative data is available less time can be spent on collecting qualitative data. Hence, this implies that the baseline as such was more costly than needed, without attaining more information.

4.2 Consequences of the data collection

The problems with the data collection implies two consequences for the study. Firstly, the baseline is not quantitative enough to base quantitative research upon. Secondly, the baseline study took more time from the organization as people had to be available for interviews.

There is not a lot of research done in the area of baselining. Much research is done in the area of measurement programs in general, but not on the baseline process as such. For future research within software engineering, these issues can be addressed in two ways. Either, the facts have to be accepted and the research has to turn more qualitative, or the industry practice on data collection has to change. Probably, the solution is a mixture of the two.

Qualitative research is sometimes considered being of less value than quantitative research in the engineering community. This discussion is not fruitful. Instead, qualitative and quantitative research should be seen as complementary. Qualitative studies should be utilised for broader, more general research questions, while quantitative studies should be used for more specific studies. By having this balanced view on research methods, a better understanding and research progress can be achieved within the software engineering domain.

Concerning the data collection in industry, there is a need for improvements. Not that the data collection as such is important for the companies, but they ought to use the measurements to enable management make decisions based on quantitative information. On financial issues, it would never be accepted to make decisions without quantitative data, while in project and product management, decisions are taken with very little support from quantitative data.

If this improvement takes place in industry, the basis for conducting empirical research will improve substantiatally. Researchers will always want more measurements, and also more specific ones, than industry. With an interest within the organization for the measurements, there will also be a bigger acceptance for measurement programs in larger. A greater interest from industry of measurement programs is also likely to pave the way for more research co-operations with academia, as the mutual benefit would increase.

5 Summary

Establishing a baseline is and important starting step in an improvement program as well as in a research program. Without knowing the characteristics of the starting point, you will never know if a change is an improvement, or just is a change. This paper reports a case study aiming at establishing a baseline for a research program.

The company in the case study uses a waterfall model for the development. The personnel has a varying experience with software development, but generally there is a low formal training level in software engineering, especially verification and validation. It is not uncommon that the development projects take more than 100% more lead-time than estimated. The quality of the delivered product is considered to be too low by the company. The internal quality of documents is generally considered to be low and the varying type of content (for example requirements or design information) depending on who is the author.

It is concluded that the existing measurement program is not sufficient to answer the questions raised, which where of quantitative nature. In the baseline, qualitative issues where investigated as well.

It is the authors experience that these problems with an inferior measurement program are by no means exclusive for the company in this case study. These problems exist at several companies. Establishing a baseline as the basis for research co-operations with companies and as a base for improvement work is difficult and often not done. As the software engineering fields matures, both research wise as well as the development practice, an even greater need for well-founded baseline is eminent. It is not enough to know where we are going, we must also know where we are.

Acknowledgements

The authors are thankful to all participants of the case study for their contribution. The work is partly funded by the Swedish Agency for Innovation Systems (VINNOVA), under grants for Lund Center for Applied Software Research (LUCAS). The authors are also grateful for comments and help with reviews from the fellow researchers at the department.

References

[1] Basili, V.R., Caldiera, C., Rombach, H.D. “Goal Question Metric Paradigm”. *Encyclopedia of Software Engineering* (Marciniak, J.J., editor). Volume 1, John Wiley & Sons, pp. 528-532, 1994b.

[2] Basili, V.R., Caldiera, C., Rombach, H.D. “Experience Factory”. *Encyclopedia of Software Engineering* (Marcin-

iak, J.J., editor). Volume 1, John Wiley & Sons, pp. 469-476, 1994b.

[3] Cogula, G., Ghezzi, C. "Software Processes: a Retrospective and a path to the Future". *Software Process - Improvements and Practice*. Vol. 4, No. 3, pp. 101-123, 1998.

[4] Fenton, N., Pleeger, S.L. *Software Metrics: A Rigorous & Practical Approach*. Second edition, International Thomson Computer Press, 1996.

[5] Glass, R. L., "The Software Research Crisis", *IEEE Software*. Vol. 11, no. 6, pp. 42-47, November 1994.

[6] Humphrey, W.S. *Managing the Software Process.* Addison-Wesley Publishing Company, 1989.

[7] Leon, A. *A guide to Software Configuration Management*. Artech House, 2000.

[8] Olsson, T., Runeson, P. "V-GQM: A feed-back approach to validation of a GQM-study". *Proceedings International Software Metrics Symposium*. Pp. 236-245, April, 2001.

[9] Robson, C. *Real World Research*. Blackwell Publishers, 1993.

[10]van Solingen, R., Berghout, E. *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill Publishing Company, 1999.

Introducing Extreme Programming – An Experience Report

Daniel Karlström
Daniel.Karlstrom@telecom.lth.se

Dept. Communication Systems,
Lund University,
Box 118, SE-221 00 Lund,
Sweden.

Abstract

This paper presents a single case study reporting the experiences of introducing extreme programming (XP) *in a small development project at Online Telemarketing in Lund, Sweden. The project was a success despite the fact that the customer had a poor idea of the system required at the start of the development. This success is partly due to the introduction of practically all of the XP practices. The practices that worked best were the planning game, collective ownership and customer on site. The practices that were found hard to introduce and not so successful were small releases and testing.*

1 Introduction

Extreme programming (XP) [1] is a methodology that has received much attention during 2000 and 2001. XP is a package of several practices and ideas, most of which are not new. The combination and packaging of all of these is, however new. One of the features that makes XP different to most other methodologies is that it is centred on the developer and gives him or her more responsibility in the creation of the product. This paper provides an experience report from the introduction of XP at Online Telemarketing in Lund, Sweden. The company decided to use XP to develop a sales support system for use in their principal line of business, telemarketing. The paper provides a brief description of extreme programming in section 2. Section 3 presents a brief introduction to qualitative research methodology, which can be said to be the research methodology used for this experience report. Section 4 contains a brief introduction to the company, followed by an introduction to the development project in section 5. The experiences of the XP practices are accounted for in section 6 and a discussion of the quality of the conclusions is accounted for in section 7. Finally the conclusions and a summary are presented in section 8.

So far, relatively few experience reports have been made available with regards to XP. Especially, well structured reports of attempts to fully introduce XP are rare. Experience reports not only provide insight into specific situations in which the method may work and not work, but also provide practical examples to illustrate the method. Organisations considering XP can gain much needed prior experience of what to expect when introducing practices, irrespective if they are implementing one practice or implementing XP fully.

2 Extreme programming

2.1 Introduction

Extreme programming is a methodology created to address the needs of small to medium software projects developing products with little or vague requirements specification [1]. The methodology has seen much attention during 2000 and 2001, but relatively few companies have tried more than one or two of the practices that make up XP [2, 3, 4, 5].

2.2 XP practices

XP is composed of a few fundamental values, principles and activities, which are implemented by 12 practices. The fundamental values are *communication*, *simplicity*, *feedback* and *courage*. The fundamental principles are: *rapid feedback*, *assume simplicity*, *incremental change*, *embracing change* and *quality work*. The basic activities are *coding*, *testing*, *listening* and *designing*.
The 12 practices that are intended to realise all this are described in the following list [1].

- *Planning Game*

Quickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes the plan, update the plan.

- *Small Releases*

Put a simple system into production quickly, and then release new versions on a very short cycle.

- *Metaphor*

Guide all development with a simple shared story of how the whole system works.

- *Simple Design*

The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.

- *Testing*

Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests demonstrating that features are finished.

- *Refactoring*

Programmers restructure the system without changing its behaviour to remove duplication,

improve communication, simplify, or add flexibility.

- *Pair Programming*

All production code, i.e. code that is actually used in the product, is written with two programmers at one machine.

- *Collective Ownership*

Anyone can change any code anywhere at any time.

- *Continuous Integration*

Integrate and build the system many times a day, every time a task is implemented.

- *40-hour Week*

Work no more than 40 hours a week as a rule. Never work overtime a second week in a row.

- *On-site Customer*

Include a real, live user on the team, available full-time to answer questions.

- *Coding Standard*

Programmers write all code in accordance with set rules emphasizing communication through code.

The practices can be introduced independently or as a whole depending on the situation in the development organisation. The practices are intended as a starting point for a development team. The team should start using the practices as they are described in XP and gradually adapt and optimise them to the team's own preferred method of working.

An XP project works best if certain roles are assigned to the team individuals so that they each have different responsibilities regarding the practices previously described. The roles do not necessarily need to represent individual persons. One role can be assumed by several people and conversely one person can assume several roles if need be. The roles are briefly described below [1].

- *Programmer*

The programmer is at the heart of XP. The programmer creates the product based on the story cards written by the customer.

- *Customer*

The customer provides the functionality to be implemented in the form of story cards. The customer also creates, sometimes with the help of the tester, functional tests to ensure that the product does what it is supposed to do.

- *Tester*

The tester role in XP is focused towards the customer to ensure that the desired functionality is implemented. The tester runs the tests regularly and broadcasts the results suitably.

- *Tracker*

The tracker role is to keep an eye on the estimates and compare these to the performance of the team. The tracker should keep an eye out for the big picture and remind the team, without disrupting it, when they are off track.

- *Coach*

The coach's role in XP is to be responsible for the process as a whole, bringing deviations to the team's notice and providing process solutions.

- *Boss*

The boss's role in XP is basically to provide for the needs of the team to ensure it get the job done.

3 Methodology

The majority of the information presented in this experience report was gathered in two different ways. The first way was by direct observation of the developers during the course of the project, and the second was by interviews with both the developers and the development management. The interviews with the developers gave a lot of information about attitudes towards different XP practices, while the interviews with the management gave information mostly about how the practices were being followed.

As the information presented is of a qualitative nature, a brief discussion of qualitative methodology and threats is in order. It should be mentioned that the study performed is not intended to be a complete formal qualitative investigation in the respect that some form of auditing is not used to validate the results [7]. This kind of validation is only applicable and practical in much larger studies. By addressing the methodology behind the research techniques we can at least make an informed attempt at improving the quality of the information obtained.

The techniques used for gathering the information, interviewing and observing, are both qualitative research techniques. In this type of research the trustworthiness of the investigation, which is usually called validity in quantitative research, can be addressed using four criteria: *credibility*, *transferability*, *dependability* and *confirmability* [6].

These criteria are briefly summarised below. Further information can be found in the works of Lincoln [6], Robson [7], and Miles and Huberman [8]. References are made in the summaries below to corresponding criteria in quantitative validity theory. Wohlin et al. [9] contains a comprehensive quantitative validity section.

- *Credibility*

Credibility corresponds to internal validity in quantitative research. The aim of this criterion is to ensure that the subject of the enquiry has been accurately identified and described. This can be achieved by using several techniques, for example triangulation of sources or methods.

- *Transferability*

Transferability corresponds to external validity in quantitative research. This criterion addresses how far outside the observed domain the results are applicable.

- *Dependability*

Dependability addresses whether the process of the study produces the same results, independent of time, researcher and method.

- *Confirmability*

Confirmability addresses the issue of researcher biases and ensures that the researcher affects the results as little as possible.

An attempt to evaluate the study according to these four criteria is performed in the quality of conclusions section, section 7.

4 Online Telemarketing

Online Telemarketing is a small company specialising in telephone-based sales of third party goods. The company has its head office in Lund, Sweden, and regional branches in Uppsala, Visby and Umeå. Recently the company has expanded internationally with operations in Denmark, Norway and Finland. The company consists of a small core of fulltime staff that manages and supports a large number of temporarily employed staff. This implies that the company has a very flat organisation as shown in figure 1. The primary task of the temporary staff is performing the actual sales calls.

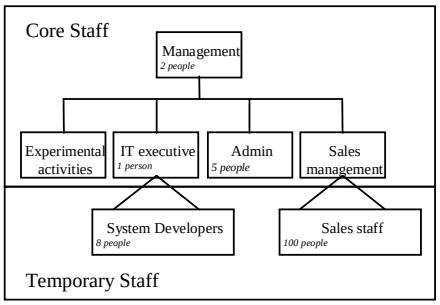


Figure 1: Online Telemarketing corporate organisation

Management realised in the autumn of 2000 that a new sales support system would be required and started planning for a system for use within the company. 'Commercial off the shelf' (COTS) alternatives were evaluated but discarded due to being too expensive and due to the fact that it would be both difficult and expensive to incorporate specialised functionality. The management at Online Telemarketing had several novel ideas for features not present in the systems available on the market that they considered crucial for the future expansion and business success of the company.

The person responsible for systems development at Online Telemarketing realised that the lack of detailed requirements from management and the fact that no similar systems had been created before meant that traditional development with a big up-front design and detailed requirements documents would prove expensive and not very efficient. An alternative was found in XP [1].

5 The development project

5.1 System overview

The system that was to be developed in the studied project is a sales support system for telephone sales. The system should cater for the needs of the sales staff while they are performing their selling tasks and the administrative staff that monitor the selling. All information is logged in a database system. A system overview is presented in figure 2. The lines in the figure represent bi-directional transfer of data. Further details of the developed system are irrelevant to this paper and therefore omitted.

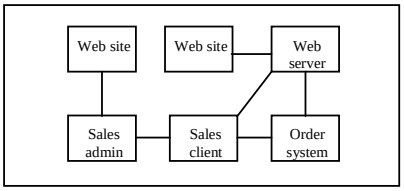


Figure 2: System structure overview

5.2 Project overview

Online Telemarketing decided on a strategy for developing the product that involved using their own system responsible person and employing part time developers to perform the coding work. To start with, four systems-engineering students were employed part time in parallel with their coursework at the university. After three months a further four people were employed and integrated into the development team in order to increase the absolute velocity of the project. The developers were employed as regular employees and there was no connection whatsoever between their position at Online and their university course-work. The employees were selected by interviewing applicants answering adverts placed throughout the student community.

The product was coded using Microsoft Visual Basic and SQL in a Microsoft development environment. The customer for the project was internal at Online Telemarketing and no considerations were made for eventually selling the product outside the company.

The size of the product is estimated to approximately 10 000 lines of code after all the initial functionality has been developed. The development was started in December 2000 and the first functional system was launched in mid April 2001. The system has been in full commercial operation since the end of August 2001.

5.3 Roles

The roles described in section 2 were assigned to the various members of the team at the start of the project. Of course, the employed developers assumed the roles of programmer. They also assumed the roles of testers, working together with the customers to create and run functional tests. The senior management at Online assumed the role of customer, as they were the people who had the original idea of the system. The tracker's responsibilities were assumed by the IT executive at Online as he had a good overview of the work performed by the group and was in direct contact with the developers daily. The coach role was assumed mainly by the IT executive, but at the beginning of the project, when XP was new to the team, the author shared some of the coach's responsibilities. Finally the Online senior management also assumed the roles of bosses for the project as they were providing all means for the development, such as computers, location and funding.

5.4 Configuration management

The configuration management was solved by a simple solution. As there were no branches in the configuration management and the system was relatively small, the team used a checkout directory to copy source code manually instead of using a tool for this purpose. This solution proved effective during the first part of the project when only two pairs of programmers were working. Common sense, combined with the fact that all the developers were in the same room, made sure that the configuration management worked well. As the product grew, and the number of developers doubled, problems did arise on occasion. One of the effects of the problems was work being deleted on a few occasions due to versions overwriting each other because of misunderstandings. When this showed to be causing problems for the developers, a quick and dirty solution was introduced. Using simple text files to administrate copies to checkout directories, the problem was solved.

Awareness of what was happening in the product was intended to be handled by the developers sitting in the same room and communicating all the time. The problem that became apparent with this strategy was when people were absent or working different schedules.

Code is integrated continuously several times a day and several times for each task. The alternative of employing a tool for the configuration management might, in retrospect, have been a more effective solution. The basic system of copying files to checkout folders solves the basic issues addressed by these tools and, as no configuration branches were to be used at all, the simple solution worked once the communication problems were fixed.

6 Experiences of the XP practices

This section discusses the experiences gathered through the observations and interviews made at Online Telemarketing regarding each specific XP practice. The developers were introduced to XP by a half-day seminar with a basic introduction and an *extreme hour* exercise [10]. The extreme hour is an exercise developed to demonstrate the planning game and gives a good overview of how XP works in practice. The developers had guidance from the coach regarding how they should implement XP at all times. XP books [1, 11, 12] were also made available to them. The developers were also instructed to look at XP websites to keep up to date on recent developments in the XP community [10, 13, 14, 15].

6.1 The planning game

Using story cards proved to be one of the greatest successes of all the XP practices. The story cards provided all parties involved with a picture of the status of the work and an overview of the product as a whole. Approximately 150 stories have been implemented in total. The stories were written by the customer and then prioritised together with the development manager as he had the best overview of the technical status of the product. The estimation worked well once the management understood the three levels of prioritisation [1, 2, 3].

New stories were added continuously during the whole project. This was due to the fact that the management did not have a clear picture of the product at the start of the project. This meant that functionality was continuously added during the entire project. The time estimation of the stories was difficult at first due to the lack of practical experience of estimating, but after a few weeks the estimating worked very well according to the group members. The estimation quality was not confirmed using quantitative methods. The whole group

performed estimations together during planning meetings.

Breaking the stories into tasks was difficult for the developers to grasp. The developers ended up drawing flow charts for the work, which was not the idea. Some of the story cards were very similar to tasks, i.e. at a too detailed a level for story cards. The problem was thought to be due to the difficulty of setting some kind of a common detail level for the stories.

The developers selected the stories to develop in conjunction with the development manager. This way of working with stories and tasks is an area that was continuously looked at and improved during the course of the project.

6.2 Small releases

Creating a minimal framework for each part of the system proved to take longer than the following smaller releases. The very first iteration took much more time than intended due to lack of experience in using the XP methods and traditional development thinking dominating. Once a complete bare working system was implemented, however, small releases were easier to implement.

During the long initial releases it was important to keep good communication between the customers and developers so that the project did not proceed in the wrong direction. As an afterthought, this practice seems fundamental to the success of XP. Maybe more effort should have been exerted to keep the initial release time shorter.

6.3 Metaphor

The system metaphor created before the actual start of the development was a little too detailed. It was almost an attempt at a complete requirements document. This was partly due to the fact that this document was written before the XP methodology was first thought of for the project. The document was not altered after XP was selected as the preferred development method. The metaphor document was also not properly updated as the system evolved during the course of the project. This is most probably also due to the too detailed level of the system metaphor. A common picture of the system was gained throughout the project by looking at the system directly and discussing individual cards. This common picture could have been improved by creating an accurate system metaphor.

6.4 Simple design

The development team has strived to implement the simplest possible solution at all times in accordance with this XP practice. A further evaluation of this practice was deemed to be difficult to perform in a reasonable amount of time.

The philosophy of always assuming simplicity was thought to have saved time in the cases where a much larger solution would otherwise have been implemented. Time was also believed to have been saved due to the fact that developers did not have to cope with a lot of unnecessarily complicated code.

6.5 Testing

Test-first programming was difficult to implement at first. Determining how to write tests for code proved difficult to master. The developers thought that the tests were hard to write and they were not used to thinking the test-first way. It was found difficult to see how many tests were enough to satisfy that the desired functionality would be implemented correctly.

The VB Unit test structure [12, 13] was used to create the automatic unit tests. VB Unit takes quite a long time to get used to and set up according to the developers. The unit tests that were written take less than one minute to run in total. The whole set of tests were run each time new code was integrated. During the course of the project the developers started to ignore writing tests first, especially when the project came under time pressure a few months in. The developers understood why tests are important but thought it involved too much work and did not see the short term benefits. It is believed that this was due to the inexperience of the developers. A more rigorous approach to the testing practices would most probably have been preferable.

The developers found programming by intention difficult. Programming by intention involves deciding the functionality and structure of the code in advance so that the test cases can be created beforehand. The development manager, who is experienced in coding these kinds of systems, found this way of working natural. He actually found that the way he usually worked was very close to the way described by XP. It was found that database code was much easier to write test code for than business rule code. The graphical user interface (GUI) code was also, as expected beforehand, hard to write automated tests for. Because of the limited nature of the GUI it was decided that an automated test tool for GUI testing would probably take longer to take into practice than manual user testing.

As the project came under pressure to release the fully operational version, the test first method of working ceased completely. The time pressure was due to the expansion of the company into a new region earlier than first expected. This meant that a portion of the new system was desired to go into operation earlier than the initial planning.

The functionality of the system was tested by the

customer before each release as well as spontaneously during the development. When the functionality was not as the customer had intended it, a correction card was written. At first the customer just interrupted the developers when they found the functionality inconsistent with the desired functionality, but this was found to be too disruptive so a correction card strategy was adopted. The functional testing provided a good view of how the product is progressing.

6.6 Refactoring

No tools for refactoring were used in the project. All project members performed minor refactoring continuously. No major refactoring of the code was performed, but assessment of the code was performed continuously regarding the benefits of a major refactoring in case it was necessary. No education or training was given either beforehand or during the course of the project in refactoring methods or theory such as those presented by Fowler [16].

6.7 Pair programming

The developers used pair programming at all times. The only exceptions were when illness intervened or the developers had demanding schedules at the university. The developers adopted pair programming cautiously at first, but then gradually started to work naturally and effectively in the pairs.

The fact that the developers had no prior professional experience probably made the introduction of pair programming much easier than if they had been used to working in a traditional single-programmer manner.

When alone, the programmers often seemed to seize and get stuck when solving a problem. Also the tendency to carry on with a nonworking solution seemed more frequent. The developers found it easier to keep their concentration on the task at hand when working in pairs.

The development leader estimates that the pair programming produced the code faster than if the same programmers would be working separately. However the inexperience of the developers made them much slower than experienced professionals.

The pair programming worked excellently when introducing new people into the project. For the first part of the project the pairs were been fixed so that the developers could synchronise their schedules easily, but during the second phase of the project when 8 people were working full time on the project, the pairs were changed continuously. The original 4 developers also chose their own pair-programming buddy, but the second group were assigned into pairs by management.

6.8 Collective ownership

Collective ownership worked well in the project. This contributed to solving some minor irritation among the developers due to defects found in the code. When the programmers thought of defects as a group issue, rather than someone else's 'private' defect the irritation disappeared and a constructive atmosphere was created. The only problem observed in this practice was due to the configuration management or rather lack of effectiveness in the communication in the handling of the configuration management. The developers were on occasion afraid to change parts of the code due to the risk of loosing work if not in direct contact with the other pairs.

6.9 Continuous integration

Continuous integration proved to be natural in the development environment created for the project. As soon as code was finished it was integrated into the product. The ease with which this practice was implemented is notable in itself.

6.10 40-hour week

As the developers all worked part time, 20-hours per week, this practice was adjusted to accommodate this. Only the development manager and senior management worked full time.

6.11 On site customer

The customer was available throughout the course of the project. This worked very well. The only problems were the flexible work hours of both developers and management and everyone's busy schedules. While the senior management of the company had the role of customer, they were not been able to devote all of their available time to this project, because of other meetings and responsibilities in running the company. At the start of the project the customer had many opinions on the functionality in the product. As soon as a release was made the customer wanted to modify or add to it. This decreased during the course of the project, partly due to the system evolving into what the customer wanted and partly due to that the customer became better at writing story cards describing the desired functionality to the developers more efficiently.

6.12 Coding standards

A coding standard document was created at the start of the project. This was used extensively at first and added to when needed. After a while the developers became more relaxed and used the coding standard less. This

was at the time identified as an issue and was re-enforced with success. The outcome of this practice has however not been evaluated by comparing sections of the actual code with the coding standard.

7 Quality of conclusions

In this section the criteria discussed in section 3, methodology, are discussed with regards to the research methodologies employed in this paper.

- *Credibility*

  The fact that both interviews and observations were used in the study increases credibility. The resulting observations do not seem to be incredible. The resulting observations seem to be correct when reviewed by the development manager at Online Telemarketing.
- *Transferability*

  The experiences from introducing XP in this project should be of considerable help to other projects introducing XP, either in part or fully. Consideration should be taken to the facts that the developers were working part time and were otherwise university students, not full time, experienced professionals.
- *Dependability*

  Due to the limited nature of the study in this experience report it is difficult to assess the dependability of the study.
- *Confirmability*

  The confirmability is also increased by the review by the development manager at Online Telemarketing.

The quality of the conclusions is increased by the triangulation of qualitative research methods. Both interviews and direct observations were used and the results were reviewed by a representative of the participating subjects.

8 Summary and conclusions

In conclusion, the project at Online Telemarketing was a success. The product was created and is now functioning live. The experiences of the actual XP practices are a mostly successes, but also a few failures. All of these experiences are relevant to projects considering introducing XP.

The planning game was easy to introduce and a great success. This can be partly due to the fact that the extreme hour, used to initially introduce XP, focuses on the planning game, as does extensive parts of the XP literature [e.g. 1, 11, 12].

Small releases proved difficult for the first releases for each part of the system. Even though they were expected to take a little longer than the rest of the releases, they took longer than planned. An increased focus on only creating an absolute minimal framework system might help this.

The system metaphor was too complicated to start with. This resulted in a document that did not keep up with the evolution of the system. It should not be difficult to keep the metaphor simple and up to date.

The simple design was thought to work well, but was not verified by code inspections. The developers believed that by thinking in terms of simple solutions as much as possible, they saved a lot of time by not having to try to understand unnecessarily complicated code.

Testing was one of the hardest practices to implement. It requires careful preparation of the testing unit and also a strict discipline among the testers to always test first. The testing practice was the first practice to cease when the project came under pressure.

Refactoring was performed on a small scale all the time. This is, however, natural in normal programming. Larger scale refactoring was not performed, although the possibility of large scale refactoring was continuously evaluated.

Pair programming worked excellently for the developers in the project. It seemed to help them solve difficult problems faster and identify potential dead end solutions earlier. The pair programming also worked very well when introducing new people into the project

Although it was different from what the developers were used to from the start, collective ownership proved to be effective for the team spirit.

Continuous integration was not hard to implement and was found a natural way to work in the development environment created in the project.

The on-site customer practice worked well. The customer solved many misunderstandings of functionality early and was available to complete or clarify any poorly written story cards. As the customer did not really know the full extent of the product at the start of the project, this practice appears to be one of the major reasons for the success of the project.

The coding standard worked well. When the developers started to get sloppy in the middle of the project, the development manager enforced the coding standard again.

Keeping in mind the issues raised in the quality of conclusions section, section 7, these experiences should be of interest to any development team considering introducing XP.

9 Acknowledgements

This work was partly funded by The Swedish Agency for Innovation Systems (VINNOVA), under a grant for the Center for Applied Software Research at Lund University (LUCAS). The author would also like to

thank Johan Norrman (Online Telemarketing) and Per Runeson (LTH) for their contributions to the paper. Finally the comments of the SERP 01 reviewers have contributed to improving the quality of the paper.

10 References

[1] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison Wesley, 1999.

[2] Beck, K., "Embracing Change with Extreme Programming", *IEEE Computer*, October 1999, pp. 70-77.

[3] Martin, R., C., "Extreme Programming Development through Dialog", *IEEE Software*, July/August 2000, pp.12-13.

[4] Haungs, J., "Pair Programming on the C3 Project", *IEEE Computer*, February 2001, pp. 118-119.

[5] Hicks, M., *XP Pros Take it to the Extreme*, ZDNet eWeek News, last confirmed 010903, http://www.zdnet.com/eweek/stories/general/0,11011,2714342,00.html .

[6] Lincoln, Y., S., Guba, E., G., *Naturalistic Inquiry*, Sage Publications, 1985.

[7] Robson,C., *Real World Research*, Blackwell Publishers, Oxford, 1993.

[8] Miles, M.B., Huberman, A.M., *Qualitative Data Analysis*, Sage Publications, 1994.

[9] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A., *Introduction to Experimentation in Software Engineering*, Kluwer Academic Publishers, 2000.

[10] Multiple Authors, *The Extreme Programming Roadmap*, last confirmed 010903, http://www.c2.com/cgi/wiki?ExtremeProgrammingRoadmap .

[11] Beck, B., Fowler, M., *Planning Extreme programming*, Addison Wesley, 2000.

[12] Jeffries, R., Anderson, A., Hendrickson, C., *Extreme Programming Installed*, Addison Wesley, 2000.

[13] Jeffries, R., (Ed.), *XProgramming.com*, last confirmed 010903, http://www.xprogramming.com .

[14] Wells, D., *Extreme Programming: A Gentle Introduction*, last confirmed 010903, http://www.extremeprogramming.org/ .

[15] Multiple authors, *Extrem Programmering*, (in Swedish), last confirmed 010903, http://oops.se/cgi-bin/wiki?ExtremProgrammering .

[16] Fowler, M., *Refactoring: Improving the Design of Existing Code*, Addison Wesley, 2000.

Software Engineering at System Level

Asmus Pandikow and Anders Törne
Real-Time Systems Laboratory, Software and Systems Division
Department of Computer and Information Science
University of Linköping
581 83 Linköping, Sweden
{asmpa , andto}@ida.liu.se

Abstract

This paper presents work that provides the means to technically bridge the gap between software and systems engineering methods. It specifically focuses on modern object oriented software engineering techniques from a system perspective. The problems caused by the gap between software and systems engineering methods are described as the difficulty to trace software engineering specifications from a system perspective down to concept level and the difficulty of information exchanges between mismatching methods. Four of the current approaches that tackle these problems are briefly presented and evaluated, resulting in the conclusion that currently there is no solution free from side-effects. A new approach bridging the gap between software and systems engineering is being presented, based on the efforts to create the forthcoming systems engineering ISO standard AP-233 and the just initiated work on a UML systems engineering profile. The approach allows to inter-link software and systems engineering techniques. It enables the traceability of single software specification elements from a systems perspective and, on the other hand, the use of systems engineering management capabilities of AP-233 in UML based developments.

1. Introduction

Software engineering has in the recent years grown to a fully-fledged engineering discipline. Software specific engineering processes, methods and specification notations have emerged and established. Additionally, to cope with the increased requirements on scalability, modularity and reuse capability, software engineering has even undergone a "paradigm shift" in the 1980's and 1990's, from previously structured development to object oriented analysis and design methods that require a different kind of thinking for developing software but provide the means to cope with the increasing requirements on software engineering. Within this period, the complexity of software systems themselves and the need for distributed development with reusable products has continuously grown up to a level where methods and tools to manage complexity were needed in order be able to keep track of things. Additionally, the portion of software in complex physical systems, such as cars, aircrafts and space vehicles, has continuously grown. Functionality that previously has purely been implemented with hardware is increasingly implemented with software.

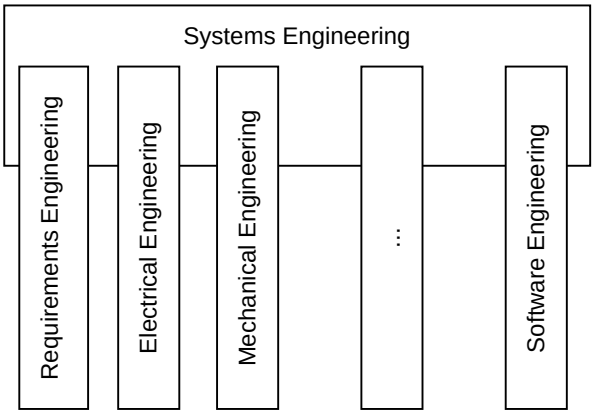


Figure 1. Systems Engineering Overview

In complex projects with interdisciplinary collaboration across several engineering disciplines, an overall system view is needed in order to manage problems of complexity and heterogeneity. The activities on this level are usually referred to as "systems engineering". Systems engineering can be summarized as encompassing and managing the engineering activities assigned to all stages of a system's lifecycle, as shown in

Figure 1. System-wide activities such as project management or version and configuration management are performed by the systems engineering discipline. Technical in-depth engineering activities are performed by the respective engineering discipline, e.g. mechanical engineering, electrical engineering or software engineering.

1.1. Paper Structure

The rest of the paper is structured as follows. Section 2 describes current problems with the integration of engineering techniques in a comprehensive systems engineering view, specifically focussing on modern object oriented software engineering. Then, the goals of this paper are outlined. Section 3 presents and evaluates current efforts towards integrating software specifications in a systems engineering view, respective viewing software engineering methods from a system perspective. The following section 4 provides an approach to bridge the gap that is still left open between software and systems engineering, even considering the above mentioned current efforts. The paper is finished with concluding remarks and an outlook on the author's future work in this area.

1.2. Term Definitions

Due to the nature of the work presented in this paper, certain terms have several, partly mismatching, definitions in different domains. The following terms are used in this paper following the given definition.

- **Technique**: A collection of methods and notations of a certain paradigm. For example, object oriented techniques include object oriented analysis and design but also the Unified Modeling Language (UML, see [12]) as notation.
- **Method**: A certain approach to solve a certain class of tasks. For example, Booch's method [2] is employed for analyzing and designing object oriented software systems.
- **Concept**: An element of a method. For example, classes are a concept of Booch's method. Concepts can consist of other methods, e.g. attributes are a concept of the concept class.

The defined terms can be viewed as being hierarchically related, techniques contain methods, methods in turn contain concepts and concepts may contain other concepts.

2. Problem Description

Although systems engineering has been performed since decades, the management of all artifacts of a system from a comprehensive systems engineering view is still difficult. Such a unified view, presumably implemented in the form of a central database, provides the means for overall system management capabilities, such as system-wide project management, version management or configuration management. It furthermore allows for system-wide inter-allocation across engineering disciplines, e.g. the allocation of a requirement to a low-level detail of a software specification, thus providing system-wide traceability of specification elements. With such a comprehensive view, product data management (PDM) techniques and tools could be applied across the engineering disciplines down to lower specification levels.

However, different engineering disciplines often employ different methods and concepts to specify their view of the system or a part of the system. The methods used differ from engineering discipline to engineering discipline, sometimes syntactically congruent but semantically different, e.g. the notation of state machines in structured design and in the UML. These cases make it especially difficult to create a unified systems engineering view encompassing the artifacts from all of the single engineering disciplines.

The gap between engineering methods is specifically apparent if software engineering is also to be included in such a comprehensive systems engineering view, as described in Cocks [3]. The integration of software engineering methods with other engineering methods comprises another difficulty, namely the fast pace of changes in software engineering techniques. It would be inappropriate to pin down the current status of software engineering for the integration efforts, because the techniques may change or even be obsolete shortly.

2.1. Goals

This paper specifically tackles the aspects of integrating modern object oriented software engineering at systems engineering level. The goals are to provide a solution of the above described problems allowing to integrate software engineering artifacts on system level and also, on the other hand, to enable the use of system engineering techniques for software engineering.

The presented solution aims at an integration on concept level, i.e. it considers file units as traceable

software artifacts as insufficient. Rather, the single elements of a software specification are to be accessible.

The discussions are, for the software engineering part, based on the UML Version 1.3, see [12]. The UML has emerged from several of the major object oriented software engineering techniques and can be considered to be the current de-facto standard in software engineering. An introduction to details of the UML can be found in Fowler and Scott [4].

2.2. Remarks

The integration problem described above can be seen from two perspectives.

First, the view from system level on software engineering highlights the difficulties of mapping modern object oriented software engineering notations to the traditional structured notations predominantly used in other engineering disciplines and systems engineering. Also, software engineering notations, such as the UML, often lack explicit support for version and configuration management, as this is often handled externally on file level in software engineering. This "snapshot" character of software engineering specifications represents another dissimilarity compared to systems engineering practice.

Second, the view from software engineering on the management capabilities of systems engineering needs also be taken into account. For example, employing the mature techniques from systems engineering for managing versions and configurations allows for handling issues such as system complexity and distributed development in software engineering.

3. Current Efforts

There are currently several efforts undertaken striving to solve the integration problem from either the software engineering or the systems engineering side. The following subsections each present and evaluate an approach integrating UML concepts with systems engineering views. The presented set of approaches is not complete, it rather contains the most important approaches, determined by the size of the associated projects and the number of related publications. Furthermore, "UML for Real-Time Control Systems" is presented as representative for smaller but similar efforts with interesting considerations.

3.1. AP-233

In the SEDRES projects, see SEDRES-2 Website [16], where the authors of this paper are involved, a

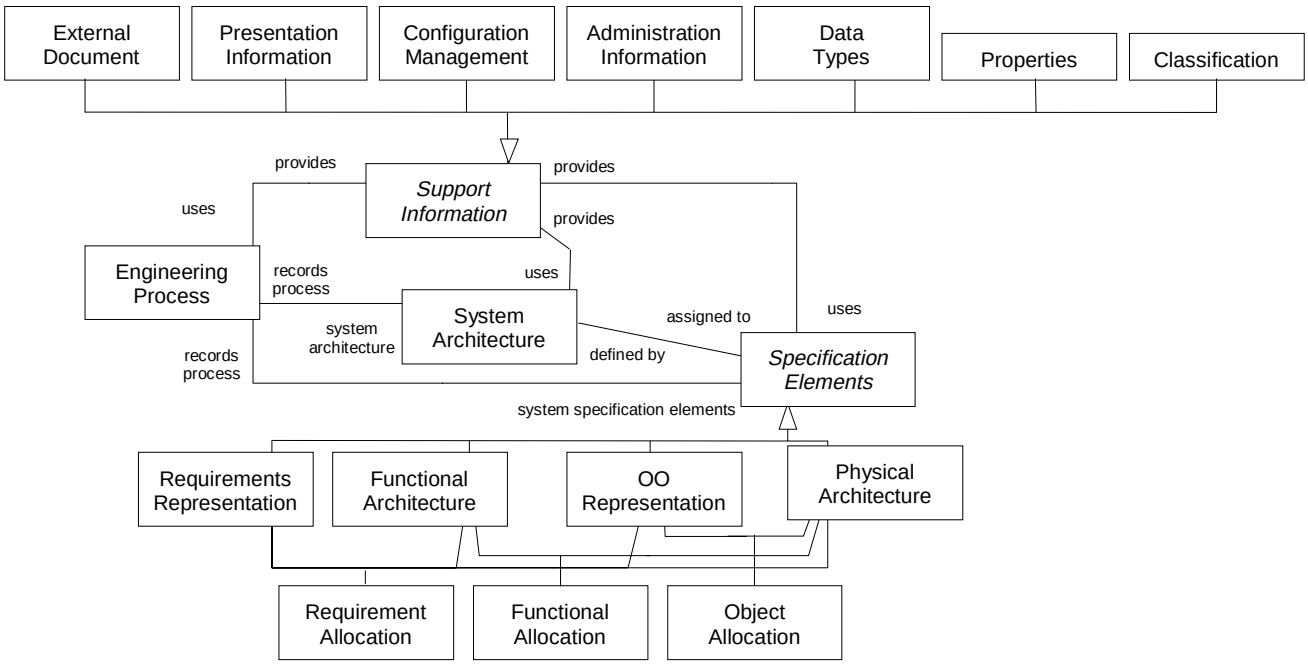


Figure 2. AP-233 Overview

comprehensive information model for data exchanges between systems engineering software tools has been created. On the basis of this information model, a working group (ISO AP-233, see [9]) within the International Standardization Organization (ISO, see [10]) has been formed in order to push the model towards an international standard for the exchange of systems engineering design tool data. The proposal for this forthcoming application protocol standard ISO 10303-233 (short: AP-233, see [15]) comprises the most important concepts supporting the current systems engineering practice.

Figure 2 shows a class diagram giving an overview of the working draft 5 of AP-233, as described in [15]. The system engineering areas that AP-233 provides support for are depicted as rectangles (classes), the relationships among them are depicted as lines (associations).

In summary, AP-233 provides support for system-wide management activities (such as project, version and configuration management), system architecture, the engineering process used within an organization to develop a system, and a number of specification elements from different engineering disciplines, including object oriented software engineering. The object oriented software engineering concepts encapsulated in "OO Representation" and "Object Allocation" are for the greater part based on the UML notation. A more detailed description of AP-233 can be obtained from Herzog and Törne [5] and [6], a description of the integration of object oriented concepts can be found in Pandikow and Törne [14].

The approach taken with AP-233 is different to the approaches presented in the following subsections. AP-233 tries to harmonize with existing concepts from systems engineering standards by including object oriented concepts (based on the UML) from a systems engineering perspective. The respective software engineering concepts are included in the management capabilities of AP-233, i.e. elements from software specifications can be addressed by system-wide functionality.

AP-233 takes a step in the right direction towards integrating software engineering concepts on system level. It shows how single software engineering elements can be included in systems engineering considerations and how object oriented concepts can be made available for a coherent use outside software engineering. However, AP-233 merely includes object oriented concepts instead of integrating them. A closer semantic integration with existing concepts of other areas of AP-233 would be desirable. This would allow for data exchanges between systems engineering tools supporting different techniques, either the object oriented or the structured paradigm.

Working draft 5 of AP-233 allows for the use of object-oriented concepts in heterogeneous system specifications and, on the other hand, also for the use of systems engineering capabilities in object-oriented specifications.

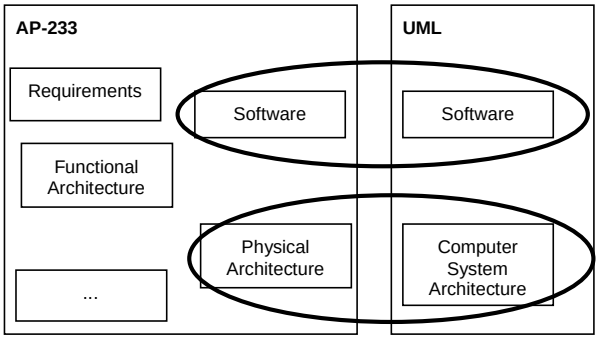


Figure 3. Standard Redundancy

Nevertheless, it has an intrinsic drawback, shown in Figure 3: AP-233 contains a set of software engineering concepts resembling the UML notation. The difficulty with this is that the UML is in flux, i.e. it will be updated and changed within short periods of time. In contrast, the forthcoming AP-233 will for the greater part be unchanged for at least five years as ISO standard. Hence, keeping the two standards consistent and synchronized with respect to their redundant areas is infeasible in the medium term.

3.2. UML Systems Engineering Profile

In the recent years, the International Council on Systems Engineering (INCOSE, see [7]), representing the worldwide systems engineering community, has recognized object orientation as alternative systems specification technique. Also, the need to integrate the increasingly important developments in software engineering with the traditional techniques used in systems engineering has been recognized.

INCOSE and the Object Management Group (OMG, see [13], responsible for the UML) are currently initializing the creation of a UML profile for systems engineering (UML SE profile). A UML profile is a standardized way to tailor the UML notation for specific purposes by restricting existing UML concepts for specific needs, as well as introducing new concepts. The scope and content of the UML SE profile are currently not yet completely defined.

This approach has the same potential as AP-233, to harmonize systems engineering practice with modern object oriented methods. However, the UML SE profile approach will, as AP-233, most likely implement redundant sets of specification concepts representing systems engineering artifacts already modeled in AP-233. Hence, the difficulty of keeping the two standards synchronized will also arise from this solution. However, in this case the sychnronization of the standards is easier to accomplish, because changes in the systems engineering practice are more unlikely to happen and easier to integrate in the anyway faster changing UML standard.

3.3. OOSEM

Lykins et al. [11] have developed an Object Oriented Systems Engineering Method (OOSEM) that adapts the UML notation for the use of engineering systems. The approach proposes solutions for the most important limitations of the UML when designing at system level, e.g. by extending the UML syntax in order to capture system requirements and by providing semantics where the specification of the UML is vague.

In summary, OOSEM is a new object oriented method for engineering systems, i.e. it allows to use concepts of modern object oriented software engineering concepts, such as use cases, to be used at system level. OOSEM is to a large extent based on the UML, thus it is obvious that it is easy to be employed and proven useful in software-intensive projects, as for example described in Steiner et al. [17].

Nevertheless, OOSEM does not attempt to harmonize with existing systems engineering standards. Furthermore, it does not explicitly focus on integrating software concepts in systems engineering views. Rather, it changes software concepts for the use at system level. Hence, this approach is not directly suitable for achieving the intended integration goals, although it contains useful extensions to the UML that improve the consistency of the notation.

3.4. UML for Real-Time Control Systems

An adaptation of the UML is developed and described by Axelsson [1]. It can be seen as a representative of several approaches from a software engineering perspective that extend the UML such that it can be used for modeling a certain aspect outside the original scope of the UML. The author proposes extensions to the UML that allow to use the notation for designing real-time control systems. Additional elements are created through UML stereotypes, i.e. with built-in means to extend the UML notation, in order to allow for a richer specification of physical architecture and for modeling continuous-time relationships. This approach bridges the gap that UML leaves in designing real-time control systems. It allows, alike the OOSEM approach, to use UML concepts to model additional aspects of software systems that are slightly outside the scope of the UML.

However, the real-time adaptations do not strive for semantic harmonization with existing systems engineering concepts and hence, do not provide the means for improved interdisciplinary collaboration based on common semantics.

3.5. Summary

None of the above presented approaches provides a solution to the integration problems being free from side-effects. The most interesting approaches, AP-233 and the UML Profile for Systems Engineering, each model the interesting elements of the respective other domain within their own domain, generating a redundant representation. This constitutes the major problem of these approaches, namely to keep the redundant models synchronized. This, in turn, is especially difficult as the chosen underlying software engineering notation is the UML, which is in the flux.

Nevertheless, different aspects of each approach contain contributions to an overall solution.

The adaptations of the UML, such as the presented OOSEM or “UML for Real-Time Control Systems”, do not primarily focus on the harmonization of systems and software engineering standards in order to allow for mutual use of specification concepts. Each of the adaptations needs to be examined individually whether its contributions may be useful to extend the core specification of the UML in its meta-model or as formal UML extension, e.g. in the form of a UML profile.

Integrating software engineering concepts in systems engineering management capabilities as performed in AP-233 is on principle a viable solution. The major drawback of the AP-233 approach is, as mentioned above, to keep the two independent standards synchronized, which in fact is practically infeasible.

The approach of creating an UML profile for systems engineering also represents a viable solution, but with the same difficulty to keep two standards synchronized, as with AP-233.

4. Approach

A combination of aspects of the AP-233 and the UML systems engineering profile approaches circumvents the common sychnronization problem. Figure 4 gives an overview on how the approach could be implemented.

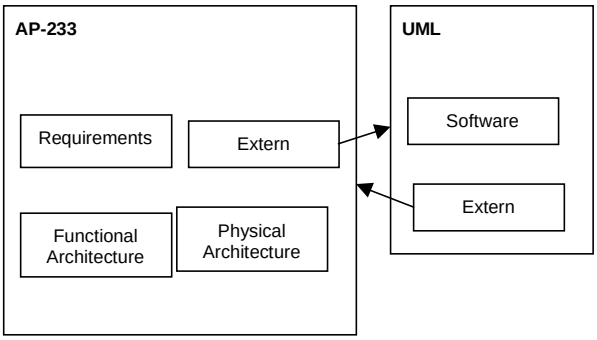


Figure 4. Solution Overview

Instead of keeping redundant implementations of respective opposite concepts in each standard, as shown in Figure 3, each standard implements an interface for accessing external capabilities and specification elements. AP-233 system specifications can access UML (software engineering) specification elements through its external interface, and, vice-versa, UML specifications can access AP-233 capabilities through the UML external interface. This eliminates the redundancy and hence the sychnronization problem, as long as the interfaces refer to the core elements of the respective other standard that are unlikely to change.

The following subsections describe the necessary implementations for this approach for each of the two standards.

4.1. AP-233 External Interface

Implementing the above described approach comprises for AP-233 (in its form described in working draft 5 [15]) the following modifications. First, all entities representing object oriented techniques need to be removed, the same applies to references to these entities within the AP-233 information model. Second, the interface for external accesses has to be included in the information model.

Figure 5 outlines a possible implementation of the interface in the form of an EXPRESS-G diagram (EXPRESS is a specification language defined in the STEP standard framework ISO 10303, namely ISO 10303-11 [8]; EXPRESS-G is a notation allowing to represent subsets of EXPRESS graphically). The entities "label", "id" and "version" refer to respective definitions in AP-233 working draft 5 [15].

The principal of the interface is to introduce a new entity for referring to external elements, may it be an element that is supposed to be included in system-wide management capabilities ("controlled_external_element"), as shown for version management, or just a reference in order to provide simple traceability between system and software specifications.

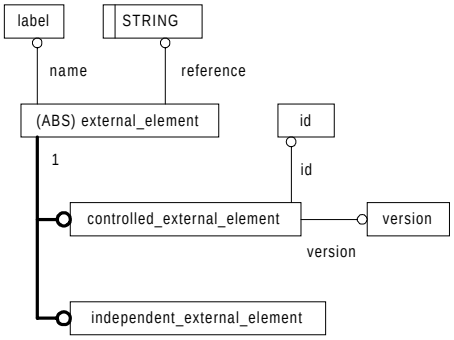


Figure 5. AP-233 External Interface

The proposed implementation can be either introduced as new constructs, as shown in Figure 5, or by extending the existing AP-233 interface to external documents through a common superclass embracing external documents and (the new) references to external elements.

4.2. UML SE Profile External Interface

To complete the bi-directional integration, the forthcoming UML systems engineering profile needs to provide a similar interface as proposed for AP-233 above. Figure 6 shows the major changes required in the UML core meta-model providing the described capabilities of referring to UML external specification elements.

The upper half of the class diagram in Figure 6 contains a proposal for a general external element that has a location outside the UML model. The location element "ExternalElementLocation" may be further specialized in subclasses in order to describe a specific location, e.g. a reference to an entity of AP-233. In order to reveal the binding of an external element, the UML core should also be extended by an explicit model management as proposed in the lower half of Figure 6. A "Model" would explicitly correspond to one of the UML diagrams, such as a use case diagram or a class diagram.

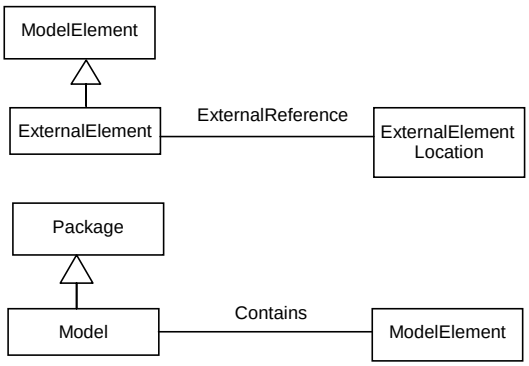


Figure 6. UML SE Profile External Interface

This extension of the UML core meta-model allows for the integration of external modeling elements in UML models. In this way, the management capabilities of AP-233 can be accessed from UML models, e.g. in order to employ the AP-233 version and configuration management.

4.3. Examples

The following examples show how the mutual integration of system and software engineering concepts would be realized using the presented approach.

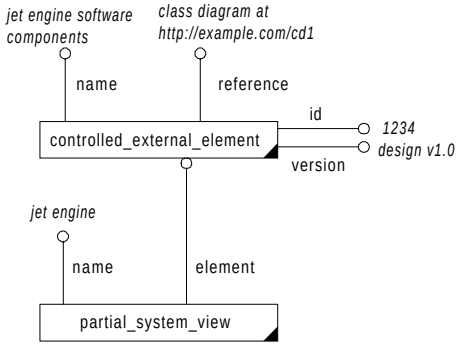


Figure 7. AP-233 Interface Example

Figure 7 shows a simplified example of a jet engine that may be part of a large system, e.g. an aircraft. The EXPRESS-G notation used for this example has been modified in order to represent instances of a class (shown with a black triangle in the lower right corner) and attribute values (shown as cursive text associated with a relationship). Furthermore, the relationship between "partial_system_view" and "controlled_external_element" is also simplified, compared to the modeling style in working draft 5 of AP-233 [15]. The interesting connection between the AP-233 specification and a UML class diagram describing the major software components of the jet engine's control software is done via the attribute "reference".

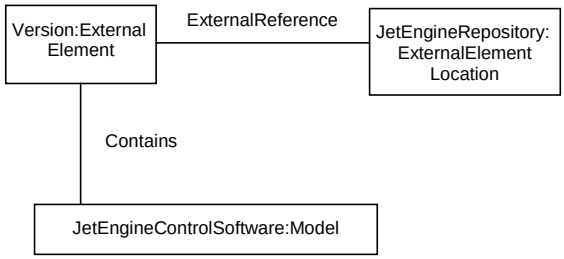


Figure 8. UML SE Interface Example

Figure 8 shows how, on the other side, UML specifications can access AP-233 elements through the proposed UML SE External Interface.

In this simplified example the UML model "JetEngineControlSoftware" accesses a specific AP-233 version, which is stored in an AP-233 based repository, through an "ExternalElement".

4.4. Evaluation

The presented approach provides the means to integrate AP-233 system specifications and UML software specifications without being dependent on the asynchronous evolution cycles of the two standards. AP-233 specifications can include UML specifications down to concept level, thus providing a system-wide traceability. On the other hand, UML specifications can access modeling elements and capabilities of an AP-233 specification, allowing to include concepts that are not covered by the UML.

Nevertheless, the major advantage of the proposed solution, namely independence from updating cycles of the respective standards, takes only effect if the mutually referenced concepts are carefully selected with respect to their probability of change. For example, the concept "class" can be considered to be unchanging throughout the next versions of the UML, and hence, is a good candidate for being included in the interface definitions.

5. Conclusions and Future Work

In this paper, the gap between software and system engineering and the resulting problems have been presented. The gap mainly originates in the increasingly diverging techniques used in the respective areas since the

introduction of object oriented software engineering. The resulting problems lie in the difficulties of tracing single elements of software specifications from system level and the difficulty to exchange information between software and systems engineering due to the mismatch of their techniques and methods.

Four of the current approaches tackling these problems have been presented and evaluated, leading to the conclusion that none of the approaches provides a solution free from major side-effects.

The integration of software engineering concepts with a systems engineering view performed in AP-233 and the potentials of the initiative to create an UML systems engineering profile have been combined into a new approach that reduces the above mentioned side-effects, namely the problem of keeping redundant parts in two independent standards consistent.

The primary author of this paper will continue to work in this field, focussing on the technical integration of object oriented techniques with traditional structured techniques. The intention is to show that object orientation does not represent a new paradigm, rather than a logical extension of traditional techniques in order to cope with the demands of complex systems and hence, can also be used outside software engineering.

6. References

[1] Axelsson, J.: "Unified Modeling of Real-Time Control Systems and their Physical Environments Using UML" in the proceedings of the "Eighth IEEE International Conference and Workshop on the Engineering of Computer Based Systems", IEEE Computer Society, 2001

[2] Booch, G.: "The Evolution of the Booch Method", Report on Object Analysis and Design, pages 2-5, May / June 1994

[3] Cocks, D.: "The Suitability of Using Objects for Modeling at the Systems Level" in the Proceedings of the Ninth Annual International Symposium of the International Council on Systems Engineering, pages 1047-1054, INCOSE, 1999

[4] Fowler M. and Scott K., "UML Distilled – Applying the Standard Object Modeling Language", Addison Wesley, 1997

[5] Herzog, E. and Törne, A.: "Towards a Standardised Systems Engineering Information Model" in the "Proceedings of the Ninth Annual International Symposium of the International Council on Systems Engineering", INCOSE, 1999

[6] Herzog, E. and Törne, A.: "AP-233 Architecture" in the "Proceedings of the Tenth Annual International Symposium of the International Council on Systems Engineering", INCOSE, 2000.

[7] INCOSE Website: Internet homepage of the International Council on Systems Engineering at http://www.incose.org, INCOSE, 2001

[8] ISO 10303-11: "Industrial automation systems and integration - product data representation and exchange - part 11: Description methods: The express language reference manual", Technical Report ISO 10303-11:1994(E), ISO, Geneva, 1994

[9] ISO AP-233 Website: Internet homepage of the AP-233 Systems Engineering Working Group at http://www.sedres.com/ap233/sedres_iso_home.html, hosted by the SEDRES project, 2001

[10] ISO Website: Internet homepage of the International Standardization Organization at http://www.iso.ch, ISO, 2001

[11] Lykins H., Friedenthal S, Meilich A.: "Adapting UML for an Object Oriented Systems Engineering Method (OOSEM)" in the "Proceedings of the Tenth Annual International Symposium of the International Council on Systems Engineering", INCOSE, 2000.

[12] OMG UML Specification v1.3, OMG Website at http://www.omg.org, 2000

[13] OMG Website: Internet homepage of the Object Management Group at http://www.omg.org, OMG, 2000.

[14] Pandikow, A. and Törne, A.: "Support for Object-Orientation in AP-233" in the Proceedings of the 11th Annual International Symposium of the International Council on Systems Engineering, INCOSE, 2001

[15] SEDRES-2 AP-233 proposal: "AP-233 Working Draft 5", SEDRES-2 Website at http://www.sedres.com/ documents/sedres_all_documents.html, SEDRES-2 project, 2001

[16] SEDRES-2 Website at http://www.sedres.com, SEDRES-2 project, 2001

[17] Steiner R., Friedenthal S., Oesterhel J. and Thaker G, "Pilot Application of the Object Oriented System Engineering Method (OOSEM) Using Rational Rose Real Time to the Navy Common Command and Decision (CC&D) Program" in the Proceedings of the 11th Annual International Symposium of the International Council on Systems Engineering, INCOSE, 2001

Acknowledgements

The authors gratefully acknowledge the hard work of the participants in the SEDRES project and the financial support from the European Commission for the SEDRES projects.

Experiences with Component-Based Software Development in Industrial Control

Frank Lüders and Ivica Crnkovic
Department of Computer Engineering
Mälardalen University
frank.luders@mdh.se

Abstract

When different business units of an international company are responsible for the development of different parts of a large system, a component-based software architecture may be a good alternative to more traditional, monolithic architectures. The new common control system, developed by ABB to replace all its existing control systems, must incorporate support for a large number of I/O systems, communication interfaces, and communication protocols. An activity has therefore been started to redesign the system's architecture, so that I/O and communication components can be implemented by different development centers around the world. This paper reports on experiences from this effort, describing the system, its current software architecture, the new component-based architecture, and the lessons learned so far.

1. Introduction

Increased globalization and the more competitive climate make it necessary for international companies to work in new ways that maximize the synergies between different business units around the world. Interestingly, this may also require the software architecture of the developed systems to be rethought. In a case where different development centers are responsible for different parts of the functionality of a large system, a component-based architecture may be a good alternative to the more traditional, monolithic architectures, usually comprising a large set of modules with many visible and invisible interdependencies. Additional, expected benefits of a component-based architecture are increased flexibility and ease of maintenance [1][2].

This short paper reports on experiences from an ongoing project at ABB to redesign the software architecture of a control system to make it possible for different development centers to incorporate support for different I/O and communication systems. While it is obvious that the component-based approach in the long run brings advantages in terms of time-to-market and less costs for system adaptability and improvements, it is also clear that the redesign itself and the additional costs for designing components to be reusable require more costs in the beginning of the process [3]. Minimizing the additional costs of the project in its starting phase was one of the main challenges. The second challenge of the project was to achieve a good design of the architecture where the interfaces between reusable parts are clear and sufficiently general. The third challenge was to keep the performance of the existing system, since the separation of system parts and introduction of generic interfaces between the parts may cause overhead in the code execution.

The remainder of the paper is organized as follows. In section two, the ABB control system is described with particular focus on I/O and communication. The software architecture and its transformation are described in more detail in section three. In section four, we analyze the experiences from the project and try to extract some lessons of general value. Section five reviews some related work in this area, and section six present our conclusions and outlines future work.

2. The ABB control system

Following a series of mergers and acquisitions, ABB now has several independently developed control systems for the process, manufacturing, substation automation and related industries. To leverage its worldwide development resources, the company has decided to continue development of only a single, common control system for these industries. One of the existing control systems was selected to be the starting point of the common system. This system is based on the IEC 61131-3 industry standard for programmable controllers [4]. The software has two main parts, the ABB Control Builder, which is a Windows application running on a standard PC, and the system software of the ABB Controller family, running on top of a real-time operating system (RTOS) on special-purpose hardware. The latter is also available as a Windows application, and is then called the ABB Soft Controller.

The ABB Control Builder is used to specify the hardware configuration of a control system, comprising one or more ABB Controllers, and to write the programs that will execute on the controllers. The configuration and the control programs together constitute a control project. When the control project is downloaded to the control system via the control network, the system software of the controllers is responsible for interpreting the configuration information and for scheduling and executing the control

programs. Only periodic execution is supported. Figure 1 shows the Control Builder with a control project opened. It consists of three structures, showing the libraries used by the control programs, the control programs themselves, and the hardware configuration, respectively. The latter structure is expanded to show a configuration of a single AC800M controller, equipped with an AI810 analogue input module, a DO810 digital output module, and a CI851 PROFIBUS-DP communication interface.

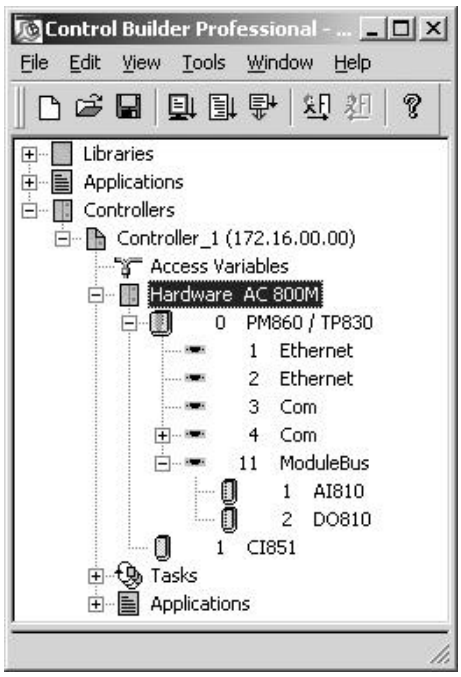


Figure 1. The ABB Control Builder.

To be attractive in all parts of the world and a wide range of industry sectors, the common control system must incorporate support for a large number of I/O systems, communication interfaces, and communication protocols. In the current system, there are two principal ways for a controller to communicate with its environment, I/O and variable communication. When using I/O, variables of the control programs are connected to channels of input and output modules using the Control Builder. For instance, a Boolean variable may be connected to a channel on a digital output module. When the program executes, the value of the variable is transferred to the output channel at the end of every execution cycle. Variables connected to input channels are set at the beginning of every execution cycle. Real-valued variables may be attached to analogue I/O modules.

To configure the I/O modules of a controller, variables declared in the programs running on that controller is associated with I/O channels using the program editor of the Control Builder. Figure 2 shows the program editor with a small program, declaring one input variable and one output variable. Notice that the I/O addresses specified for the two variables correspond to the position of the two I/O modules in Figure 1.

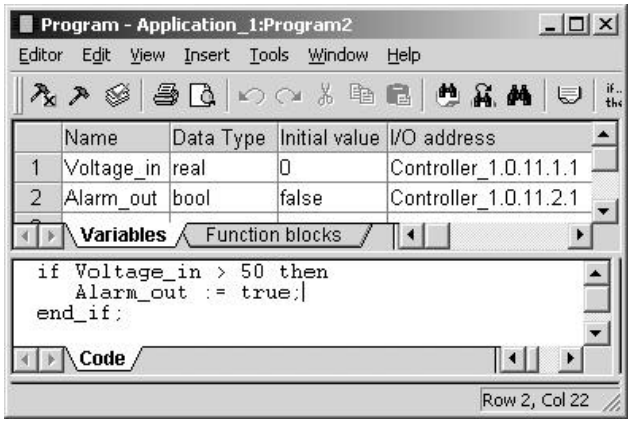


Figure 2. The program editor of the Control Builder.

Variable communication is a form of client/server communication and is not synchronized with the cyclic program execution. A server supports one of several possible protocols and has a set of named variables that may be read or written by clients that implement the same protocol. An ABB Controller can be made a server by connecting program variables to so-called access variables in a special section of the Control Builder. Servers may also be other devices, such as field-bus devices. Any controller, equipped with a suitable communication interface, can act as a client by using special routines for connecting to a server and reading and writing variables via the connection. Such routines for a collection of protocols are available in the Communication Library, which is delivered with the Control Builder.

3. Componentization

3.1. Current software architecture

The software of the ABB Control System consists of a large number of source code modules, each of which are used to build the Control Builder or the controller system software or both. Figure 3 depicts this architecture, with emphasis on I/O and communication. The boxes in the figure represent logical components of related functionality. Each logical component is implemented by a number of modules, and is not readily visible in the source code.

To see the reason for the overlap in the source code of the Control Builder and that of the controller system software, we look at the handling of hardware configurations. The configuration is specified using the control builder. For each controller in the system, it is

specified what additional hardware, such as I/O modules and communication interfaces, it is equipped with. Further configuration information can be supplied for each piece of hardware, leading to a hierarchic organization of information, called the hardware configuration tree. The code that builds this tree in the Control Builder is also used in the controller system software to build the same tree there when the project is downloaded. If the configuration is modified in the Control Builder and downloaded again, only a description of what has changed in the tree is sent to the controller.

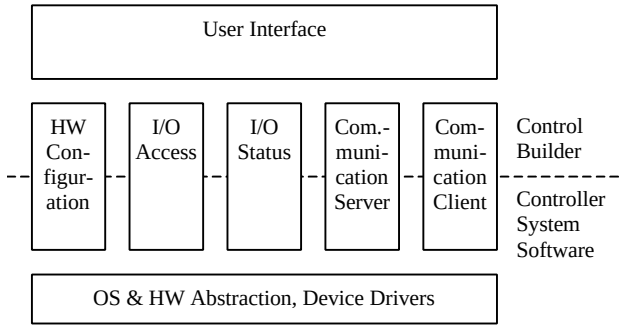


Figure 3. The current software architecture.

The main problem with the current software architecture is related to the work required to add support for new I/O modules, communication interfaces, and protocols. For instance, adding support for a new I/O system may require source code updates in all the components except the User Interface and the Communication Server, while a new communication interface and protocol may require all components except I/O Access to be updated.

As an example of what type of modifications may be needed to the software, we consider the incorporation of a new type of I/O module. To be able to include a device, such as an I/O module, in a configuration, a hardware definition file for that type of device must be present on the computer running the Control Builder. For an I/O module, this file defines the number and types of input and output channels. The Control Builder uses this information to allow the module and its channels to be configured using a generic configuration editor. This explains why the user interface does not need to be updated to support a new I/O module. The hardware definition file also defines the memory layout of the module, so that the transmission of data between program variables and I/O channels can be implemented in a generic way.

For most I/O modules, however, the system is required to perform certain tasks, for instance when the configuration is compiled in the Control Builder or during start-up and shutdown in the controller. In today's system, routines to handle such tasks must be hard-coded for every type of I/O module supported. This requires software developers with a thorough knowledge of the source code. The situation is similar when adding support for communication interfaces and protocols. The limited number of such developers therefore constitutes a bottleneck in the effort to keep the system open to the many I/O and communication systems found in industry.

3.1. Component-based software architecture

To make it much easier to add support for new types of I/O and communication, it was decided to split the components mentioned above into their generic and non-generic parts. The generic parts, commonly called the generic I/O and communication framework, contains code that is shared by all hardware and protocols implementing certain functionality. Routines that are special to a particular hardware or protocol are implemented in separate components, called protocol handlers, installed on the PC running the Control Builder or on the controllers. This component-based architecture is illustrated in Figure 4. To add support for a new I/O module, communication interface, or protocol to this system, it is only necessary to add protocol handlers for the PC and the controller along with a hardware definition file. The format of hardware definition files is extended to include the identities of the protocol handlers.

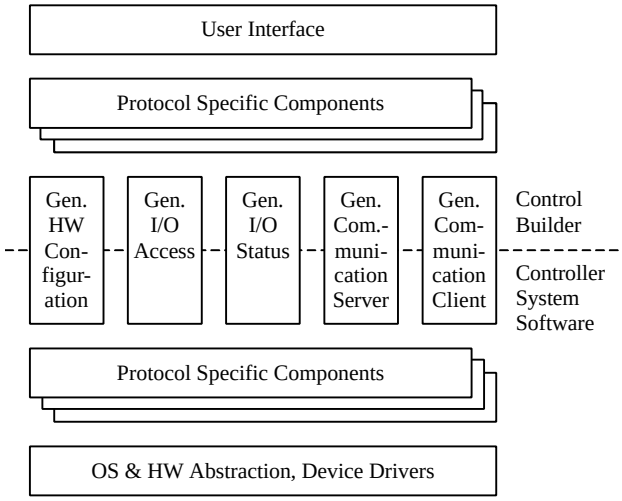


Figure 4. Component-based software architecture.

Essential to the success of the approach, is that the dependencies between the framework and the protocol handlers are fairly limited and, even more importantly, well specified. One common way of dealing with such dependencies is to specify the interfaces provided and required by each component. ABB's component-based

control system uses Microsoft's Component Object Model (COM) [5] to specify these interfaces, since COM provides suitable formats both for writing interface specification, using the COM Interface Description Language (IDL), and for run-time interoperability between components. For each of the generic components, two interfaces are specified: one that is provided by the framework and one that may be provided by protocol handlers. Interfaces are also defined for interaction between protocol handlers and device drivers. The identities of protocol handlers are provided in the hardware definition files as the Globally Unique Identifiers (GUIDs) of the COM classes that implement them.

The use of COM implies that all invocations of an interface's methods are sent to a particular object. The use of objects turns out to work very well for the system in question. It allows several instances of the same protocol handlers to be created. This is useful, for instance, when a controller is connected to two separate networks of the same type. Also, it is useful to create one instance of the object implementing an interface provided by the framework for each protocol handler that requires the interface. An additional reason that COM is the technology of choice is that it is expected to be available on all operating systems that the software will be released on in the future. The Control Builder is only released on Windows, and an effort has been started to port the controller system software from pSOS to VxWorks. In the first release of the system, which will be on pSOS, the protocol handlers will be implemented as C++ classes, which will be linked statically with the framework. This works well because of the close correspondence between COM and C++, where every COM interface has an equivalent abstract C++ class.

When a control system is configured to use a particular device or protocol, the Control Builder uses the information in the hardware definition file to load the protocol handler on the PC and execute the protocol specific routines it implements. During download, the identity of the protocol handler on the controller is sent along with the other configuration information. The controller system software then tries to load this protocol handler. If this fails, the download is aborted and an error message displayed by the Control Builder. This is very similar to what happens if one tries to download a configuration, which includes a device that is not physically present. If the protocol handler is available, an object is created and the required interface pointers obtained. Objects are then created in the framework and interface pointers to these passed to the protocol handler. After the connections between the framework and the protocol handler has been set up through the exchange of interface pointers, a method will usually be called on the protocol handler object that causes it to continue executing in a thread of its own. Since the interface pointers held by the protocol handler references objects in the framework, which are not used by anyone else, all synchronization between concurrently active protocol handlers can be done inside the framework.

To make this a little bit more concrete, consider the interface pair IGenClient, which is provided by the framework, and IPhClient, which is provided by protocol handlers implementing the client side of a communication protocol. IPhClient has a method

```
HRESULT SetClientCallback(IGenClient *pGenClient)
```

which is called to pass an interface pointer to an object in the framework to the protocol handler. There is a similar method for passing an interface pointer providing access to a device driver. After the interface pointers have been handed over, the framework can start the execution of the protocol handler in a separate thread. The code in this thread will then mediate message between control programs and a communication interface via the device driver.

4. Lessons learned

The definitive measure of the success of the project described in this paper will be how large the effort required to redesign the software architecture has been compared to the effort saved by the new way of adding I/O and communication support. It is important to remember, however, that in addition to this cost balance, the business benefits gained by shortening the time to market must be taken into account. Also important, although harder to assess, are the long time advantages of the increased flexibility that the component-based software architecture is hoped to provide.

At the time of writing, the design of the framework, including the specification of interfaces, is largely completed and implementation has started. It is thus too early to say exactly how much work has been needed, but it seems safe to conclude that the efforts are of the same order of magnitude as the work required to add support for an advanced I/O or communication system the old way, that is by adding code to the affected modules. From this we can infer, that if the new software architecture makes it substantially easier to add support for such systems, the effort has been worthwhile. We therefore find that the experiences with the ABB control system supports our hypothesis that a component-based software architecture is an efficient means for supporting distributed development of complex systems.

Another lesson of general value is that it seems that a component technology, such as COM, can very well be used on embedded platforms and even platforms where run-time support for the technology is not available. Firstly, we have seen that the overhead that follows from using COM is not larger than what can be afforded in many embedded

systems. In fact, used with some care, COM does not introduce much more overhead than do virtual methods in C++. Secondly, in systems where no such overhead can be allowed, or systems that run on platforms without support for COM, IDL can still be used to define interfaces between components, thus making a future transition to COM straightforward. This takes advantage of the fact that the Microsoft IDL compiler generates C and C++ code corresponding to the interfaces defined in an IDL file as well as COM type libraries. Thus, the same interface definitions can be used with systems of separately linked COM components and statically linked systems where each component is realized as a C++ class or C module.

An interesting experience from the project is that techniques that were originally developed to deal with dynamic hardware configurations have been successfully extended to cover dynamic configuration of software components. In the ABB control system, hardware definition files are used to specify what hardware components a controller may be equipped with and how the system software should interact with different types of components. In the redesigned system, the format of these files has been extended to specify which software components may be used in the system. The true power of this commonality is that existing mechanisms for handling hardware configurations, such as manipulating configuration trees in the Control Builder, downloading configuration information to a control system, and dealing with invalid configurations, can be reused largely as is. The idea that component-based software systems can benefit by learning from hardware design is also aired in [1].

5. Related work

The use of component-based software architecture in real-time, industrial control has not been extensively studied, as far as we know. One example is documented in [7]. This work is not based on experiences from industrial development, however, but rather from the construction of a prototype, developed in academia for non-real-time platforms with input from industry. It also differs from our work in that it focuses on the possibility of replacing the multiple controllers usually found in a production cell with a single controller, rather than on supporting distributed development.

6. Conclusions and future work

The initial experiences from the effort to redesign the software architecture of ABB's control system to support component-based development are promising, in that the developers have managed to define interfaces between the framework and the protocol handlers. Since the effort to redesign the system has not been too extensive, we conclude that the project has met its first challenge successfully. An assessment of how the remaining challenges of achieving sufficiently general interfaces while maintaining an acceptable performance have been met would be premature at this point.

An issue that may be addressed in the future development at ABB is richer specifications of interfaces. COM IDL only specifies the syntax of interfaces, but it is also useful to specify loose semantics, such as the allowed parameters and possible return values of methods, and timing constraints. Since UML has already been adopted as a design notation, one possibility is to use the specification style suggested in [6]. In our continued research concerning this effort we plan to study in more detail how non-functional requirement are addressed by the software architecture. We will, for instance, look at reliability, which is an obvious concern when externally developed software components are integrated into an industrial system.

7. Acknowledgements

We gratefully acknowledge the financial support of ABB Automation Products, Sweden and the Swedish KK Foundation.

8. References

[1] Clemens Szyperski, *Component Software – Beyond Object-Oriented Programming*, Addison-Wesley, 1997.

[2] H. Hermansson, M. Johansson, L. Lundberg, "A Distributed Component Architecture for a Large Telecommunication Application", *Proceedings of the Seventh Asia-Pacific Software Engineering Conference*, December 2000.

[3] I. Crnkovic, M. Larsson, "A Case Study: Demands on Component-Based Development", *Proceedings of 22nd International Conference of Software Engineering*, May 2000.

[4] International Engineering Consortium, *IEC Standard. 61131-3*.

[5] Microsoft Corporation, *The Component Object Model Specification, Version 0.9*, October 1995.

[6] A. Speck, "Component-Based Control System", *Proceedings of the Seventh IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, April 2000.

[7] John Cheesman, John Daniels, *UML Components – A Simple Process for Specifying Component-Based Software*, Addison-Wesley, 2001.

A Survey of Capture-Recapture in Software Inspections

Håkan Petersson
Dept. of Communication Systems
Lund University
hakan.petersson@telecom.lth.se

Thomas Thelin
Dept. of Communication Systems
Lund University
thomas.thelin@telecom.lth.se

Abstract

Software inspection is a method to detect faults in the early phases of the software life cycle. One piece of information available after an inspection session is the number of found faults. However, more important information is the number of faults not found. In order to estimate this, capture-recapture was introduced for software inspections in 1992. Since then, several papers have been written in the area. This paper summarizes the work made in capture-recapture for software inspections during these years. Furthermore, and more importantly, the papers are classified in order to facilitate other researchers' work as well as highlighting the areas of research that need further work.

1. Introduction

Software inspection [18][29] is an efficient method to detect faults in software artefacts. It was first described by Fagan [25], and since then inspections have evolved to become a mature empirical research area. The research has addressed changes to the inspection process, e.g. [8][40], [33][30], support to the process, e.g. [2][20] and empirical studies, e.g. [45][53]. The support to the inspection process includes reading techniques [2] and the use of capture-recapture techniques to estimate the remaining number of faults after an inspection [20]. Furthermore, industry has studied the benefits of conducting software inspections [58]. Reading techniques are applied to the individual part of inspections in order to aid reviewers with more information of how to read when inspecting. The purpose is to increase the efficiency and effectiveness. Several reading techniques have been proposed, checklist-based reading [26], defect-based reading [45] perspective-based reading (PBR) [2], traceability-based reading [56] and usage-based reading [55].

Capture-recapture [17] is a statistical method that can be utilized with software inspections to estimate the fault content of an artefact. Capture-recapture was first introduced in software inspections by Eick et al. 1992 [20], and since then a number of papers has been published that evaluates and improves capture-recapture for software inspections. The method uses the overlap among reviewers to estimate the fault content. It is assumed that the reviewers work independently of each other and therefore the fault searching has to be performed before, and not during, an inspection meeting. The size of the overlap indicates the number of faults left; if the overlap is large, it indicates that few faults are left to be detected; if the overlap is small, it indicates that many faults are undetected. Using statistical methods, an estimation value and a confidence interval can be calculated. This information can be used by inspection coordinators and project managers to take informed decisions, which is exemplified in Section 2.

The first known use of capture-recapture was by Laplace 1786 [31], who used it to estimate the population size of France [44]. In biology, capture-recapture is used to estimate the population size of animals, e.g. the number of fish in a lake. Several different types of capture-recapture models exist. Capture-recapture methods have also been utilized in other areas, e.g. software testing [50][63] and medical research [17].

The purpose of this paper is to summarize the capture-recapture research in software inspections during the past ten years. During these years, a number of research papers have been published. By categorizing the papers, this is intended to facilitate other researchers' work as well as highlight the areas of research that need further work. The papers have been classified into three main categories *theory*, *evaluation* and *application*. Several papers have considered the theory and evaluation of capture-recapture. Only one published paper has tried to apply capture-recapture in an industrial environment.

To find the relevant literature for the survey, a literature search was carried out. This was made through searching the databases INSPECT, IEEE online, Science Direct and Association of Computing Machinery (ACM) using the keywords "capture recapture", "defect content estimation" and "fault content estimation". In addition, some papers were obtained by personal communications with researchers. Finally, all references in the papers were checked to guarantee that no referenced paper was missed.

The paper is outlined as follows. In Section 2, an inspection process using capture-recapture is outlined and the theory of capture-recapture is presented. A summary of capture-recapture papers written is provided in Section 3. In Section 4, future research is suggested and in Section 5 a summary of this paper is provided.

2. Capture-Recapture

2.1. Inspection Process

An example of a process that can be used to combine inspections and capture-recapture is described in this section and shown in Figure 1.

After the individual inspection (1), the inspection records are handed in to an inspection coordinator who compiles the faults into one document (2). The inspection coordinator uses the overlap of the faults that are found to make an estimate of the number of remaining faults. This information can then be used to decide whether a meeting is necessary, more reviewers are needed, reinspection are needed, rework of the document should be made and so forth. During the meeting (3), only the reviewers that are needed to participate attend. The inspection coordinator decides who will attend, based on the number of faults left, the preparation time and the overlap information. The main purpose of the meeting is to find new faults, but also to spread information and take informed decisions about the artefact inspected. After the meeting, a new estimate of the fault content can be made (the confidence is probably better in this estimation, since reviewers have agreed upon the faults in the artefacts). The inspection coordinator may then use the inspection record for process improvement and gives the record to the authors of the artefact to correct the faults (4). In the described process, the main purpose is to find as many faults as possible. There are other possibilities where the goal is not necessarily to find all faults. Thelin et. al. [51] describes an inspection process where, in order to save inspection effort, the documents are sampled. Then, based on a pre-inspection on the samples, the inspection effort is focused on the documents that need it the most.

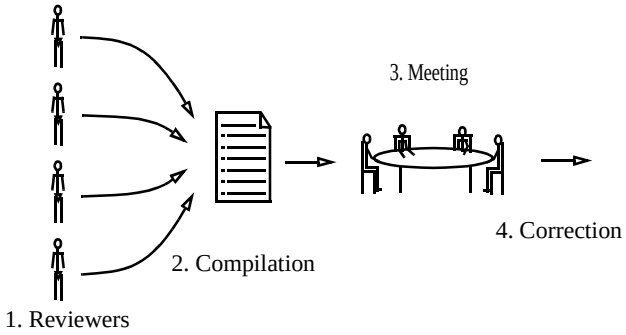


Figure 1. An example of an inspection process using capture-recapture estimations.

2.2. Models

There are many different models and estimators in capture-recapture. An *estimator* is a formula used to predict the number of faults remaining in an artefact. *A model* is the umbrella term for a number of estimators with the same prerequisites. Four basic capture-recapture models are used for software inspections, see Table 1. However, more models have been developed for other domains, but have not yet been investigated for software inspections.

The overlap among the faults that the reviewers find is used as a basis for the estimation. The smaller overlap among the reviewers the more faults are assumed to remain, and the larger overlap the fewer faults are assumed to remain. Two extreme cases can occur. Either, all reviewers have found exactly the same faults, which means that there are probably not any faults left, or none of the reviewers has found a fault that another reviewer has found, which indicates that there are probably many faults left. To estimate the number of faults left, statistical estimators are used, which are designed to model different variations in software inspections.

The models handle variations in the ability of the reviewers to find faults as well as the faults' probability to be found. The most basic model (M0) assumes that all faults are equally probable to be found and that all reviewers have equal abilities to find faults. More advanced models use either the assumption that the probabilities of faults vary (Mh), or the abilities of reviewers vary (Mt), or both (Mth), see Table 1. Within each model, a number of estimators have been developed.

In addition to capture-recapture, two other fault content estimation methods have been developed that utilize the overlap information, curve-fitting models and subjective estimations. The curve fitting models use a mathematical function, which is fitted to the inspection data and extrapolated to a limit value. The most commonly used curve fitting method, which is called detection profile method (DPM) [61], uses an exponential function. Subjective estimations use the knowledge of reviewers to estimate fault content after inspections. The reviewers estimate the most probable value of the number of faults left [23]. Enhanced methods have been developed [3][62], where reviewers' estimations are combined. Some of these models require that the most probable, minimum and maximum values are estimated.

Table 1: The models, prerequisites and estimators in capture-recapture. More models exist in capture-recapture but have not been used for software inspections.

Model	Prerequisites	Estimators
M0	All faults have equal detection probability. All reviewers have equal detection ability.	M0-ML – Maximum likelihood [39]
Mt	All faults have equal detection probability. Reviewers may have different detection abilities.	Mt-ML – Maximum likelihood [39] Mt-Ch – Chao's estimator [15]
Mh	Faults may have different detection probabilities. All reviewers have equal detection ability.	Mh-JK – Jackknife [13] Mh-Ch – Chao's estimator [14]
Mth	Faults may have different detection probabilities. Reviewers may have different detection abilities.	Mth-Ch – Chao's estimator [16]

3. State-of-the-Art

This section summarizes and classifies the papers written in the area of capture-recapture connected with software inspections, see Table 2. Research contributions can from a general point of view be divided into *theory*, *evaluation* and *application*. *Theory* includes basic research, which investigates and describes the fundamentals of capture-recapture models and estimators. This has been extensively investigated in biostatistics, but has been further developed and transferred to software engineering and especially inspections. The second step in a research chain is to *evaluate* the proposed methods and to improve them. The third main area is *application*, where the results from theory and evaluation are transferred to be used in an industrial setting. Moving from theory to application takes a long time [46]. Researchers need to establish basic results before software organizations are willing to adopt them. In this survey, only one paper has been classified as an application paper. That paper only uses results from Eick et al. [20] and no other conducted research.

A secondary classification is made to classify the papers. in subsets of the main classification. Only the subsets needed for the summarized papers are included, i.e. it is not necessarily exhaustive. The classification shows the main ideas of the papers and is intended to guide the readers to help them understand the research conducted. These topics are further discussed in Section 4, where future research is pointed out. The classification of a paper includes a primary classification denoted with an "x" and a secondary classification, denoted with an "(x)". Although a paper makes a contribution in one area it may also contain smaller contributions in other areas. These smaller contributions receive a secondary classification.

Each paper classified as primary is described in one paragraph. To recognize these papers, the references are marked in bold. All papers are at least classified as primary once, but may also be classified once or more as secondary. This leads to that some papers are described more than once, and hence some recurrences are inevitable.

3.1. Basic Theory

Most of the basic capture-recapture theory as well as the derivation of all the models and estimators have been described and developed within the research area of biostatistics. Capture-recapture in software inspections is an adaptation of an old technique into a new area. There are, however, some papers published within the software inspection community, which contribute to the investigation and evolvement of the basic theory. This includes theory concerning the assumptions or the introduction of new theoretical concepts that arise because of inspections being a new area of application. In the case of Freimut's master thesis [27], it is included in this category because of being the first comprehensive description of all capture-recapture models suitable to be evaluated for use in software inspections.

The first paper to apply capture-recapture to software inspections was written by Eick et al. **[20]**. Eick et al. describe a model showing how faults propagate through the development phases. At each phase, some faults are detected but at the same time new ones are introduced. Capture-recapture is introduced as a technique to estimate the number of residual faults after each phase. To test the technique Eick et al. designed a study that when the paper was written had produced some preliminary results. The paper does not only concern capture-recapture findings but it also reports on inspection observations in general such as the lack of synergy effect at the inspection meeting. The statistics of the capture-recapture estimator Mt-ML is described and some preliminary results from the experimental study are presented. These results show the relation between the predicted number of faults, the number found during preparation and total number found including those faults discovered at the inspection meeting. However, the true number of faults was not known.

Table 2: The classification of the capture-recapture papers.

			Theory		Evaluation			Application
First Author	**Year**	**Ref.**	**Basic Theory**	**New Approaches**	**Evaluation of Estimators**	**Improvements of Estimators**	**Estimators and PBR**	**Experience Reports**
Eick	1992	[20]	x					
Eick	1993	[21]	x					
Vander Wiel	1993	[57]	(x)		x	(x)		
Wohlin	1995	[60]				x		
Briand	1997	[9]			x			
Ebrahimi	1997	[19]	(x)	x	(x)			
Freimut	1997	[27]	x		(x)		(x)	
Ardissone	1998	[1]						x
Briand	1998	[10]				x		
Ekros	1998	[22]	x		(x)			
Runeson	1998	[49]			x			
Wohlin	1998	[61]		x	(x)			
Miller	1999	[35]			x			
Petersson	1999	[41]			x			
Petersson	1999	[42]				x		
Thelin	1999	[52]					x	
Biffl	2000	[3]			x			
Briand	2000	[12]			x			
Petersson	2000	[43]				x		
Thelin	2000	[53]	(x)			x		
Thelin	2000	[54]	(x)			x		
Biffl	2001	[6]		(x)	x			
El Emam	2001	[24]	(x)		x			
Freimut	2001	[28]					x	
Wohlin	2001	[62]			x			

In **[21]**, Eick et al. argue that most of the earlier statistical analyses of software have focused on the last two development phases, i.e. testing and release. In their paper, they introduce capture-recapture that brings statistical analysis to two of the earliest stages, generation of requirements and specification of design. First, Eick et al. introduce one of the most basic capture-recapture estimators from biology (the Lincoln-Peterson estimator [39]). This estimator uses only two samples (reviewers) and assumes equal probabilities (Mt). Eick et al. identify four issues concerning software inspections that make this simple approach unsuitable. In software inspections: (a) most of the time there are more than two reviewers, (b) reviewers are not equal in their ability to find defects which results in unequal probabilities, (c) there is a risk that the reviewers cooperate, which violates the assumption of independence between "trapping occasions" and finally (d) the faults are not equally difficult to discover which also affects the probability in the statistical model. Eick et al. take care of a, b and d by presenting and explaining estimators of models Mt and Mh. Both these models handle cases with more than two reviewers. Issue c is tackled by introducing a statistical method to identify collusion and specialization, i.e. the lack of independence, among the reviewers. Also included in the paper is a brief description of an experiment where Eick et al. analysed data from thirteen reviews. The estimation models give indications that about 20% of the faults are not detected during the first inspection. Another result is that the indication of quality that was given by the estimate agreed with the reviewers' opinion of the inspected documents.

Freimut makes an extensive overview and introduction to the mathematical theory underlying capture-recapture in his master thesis **[27]**. The theory behind capture-recapture in general as well as for all of the common capture-recapture models is presented.

The paper by Ekros et al. **[22]** can be regarded as a critique against using capture-recapture in software engineering without investigating the underlying model. Ekros et al. claim that the most important aspect when adapting new methods to new areas is that of the underlying models. They identify the absence of investigations concerning model validity in earlier papers and set out to fill this gap. Three hypotheses are tested (1) reviewers find the same number of faults, (2) faults are equally easy to detect and (3) reviewers find the same faults. Statistical test methods are derived and applied to data sets taken from [27][38][60]. The results

give no support for any of the three hypotheses. This, according to Ekros et al., implies that the used capture-recapture models are not suitable to use when analysing the software engineering data in their study.

El Emam and Laitenberger, in [23][24], include the concept of using an alternative evaluation measure when investigating capture-recapture estimators, i.e. Relative Decision Accuracy (RDA). RDA evaluates how the estimators actually are utilized within the inspection process, in contrast to evaluating on only how close the estimations are to the true value. A contribution to how to handle the problem with dependent reviewers is made by Ebrahimi in [19]. A non-independent model is presented, where non-parametric kernel smoothing is combined with a Maximum likelihood estimator to estimate the number of remaining faults. Vander Wiel and Votta [57], illustrate an approach to improve Mt-ML by grouping the faults and estimate each group separately in order to fulfil the assumption. This idea is further elaborated by Wohlin et al. [60] and Runeson and Wohlin [49] and originates from White et al. ([59] p. 163). The question of which estimation model to select in a certain case is investigated with the help of chi-square tests by Thelin and Runeson [54] and is further analyzed by the use of Akaike model selection criterion by Thelin and Runeson [53].

3.2. New Approaches

During the years of research on capture-recapture within software inspections, new ideas and suggestions that reach beyond the mere application of existing estimators have emerged. This aim is important since the conditions of capture-recapture in software inspections is in many ways different to the conditions in biological settings. The following papers have contributed with knowledge that can be classified as more than only improvements.

Ebrahimi argues in **[19]** that in the software development environment, some degree of collusion cannot be avoided. Since reviewers are selected to cover different aspects of the document, specialization is unavoidable too. This leads to that the binomial distribution is not valid to use for estimations of the total number of faults. To solve this, Ebrahimi models a likelihood function with independent faults but allows for dependence among the reviewers. These unknown parameters in the likelihood function are estimated using non-parametric kernal smoothing. To evaluate the estimator, Ebrahimi applies it to two data sets (taken from [21] and [29]). Both the unmodified data sets and the same with dependence introduced are used. Ebrahimi concludes that the estimates differ from what Mt-ML produces for the modified variants but produces similar results when applied to the unmodified data.

As in [19] Wohlin and Runeson **[61]**, observe that the assumptions of Mt and Mh are unlikely to be valid. Instead, they introduce two methods similar to the approaches that use reliability growth models to estimate software reliability [37]. Wohlin and Runeson propose methods that are based on the shape of the data when plotting. This idea is mentioned in [21] where Eick et al. state that Mh-JK is equal to "*...fitting a k-1 degree polynom... and extrapolating to estimating [the number of remaining faults]*". In [61], the inspections data are sorted (and plotted) according to certain rules and mathematical functions are then fitted to the data. Two methods are proposed, the Detection Profile Method (DPM) and the Cumulative Method. In DPM, the plot shows the number of reviewers that found a specific fault sorted in decreasing order while the cumulative method plots the cumulative sum of faults that are found. These estimators are then evaluated by comparing them to Mt-ML. The DPM estimates the best of the three, however, not significantly better. The cumulative estimates the worst. The cumulative method is constructed to make overestimations. Another approach tested is to combine the three estimators and use the average as the estimate. However, the average method does not significantly outperform any of the others.

Biffl and Grossman [6] investigate an approach of utilizing the information from a second inspection cycle (reinspection). They investigate how estimations from two consecutive inspection sessions should be combined to gain the most accurate estimate. The best approach was to make one estimate from the combined data of the two inspection sessions.

3.3. Evaluation of Estimators

An important part of the capture-recapture research has been to evaluate (a) estimators designed in biostatistics research (b) new proposed estimators and (c) improvements and variants of estimators. This has resulted in a number of papers that evaluate the estimators. As the research has matured, most capture-recapture researchers agree on that Mh-JK seems to be the best suited estimator for software inspections. In addition, some proposed improvements are evaluated in the papers in this section, e.g. DPM, but still many of these improvements need to be replicated by other researchers.

Vander Wiel and Votta **[57]** evaluate Mt-ML and Mh-JK using a simulation. Furthermore, they evaluate confidence intervals and whether the estimations can be improved by grouping the faults into classes. The confidence intervals are Walds and Likelihood confidence interval. Observations of Mt-ML lead to the suggestion of grouping faults, which has also been mentioned by White et al. ([59] p. 163). The parameters of the simulation in the paper have been designed after interviewing over 100 reviewers and observing over 50 inspections. Vander Wiel and Votta found that (1) Mh-JK overestimates when reviewers' detection abilities

differ, (2) both Mh-JK and Mt-ML have a large bias when some faults are easy to find and some are difficult, (3) the bias of Mt-ML can be reduced by grouping the faults into classes (however, at the cost of an increased variance of the estimations). Furthermore, they recommend using the Likelihood confidence interval instead of Walds, since the former includes the correct number of faults in most cases. However, the Likelihood confidence interval is too conservative, which often leads to broad intervals.

Briand et al. **[9]** wrote the first research paper where all four models, presented in Section 2, are described. The estimators used besides Mh-JK and Mt-ML are M0-ML, Mt-Ch, Mh-Ch and Mth-Ch. Briand et al. use data from an earlier conducted perspective-based reading (PBR) experiment [2] to evaluate the estimators. The main purpose of the paper is to investigate the relative error, the variance and the failure rate of the estimators when having 2 to 6 reviewers. Their approach to find the best estimator is to use paired t-tests [36] of the bias together with analysis of variance, outliers and failure rate. They found that (1) most estimators underestimate, (2) no model estimates satisfactory for 2 and 3 reviewers and (3) Mh-JK is recommended for 4 and 5 reviewers.

Runeson and Wohlin **[49]** conducted a code inspection experiment to evaluate one improved version of a filtering approach proposed by Wohlin and Runeson [60]. The purpose of the filtering is to divide the data into two classes, one class for all faults detected by one reviewer (class 1) and one class for the rest of the faults (class 2). The faults in class 2 are used as input to Mt-ML and faults in class 1 are multiplied with a predetermined factor, which is based on historical data. The predetermined factor is defined as the average of the outcome of a number of previous conducted inspections. The relative error and variance are used for evaluation. The results are (1) the mean is not improved with the filtering approach (2) the variance is smaller using the filtering approach (3) Mt-ML overestimates in all cases.

Miller **[35]** evaluates the same estimators as Briand et al. [9], with the exception that Miller uses all Mh-JK's orders (1-5) and Mth-Ch orders (1-3). Miller argues that the procedure used to selecting among Mh-JK's and Mth-Ch's orders as implemented in the program CAPTURE [48] is not applicable for software inspections. The results may be that one of the subestimators might produce a more accurate result than the full estimator. Therefore, Miller uses all orders separately in the evaluation. Furthermore, Miller translates the prerequisites from biology to software inspection terms and summarizes all closed model estimators used in biology. The data come from two previously conducted experiments [32][34], where the purposes were to evaluate tool inspections and defect-based reading, respectively. The investigation uses relative error and box plots to evaluate the estimators for 3 to 6 reviewers. The results are (1) the estimators underestimate (2) Mh-JK order 1 is the best estimator and can be used for 3 to 6 reviewers (3) Mth-Ch (order 1 and 3) might be appropriate to use if many reviewers inspect.

Petersson and Wohlin **[41]** replicated the improvement approach made by Briand et al. [10]. Briand et al. developed two criteria to select among a linear model, an exponential model (DPM) and Mh-JK. Briand et al. called the new approach enhanced DPM (EDPM). Petersson and Wohlin replicated this approach by using three data sets from conducted experiments [60][49][47]. Furthermore, they apply another criterion to be used for choosing between the linear and exponential model. The purpose was to reduce the number of outliers. The results do not confirm the results achieved by Briand et al. [10]. Further results are (1) neither of EDPM, Mh-JK, improved EPDM can be considered to be best and (2) EDPM may be suitable for cases with few reviewers.

El Emam and Laitenberger **[24]** investigate capture-recapture estimators when two reviewers inspect. They simulate inspections to evaluate the estimators M0-ML, Mt-ML, Mt-Ch, Mh-Ch, Mh-JK and Mth-Ch. As in the paper by Vander Wiel and Votta [57], they use two classes of faults (hard and easy to find) together with 48 study points. As evaluation criteria, relative error, variance, regression trees and relative decisions accuracy (RDA) are used. RDA measures whether a capture-recapture estimator is beneficial to use as a decision criterion for reinspection. The input to RDA is an inspection effectiveness threshold. Two different threshold values are used (0.57 and 0.7), which are based on code inspections [11]. The results are (1) Mt-Ch (also known as Chapman) is the best estimator for 2 reviewers, (2) only Mh-JK and Mt-Ch do not exhibit frequent failures, (3) Mh-JK is non-robust and produced underestimates.

Briand et al. **[12]** investigate four specific issues concerning capture-recapture estimation. The evaluation is connected with the master thesis work performed by Freimut [27]. The purpose of the paper is to evaluate the estimators M0-ML, Mt-ML, Mt-Ch, Mh-JK, Mh-Ch and Mth-Ch for 2 to 6 reviewers, investigate the impact of the number of faults and investigate the impact of the number of reviewers. Furthermore, they also make an analysis of the filtering method proposed by Wohlin and Runeson [60][49]. Using the filtering method increases the variance, which makes it less useful. The evaluation measures for the estimators are relative error, variance and failure rate. The main results are (1) Mh-JK is the best estimator and can be used for 4-6 reviewers, (2) no estimator is good enough to be used for less than 4 reviewers, (3) there is a tendency for underestimation for the estimators and (4) increasing the number of faults reduce the variance, but does not affect the relative error significantly.

Biffl **[3]** uses data from a large experiment [7] to evaluate capture-recapture estimators against subjective estimations. The subjective estimations are combined with different formulae in order to merge several reviewers' subjective estimation into one. In the experiment, each reviewer gave three estimations, maximum, minimum and most likely number of faults left. The evaluation measures used are relative error and standard deviation. Furthermore, confidence intervals are investigated. The capture-recapture estimators included in the study are M0-ML, Mt-ML, Mt-Ch, Mh-JK, Mh-Ch, Mth-Ch and DPM. The results are (1) all estimators underestimate, (2) Mh-JK estimates most accurate of the capture-recapture estimators, (3) subjective estimations have smaller bias, but larger variance and (4) neither estimator has a reliable confidence interval.

Biffl and Grossing **[6]**, suggest and evaluate three different formulae to be used for capture-recapture estimators when reinspections are performed. The approaches are either to (a) first combine the data from the inspections and then estimate, (b) add the number of faults detected in the first inspection to an estimate of the reinspection or (c) capture-recapture estimate the first inspection and the reinspection and then add their results. The data come from the experiment described in [7]. The measures used for evaluation are relative error and variance. The results are (1) the best approach is to first combine the data and then estimate, (2) the estimators improved significantly when this approach was used. The interpretation of the results is that the more time used in inspection, the more accurate estimation results are obtained.

Biffl **[7]** describes a large experiment (169 students) conducted 1998 investigating several different research questions. Some of the investigated areas are capture-recapture, reading techniques (PBR) and reliability models for software inspections. The papers evaluating capture-recapture are summarized and referenced in other parts in this paper. However, three papers are under submission [4][5] and are therefore not included.

Wohlin et al. investigate in **[62]** as in [23], the difficulty of achieving reliable estimations for two reviewers. Wohlin et al. present three variants of experience-based methods and compare them to M0-ML, Mt-ML, Mh-JK, Mth-Ch and Mh-Ch. The variants of the experience-based methods are all based on reviewers' effectiveness. The effectiveness history is logged either individually, by groups or as an average of all reviewers. The experience-based methods have less bias than any of the capture-recapture estimators, but not significantly lower. The absolute bias of the relative error as well as the standard deviation is around 20% for the experience models.

Freimut [27] describes and evaluates many estimators derived in biostatistics. Most of the results are described in [12]. Ebrahimi [19] derives a new estimator and makes a comparison with Mt-ML. It is concluded that since the new estimator gives different results it may be appropriate to use for capture-recapture estimations. Wohlin and Runeson [61] evaluate two curve fitting methods against Mt-ML in an experiment. The curve fitting method DPM estimates most accurately, though not significantly better than Mt-ML. Freimut et al. [28] evaluates the capture-recapture estimators with DPM and subjective estimations. They find that Mh-JK is the most accurate estimator considering bias and variance.

3.4. Improvement of Estimators

The evaluation of capture-recapture in software inspections shows results that encourage improvements. A number of approaches to improve the estimations have been investigated. Among these, there are model selection approaches, usage of historical data and approaches that make the data fit the model assumptions better.

Wohlin et al. **[60]** conducted the first experiment with the purpose to evaluate capture-recapture estimators for software inspections. They observed three main problems (1) some faults have low probability to be found, (2) if many faults are found during the inspection meeting, it indicates that a large number of faults exist, but capture-recapture implies the opposite, (3) if no overlap exists, capture-recapture does not work. The first two problems are connected to the assumption of equal probabilities for all faults. Wohlin et al. continue the development of the idea presented by Vander Wiel and Votta, i.e. grouping the faults to improve Mt-ML's estimates. They identify two possible ways of creating a filter that divides the found faults into groups. Filter one is based on the percentage of the reviewers that found a fault. Filter two selects all faults only found by one reviewer and then estimates the other faults separately. Faults found by only one reviewer are multiplied with an experience-based factor, since no overlap exists. Applying these filtering techniques, they manage to improve the Mt-ML estimates.

A problem with all estimators is that they have a tendency to produce extreme under/over estimations. Briand et al. **[10]** argue that this behaviour can discourage their use because of the risk involved in not knowing when such extreme estimations occur. In their study, they evaluate a selection procedure that, based on certain criteria, chooses between using an enhanced version of DPM (EDPM) and Mh-JK. They note that there are cases where DPM makes large overestimations, e.g. when there are no faults found by only one reviewer. To avoid this, they let EDPM be based on an ordering criterion that chooses between using the original exponential fit or a linear fit. If the goodness-of-fit of the EDPM's curve fit is less than a threshold, they select Mh-JK instead. This approach shows a small overall improvement of reducing the outliers.

Petersson and Wohlin **[42]** investigate the use of experience data for capture-recapture estimators. The study explores the estimation of intervals. They note that the best would be to find one estimator that always underestimates and one estimator that always overestimates. These could be used to (a) provide a 100% confidence interval, (b) provide an improved estimate by interpolating between the two limits and (c) improve existing estimators by providing an interval that cut of extreme estimations. They force the estimators to overestimate in two different ways. Either by adding a parameter multiplied with the number of found faults (method M1) to Mh-JK's estimate and use Mt-ML as the lower limit or by not using limits based on fixed constants p_{upper} and p_{lower} multiplied with the number of found faults (Method M2). The constants are generated from historical data. Method M1 improves the bias, but slightly increased variance. Another result is that the interval managed to cut the estimates of Mth-Ch to avoid all of its extreme outliers.

Based on the observations, Petersson and Wohlin **[43]** investigate two alternative ways of improving DPM. The alternatives are to (a) have a variable point of estimation limit that is calculated from historical data, or (b) use the derivative of the curve to determine where to select the estimate. The derivative thresholds are based on historical behaviour and are allowed to vary with the number of reviewers. The derivative DPM managed to improve the original DPM but is no improvement compared to Mh-JK. A brief investigation in the study indicates that the derivative DPM can be further improved if less variant data are used to calibrate the parameters. The recommendation from Petersson and Wohlin is to use Mh-JK together with a simple experience-based estimator, either derivative DPM or mean bias correction as presented in [42], and use data from within the company to generate the experience-based parameters.

Thelin and Runeson **[53]** evaluate the use of an information-theoretic approach in the form of Akaike's model selection criterion. This statistical method has been utilized successfully on capture-recapture in connection to biology. The Akaike method estimates a distance between a model fitted to the observed data and the true, but unknown, model. The estimators used in the paper are curve fitting models. Both linear, quadratic, potential as well as DPM and an extended variant of exponential curves are used and evaluated using a number of data sets together with Akaike's criterion. The estimators are compared to Mh-JK as a benchmark estimator. Their conclusions are that Akaike's criterion fails to select the best estimator sufficient number of times to be useful and no curve fitting method evaluated estimates better than DPM. Furthermore, Mh-JK estimates most accurately.

Thelin and Runeson **[54]** introduce distance measures, chi-square tests and smoothing algorithms with the aim to find robust estimators. The purpose of the distance measures and chi-square tests are to investigate the prerequisites of the capture-recapture models. The chi-square tests, described in [39], are used to design an algorithm with the aim to choose the best model for each estimation occasion. It is shown that neither the distance measures nor the chi-square algorithm could be used for model selection. The conclusion of this is that other parameters are more important than the prerequisites if accurate estimations are to be achieved. Some of these parameters are the number of reviewers and detection ability. The smoothing algorithms use two or more estimators and calculate the mean, median or chooses the two estimation results that are closest to each other. The conclusion is that the smoothing algorithms work well for the data sets used in the paper.

Vander Wiel and Votta [57] introduce the use of grouping faults to better conform to Mt-ML's assumption of equal capture ability among the faults. Vander Wiel and Votta have two types of faults in their study, easy and difficult to find, which they estimate separately. This approach give smaller bias, but larger variance.

3.5. Estimators and PBR

Perspective-based reading (PBR) and capture-recapture have prerequisites that seem to be contradictory. PBR inspections use perspectives applied to the reviewers and the goal is to minimize the overlap of the faults that the reviewers find. Since capture-recapture uses the overlap to estimate the fault content, PBR may affect the estimation result. In all investigations, the same conclusion is drawn, which is that capture-recapture can be used in combination with PBR with little or no impact on the estimation results.

Thelin and Runeson **[52]** investigate the impact of PBR on capture-recapture estimators. The estimators used are M0-ML, Mt-ML, Mh-JK, Mh-Ch, Mth-Ch and DPM. They simulate 19 cases with different reviewer abilities. For each case, two reviewers have low ability and one has high ability to detect faults. The investigation is made for 3 and 6 reviewers and the relative error and variance are used for evaluation. The result is (1) capture-recapture estimations can be applied even when using PBR, (2) for 3 reviewers Mh-JK and DPM estimates best (3) for 6 reviewers all estimators except DPM estimate well and (4) most estimators overestimate when using PBR.

Freimut et al. **[28]** evaluate a combination of PBR and traceability-based reading (TBR) and compare the estimates against using estimations from reviewers using checklist-based reading. Furthermore, they investigate which estimator that work best and whether it estimates more accurately than the subjective approach. The estimators used are M0-ML, Mt-ML, Mt-Ch, Mh-JK, Mh-Ch, Mth-Ch and DPM. The data come from an experiment where 169 students inspected according to PBR and CBR [7]. The results are (1)

estimations are not affected if PBR is used, (2) Mh-JK shows most accurate estimation results, (3) subjective estimations work well, but have larger variance than Mh-JK and (4) capture-recapture estimators underestimate, even when data from PBR inspections are used.

Freimut [27] describes the contradiction between PBR and capture-recapture in his master thesis. Using the data from two PBR experiments [2][27], the impact of PBR on capture-recapture estimations are investigated. The evaluation shows that no such contradiction can be proved empirically, i.e. there is no difference between the estimators when using no reading technique (ad hoc) against using PBR. The evaluation is made for 3 to 6 reviewers with the measures, relative mean, variance and failure rate.

3.6. Experience Reports

Few papers report on industrial use and integration of capture-recapture into their inspection process. As described in previous sections, there have been controlled experiments performed in industry but they are outside the classification of application.

Ardissone et al. **[1]** describe inspections and capture-recapture in general as well as Mt-ML. They report on an implemented tool used for capture-recapture calculations that has been on trial use in an Italian telecom company. Together with the tool, a decision grid is described that determines the inspected objects future, based on two estimates; one estimate for faults that are easy to find and one for faults that are difficult to find. The distinction of is are made to make the prerequisites of the Mt model more valid, as in [57][60][49]. The use of the tool is reported to be adopted with encouraging results, but includes little quantitative data.

4. Further Research in Capture-Recapture

The summarized capture-recapture papers have been classified into three main research directions: theory, evaluation and application. The main purpose of this classification is to show how the research has progressed during time. As pointed out by Redwine and Riddle [46], moving from theory to application takes a long time. Although some issues need to be further evaluated and some more basic research need to carried out, the future challenge for capture-recapture researchers is to apply the knowledge gained during the ten years of capture-recapture research for software inspections.

In this section, we extract the knowledge gained from the summaries in Section 3, in order to aid researchers and software organizations with common knowledge collected over the years. Furthermore, examples of important future research based on the papers are discussed.

In the category *theory*, the main contribution has been to make the transfer into the software engineering area and summarize the literature in biostatistics. So far, closed models have been extensively investigated and described. In biostatistics, there are still other areas that have not been explored, e.g. open models and change-in-ratio models [44]. An interpretation of open models for software inspections is that faults may be introduced or removed during inspection. This has been regarded as not useful for software inspections, but none has actually explored this subject. Change-in-ratio makes an estimation using the difference between the number of faults in several fault classes over time. This could, for example, be used as risk management information in a spiral or incremental development process to estimate the fault content after design using data from the requirements and the design phase. These are just two topics needed to be looked into. Other future research points are listed below:

- Gain further knowledge of the models of capture-recapture and evaluate whether these are appropriate for software inspections [22].
- The relative decision accuracy needs to be further evaluated [23][53]. In addition, other measures, which try to capture important inspection statistics need to be designed.
- The estimator developed by Ebrahimi [19] should be replicated.
- The connection between curve fitting methods, their prerequisites and Mh-JK needs to be investigated.

In the *evaluation* category, many papers have been written to evaluate the estimators. Evaluations have been made for reading techniques and for different software documents. The common knowledge in this area is (1) most estimators underestimate, (2) Mh-JK is the best estimator for software inspections, (3) Mh-JK is appropriate to use for 4 reviewers and more (4) DPM is the best curve fitting method and (5) capture-recapture estimators can be used together with PBR. There are some papers improving the estimation results. However, they need to be replicated by other researcher in order to know whether they work in different settings. Main future research points are to:

- investigate whether one of the subestimators is better than the full estimator of Mh-JK [35].
- evaluate whether PBR makes estimators overestimate or not [52]. In addition, other reading techniques than PBR should be investigated together with capture-recapture.
- replicate suggested improvements, e.g. [10][42][54].
- evaluate the use of confidence intervals for the estimators [3][20].
- investigate capture-recapture for 2 and 3 reviewers [24].

In the *application* category, only one paper has been written, although we suspect that more application investigations have been conducted. Still, they need to be reported in order to build a body of knowledge of capture-recapture for software inspections. Another thing worth noting is that the paper classified in this area does not use any of the research results of the theory or evaluation area other than [20]. Consequently, the main research in this area should focus upon transfer capture-recapture into software organizations and report the results as case studies or surveys.

5. Summary

This paper discusses the status of capture-recapture research for software inspections. In this survey, all available papers within the area have been summarized. From these summaries a number of pointers for future research is stated. The papers are also categorized in order to facilitate other researchers' work as well as highlighting the areas of research that need further work.

Three main categories have been used together with a number of sub-categories within each main category.

- Theory – Basic Theory, New Approaches
- Evaluation – Evaluation of Estimators, Improvements of Estimators, Estimators and PBR
- Application – Experience Reports

The most apparent result from the classification is the lack of papers in the *application* area. Only one published paper has tried to apply capture-recapture in an industrial environment (not including the controlled experiments).

Acknowledgements

The authors would like to thank Dr. Per Runeson for valuable comments on an earlier draft of this paper. This work was partly funded by The Swedish Agency for Innovation Systems (VINNOVA), under a grant for the Center for Applied Software Research at Lund University (LUCAS).

References

[1] Ardissone, M. P., Spolverini, M. and Valentini, M., "Statistical Decision Support Method for In-Process Inspections", *Proc. of the 4th International Conference on Achieving Quality in Software*, pp. 135-143, 1998.

[2] Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S. and Zelkowitz, M. V., "The Empirical Investigation of Perspective-Based Reading", *Empirical Software Engineering*, 1(2):133-164, 1996.

[3] Biffl, S., "Using Inspection Data for Defect Estimation", *IEEE Software*, 17(6):36-43, 2000.

[4] Biffl, S., "Evaluating Defect Estimation Models with Major Defects", Submitted to the *Journal of Systems and Software*, 2000.

[5] Biffl, S. and Gutjar, W., "Using a Reliability Growth Model to Control Software Inspection", Submitted to *Empirical Software Engineering: An International Journal*, 2000.

[6] Biffl, S. and Grossman, W., "Evaluating the Accuracy of Defect Estimation Models Based on Inspection Data From Two Inspection Cycles", *Proc. of the 23th International Conference on Software Engineering*, pp. 145-154, 2001.

[7] Biffl, S., "Software Inspection Techniques to Support Project and Quality Management", Habilitationsschrift, Technischen Universität, Austria, 2001.

[8] Bisant, D. B. and Lyle, J. R., "A Two-Person Inspection Method to Improve Programming Productivity", *IEEE Transactions on Software Engineering*, 15(10):1294-1304, 1989.

[9] Briand, L., El Emam, K., Freimut, B. and Laitenberger, O., "Quantitative Evaluation of Capture-Recapture Models to Control Software Inspections", *Proc. of the 8:th International Symposium on Software Reliability Engineering*, pp. 234-244, 1997.

[10] Briand, L., El Emam, K. and Freimut, B., "A Comparison and Integration of Capture-Recapture Models and the Detection Profile Method", *Proc. of the 9th International Symposium on Software Reliability Engineering*, pp. 32-41, 1998.

[11] Briand, L., El Emam, K., Laitenberger, O. and Fussbroich, T., "Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects", *Proc. of the 20th International Conference on Software Engineering*, pp. 340-349, 1998.

[12] Briand, L., El Emam, K. and Freimut, B., "A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content", *IEEE Transactions on Software Engineering*, 26(6):518-540, 2000.

[13] Burnham, K. P. and Overton, W. S., "Estimation of the Size of a Closed Population when Capture-Recapture Probabilities Vary Among Animals", *Biometrika*, 65:625-633, 1978.

[14] Chao, A., "Estimating the Population Size for Capture-Recapture Data with Unequal Catchability", *Biometrics*, 43:783-791, 1987.

[15] Chao, A., "Estimating Population Size for Sparse Data in Capture-Recapture Experiments", *Biometrics*, 45:427-438, 1989.

[16] Chao, A., Lee, S. M. and Jeng, S. L., "Estimating Population Size for Capture-Recapture Data when Capture Probabilities Vary by Time and Individual Animal", *Biometrics*, 48:201-216, 1992.

[17] Chao, A., "Capture-Recapture Models", *Encyclopaedia of Biostatistics, Editors: Armitage & Colton*, Wiley, New York, 1998.

[18] Ebenau, R. G. and Strauss, S. H., *Software Inspection Process*, McGraw-Hill, New York, 1994.

[19] Ebrahimi, N., "On the Statistical Analysis of the Number of Errors Remaining in a Software Design Document after Inspection", *IEEE Transactions on Software Engineering*, 23(8):529-532, 1997.

[20] Eick, S. G., Loader, C. R., Long, M. D., Votta, L. G. and Vander Wiel, S. A., "Estimating Software Fault Content Before Coding", *Proc. of the 14th International Conference on Software Engineering*, pp. 59-65, 1992.

[21] Eick, S. G., Loader, C. R., Vander Wiel, S. A. and Votta, L. G.,

"How Many Errors Remain in a Software Design Document after Inspection?", *Proc. of the 25th Symposium on the Interface*, pp. 195-202, 1993.

[22] Ekros, J-P., Subotic, A. and Bergman, B., "Capture-Recapture – Models, Methods, and the Reality", *Proc. of the 23rd Annual NASA Software Engineering Workshop,*1998.

[23] El Emam, K. and Laitenberger, O., "The Application of Subjective Estimates of Effectiveness to Controlling Software Inspections", *Journal of Systems and Software*, 54(2):119-136, 2000.

[24] El Emam, K. and Laitenberger, O., "Evaluating Capture-Recapture Models with Two Inspectors", *IEEE Transactions on Software Engineering*, 27(9):851-864, 2001.

[25] Fagan, M. E., "Design and Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal*, 15(3):182-211, 1976.

[26] Fagan, M. E., "Advances in Software Inspections", *IEEE Transactions on Software Engineering*, 12(7):744-751, 1986.

[27] Freimut. B., "Capture-Recapture Models to Estimate Software Fault Content", Diploma Thesis, University of Kaiserslautern, Germany, 1997.

[28] Freimut, B., Laitenberger, O., Biffl,. S., "Investigating the Impact of Reading Techniques on the Accuracy of Different Defect Content Estimation Techniques", *Proc. of the 7th International Software Metrics Symposium*, pp. 51-62, 2001

[29] Gilb, T. and Graham, D., *Software Inspections*, Addison-Wesley, UK, 1993.

[30] Knight, J. C. and Myers, A. E., "An Improved Inspection Technique", *Communications of ACM*, 36(11):50-69, 1993.

[31] Laplace, P. S., "Sur les Naissances, les Mariages et les Morts" *Histoire de L'Académie Royale des Sciences*, Paris, 1786.

[32] MacDonald, F., "Computer-Supported Software Inspection", PhD Thesis, Dept. of Computer Science, University of Strathclyde, UK, 1998.

[33] Martin, J. and Tsai, W. T., "N-fold Inspections: A Requirements Analysis Technique", *Communications of the ACM*, 36(11):51-61, 1990.

[34] Miller, J., Wood, M. and Roper, M., "Further Experiences with Scenarios and Checklists", *Empirical Software Engineering*, 3(3):37-64, 1998.

[35] Miller, J., "Estimating the Number of Remaining Defects after Inspection", *Software Testing, Verification and Reliability*, 9(4):167-189, 1999.

[36] Montgomery, D., *Design and Analysis of Experiments*, John Wiley and Sons, USA, 1997.

[37] Musa, J. D., *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*, McGraw-Hill, USA, 1998.

[38] Myers, G. J., "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections" *Communication of ACM*, 29(9):760-768, 1978.

[39] Otis, D. L., Burnham, K. P., White, G. C. and Anderson, D. R., "Statistical Inference from Capture Data on Closed Animal Populations", *Wildlife Monographs*, 62, 1978.

[40] Parnas, D. L. and Weiss, D. M., "Active Design Reviews: Principles and Practices", *Proc. of the 8th International Conference on Software Engineering*, pp. 418-426, 1985.

[41] Petersson, H. and Wohlin, C., "Evaluation of using Capture-Recapture Methods in Software Review Data", *Proc. of the 3rd International Conference on Empirical Assessment & Evaluation in Software Engineering*, 1999.

[42] Petersson, H. and Wohlin, C., "An Empirical Study of Experience-Based Software Defect Content Estimation Methods", *Proc. of the 10th International Symposium on Software Reliability Engineeing*, pp. 126-135, 1999.

[43] Petersson, H. and Wohlin, C., "Evaluating Defect Content Estimation Rules in Software Inspections", *Proc. of the 4th International Conference on Empirical Assessment & Evaluation in Software Engineering*, 2000.

[44] Pollock, K. H., "Modeling Capture, Recapture, and Removal Statistics for Estimation of Demographic Parameters for Fish and Wildlife Populations: Past, Present, and Future", *Journal of the American Statistical Association*, 86(413):225-238, 1991.

[45] Porter, A., Votta, L. and Basili, V. R., "Comparing Detection Methods for Software Requirements Inspection: A Replicated Experiment", *IEEE Transactions on Software Engineering*, 21(6):563-575, 1995.

[46] Redwine. S. and Riddle, W., "Software Technology Maturation", *Proc. of the 8th International Conference on Software Engineering*, pp. 189-200, 1985.

[47] Regnell, B., Runeson, P. and Thelin, T., "Are the Perspectives Really Different? - Further Experimentation on Scenario-Based Reading of Requirements", *Empirical Software Engineering: An International Journal*, 5(4):331-356, 2000.

[48] Rexstad, E. and Burnham, K. P., "User's Guide for Interactive Program CAPTURE", Colorado Cooperative Fish and Wildlife Research Unit, Colorado State University, Fort Collins, CO 80523, USA, 1991.

[49] Runeson, P. and Wohlin, C., "An Experimental Evaluation of an Experience-Based Capture-Recapture Method in Software Code Inspections", *Empirical Software Engineering*, 3(4):381-406, 1998.

[50] Stringfellow, C., von Mayrhauser, A., Wohlin, C. and Petersson, H., "Estimating the Number of Components with Defects Post-Release that Showed No Defects in Testing", to appear in *Software Testing, Verification & Reliability*, 2001.

[51] Thelin, T., Petersson, H. and Wohlin, C., "Sample-Driven Inspections", *Proc. Workshop on Inspection in Software Engineering*, pp. 81-91, 2001.

[52] Thelin, T. and Runeson, P., "Capture-Recapture Estimations for Perspective-Based Reading – A Simulated Experiment", *Proc. of the International Conference on Product Focused Software Process Improvement*, pp. 182-200, 1999.

[53] Thelin, T. and Runeson, P., "Fault Content Estimations using Extended Curve Fitting Models and Model Selection", *Proc. of the 4th International Conference on Empirical Assessment & Evaluation in Software Engineering*, 2000.

[54] Thelin, T. and Runeson, P., "Robust Estimations of Fault Content with Capture-Recapture and Detection Profile Estimators", *Journal of Systems and Software*, 52(2-3):139-148, 2000.

[55] Thelin, T., Runeson, P. and Regnell, B., "Usage-Based Reading – An Experiment to Guide Reviewers with Use Cases", to appear in *Information and Software Technology*, 2001.

[56] Travassos, G., Shull, F., Fredericks, M., Basili, V. R., "Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality", *Proc. of the International Conference on Object-Oriented Programming Systems, Languages &* Applications, 1999.

[57] Vander Wiel, S. A. and Votta, L. G., "Assessing Software Design Using Capture-Recapture Methods", *IEEE Transactions on Software Engineering* 19(11):1045-1054, 1993.

[58] Weller, E. F., "Lessons from Three Years of Inspection Data", *IEEE Software*, 10(5):38-45, 1993.

[59] White, G. C., Anderson, D. R., Burnham, K. P. and Otis, D. L., "Capture-Recapture and Removal Methods for Sampling Closed Populations", Technical Report, Los Alomos National Laboratory, 1982.

[60] Wohlin, C., Runeson, P. and Brantestam, J., "An Experimental Evaluation of Capture-Recapture in Software Inspections", *Software Testing, Verification & Reliability*, 5(4):213-232, 1995.

[61] Wohlin, C. and Runeson, P., "Defect Content Estimation from Review Data", *Proc. of the 20th International Conference on Software Engineering*, pp. 400-409, 1998.

[62] Wohlin, C., Petersson, H., Höst, M. and Runeson, P., "Defect Content Estimation for Two Reviewers", to appear in *Proc. of the 12th International Symposium on Software Reliability Engineering*, 2001.

[63] Yang, M. C. K. and Chao, A., "Reliability-Estimations & Stopping-Rules for Software Testing, Based on Repeated Appearance of Bugs", *IEEE Transactions on Reliability*, 44(2):315-321, 1995.

Requirements Mean Decisions! – Research issues for understanding and supporting decision-making in Requirements Engineering

Björn Regnell[1], Barbara Paech[2], Aybüke Aurum[3], Claes Wohlin[4],
Allen Dutoit[5] and Johan Natt och Dag[1]

[1]Dept. of Communication Systems, Lund Univ., Sweden
[2]Fraunhofer Inst. for Experimental Software Eng., Germany
[3]Sch. of Information Systems, Techn. & Mgmt, Univ. of New South Wales, Australia,
[4]Dept. of Software Eng. and Computer Sci., Blekinge Inst. of Techn., Sweden
[5]Inst. für Informatik, Techn. Univ. München, Germany

bjorn.regnell@telecom.lth.se; paech@iese.fhg.de; aybuke@unsw.edu.au;
claes.wohlin@bth.se; dutoit@in.tum.de; johan.nattochdag@telecom.lth.se

Abstract

Requirements result from stakeholders' decisions. These decisions are governed by hard issues such as the balance between cost and functionality, and soft issues such as social processes and organisational politics. The quality of the decision-making process is crucial as good-enough requirements is the foundation for a successful focusing of the available development resources. In this paper it is argued that research should focus more on Requirements Engineering (RE) as a decision-making process with focus on describing and understanding it, and on providing and evaluating methods to improve and support RE decision-making. There are many opportunities of fruitful interdisciplinary research when combining RE with areas such as decision theory, decision support systems, operations research and management science. A number of research issues are identified and several aspects of RE decision-making are described, with the aim of promoting research on methods which can better support requirements engineers in their decision-making.

1 Introduction

Requirements can be viewed as the results of stakeholders' decisions regarding the functionality and quality of the software product to be constructed. Furthermore, the Requirements Engineering (RE) process needs staffing, planning, control, and organisation; all these issues are related to decision-making.

There are already existing theories and methods for decision-making in research areas such as decision theory, decision support systems, operations research and management science. The previously established large base of research results in these areas is a great resource for RE researchers to take advantage of when conducting interdisciplinary research. The objective of the presented work is to identify both descriptive research issues for *understanding* (Section 2) and prescriptive research issues for *supporting* (Section 3) RE decision-making.

2 Understanding the RE decision-making process

Although certain aspects of RE decision-making may be specific to RE, there are also many aspects which are general. Hence, RE decision-making may in part be explained using frameworks from classical decision-making theory [1, 2]. By taking existing frameworks, and relate them to decision-making in RE, a number of descriptive research issues can be identified. A number of such issues are discussed subsequently.

The RE process is communication intensive. The requirements are interpreted and decisions are made in a so called *mutual knowledge exchange process* [3]. Many stakeholders who are involved in the process make a variety of decisions that ultimately affect the effectiveness and efficiency of the software product. This process is a typical *group problem solving process*. A major challenge for RE research is thus to understand this group process and, based on this understanding, find efficient ways of supporting groups of stakeholders in solving the problem of deciding what to build.

From a management perspective, each 'requirement' takes the place of a 'decision' [4]. The decision process is both an evolutionary process and a problem solving activity, and it involves many decisions that are continuous with several levels and review points with iterations. Classical theo-

ries of decision-making in an organizational context involve three main activities: *strategic planning*, *management control* and *operational control* [5]. The strategic planning deals with decisions that are related to policy setting, choosing objectives and identifying resources. Management control deals with decisions related to assuring efficiency and effectiveness in the use of resources. Operational control deals with assuring effectiveness in performing operations.

Fig. 1 describes RE decision-making in an organizational context [5]. Strategic planning and management control in RE may include decisions such as:

(1) *scope decisions* dealing with whether a requirement is consistent with the product strategy,

(2) *resource decisions* regarding for example if more effort should be put on RE, and

(3) *responsibility decisions* where it is decided who is responsible for what in the RE process. The requirements are designed at the operational level.

Operational control may include decisions such as:

(4) *quality assessment decisions* where it is decided if a requirement is of good-enough quality,

(5) *classification decisions* where it is decided that a requirement is of a certain type, which in turn may imply specific actions, and

(6) *property decisions* where it is decided that a requirement has a certain property or value (e.g., req. X has implementation cost Y and depends on req. Z).

These decisions are made in various, inter-related and overlapping contexts such as:

(a) customer-specific systems,

(b) off-the-shelf systems,

(c) embedded systems,

(d) safety-critical systems,

(e) data-base centric systems.

A number of important research issues are related to the investigation of the nature of decision types (such as 1-6) in various contexts (such as a-e). Empirical studies of real projects with real requirements can give us a thorough understanding of types and qualities of decisions, with the benefit of providing insight into what types of decisions need what type of support in what context.

Each requirement can be viewed as an information element that is elevated in terms of quality throughout development. This view of RE as a continuous process of asynchronous information refinement is especially salient in market-driven RE [6]. The "salmon ladder" metaphor in Fig. 2 can be used to describe the life-cycle of each individual requirement in such a process. Each transition in the salmon ladder implies an operational or strategic decision. Consequently, RE research should investigate the nature of these decisions. In order to find ways of supporting decision-making in RE we need to understand issues such as: How many requirements are discarded either too early or too late? How often are requirements specified which are never released? What is the adequate quality of a requirement before it is allowed to enter the process?

3 Supporting RE decision-making

Why does RE sometimes fail? One reason may be that bad decisions are made by requirements engineers and managers during system definition. In turn, these decisions may lead to wrong or poor requirements, which subsequently may lead to a software product not fit for purpose, which eventually is rejected by the market. Consequently, a major issue for RE research is to prescribe methods and tools that can support better decision-making. This includes providing comprehensive information and stable grounds for timely decisions. For a complex system with many stakeholders, the amount of information to be handled by requirements engineers is immense. Providing structure and overview in this confusion is a central quest in order to pave the way for better decisions. Hence, *support for measurements on requirements* both for decision making in the RE proc-

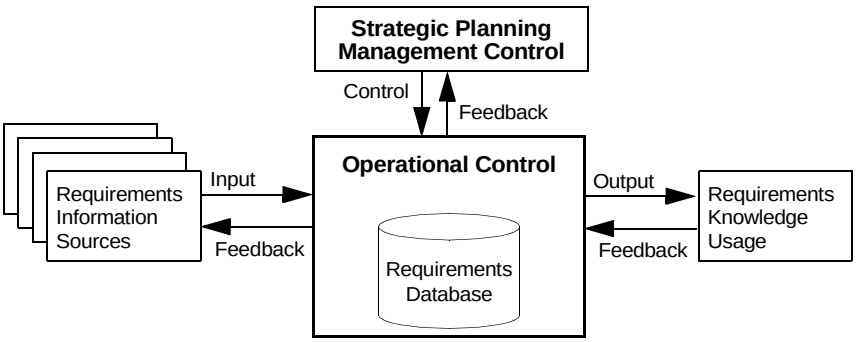


Fig. 1. Decision-making in RE at different levels, shown in an organizational context.

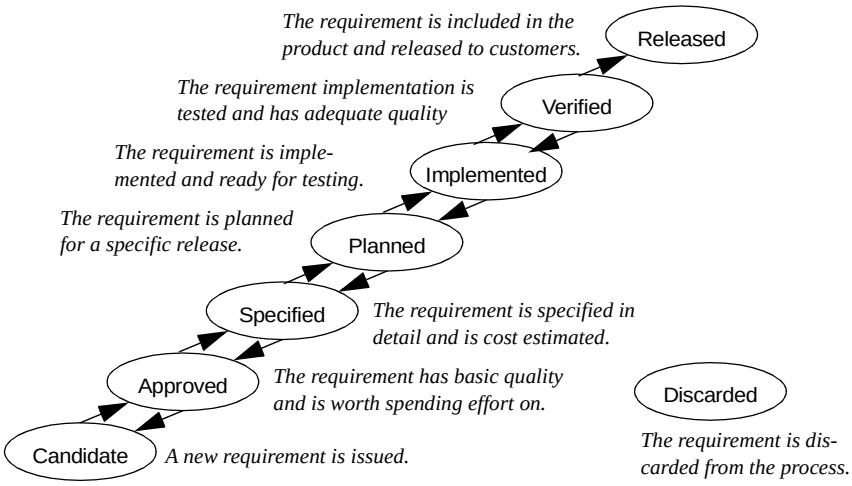


Fig. 2. An example of a "salmon ladder" where requirements are decided to be elevated or downgraded individually in a continous, asynchronous refinement process.

ess and in related processes, such as release planning and architectural design, is of great interest. Strong support for visualizing metrics allows requirements engineers to continuously answer questions such as: How much of the available construction effort is currently planned for the next software release? Which customer category will be most satisfied with the current set of planned requirements? How long does it on average take for a requirement to go from approved to specified?

The research on requirements prioritisation [7] is a striking example of how an old technique from decision theory - Analytical Hierarchy Process (AHP) [8] - after adaptation to RE, can support and improve decision-making in a new context. When adopting existing decision support methods to the special case of software requirements, it is important to investigate the underlying assumption of the methods in relation to the RE context. For example, AHP assumes that decision objects (requirements) can be treated independently, although we know that requirements depend on each other in various complex ways. Hence, research is needed on *support for management of requirements dependencies* in a cost-efficient way [6]. A related issue is *support for impact analysis*, in connection with changed decisions.

During the proposal of candidate requirements, and their subsequent approval or discarding (see Fig. 2), the decision process is characterized by intensive negotiation among multiple stakeholders. Thus, decisions made during this process are the result of the evaluation and refinement of different options. However, often only the selected option is documented in the requirements specification and the discarded options are lost. This information loss leads to costly misunderstandings about the options between the different stakeholders and a lack of *support when revising decisions*. Rationale methods [9, 10] are used to explicitly capture and manage options, decisions, and their justifications [11]. *Support for negotiation* is needed to make sure all relevant positions are represented and respected. Providing rationale-based tools to make decision steps explicit can do this. While such tools have been successful during the elaboration of complex decisions [12], several issues remain to be solved, such as training stakeholders and decreasing overhead.

Support for decision recording is needed once consensus has been achieved. When going up and down the salmon ladder, many decisions will be re-opened, sometimes without all stakeholders being available. Restructuring of the model produced during negotiation can be used for recording decisions. However, the restructuring process (e.g. identification of missing steps or obsolete decisions) using current techniques is not cost effective. The issue of cost-effectiveness in decision recording is hence a key challenge for research.

Supporting traceability between requirement decisions and their corresponding rationale is needed to assess the consistency and the impact of change to existing decisions based on the existing rationale. While this sounds straightforward, maintaining traceability is also an added cost and may not be useful at all granularity levels.

The available solutions for the issues above have had little acceptance so far, due to their lack of integration with processes and tools [13]. Thus, a major issue is to bridge the gap between group decisions support, recording decisions, and traceability. For an integrated approach to be accepted by a software development organisation, a systematic, incremental, and experimental approach should be adopted. We need to identify the applicability of solutions and evaluate the cost benefit trade-offs, reinforcing the issue of measurement on RE products and processes.

4 Conclusions

Previous research in requirements engineering has to a large extent been focused on the creation of a specification document in a contract-driven development situation. We argue that interdisciplinary research using empirical methods is needed in order to describe and understand RE as a *decision-making process* in a product development context. The major motivation from an engineering perspective for such research is to provide the basis for prescribing effective and efficient decision support. Methods and tools are needed to support areas such as: decision information management and retrieval, requirements metrics, requirements dependencies, revising decisions, and negotiation.

In summary, the following research areas have been identified and motivated:

- decisions on strategic level
- decisions on operational level
- decision contexts
- product and process metrics
- management of dependencies
- impact analysis
- decision revisioning
- negotiation
- decision recording
- traceability

These areas should be treated both descriptively and prescriptively. Research questions of a descriptive nature can provide a deeper understanding of the RE process from a decision-makers point of view. Many different kinds of empirical studies are needed in order to gain such a deep understanding. Ultimately, prescriptive research may provide empirically grounded guidelines on what methods and tools to use in what contexts, with a quantified expectancy on benefits and costs.

Acknowledgements

The presented work is a result of a research co-operation including Lund University, Fraunhofer Institute for Experimental Software Engineering, University of New South Wales and Blekinge Institute of Technology. The authors would like to express their gratitude for financing made available from each organisation. Travel funds have also been given from the Ericsson Research Exchange Scholarship.

References

[1] Vliegen, H. J. W. and Van Mal, H. H. “Rational Decision Making: Structuring Design Meetings”, IEEE Engineering Management, Vol. 37(3), pp. 185-190, 1990.

[2] Aurum, A. and Martin, E. “Managing both Individual and Collective participation in Software Requirement Elicitation Process”. Proc. 14th Int. Symposium on Computer and Information Sciences (ISCIS'99), pp. 124-131, 1990.

[3] Mallick, S. and Krishna, S. “Requirements Engineering: Problem Domain Knowledge Capture and the Deliberation Process Support”, Proc. 10th Int. Workshop on Database & Expert Systems Applications, pp. 392-397, 1990.

[4] Evans, R., Park, S. and Alberts, H. “Decisions not Requirements: Decision-Centered Engineering of Computer-Based Systems”. Proc IEEE Int. Conference and Workshop on Engineering of Computer-Based Systems, pp. 435-442, 1997.

[5] Anthony, R. N. *Planning and Control Systems: A Framework for Analysis*. Harward University, Boston, USA, 1965

[6] Carlshamre, P. and Regnell, B. “Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes”, Proc. IEEE Int. Workshop on the Requirements Engineering Process (REP’2000), 6th-8th of September 2000, Greenwich UK.

[7] Karlsson, J. and Ryan, K. “A Cost-Value Approach for Prioritizing Requirements”, *IEEE Software*, September/October 1997.

[8] Saaty, T. L. *The Analytical Hierarchy Process*, McGraw-Hill, 1980.

[9] Moran, P. and Carroll J.M. *Design Rationale: Concepts, Techniques, and Use*, Lawrence Erlbaum Associates, Mahwah, NJ, 1996.

[10] Toulmin, S. *The Uses of Argument*, Cambridge University Press, 1958.

[11] Dutoit, A. H. and Paech, B. “Rationale Management in Software Engineering”, *Handbook on Software Engineering and Knowledge Engineerin*g, World Scientific, December 2001.

[12] Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A., and Madachy, R.”Using the WinWin Spiral Model: A Case Study”, *IEEE Computer*, Vol.31(7), pp. 33-44, July, 1998.

[13] Balasubramaniam, R. and Kannan, M. “Integrating Group Decision and Negotiation Support Systems with Work Processes”, Proc. 34th Int. Conference on System Sciences, Hawaii, January 2001.

Error Management with Design Contracts

Eivind J. Nordby, Martin Blom, Anna Brunstrom
Computer Science, Karlstad University
SE-651 88 Karlstad, Sweden
E-mail: {Eivind.Nordby, Martin.Blom, Anna.Brunstrom}@kau.se

Abstract

When designing a software module or system, a software engineer needs to consider and differentiate between how the system handles external and internal errors. External errors must be tolerated by the system, while internal errors should be discovered and eliminated. This paper presents a development strategy based on design contracts to minimize the amount of internal errors in a software system while accommodating external errors. A distinction is made between weak and strong contracts that corresponds to the distinction between external and internal errors. According to the strategy, strong contracts should be applied initially to promote the correctness of the system. Before release, the contracts governing external interfaces should be weakened and error management of external errors enabled. This transformation of a strong contract to a weak one is harmless to client modules. In addition to presenting the strategy, the paper also presents a case study of an industrial project where this strategy was successfully applied.

1 Introduction

When designing a software module or system, a software engineer needs to consider and differentiate between how the system handles external and internal errors. Incorrect behaviour by end users and by external systems are typical examples of external errors. Design and programming errors are typical examples of internal errors. Such errors result in faults in the system being built. External errors have to be be tolerated by the system, while internal errors should be minimized and the faults they result in should be discovered and removed.

This paper presents a development strategy based on design contracts for error management in software development. The strategy is based on three principles. One is to make a distinction between weak and strong contracts. Another principle is the correspondance between external and internal errors and weak and strong contracts respectively. The third one is Liskov's principle of substitutability [2], which implies that a strong contract may be replaced by a weaker one without harm. This is exploited by the strategy. It prescribes to first use strong contracts to minimize internal errors and then weaken selected contracts to tolerate external errors.

During the spring of 1999, a case study was conducted of an industrial development project where the strategy described was applied. Some software modules were designed using strong contracts. Towards the end of the project, some of these contracts were weakened in order to accommodate external errors in the user interfaces. This paper also reports on the experiences from this industrial project.

The remainder of the paper is organized in two major parts, a strategy part and a case study part, followed by a conlusion. In the strategy part, the two different kinds of errors that a software engineer has to face are first presented and related to strong and weak contracts. Then, Liskov's substitution principle and Meyer's assertion redeclaration rule [3] are presented and combined, showing that the transformation of a strong contract to a weak one is a harmless operation, confirming with Liskov subtyping. The development strategy, which is to start out with strong contracts and then weakening selected contracts, is then deduced from these principles. In the case study part, the project studied is presented and the experiences from applying the strategy in this project are summarized. The conclusion from this case study is that the application of the strategy gave a positive contribution to a successful result.

2 Errors and design contracts

This section starts by briefly reviewing the distinction between external and internal errors. It then summarizes the principles for design contracts and introduces two categories of contracts, called weak and strong contracts in this paper. Finally, a correspondance is established between these contract categories and external and internal errors.

2.1 External and internal errors

This paper uses the terms error, fault and failure according to Fenton and Pfleeger [1]. An error is a dynamic prop-

erty of an actor, something wrong being done by someone or something, for instance a user misusing the system, an external system not responding correctly or a designer misunderstanding a specification. An error, for instance during design or implementation, may result in a fault, which is a static product property, a deviation from the correct implementation. A fault may cause a failure, which is a dynamic product property, implying that the system does not behave as intended. In brief, an actor may commit an error, a system may contain a fault and, as a consequence, the system may fail.

A software engineer has to face external and internal errors in development work. External errors are errors committed by actors external to the system. An end user, for instance, may enter some illegal or meaningless input, like typing letters in a number field or entering a value out of range. Similarly, an external system may malfunction, possibly because of a physical or logical fault. Examples include external storage device errors and networking problems.

Internal errors are errors committed by designers or programmers in the development team. They result in faults being built into the system itself. These faults should never have been introduced, and the ones that are should be discovered and removed as soon as possible.

One important distinction between external and internal errors is that external errors arise when the system is used while internal errors arise when it is created. External errors affect the system dynamically from the outside without modifying the software itself. To maintain system integrity and user friendliness, these errors should be tolerated and dealt with by the system. The internal errors, on the other hand, are committed by the system designers and programmers while the system is being developed. They introduce static faults into the system software. Once introduced, these faults remain in the system until they are detected and removed, potentially causing the system to fail even when used correctly. Even if internal errors can never be totally eliminated, their number should be kept low and the faults they result in should be detected and removed.

2.2 Weak and strong contracts

Design contracts are used to define the semantics of operations and to specify the responsibilities of both the client and the supplier of the operation. A contract consists of a precondition and a postcondition [3]. Correctness is achieved when the client software satisfies the precondition before calling the operation and the supplier implementation satisfies the postcondition when terminating. In the case when the precondition is not initially satisfied by the client, the supplier is not bound to satisfying the postcondition. In such a situation, the outcome of the operation is explicitly left undefined [3], [5]. This leaves the supplier the freedom to produce any result, to not terminate or to abort the execution, to name but a few examples. The actual choice is a matter of convenience. It is not a correctness issue but is considered part of the robustness. The classic example, which will also be used in this paper, is the stack and the operation `top`, returning the topmost element of the stack. This operation may be successfully completed according to this description only if the stack actually has a top element.

Two major categories of contracts are identified, called weak and strong contracts respectively in this paper. The weak contracts typically have the precondition `true`, implying that the client does not have any obligations whatsoever. Instead, the supplier must be prepared to handle even meaningless calls, like `top` being called when the stack is empty, and the definition of the operation must prescribe the outcome in such cases. Meyer [3] refers to this as the tolerant approach to contract design. The outcome will typically be some kind of error indication, like a status value being defined or an exception being thrown. This approach is illustrated in Figure 1. The notations `some property@pre` and `result = some expression` used in the figure are OCL-notations[1] for the value of the property `some property` at the start of the execution of the operation and the value returned from the operation respectively [4].

```
Precondition:  true
Postcondition: if empty@pre
               then EmptyException thrown
               else result = top element
```

Figure 1: A weak contract for `top`

Strong contracts require that the client satisfies a specific precondition, as shown in Figure 2. The postcondition only states the outcome in the legal situations, that is the situations where this precondition was true. Meyer [3] refers to this as the demanding approach to contract design or the "tough love" approach.

```
Precondition:  not empty
Postcondition: result = top element
```

Figure 2: A strong contract for `top`

2.3 Relation between contracts and errors

An interface exposed to an external system or an end user will be called an external interface, and one exposed to another part of the same system will be called an internal interface. A weak contract corresponds to accepting input errors and is suitable for external interfaces, which are

[1] OCL, the Object Contraint Language, defines a syntax to express preconditions, postconditions and other assertions. It was initially defined by IBM and is now included in the family of standards managed by by the Object Management Group (OMG).

exposed to external errors. A strong contract assumes that it is possible for the operation to be called correctly. That is possible only for operations in internal interfaces, but even clients to such operations will contain faults resulting from internal errors.

Applying weak contracts, as illustrated in Figure 1, for internal errors would correspond to accepting unacceptable situations that ultimately result from program faults. A weak contract allows meaningless calls to a supplier operation. The supplier is required to detect the error and will return the responsibility back to the client software, expecting it to take care of the returned error indication. However, handling this error indication requires as much effort from the client as assuring a correct call in the first place.

3 Transforming strong contracts to weak

One powerful property of contracts is that they may be modified under the control of logical statements. The basic principle is stated by Barbara Liskov in her principle of substitutability [2]. This principle is refined by Bertrand Meyer for class inheritance in his Assertion Redeclaration rule [3]. The same logic can be applied to the contract of an operation to predict whether the modification of the operation will affect the clients or be unnoticeable to them.

3.1 Liskov's substitution principle

Barbara Liskov stated her principle of substitutability while relating the usefulness of inheritance hierarchies in program development to data abstraction [2]:

> If for each object o_1 of type S there is an object o_2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o_1 is substituted for o_2, then S is a subtype of T.

In summary, it states that the type of an object is a subtype of the type of another object if it is impossible to observe any difference in behavior when the latter object is substituted by the former. This property is also wanted when a contract is replaced by another in a module, since it allows the client environment of the module to remain unchanged across the modification. We therefore need a principle of substitutability for contracts, answering the question when a contract defines a module to be a subtype of another.

3.2 Transparent transformations

If a contract defines one module to be a subtype of another, replacing the latter by the former is a transparent operation as seen from the clients' point of view. This corresponds to Meyer's Assertion Redeclaration rule for classes [3]. It expresses when an object of a subclass can replace an object of its superclass without affecting the clients of the class.

> A routine redeclaration may only replace the original precondition by one equal or weaker, and the original postcondition by one equal or stronger.

3.3 Definition of strong and weak contracts

Up till now, the terms strong and weak contract have been defined intuitively only. Now, they can be defined in a somewhat more precise way. A contract is strong or weak relative to another one. That means that a contract can be stronger than or weaker than another one but no absolute measure of "strongness" is defined. Our definition of when a contract is stronger than another is given below:

> If two contracts obey the Assertion Redeclaration rule of Section 3.2, then the original contract is said to be stronger than the redefined one.

This definition automatically implies that the transformation of a contract to a weaker one follows the Assertion Redeclaration rule. Such a transformation is transparent to clients of the operation since it does not affect their behavior, as paralleled by Liskov's principle of substitutability.

3.4 Development strategy and expected effects

The main principles discussed so far are summarized below.

- External errors should be managed by the system.
- Internal errors should be minimized and the faults introduced identified and removed.
- Weak contracts are useful for tolerating external errors.
- Strong contracts are useful for detecting and removing faults introduced by internal errors.
- Strong contracts can be weakened without affecting their clients.

Combining these observations, we propose the following development strategy.

> When developing a system with external interfaces, start out with strong contracts for all operations and equip the operations with a contract violation detection mechanism. Then weaken selectively the contracts of the external interfaces to tolerate external errors and add robustness in the external interface.

A discussion of contract violation detection mechanisms is outside the scope of this paper, but inspections and runtime monitoring are two relevant alternatives. They may be used alone or in combination. Similarly, a discussion of alternative techniques for contract weakening is also outside the scope of this paper. The alternatives include modification of the interface, inheritance and wrapping.

The proposed strategy focuses on correctness and contract conformity. The primary expected effect of this is a decrease in the number of faults. It also focuses on a consistent use of strong contracts. An expected effect of this is a lower product complexity, which in turn is expected to reduce both development time and the number of faults. As a result of the reduced number of faults, the time spent on testing and fault correction is also extepted to be reduced. The final weakening of the contracts will probably contribute to some increase in development time, but planning for this weakening should minimize the extra effort and time needed. This increase in time should also be compensated for by the savings mentioned. The case study reported in the rest of this paper supports these expectations, but more research is needed to draw decisive conclusions.

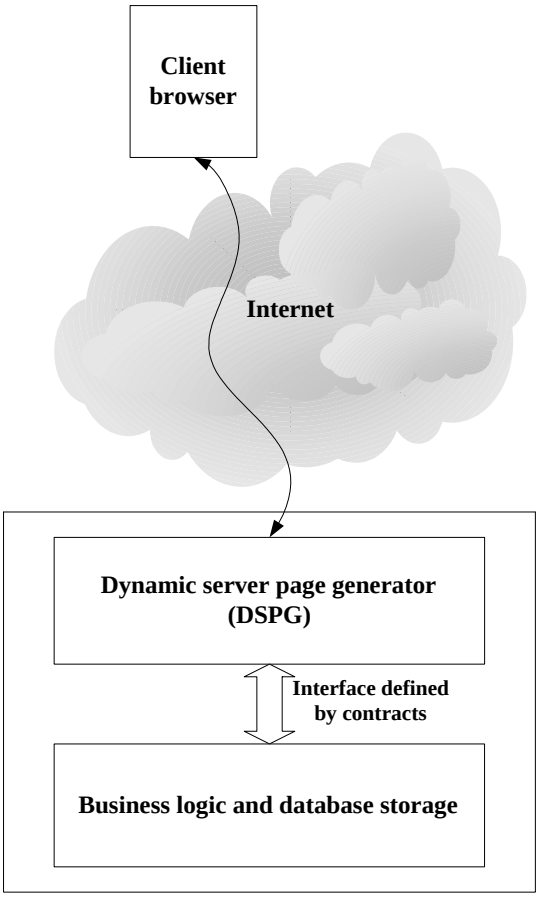


Figure 3: Architectural overview of the system

4 Presentation of the case study

As mentioned initially, a case study of an industrial project where the strategy presented above was applied has been conducted. The remainder of this paper presents the case study and the experiences gained from it. This section starts by an overall presentation of the nature and architecture of the software system produced in the industrial project. It then identifies the nature of the interfaces in the system and presents the contracts used initially by one of the system modules.

4.1 Overall system description

The system produced by the project studied is a wap[2] server that also includes a web interface. It uses dynamic server pages technology to allow the end users to access and modify user defined menu structures in a data base hosted by the server. Access to the system is through wap enabled telephones or through standard web browsers, at the user's discretion. The parts of the overall system architecture of relevance to this paper are shown in Figure 3.

The project was of medium size. It involved about 10 persons, most of them full time, for a period of 6 months. The size of the resulting software produced during the project is 15,800 new lines of code, including comments and empty lines.

The whole system consists of a client browser and the server system, the latter being divided into the dynamic server page generator (DSPG) and the business logic and database storage. The user interacts with the system by selecting a menu alternative or by clicking on a button in the wap or web pages displayed in the browser. A user command is transformed into a URL with parameters and transmitted across Internet. On the server side, a dispatcher transforms it into a call with parameters to an operation in the DSPG module. The business logic part supports this module with tailored operations on the data structure, which is stored in the database.

Much of the functionality in the system consists of routines to allow the user to manipulate the menus to be used from the wap telephone. A user will define menus containing his or her most common telecom services or links to frequently visited wap or web pages. These menus are presented as a line oriented series of choices. The operations the user can use to configure the wap menus include operations to add a new menu selection, to move a selection within a menu to another menu, to define the details of a selection and to remove a selection. The user can also define new menus, link menus to each other and delete menus.

[2]Wireless Application Protocol, a standard for providing Internet communications on digital mobile phones and other wireless terminals

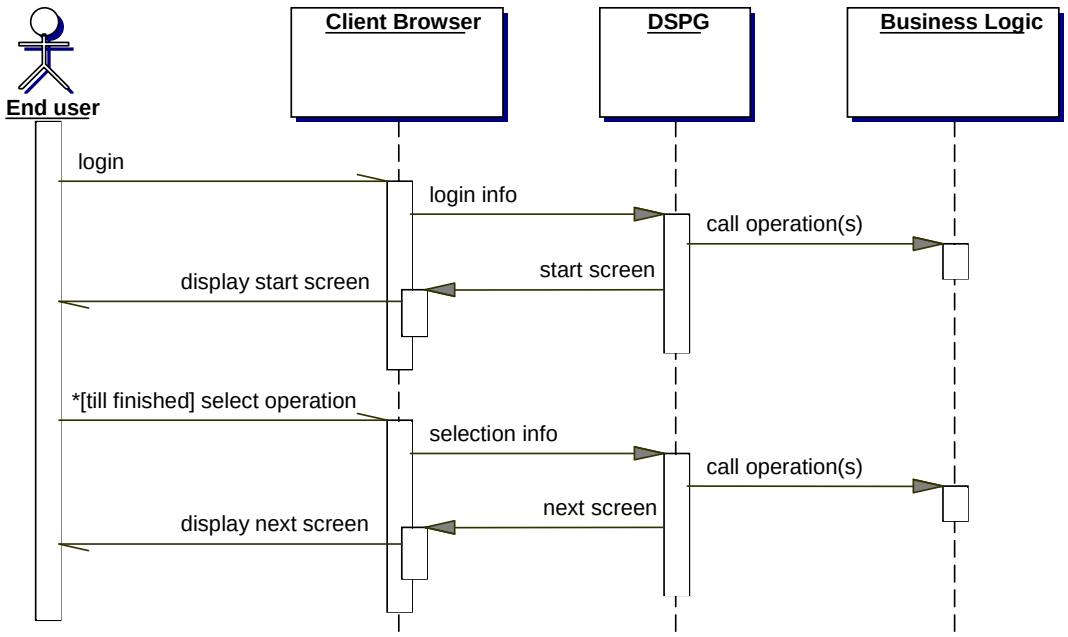


Figure 4: Browser/server interaction

4.2 Identification of interfaces

Three principal interfaces can be identified in this architecture. One is the user interface, represented by the wap and web pages in the client browser. The next one is the server interface, managed by the DSPG. Finally, there is the business logic interface. Of these, the first one is an external interface and the last two are logically internal interfaces, since they are under the direct control of the software. In this setup, the user may only call the operations and supply the arguments defined by the current browser page, which in turn is defined by the DSPG. These interactions are illustrated in Figure 4.

4.3 The initial choice of contracts

Consistent with the strategy proposed in this paper, the interface to the business logic module was defined with strong contracts. The module contained 17 classes with a total of about 70 public operations defined using strong contracts. Including support operations, this accounts for about 6,000 lines of code, including comments and empty lines, or about 40% of the total code size. The contract for the operation to retrieve the details of a menu selection can be taken as an example. It is shown in Figure 5.

```
Precondition:  the item exists in the menu
Postcondition: result = details for item
```

Figure 5: The contract for retrieving the details for a menu selection

According to the contract theory, the implementation of this operation will assume that the item is actually present in the menu, so this condition will not be checked by the code. The resulting implementation is illustrated by the pseudo-code in Figure 6.

```
loop from first item
  compare current item with parameter
until parameter item found
return the details of the current item
```

Figure 6: The pseudocode for retrieving the details for a menu selection

This implementation is consistent with the precondition, which states that the item searched exists in the menu. That implies for instance that the menu contains at least one element, so the loop will run at least once. It also implies that there is no need to check for the end of the list, since the element searched will always be found before the end of the list is reached. Also consistent with the contract principle is the fact that there is no strategy to recover in case the precondition is not satisfied. It is assumed to be satisfied.

5 Experiences from strong contracts

This section summarizes some experiences from the application of strong contracts in the business layer of the project. It starts with a description of how the focus on correctness allowed a fast implementation of the business logic part. This is followed by a report on three effects of the strong contracts on the programming of the DSPG software. Thanks to the strong contracts, an error detection mechanism could pinpoint violations made by the

DSPG programmers. They soon learned to respect the contracts and this helped them to keep the number of faults introduced in the system down. Strong contracts in combination with some kind of violation detection mechanism showed to be a strong tool for correctness. During testing, the system also exhibited a more stable failure profile than an earlier, comparable project.

5.1 Focus on correctness

As usual, the project was under time pressure, and the business logic part was essential for the DSPG part to progress. Two designers were assigned the responsibility to produce the business logic part. After an initial phase settling the design principles, the contracts for the operations were defined. After that, the operations could be rapidly implemented, focusing on correct functionality and avoiding error checking of input parameters. The implementation assumed that a precondition that was stated in the contract was always satisfied. This procedure allowed a fast and fault free implementation of the module. Only on two occasions after the internal delivery to the project were minor adjustments needed in this part of the system.

5.2 Use of violation detection

With the finished business logic module, the DSPG programmers could progress. In this stage, the project took advantage of the potential in the contracts to detect and signal contract violations. The error checking mechanisms in Java were exploited to detect contract violations. The business logic module had no error checking in it, but as soon as a precondition of an operation was not satisfied, the operation would perform some kind of illegal operation, for instance indexing an element out of bounds or attempting to reference an object through a null pointer. This would be caught by the built-in error control in Java. Typically, the program would then crash with a run-time exception.

5.3 Client programmers conforming to the rules

To start with, the information to the DSPG programmers about the strong contracts used was insufficient. Being used to less strict function definitions, they did not pay so much attention to the details in the calls and frequently made the error to violate the preconditions. Since violations of the preconditions caused the program to crash, they could not progress with their work until they conformed with the contracts. This, of course, caused a lot of frustration and was a very strong motivation to study and conform with the rules set up and to produce fault-free code.

5.4 Absence of errors in the client modules

All the frustration and system crashes were not in vain. Two facts could be noted. One, already mentioned, was that the business logic software produced the correct result. Only two faults were reported during testing and both were easily corrected. The other fact is that even the DSPG software was free from faults in its communication with the business logic part. The programmers made fewer and fewer errors and the faults that existed during development were rapidly discovered and removed during module testing.

5.5 Stability with respect to failures

During system testing, if a module had a fault, it was discovered during the early test cases. If for instance ten test cases were run without failure before delivery, subsequent test cases run by the customer were also failure free. This is different from earlier projects not using contracts, where ten test cases could be run without failure but later test cases run by the customer after delivery could experience transient failures, revealing a fault. For the project reported in this paper, transient failures were not a problem and new faults were normally not discovered after delivery.

6 Weakening the contracts

Some of the operations defined by strong contracts were exposed to external errors through an interface accessible to the end users. These operations therefore had to use weaker contracts in the finished product. The change from strong to weak contracts was done late in the project. The operations, whose contracts needed to be weakened, as well as the weakening strategy chosen are presented in this section. After that, the experiences gained during this process are presented, showing that the weakening of the contracts did not cause any noticeable problems.

6.1 Identification of contracts and strategy

The first step was to identify the contracts that needed to be weakened. The end user interface, as it appears in the wap or web browsers, are under the control of the system. This forces a correct use of most of the operations. However, the calls from the client browser to the server pass as standard URL strings that may be entered or repeated by the end user, potentially producing an illegal call to the server. Therefore, the operations accessible directly from the Internet interface were the candidates for weaker contracts. The project identified 16 such operations.

Two main strategies were considered to make these operations tolerant to external errors. One was to implement explicit inquiry operations for the contract preconditions and implement a wrapper module with the weaker contracts. The other was to modify the operations themselves. The first strategy was the most modular one, but the second one was chosen. The main reason for this choice, was that it could be implemented faster. The modification to weaker contracts had not been anticipated sufficiently

early, so there was no support for the first strategy in the module and there was no time to implement it.

6.2 Experience from contracts weakening

With a strong contract, all the input conditions that do not satisfy the precondition are invalid. When the contract is weakened, some or all of these conditions are defined to be valid and special cases are added to the postcondition to specify their result. The implementation of the operation must then be modified to take care of these extra cases according to the new postcondition. The project confirmed that this could be done and that the modified operations did not affect existing client code that already satisfied the stronger contract.

6.3 Adaptation of client modules

The contracts were weakened by specifying that some exceptions should be thrown in the new special cases now allowed in the preconditions. In order to accommodate external errors, some client code was modified to catch the new return situations defined by the new postconditions. This was easily done by adding code to catch the exceptions thrown and display a user message stating that the call was not valid. All these modifications were easy to control and did not cause problems or introduce new faults.

7 Conclusions and furher study

We have presented a strategy based on design contracts for error management during software development. The strategy states that the development of a subsystem should be based on strong contracts in order to identify and eliminate internal errors. Before delivery, the contracts for subsystems with external interfaces should be weakened in order to tolerate external errors. The strategy is based on the mapping that exists between contracts and errors, where weak contracts are appropriate for interfaces exposed to external errors and strong contracts are appropriate for interfaces exposed to internal errors. As an extension to Liskov's principle of substitutability, rules were provided for the transformation between strong and weak contracts. A strong contract can be substituted by a weaker one without affecting the clients of the operation defined by the contract. This is based on the same reasoning as Meyer's Assertion Redeclaration rule for subclassing.

We have also presented a case study of an industrial project where the strategy was successfully applied. The interface to the business logic module was defined with strong contracts. This proved efficient in keeping down the number of faults in both the business logic module itself and its clients. It also contributed to making the system stable with respect to failures with few transient failures both before and after delivery. Late in the project, the contracts of the operations accessible to the external user interface were weakened to tolerate external end user errors. The adaptation of the implementation to these weakened contracts did not introduce new faults.

The expected effects of the strategy is a total gain in time and quality, as presented in Section 3.4. Although the case study supports the positive effects of the strategy, more research is required to be conslusive.

References

[1] Fenton, N. E., Pfleeger, S. L., *Software Metrics, A Rigorous & Practical Approach, second edition*, PWS Publishing Company 1997

[2] Liskov, B., *Data Abstraction and Hierarchy* OOPSLA '87 Addendum to the Proceedings, October 1987.

[3] Meyer, B., *Object Oriented Software Construction, 2nd edition*, Prentice Hall, 1997

[4] Warmer, J., Kleppe, A., *The Object Constraint Language, Precise Modeling with UML*, Addison Wesley, 1999.

[5] Rumbaugh, J. et al, *The Unified Modeling Language Reference Manual*, Addison Wesley, 1999.

First Swedish Conference on Software Engineering Research and Practise : Proceedings
Editor: PerOlof Bengtsson

ISSN 1103-1581
ISRN BTH-RES--01/10--SE

Printed by Kaserntryckeriet AB, Karlskrona 2001