Exploring Industrial Data Repositories: Where Software Development Approaches Meet

Parastoo Mohagheghi, Reidar Conradi
Department of Computer and Information Science, NTNU, NO-7491 Trondheim, Norway
parastoo@idi.ntnu.no, conradi@idi.ntnu.no

Abstract

Lots of data are gathered during the lifetime of a product or project in different data repositories that may be part of a measurement program or not. Analyzing this data is useful in exploring relations, verifying hypotheses or theories, and in evaluating and improving companies' data collection systems. The paper presents a method for exploring industrial data repositories in empirical research and describes experiences from three cases of exploring data repositories of a large-scale telecom system: A study of defect reports, a study of change requests, and a study of effort. The system in study is developed incrementally, software is reused in a product line approach, and the architecture is component-based. One main challenge is the integration of the results of studies with one another and with theory. We discuss that the challenges of integration especially arise when development approaches meet one another, while metrics and measurement programs do not. In order to develop advanced theories on the relations between development approaches and their impacts, measurement programs should be updated to collect some basic data that meets all the development approaches. A set of metrics for incremental, reuse-, and component-based development is identified.

Keywords: Data repositories, data mining, , metrics, component-based development, incremental development, reuse.

1. Introduction

Exploring industrial data repositories for valuable information has been performed for many decades and the fields of *Data Mining*

and *Exploratory Data Analysis* (EDA) have grown to become own branches of computer science. With the growing rate of empirical studies in software engineering and the gained approval of such studies for assessing development approaches and verifying theories, exploring data collected in industrial data repositories is more often performed, standing alongside other empirical methods. The goals of such studies can be exploratory (finding relations or distributions), confirmatory (verifying relations or theories), or used in triangulation for putting different sources of information against each other. Data repositories are also used in searching for design patterns, user interaction patterns, or reengineering legacy systems. For companies, the studies are useful to give insight into their collected data and to assess internal measurement programs and data collection systems. The focus of this paper is on data that can be used to assess quality of software or software development processes.

We present three empirical studies of exploring data repositories of a large telecom system developed by an Ericsson organization in Grimstad-Norway. These repositories contained defect reports, change requests, and effort reports for several releases. We also used data from the configuration management system on software size. Data for 3 years of development is collected in 2003 and 2004. The goals of the studies were to: a) quantitatively assess hypotheses related to reuse and quality metrics such as defect-density and stability of software components, b) explore the origin of software changes, and c) adopt an estimation method for incremental development of software. We describe steps in exploring data repositories, the role of literature search in the process, and the importance of relating hypotheses to one another and to a theory or model. We describe the challenges of integrating the results of these studies. The first challenge is the physical challenge since data is stored in different data repositories and in multiple formats. The second challenge is related to the conceptual integration of results for comparing and combining these in order to build theories. We discuss that problems in combining results especially arise when development approaches meet one another, while metrics are not defined to do so. In this case, incremental, use-case driven, reuse, product line, and component-based development approaches are used in parallel. We propose therefore to define metrics in a way that we can collect data to assess each

approach, the combinations of these, and their impacts on one another.

The remainder of this paper is organized as follows. Section 2 discusses research methods, the role of exploring industrial data repositories in empirical research, and steps in such a study. Section 3 presents the studies performed in Ericsson, while Section 4 summarizes the research challenges. Section 5 presents metrics for a combination of development approaches. The paper is concluded in Section 6.

2. Exploring industrial data repositories in empirical research

2.1. Research classifications

Coop et al. classify research design using 8 descriptors. One of the descriptors is the degree to which the research question has been crystallized, which divides research into *exploratory* and *formal* research [Coop01]. The objective of an exploratory study is to develop research questions or hypotheses and is loosely structured. The goal of a formal research is to test the hypotheses or answer the research questions.

Empirical research is research based on the scientific paradigm of observation, reflection, and experimentation. Empirical studies may be exploratory or formal as any other research. Empirical studies vary in scope, degree of control that the researcher has, and the risk associated with such studies. Wohlin et al. classify empirical strategies in three categories [Wohl00]: *surveys*, *case studies*, and *experiments*. Yin extends research strategies to five, adding *archival analysis* and *history* to research strategies [Yin02]. He does not provide further description of these strategies, except for defining archival analysis most suitable for exploratory studies, while history analysis is proposed for explanatory studies (answering how and why questions). Zelkowitz et al. classify validation methods as *observational*, *historical*, and *controlled*, which can be referred as research methods as well [Zelk98]. Wohlin et al. also divide empirical research into being *quantitative* (quantifying a relation) or *qualitative* (handling other data than numbers; i.e. texts, pictures, interview results, etc). A theory or even a hypothesis should be studied by a combination of methods. For example, the Conjecture 9 in [Endr04] says, “learning is best accelerated by a combination of controlled experiments and case studies”.

Coop et al. define data mining as “the process of discovering knowledge from

databases stored in data marts or data warehouses [Coop01]. The purpose is to identify valid, novel, useful, and ultimately understandable patterns in data. It is a step in the evolution from business data to information". They add, "data mining tools perform exploratory and confirmatory statistical analyses to discover and validate relationships". When data is stored in repositories with little or no facilities for mining with data mining tools, other research methods should be applied.

2.2. Role of exploring industrial data repositories in empirical research

With *industrial data repositories*, we mean contents of defect reporting systems, source control systems, or any other data repository containing information on a software product or a software project. This is data that is gathered during the lifetime of a product or project and may be part of a measurement program or not. Some of this data is stored in databases that have facilities for search or mining, while others are not.

Zelkowitz et al. define examining data from completed projects as a type of *historical study* [Zelk98]. Using Yin's terminology, it is classified either as archival analysis or history. We mean that this is a quantitative technique where the results should be combined with other studies of both types in order to understand the practice or to develop theories.

As the fields of Software Process Improvement (SPI) and empirical research have matured, these communities have increasingly focused on gathering data consciously, according to defined goals. This is best reflected in the Goal-Question-Metric (GQM) paradigm developed first by Basili [Basi94]. It states that data collection should proceed in a top-down rather than a bottom-up fashion. However, some reasons why bottom-up studies are useful are:

1. There is a gap between the state of the art (best theories) and the state of the practice (current practices). Therefore, most data gathered in companies' repositories are not collected following the GQM paradigm.
2. Many projects have been running for a while without having improvement programs and may later want to start one. The projects want to assess the usefulness of the data that is already collected and to relate data to goals (reverse GQM).
3. Even if a company has a measurement program with defined goals and

metrics, these programs need improvements from bottom-up studies.

Exploring industrial data repositories can be part of an exploratory (identifying relations or trends in data) or formal (confirmatory; validate theories on other data that the theories were built on) empirical research; e.g. in order to study new tools, techniques or development approaches. It may be used in *triangulation* as well; i.e. setting different sources of information against each other.

Exploring industrial data repositories may be relatively cheap to perform since data is already collected. It has no risks for the company for interfering with on-going activities. Sometimes extra effort is needed to process the data and insert it in a powerful database. An important aspect is the ethical one; i.e. having the permission to perform such studies in companies and publish the results. The limitations are that the quality of the gathered data is sometimes questionable, data needs cleaning or normalization and other types of preparation before it may be used, and the hypotheses are limited to the available data. Limitations have impact on validity of the results. For example:

- Missing data can reduce the power of statistical tests in hypotheses testing.
- Generalization of results from single studies needs a clear definition of population. Some researchers mean that generalisation based on single studies is possible if the context is well packaged and the case is carefully selected [Flyv91].

2.3. Steps in exploring industrial data repositories

Figure 2 shows the main steps in our research. A description of each step is given below.

The *theoretical phase* of the study starts either with a defined hypothesis or theory to assess, or some research or management question to answer. We emphasize the role of literature research or other secondary data analysis in the process. With such a study, possible results will be integrated into the total body of knowledge; i.e. not stay stand-alone and without any connection to a model or theory.

The *preparation phase* consists of a pre-study of data and definition of hypotheses or theory for the context (the particular product, project, and environment). The researcher must decide whether to use the entire data or a sample of it. After the data set is selected, it

should be explored visually or numerically for trends or patterns. EDA techniques are also used in the exploring. Most EDA techniques are graphical such as plotting the raw data, with the means and standard deviations etc. Together with the pre-study of data, tools and statistical techniques for the analysis should be selected. Results of the preparation phase may invoke further need for literature search or refinement of research questions.

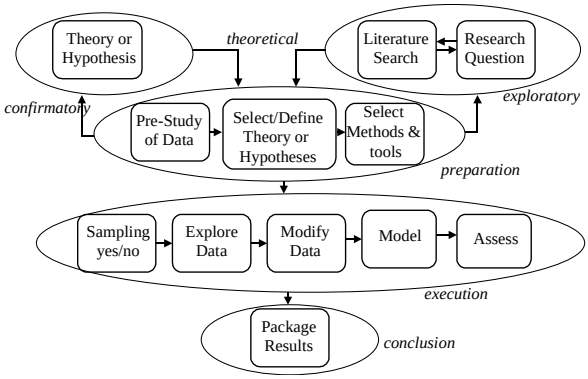

Figure 1. Steps in the process of exploring industrial data repositories in empirical research.

The *execution phase* consists of steps of a data mining process as described in [Coop01]. The data is formally sampled if necessary and fully explored. Data may need modification, e.g. clustering, data reduction, or transformation. Cooper et al. call the next step for modelling, which uses modelling techniques in data mining (neural networks, decision trees etc.). In the last step of the execution phase, hypotheses or theories should be assessed or research questions should be answered. Finally the results and the context are packaged and reported in the *conclusion phase*.

Very much like GQM, there is a hierarchy of goals, questions and metrics in Figure 1. But there is also a feedback loop between preparation and theoretical phases, due to the impact of the bottom-up approach. Questions may be redefined or hypotheses may be dropped if we do not data to assess them. However, there is no control of treatments, although the study may be applied to contemporary events as well.

There are several interesting examples of successful use of industrial databases for developing theories; e.g. Lehman developed the laws of software evolution by studying release-based evolution of a limited number of systems [Lehm96].

3. Empirical studies in Ericsson

3.1. The context

Ericsson has developed several releases of two large-scale telecom systems using component-based development and a product

line approach based on reusing software architecture and software components. Systems are developed incrementally and new features are added to each release of them.

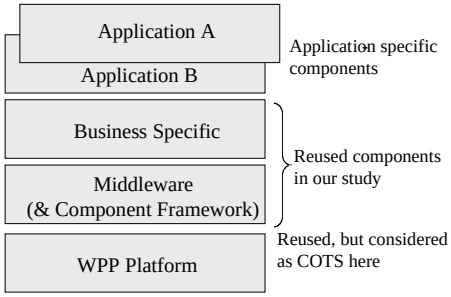

Figure 2. High-level architecture of systems A & B

The high-level software architecture is shown in Figure 2. The first system (system *A*) was originally developed to provide packet data capability to the GSM (Global System for Mobile communication) cellular network. A later recognition of common requirements with the forthcoming WCDMA system (Wide-band Code Division Multiple Access) lead to reverse engineering of the original software architecture to identify reusable parts across the two systems. The two systems *A* and *B* in Figure 2 share the system platform, which is considered here as a Commercial-Off-The-Shelf (COTS) component developed by another Ericsson organization. Components in the middleware and business specific layers are shared between the systems and are hereby called for *reused components* (reused in two distinct products and organizations and not only across releases). Components in the application-specific layer are specific to applications and are called for *non-reused components*. All components in the middleware, business specific, and application-specific layers are built in-house.

The term *component* is used on two levels: for *subsystems* at the highest level of granularity and for *blocks.* The system is decomposed in a number of subsystems. Each subsystem is a collection of blocks and blocks are decomposed in a number of units, while each unit is a collection of software source code modules. Subsystems and blocks have interfaces defined in IDL (Interface Definition Language) and communication between blocks inside a subsystem or between subsystems happens through these interfaces. Communication within a block or unit is more informal and may happen without going through an external interface.

The systems' GUIs are programmed in Java, while business functionality is programmed in Erlang and C. Erlang is a functional language for programming concurrent, real-time, distributed, and fault-tolerant systems. The size of systems measured in equivalent C code is more that one million lines of non-commented source

code. The development process is an adaptation of the Rational Unified Process (RUP) [RUP]. RUP is an incremental, use-case driven, and UML-based approach.

We collected and analyzed data gathered in the defect reporting, configuration management, change management, and effort reporting systems for three years of software development. Some results are described in [Moha04a] [Moha04b]. In [Moha03], we discuss how the results can be used to assess development approaches and measurement programs. We give a brief overview of three studies here. The external validity of all studies is threatened by the fact that the entire data set is taken from only one company. The results may be generalized to other systems within the same company or in similar domains.

3.2. Study of defect-density and stability of software components in the context of reuse

In order to quantitatively assess the impact of reuse on software quality, we decided to analyze data that is collected in the defect reporting and the configuration management systems. The defect reporting system included over 13000 defect reports (corrective maintenance activity) for several releases of the systems. For three releases of system *A*, we had data on the components' size in Lines of Code (LOC) from the configuration management system.

Theory and preparation: Study of defects is usually reported connected to the subject of *reliability* (the ability of a system to provide services as defined), which is thoroughly studied in literature. However, reliability of component-based systems is a new field with few reported empirical studies. Based on the literature search and a pre-study of the available data, we found two groups of research goals:

1. Earlier theories or observations such as correlation between size of a component and its defect-density or the number of defects. Some studies report such a correlation while others not.
2. Studying relations between reuse and software quality metrics. Some results are reported from case studies in industry or university experiments.

We defined 4 hypotheses for quantitative assessment. We decided to assess whether reused components have less defect-density than non-reused ones have and are more stable (less modified between releases). We also decided to assess whether there is a

relation between component size, and number of defects or defect-density, for all components and reused vs. non-reused ones (combining group 1 and 2). We chose Microsoft Excel and Minitab for performing statistical analysis.

Execution and results: We did not take a sample but used the whole dataset of some releases. All data on defects and components' size were inserted in a Microsoft SQL database using a C# program. Data for two releases of system *A* were used to assess hypotheses. Our results showed that size did not correlate with defect-density. Only for non-reused components, size correlated with the number of defects. Reused components had significantly less defect-density than non-reused ones and were less modified between releases. We concluded that reused components are designed more thoroughly and are changed with more care. One confounding factor is the type of functionality since non-reused components have more external interfaces than the reused ones have.

Contributions and experiences: Besides answering the research questions, the study was also useful for assessing the defect reporting system. The templates for reporting a defect had changed several times, introducing inconsistencies. Many Trouble reports had missing fields that reduced the internal validity of the results.

Research challenges: We met several challenges in the study:

1. The granularity of component definition: Some defect reports have registered only the subsystem name, while others have registered block name, unit name, or software module name. The main reason is that the origin of fault was not known when the defect was reported and the defect reports are not updated later for this information. We assessed our hypotheses both with subsystems and blocks with similar results. However, the number of subsystems was too low (9-10) for statistical tests.
2. The concept of reuse: Reuse may happen in the releases of the same product, or in multiple products and across organizations. Some mean that the first type cannot be classified as reuse. We defined a reused component to be a component that is used in more than one product.
3. Incremental and component-based development: Ideally hypotheses on defect-density should be assessed for both pre-release and post-release

defects. As mentioned by Fenton [Fent00a], the results may differ and those modules that have most pre-release faults, may have least post-release faults. But this turned out to be difficult, if not impossible with the current data of several reasons: Only the whole system is labeled with a release date and not components, the development of a new release is usually running in parallel with testing of the previous one, a component is usually involved in several use cases and is therefore updated and tested by several teams etc. Thus, relating defects to component releases or life-cycle phases was difficult.

3.3. Study of software change

We performed an exploratory study of the contents of the change management system. The database consisted of 160 Change Requests or CRs of 4 releases of system *A*. CRs are issued to add, delete, or modify a requirement after requirement baseline, or to add or modify a solution or documentation. The quality attributes related to software change are *stability, evolvability* or *maintainability* (or need for such).

Exploring the database: The variables that we had data on were size of components in LOC, type of components (reused or non-reused), and CRs in different releases. CRs are written in FrameMaker and Word using templates that have changed a few times and contain information on reason for the request, consequences, affected components, estimated effort etc.

Hypotheses selection based on literature and data: We found studies on distribution of maintenance activities and one study on the improvement of maintainability using a component-based architecture. Studies on requirement engineering have assumed that most changes are due to external factors (changing environment or customer needs). We found no study that on the origin of changes in more details. We decided to assess the distribution of change requests in the categories used in other studies (perfective, adaptive, preventive), over functional vs. non-functional reasons, phase (pre-or post delivery, before or after implementation), and to compare change-proneness in the number of CRs/size for reused vs. non-reused components.

Selecting and normalizing data: Data from CRs were inserted in a Microsoft SQL database using a C# program and partly

manually. We noticed the same problems as described in Section 3.2 with missing data.

Contributions of the study: Our study showed that most CRs are initiated by the organization itself in order to improve a quality attribute (perfective and non-functional). The shares of adaptive/preventive changes are lower, but still not as low as reported in some previous studies. The study helped therefore to understand the origin of changes. We did not identify any significant difference between the change-proneness of reused and non-reused components. Most changes only affect one or two subsystems (high-level components). The study also showed that the percentage of accepted CRs is increasing over releases, which could be subject of further study. Performing such a study early would be useful to improve the CR reporting system. On some occasions, e.g. caused by coarse-granular components, we have too little data, which impacts conclusion validity. Missing data in some CRs is the biggest threat to internal validity.

Research challenges: We met again the challenge of the granularity of component definition: Change-proneness and the impact of CRs on sub-components could not be assessed since CRs only have information on affected subsystems and not blocks. We used the delivery data of the whole system for differing pre- and post-release CRs.

3.4. Study of effort

We have collected and partly analyzed data on the effort spent in 2 releases of system *A*. The goal of this study is to calibrate an estimation method based on use cases. This study is still going on, but it gave us insight on how effort is spent in different activities in several releases.

Selecting and normalizing data: Effort is registered using development phases such as analysis, coding, unit testing etc. for each member of a team. Teams are organized in different ways; i.e. around use cases, non-functional requirements such as performance, features that cross use cases, or 'just-in-time' for an extra task such as reengineering or re-factoring a solution or a component. There are also teams for handling methods and tools, configuration management, and system test. We received some effort data in printed form and some in Excel sheets. We had to parse the data, make consistent categories, re-group the data, insert it into new Excel sheets, and summarize it.

Experiences: There are inconsistencies in categories used in different releases and the

effort reporting system has changed in the middle of one release.

Research challenges: We met the following challenges:

1. Organizational: We had data on effort spent by each team, but teams did not record their tasks detailed enough to divide the total effort between use cases, features, or non-functional requirements. Teams are also organized in different ways, making it difficult to map teams to requirements.
2. Use-case driven approach and component-based development: Ivar Jacobson, one of the pioneers of UML, the Unified Process (UP), and use cases writes that "a component realizes pieces of many use cases and a use case is usually realized by code in many components" [Jaco03]. These two decomposition effects are known as *tangling* and *scattering* [Tarr99]. Although these effects are well known and discussed both by Jacobson and others (recently especially by the Aspect Oriented Programming community), the impacts on metrics programs and effort reporting systems are not discussed. When effort is recorded per use case, it is spread over components and vice versa.
3. Use case driven and product line development: Requirements are first defined in features that are characteristics for product line development and later mapped to use cases. Tangling and scattering effects are observed here as well.

4. Discussion of research challenges

We faced two major challenges in comparing and combining results of the studies, which are discussed in other work as well (although with other labels), but not properly solved yet. We refer to them as the *challenges of integration* in two dimensions:

Physical integration refers to integration of databases. The research method may be shared, but the techniques used for exploration of data are very context dependent. In our examples, data on defects and CRs are stored in separate data repositories without having a common interface or analysis tool. One attempt to answer the challenge of physical integration is described in [Kitc01]. The authors' measurement model consists of three layers: The generic domain, the development model

domain, and the project domain. The first two domains define the *metadata* for data sets. In this study, we achieved physical integration by inserting all data extracted in the three studies in a SQL database.

Conceptual integration refers to integrating the results of separate studies and integration of results into theories; either existing or new ones. This is not specific to this type of research and empirical studies generally suffer from lack of theories that bind several observations to one another. We observe that the conceptual challenges listed in Sections 3.2, 3.3, and 3.4 are mostly introduced in the intersection between development approaches:

- The granularity problem arises when the old decomposition system in industry meets the component-based development approach and when data is not collected consistently. For example, we could only compare change-proneness and defect-proneness of components in the highest level (subsystems) and did not have data on change-proneness of blocks.
- The reuse definition problem arises with the introduction of product line development without having consensus on definitions.
- Incremental and component-based development: metrics are either defined for the one or other approach.
- Use-case driven approach, product line development, and component-based development: effort reporting system is neither suitable for finding effort per use case or feature, nor per component.

We suggest two steps for solving these challenges and also integrating the results; both physically and conceptually:

1. Using a common database for data collection with facilities for search and data mining.
2. Defining metrics that are adapted for the combination of development approaches.

Some commercial metrics tools are available, but we have not studied them thoroughly enough to answer whether these are suitable for our purpose. The second step is the subject of the next section.

5. Metrics for incremental, reuse, and component-based development

Fenton et al. write: “Most objectives can be met with a very simple set of metrics, many of which should be in any case be available as part of a good configuration management system. This includes notably: information

about faults, failures and changes discovered at different life-cycle phases; traceability of these to specific system ‘modules’ at an appropriate level of granularity; and ‘census’ information about such modules (size, effort to code/test)” [Fent00b]. We can’t agree more, but also add that metrics should be adapted for a mixture of development approaches.

We use experiences in the three above examples and other studies we have performed in Ericsson to propose improvements and identify metrics as described:

1. Decide the granularity of ‘modules’ or ‘components’ and use it consistently in metrics. Don’t define some metrics with one component granularity and others with another, unless it is clear how to combine or compare such metrics.
2. The following data should be gathered for components:
   2.1. Size (in Lines of Code if developed in-house or if source code is available, or in other proper metrics) at the end of each release,
   2.2. Size of modified code between releases,
   2.3. Faults (or defects), with information on life-cycle phase, release and product identity,
   2.4. Effort spent on each component in each release,
   2.5. Trace to requirement or use case (this is also useful for documentation and debugging) that could be updated when the component is taken in use,
   2.6. Type: new, reused-as-is or modified,
   2.7. Change requests in each release,
   2.8. Date of delivery in a release that can be set by a configuration management system and be easily used later to decide whether a fault is detected pre- or post-release, or whether a change request is issued pre- or post-delivery.
3. The following data should be gathered for increments or releases:
   3.1. Total size of the release,
   3.2. Size of new and modified code,
   3.3. Requirements or use cases implemented,
   3.4. Effort spent in the release.
4. Effort should be recorded both per component and per use case or feature.

The list shows that it doesn’t help to define a set of metrics for a development approach without considering the impact of other

approaches. Having this data available would make it possible to assess software quality in different dimensions and answer questions such as: Are defect-density and change-proneness of components correlated? Can we estimate effort based on the number or complexity of use cases, or changes in components? Which components change most between releases? What is the impact of reuse, component-based, incremental development, or a combination of these on needed effort? Hence, we could build theories that combine development approaches.

6. Conclusions and future work

We presented three empirical studies performed by exploring industrial data repositories. We could verify hypotheses on the benefits of reuse, explore the origin the changes for future studies, and study effort distribution and adapt an estimation method, *empirically* and *quantitatively*. As our examples show, quantitative techniques may be used in different types of research. In many cases, exploring industrial data repositories is the only possible way to assess a theory in the real world.

While some concrete results are already published, this paper has the following contributions:

1. Promote the discussion on exploring industrial data repositories as an empirical research method, its advantages and limitations, and presenting a simple method to do so. The method described in Section 2.3 combines the theoretical and preparation phases defined by us, with steps of a data-mining process as defined in [Coop01].
2. Getting insight into the challenges of defining and collecting metrics when development approaches are used in parallel.
3. Identifying a basic set of metrics for incremental, component-based, and reuse-based development.

The set of metrics proposed in Section 5 does not contain any new metrics, but emphasizes that metrics should be adapted for a combination of development approaches. This basic set should be collected before we can build advanced theories on the relations between development approaches.

We plan to work further on the physical and conceptual challenges meeting measurement programs, with focus on evolution of

component-based systems in the upcoming SEVO project (Software Evolution in Component-Based Software Engineering) [SEVO04].

7. Acknowledgements

The work is done in the context of the INCO project (INcremental and COmponent-based Software Development [INCO01]), a Norwegian R&D project in 2001-2004 and as part of the first author's PhD study. We thank Ericsson in Grimstad for the opportunity to perform the studies.

8. References

[Basi94] Basili, V.R., Calidiera, G., Rombach, H.D., "Goal Question Metric Paradigm", In: Marciniak, J.J. (ed.): *Encyclopaedia of Software Engineering*. New York Wiley 1994, pp. 528-532.

[Coop01] Cooper, D.R., Schindler, P.S., *Business Research Methods*, McGraw-Hill International edition, seventh edition, 2001.

[Fent00a] Fenton, N.E., Ohlsson, N., "Quantitative Analysis of Faults and Failures in a Complex Software System", *IEEE Trans. Software Engineering*, 26(8), 2000, pp. 797-814.

[Fent00b] Fenton, N.E., Neil, M., "Software Metrics: Roadmap", *Proc. of the Conference on the Future of Software Engineering*, June 04-11, 2000, Limerick, Ireland, pp. 357-370.

[Flyv91] Flyvbjerg, B., *Rationalitet og Magt I- det konkretes videnskab*, Akademisk Forlag, Odense, Denmark, 1991.

[Jaco03] Jacobson, I., "Use Cases and Aspects- Working Seamlessly Together", *Journal of Object Technology*, 2(4): 7-28, July-August 2003, online at: http://www.jot.fm

[INCO01] The INCO Project: http://www.ifi.uio.no/~isu/INCO/

[Jørg04] Jørgensen, M., Sjøberg, D., "Generalization and Theory Building in Software Engineering Research", Accepted in the *8th International Conference on Empirical Assessment in Software Engineering (EASE2004)*, 24-25 May 2004, Edinburgh, Scotland.

[Lehm96] Lehman, M.M., "Laws of Software Evolution Revisited", In Carlo Montangero (Ed.), *Proc. European Workshop on Software Process Technology* (EWSPT96), Nancy, France, 9-11 Oct. 1996, *Springer LNCS 1149*, pp. 108-124.

[Kitc01] Kitchenham, B.A., Hughes, R.T., Linkman, S.G., "Modeling Software Measurement Data", *IEEE Trans. Software*

Engineering, 27(9): 788-804, September 2001.

[Moha03] Mohagheghi, P., Conradi, R., "Using Empirical Studies to Assess Software Development Approaches and Measurement Programs", *Proc. 2nd Workshop in Workshop Series on Empirical Software Engineering - The Future of Empirical Studies in Software Engineering (WSESE'03)*, Rome, 29 Sept. 2003, pp. 65-76.

[Moha04a] Mohagheghi, P., Conradi, R., Killi, O.M., Schwarz, H., "An Empirical Study of Software Reuse vs. Defect-Density and Stability", Proc. of the 26th International Conference on Software Engineering (ICSE'04), *IEEE Computer Society* Order Number P2163, pp.282-292.

[Moha04b] Mohagheghi, P., Conradi, R., "An Empirical Study of Software Change: Origin, Acceptance Rate, and Functionality vs. Quality Attributes", Accepted in the *2004 ACM- IEEE International Symposium on Empirical Software Engineering (ISESE'04)*, 10 p.

[Tarr99] Tarr, P., Ossher, H., Harrison, W., Sutton, S., "N Degrees of Separation: Multi-Dimensional Separation of Concerns", *Proc. of ICSE* 1999, pp.107-119, 1999.

[RUP] www.rational.com

[SEVO04] The SEVO project: http://www.idi.ntnu.no/grupper/su/sevo.html

[Zelk98] Zelkowitz, M.V., Wallace, D.R., "Experimental Models for Validating Technology", *IEEE Computer*, Vol. 16, pp. 191-222.

[Wohl00] Wohlin, C., Runeseon, P., M. Höst, Ohlsson, M.C., Regnell, B., Wesslen, A., *Experimentation in Software Engineering*, Kluwer Academic Publications, 2000.

[Yin02] Yin, R.K., "Case Study Research, Design and Methods", *Sage Publications*, 2002.