Empirical Assessment of Cost Factors and Productivity during Software Evolution through the Analysis of Software Change Effort

Hans Christian Benestad

Thesis submitted for the degree of Ph.D.

Department of Informatics
Faculty of Mathematics and Natural Sciences
University of Oslo
June 2009

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
Nr. 871

ISSN 1501-7710

Cover: Inger Sandved Anfinsen.
Printed in Norway: AiT e-dit AS, Oslo, 2009.

Produced in co-operation with Unipub AS.
The thesis is produced by Unipub AS merely in connection with the
thesis defence. Kindly direct all inquiries regarding the thesis to the copyright
holder or the unit which grants the doctorate.

Unipub AS is owned by
The University Foundation for Student Life (SiO)

Abstract

Changes and improvements to software can generate large benefits to users and to society as a whole, but can also be costly. The aggregated effort to change software normally constitutes a significant part of total lifecycle development costs. This thesis investigates such costs through the analysis of *change effort*, i.e., the effort spent by developers to perform software changes. The first goal was to identify factors that significantly and consistently affect change effort. With a better understanding of these factors, development technologies and practices could be improved to more effectively manipulate those factors. Candidate factors pertained to the people involved in making changes, to the practices they used, to the software code they changed, and to the tasks they performed. The second goal was to devise a method for software development organizations to assess trends in productivity as their software evolves. Under certain assumptions, trends in productivity can be captured by analyzing trends in change effort. With better methods to assess productivity trends, software development organizations can identify needs for improvements, and evaluate improvement initiatives.

The thesis focuses on software development organized around the implementation of change requests from stakeholders of the software. A systematic literature review established a framework for measuring changes, and summarized existing evidence on factors affecting change effort. Propositions generated from the review were then investigated in a case study in two commercial software development organizations. To identify relationships between properties of changes and the expended change effort, data on changes from version control systems and change trackers was quantitatively analyzed. Semi-structured developer interviews about recently completed changes refined and complemented the quantitative analysis.

One contribution of this thesis is to advance designs of change-based studies. The systematic review of such studies enabled a case study design that separated confirmatory and explorative analysis. In the confirmatory part, propositions were generated from the summary of factors affecting change effort in earlier studies. Testing existing evidence in new studies is useful to accumulate and generalize knowledge between contexts. The exploratory analyses discovered additional relationships in the data sets, potentially useful as a basis for new propositions in subsequent studies. The results support earlier findings that factors captured from change management data can explain only some of the

variability in change effort. To find complementary evidence, it was therefore helpful to further investigate the changes that corresponded to large model residuals.

One central result is that dispersion of code changes over source components had a consistent effect on change effort, beyond that explained by simple size effects. The qualitative analysis suggested that the effort spent on comprehending dispersed code was an important underlying cost driver. Comprehension typically occurred along the execution paths of the changed user scenarios, rather than within architectural units such as files and classes. These findings strengthen and refine earlier results on the effects of dispersion from laboratory experiments. The evidence points to design practices and tools that recognize developers' need to comprehend functional crosscuts of the software.

A second central result confirmed that the number of updates to the change request was positively correlated with change effort. The effect on change effort was particularly strong when frequent updates reflected difficulties in clarifying impacts on other parts of the systems. The developers faced such difficulties in cases where they had insufficient knowledge about the affected business rules and the domain experts had insufficient knowledge about the software. To better envision impacts of changes, software organizations should appreciate and cultivate knowledge in this boundary between the software and the business domain.

Furthermore, this thesis shows how analysis of change effort can capture trends in productivity. The method consists of four indicators based on a common, conceptual definition of productivity trend, and a set of procedures to evaluate the validity of the indicators in a given assessment. Consistent with the subjective experiences of the developers in two software organizations, the assessment indicated significant change in productivity between two time periods. In a third organization, the productivity assessment enabled more insight into the effects of a new development practice. The proposed method may represent a step towards more practical and trustworthy measurement practices that accelerate the adoption of measurement-based improvement frameworks in the software development industry.

In conclusion, the analysis of individual software changes proved effective both to identify factors that affect development costs, and to assess trends in productivity during software evolution. The results contribute towards more cost-effective and better managed software evolution.

Acknowledgement

Many people have been important in the process that materialized in this thesis. First of all, I am indebted to my supervisors Bente Anda and Erik Arisholm for sharing their profound knowledge on scientific methods and software engineering research, and for ensuring that my work was on track and progressing satisfactorily. I wish to thank Dag Sjøberg for recommending me for a PhD-position at Simula Research Laboratory, and for the strategic advice he provided at critical points. Magne Jørgensen challenged my initial research proposal and thereby significantly influenced the eventual focus for the thesis. I greatly enjoyed and benefited from the daily discussions about research methods and life in general with Jo Hannay in Room 453 at Simula. Stein Grimstad showed continuous interest in my work and contributed excellent ideas for improving the study and the papers. Audris Mockus provided inspiring feedback on two early drafts of one of the included papers. I thank Kjetil Moløkken-Østvold and Nils Christian Haugen for inviting me to participate in their research at Vegvesenet. Oxford Editing and Matt J. Duffy are to be thanked for proofreading the thesis' summary, and two of the included papers.

I am grateful to Simula Research Laboratory and the Simula School of Research and Innovation for accepting me as a PhD-student, and for the excellent environment they offer for learning the research discipline and conducting research. I thank all the people in the SE-department for generating such an inspiring environment with a fine balance between a continuous pursuit for excellence, and gallows humour when required by the situation.

The research would not have been possible without support from the people at Esito, Unified Consulting and KnowIT Objectnet. I thank Lars Ivar Næss and Mette Wam for allocating time and space for this research, and the developers who participated in data collection with a positive and open attitude. The IT-departments at NSB, the Research Council of Norway, and Vegvesenet are to be thanked for allowing me to use their projects as study objects.

Last but not least, I am grateful to my family and friends who strongly supported my decision to pursue a PhD-degree, and who continuously challenged me to explain the contribution and consequences of my work. My closest family deserves a special thank, Hildegunn, August (9) and Ingvald (6), and my dear mother and father.

List of papers

The following papers are included in this thesis:

1. Understanding software maintenance and evolution by analyzing individual changes: A literature review

H. C. Benestad, B. C. D. Anda, and E. Arisholm

Revision submitted to the Journal of Software Maintenance and Evolution: Research and Practice, November, 2008.

2. Understanding the cost of change: A quantitative and qualitative investigation of change effort during evolution of two software projects

H. C. Benestad, B. C. D. Anda, and E. Arisholm

Revision submitted to the Journal of Empirical Software Engineering, March 2009

3. Are we more productive now? Analyzing change tasks to assess trends in productivity during software evolution

H. C. Benestad, B. C. D. Anda, and E. Arisholm

Presented at the Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE) 2009

4. Using planning poker for combining expert estimates in software projects

K. J. Moløkken-Østvold, N. C. Haugen, and H. C. Benestad.

Published in Journal of Systems and Software, December 2008.

My contributions

For papers 1, 2 and 3, I was responsible for the idea, the study design, data collection, analysis and writing. My supervisors contributed with both general advice and concrete suggestions for improvements in all phases of the work. For paper 4, I was responsible for the parts that concerned code analysis, including the writing. For the other parts in paper 4, I contributed with comments and improvements to the manuscript.

I was the first author and main contributor in all parts of the work for two other papers that are not included as part of this thesis.

A. Assessing Software Product Maintainability Based on Class-Level Structural Measures

H. C. Benestad, B. C. D. Anda, and E. Arisholm

In: 7th International Conference on Product-focused Software Process Improvement (PROFES), ed. by Jürgen Münch, pp. 94-111, Springer-Verlag. Lecture Notes in Computer Science (ISBN: 0302-9743), 2006.

This paper proposes methods to aggregate fine-grained measures of structural properties for an assessment of maintainability at the system level. Maintainability was compared between four software systems, developed with the same requirement specification. The results were compared with judgements made by expert developers. Assessment of maintainability is equivalent to making predictions about the ease with which changes can be made to software. The paper is not included in the thesis, because I chose to focus directly on investigating change effort, rather than on predicting change effort indirectly from structural properties of source code.

B. Assessing the reliability of developers' classification of change tasks: A field experiment

H.C. Benestad, Technical report 12-2008, Simula Research Laboratory

This paper reports on a field experiment in one of the organizations participating in the data collection for the thesis. The goal was to investigate whether changes could be reliably classified by the developers. Reliable classification of changes can be important for empirical research, but is also of interest to software project managers and sponsors. Two alternative classification schemes were compared, one based on the dimensions of maintenance proposed by Swanson [1], the second based on the ISO 9126 product quality model [2]. The results affected the plans for data collection for this thesis, and provided some upfront warnings against putting too much trust in change classifications made by developers. The topic for this report is directly relevant to the thesis, but the manuscript has not yet been sufficiently developed to be published, or included in the thesis.

Contents

Summary

1 Introduction

1.1 Motivation and objectives

The statement *Our civilization runs on software* [3] is no longer controversial. More and improved software is a key driver behind advances in public and private services, communication, transportation, production systems and entertainment. To remain competitive, organizations involved in software development must continuously make wise decisions about software qualities to improve. For example, end users require better functionality, usability and dependability, while IT operations require the software is adapted to the currently supported technological platform. Consequently, the quality goal *ease of change* is important for development organizations, and for any stakeholder concerned with lifecycle costs. The potential for cost savings is huge if software could be changed more easily. Conservatively estimated, 50 billion USD worth of development effort is expended annually to make changes to operational software[1].

The importance of software change was recognized in early software engineering research. Lehman [4] expressed the necessity of change in what he termed the *first law of software evolution*:

> *A program that is used and that as an implementation of its specification reflects some other reality, undergoes continual change or becomes progressively less useful. The change or decay process continues until it is judged more cost effective to replace the system with a recreated version.*

[1] Estimate based on 5 million software programmers globally, with an average annual cost of 20 000USD, and 50% of total programming effort in maintenance and evolution.

Software changes differ with respect to factors such as their purpose, priority, size and complexity. Swanson proposed a first-cut categorization of changes into corrective, adaptive and perfective changes [1]. These *dimensions of software maintenance* can also be considered causes of the continuous need for change: changes in the technological environment trigger *adaptive* changes, errors committed by the development organization trigger *corrective* changes, while new requirements from stakeholders trigger *perfective* changes.

One goal of this research was to identify factors that significantly and consistently affect *change effort*, i.e., the effort spent by developers to perform software changes. Candidate factors are associated with people, practices, product, and the performed changes. Some of these factors may be manipulated through process or product improvements, leading to more cost-effective software evolution. The second goal was to devise a method to measure trends in change effort in ongoing software projects. Under certain assumptions, such measurement can be seen as equivalent to measuring trends in productivity during software evolution. With practical and trustworthy indicators of productivity trends, organizations that maintain and evolve software can identify needs for improvements, and evaluate the effects of improvement initiatives.

Today, trends in the IT industry indicate a unification of initial development and evolution. First, the omnipresence of software implies that even new projects must somehow cope with a legacy code base [5]. Second, software is increasingly developed in the shape of services that fit into already operational, loosely coupled architectures [6]. Third, popular agile development processes recommend that new software is deployed early and incrementally [7]. These trends make the topic for this thesis relevant to the complete software lifecycle.

1.2 Contributions

A systematic literature review and case studies in three industrial software organizations were conducted. The contributions from these studies can be described from three perspectives. For the software engineering discipline the main contributions are:

- *Analysis of field data to refine empirical evidence on the effect of* ***dispersion***. Developers' effort to comprehend and modify source code that was dispersed over many components and component types was higher than what could be explained by simple size effects. The comprehension activity typically occurred along the paths of the changed user scenarios, rather than within architectural units. These results point to

tools and design practices that recognize the importance of comprehending functional crosscuts of software, particularly when different languages and technologies are involved.

- *Refined evidence on the effect of volatile change requirements***.** Consistent empirical evidence suggests that measures of *change requests volatility* are useful predictors in effort estimation models. The analysis discovered a particularly strong effect of volatile requirements on change effort when the development group had insufficient knowledge about affected business rules and – simultaneously - domain experts had insufficient knowledge about the relevant parts of the software. Rather than unreflectively embracing evolving requirements, the results suggest that software organizations should cultivate knowledge in the boundary between software and the business domain, aiming at more complete specifications early in the change process. Other kinds of modifications to requirements, such as refinements to GUI design, can have inherent advantages and do not necessarily have severe effects on development effort.
- *A method to assess trends in productivity during software evolution using quantitative data extracted from change management systems.* Application of the method in three different software organizations showed that it was feasible to detect major trends in productivity, even with a limited number of data points. The method includes procedures to evaluate the validity of the indicators in a given context. Analyses can be almost fully automated, an important criterion for such methods to be adopted by the software industry.

From the perspective of methodology in empirical software engineering, the thesis contributes with:

- *A framework for measuring and analyzing changes, developed as part of a systematic review [8].* The framework consists of a (i) conceptual model that clarifies the concepts involved in analysis of changes, (ii) a comprehensive set of candidate measures for such analysis, and (iii) a summary of results and contributions from change-based studies. A common conceptual basis and systematically collected evidence from existing change-based studies contribute to more effective accumulation of knowledge from such studies. This is important for the longer term goal of developing scientific theories in the field of software evolution.
- *A methodology to combine quantitative and qualitative analysis of changes, by systematic investigation of changes that correspond to large model residuals.* The

methodology improved the empirical investigation with respect to causal analysis and construct validity issues, and provided more complete and refined evidence on the phenomenon under study. Because these are general concerns in empirical software engineering studies, the methodology could be useful with other units of analysis and in other contexts.

In addition, the individual organizations that participated in this research benefited from:

- *Proposals for improvements to judgement-based estimation practices.* The analysis pointed to specific factors that were not sufficiently accounted for by current practices.
- *Proposals on possible process improvements to reduce the effort expended to perform software changes.* These proposals included tool acquisitions and refactoring targets within the software.

1.3 Thesis Structure

The thesis is structured as follows*:*

Summary. This part explains the research conducted for this thesis and introduces the included papers. Section 2 describes the thesis' goals, motivation and focus. Section 3 presents the research method, including an overview of the study procedures. Section 4 summarizes the results, while Section 5 discusses the implications for practice and research. Section 6 concludes.

Papers. The rest of the thesis consists of four papers submitted or accepted for publication in international journals and at one peer-reviewed conference. An overview of these papers is given on page ii, in the introductory part of thesis.

2 Research Goals

In this thesis, the main variable of interest is *change effort.* This variable reflects the quality goal *ease of change* - the ease with which developers can change software. Concepts such as maintainability, evolvability and changeability are also related to this quality goal, but are often assumed to reflect the internal properties of the software. For example, the Maintainability index is a one-valued indicator of maintainability calculated from measures of such internal properties [9]. In contrast, this thesis assesses the ease of change as reflected by the externally observable human effort spent on performing changes. This effort is affected by the internal properties of the software artifacts, but also by properties of the software organization and its people, of the supporting practices and technologies, and of the actual change tasks.

2.1 Goal 1: Identify Factors that Affect Change Effort

The first goal was to identify significant and consistent factors that affect change effort. The motivation is that the aggregated effort to perform software changes constitutes a substantial part of lifecycle development costs. A better understanding of such costs has interested researchers since the infancy of the software engineering discipline. Three decades ago, the initial laws of software evolution were proposed [4, 10], in essence stating (i) that change is inevitable and (ii) that evolving software tends to become increasingly complex and difficult to change. Better practices during initial development and evolution are assumed to counteract such difficulties. However, to devise effective practices, it is useful to understand the underlying factors and mechanisms that make change easier or more difficult.

2.1.1 A Holistic View on the Ease of Change

Existing research related to Goal 1 has investigated different classes of factors. One line of research focused solely on internal properties of the software. For example, the effect of structural properties of object-oriented code on quality properties of software received considerable attention among researchers over the last two decades [11]. Knowing how structural properties affect the ease of change is useful because it can lead to improved design or coding strategies. A method to assess the ease of change using fine-grained measures of structural properties was proposed by this author in a study that led to the eventual focus of this thesis [12].

The most optimistic promises of this research are moderated by the inevitable interactions between internal software properties and the properties of the software's environment [13]. Controlled experiments have shown that the advantages of certain design principles depend on the level of experience and skills of the developers exposed to the design in subsequent change tasks [14]. Furthermore, a result from a large multidisciplinary investigation on the symptoms, causes and remedies for *code decay* (another term closely related to *ease of change*) was that the history of code changes was more indicative of problems than measures of structural properties of the code [15]. Mockus and Weiss argued that properties of the changes themselves are the most fundamental and immediate concern in a software project [16], implying that assessments of ease of change can be imprecise without considering actual change tasks. In this thesis, a holistic view assumes that change effort can be affected by properties of the involved people, the practices and technologies they use, the software code they change, and the tasks they perform to reach the development goals.

2.1.2 Change-Based Studies – Key Concepts and Motivation

Studies on factors that affect software development and evolution have been performed at different levels of granularity. Wohlin and Andrews proposed a methodology to investigate factors that pertain to the full lifecycle of a project, such as project management principles, personnel turnover, geographic distribution, and tools [17]. Lehman's laws of software evolution were developed by studying data on *releases* during the lifecycle of one specific software system [10]. At a very fine level of granularity are a series of maintenance studies by Von Mayrhauser [18-21], and research summarized by Detienne [22]. The goal of this research was to understand the cognitive processes of people who perform software development activities, under the premise that *code comprehension* is an essential activity.

Figure 1 illustrates some important concepts in the thesis. *Software evolution* is seen as the aggregation of individual *changes*. A *change ta*sk is a cohesive and self-contained unit of work that is triggered by a *change request*. A change task involves detailed design, coding, unit testing and integration. *Change effort*, as measured in this research, is the total development time expended for these activities. A change task is manifested in a *change set*, which is the tangible set of changes to one or more of the source components. The term *change* conveniently combines these facets when their distinction is not important. A change (or the associated change request, change task or change set) can have attributes, henceforth referred to as *change attributes*. Examples of such change attributes are the size

and type of change. Empirical case studies using changes as the main unit of analysis are referred to as *change-based studies*.

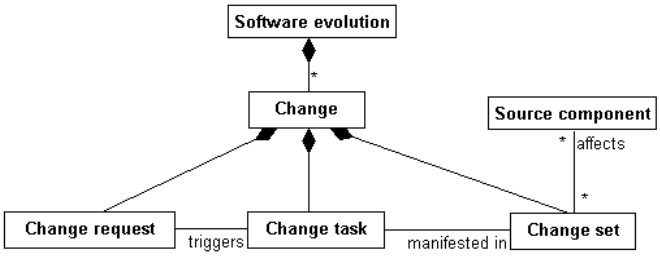


Figure 1. Software evolution as the composition of changes

The thesis focuses on factors that vary across change tasks, and that primarily capture coding-centric activities. Nevertheless, a broad set of factors was investigated, including factors associated with developer experience, collaboration in the development group, size and complexity of changes, and the structural properties of the affected software.

Change-based analysis is a practically feasible, analytically powerful and industrially relevant approach to the analysis of development activities during software evolution. Practical feasibility comes from the widespread use of change management systems, in the form of version control systems and change trackers. Data for a change-based analysis can be retrieved from the repositories of such tools, sometimes with little or no measurement overhead. Analytic power comes from the cohesiveness and fine granularity of changes. Although there will inevitably be interrelationships between changes, each change can be considered a small project going through the phases of analysis, design, coding, test and integration. Even in moderately sized software development projects, enough changes are normally completed to be able to perform powerful statistical analysis. Graves and Mockus argued that change-based analysis makes it possible to discover factors that would be hidden at more aggregated levels [23]. Industry relevance comes from the direct relationship between effort for individual change tasks and total costs of software evolution. If improvement in development practices and tools meant that changes were completed with less effort, then the total cost of software evolution would be reduced.

2.1.3 Related Research and Research Gaps

Several researchers have investigated relationships between change attributes and change effort. For example, Niessink [24, 25] and Jørgensen [26, 27] analyzed field data to evaluate the accuracy of models to predict change effort from change attributes. In the *code decay project*, the researchers used a small set of preselected change attributes to understand the factors that affected change effort, including the effect of evolution itself [15, 28].

Existing studies related to Goal 1 have to a limited degree been able to base the research designs on earlier results, for example by proposing hypotheses on the basis of existing empirical evidence. This situation has resulted in scattered evidence that is difficult to conceptualize and aggregate. To more effectively collect evidence from existing and future change-based studies, a common conceptual basis is needed. This premise motivated a systematic review (reported in Paper 1) that (i) builds an ontology for change-based studies, (ii) generalizes measures used in change-based studies into a set of conceptual change attributes, and (iii) summarizes contributions and specific evidence from individual change-based studies.

The systematic review could not satisfactorily clarify whether the identified factors are stable across study contexts, nor shed much light on important contextual factors. Moreover, because most of the studies were based on correlation analysis of field data, only tentative propositions on causality could be claimed.

Another open issue was whether it is feasible to construct context-specific change effort models that are sufficiently accurate to be used for effort estimation purposes. The accuracy of change effort models was also important for the proposed method to assess productivity trends, described in Section 2.3. The studies by Niessink and Jørgensen indicated that the prediction accuracy in quantitative models of change effort could be expected to be moderate or poor by normal standards, such as the standards proposed by Conte et al [29].

To investigate these open issues, a change-based field study with the following properties was planned:

- Evidence-driven analysis investigating the effect on change effort of a small set of factors, selected on the basis of existing empirical evidence.
- Data-driven analysis investigating the effect of a larger set of candidate factors, also assessing the potential accuracy for change effort models.

- Qualitative analysis based on series of developer interviews to complement, refine and explain the quantitative results.
- Investigation in two organizations to contrast and compare context factors.

2.2 Goal 2: Improve Methods to Assess Trends in Productivity during Software Evolution

The second goal of the thesis was to devise better methods to assess trends in productivity in ongoing software projects. Such methods would enable software development organizations to evaluate the need for improvement initiatives, and to evaluate the effects of such initiatives. For example, the three projects that contributed data for evaluating the proposed method had the following motivations:

- One project wanted to assess the effect on productivity of using two alternative practices for estimating and planning change tasks.
- A second project had planned to restructure parts of the source code and wanted to assess whether productivity had changed after this effort.
- A third project had been in a phase of intensive change to the software and wanted to make an informed decision on whether actions were needed to ease future change.

Productivity is generally defined as the proportion of output production to input effort [30]. Defining meaningful measures of output production is the essence of, and the core difficulty with, assessing productivity for software development processes. Proposed measures of output production in the context of initial development include lines of code, function points [31] and specification weight metrics [32]. Productivity indicators based on such measures target effort prediction for new projects, as well as post-hoc project evaluation.

Existing research has attempted to adapt basic measures of output production to software evolution, for example by quantifying the extent of changes to existing components [24, 33, 34]. However, no generally accepted approach to assessing productivity during evolution has yet been developed [35]. It is perhaps a more realistic ambition to define productivity indicators within clearly defined scopes. In this thesis, the scope for the productivity assessment is limited to cases of software evolution where the software organization considers it meaningful to compare change tasks and change effort between two contexts, such as two time intervals. Furthermore, rather than evaluating indicators for general validity, it is more realistic to define validation procedures to be used in conjunction with the productivity indicators.

The proposed method assumes that the completed change task is a fundamental unit of output production. A simple indicator of productivity trend simply compares the average change effort between groups of change tasks. A limitation of this indicator is that it does not account for possible systematic differences in the properties of the compared change tasks. Three of the proposed indicators address this issue by:

- Analysis of convariance (ANCOVA) to investigate the effect of time, or some other factor, on change effort, inspired by Graves and Mockus' approach to investigating *code decay* [23].
- Comparison of actual change effort with predictions obtained from change effort models, inspired by Kitchenham's and Mendes' method to assess productivity of completed projects [36].
- Repeated benchmarking of identical change tasks, inspired by the definition of *changeability decay* by Arisholm and Sjøberg [37].

The proposed productivity indicators are based on a common set of definitions of what constitutes *change in productivity* during software evolution. These definitions make it simpler to define, interpret, validate and compare new or modified indicators targeting the same scope.

3 Research Method

3.1 Overview of Methodology

The main research methods employed for this thesis are *case study research* and *systematic literature review*. A case study is an empirical investigation of a phenomenon in a real-life context, particularly suited when the phenomenon and the context are difficult to separate [38]. Because there are typically more variables than data points in a case study, propositions and generalizations rely on multiple sources of evidence, and the use of theory. Single cases or multiple cases can be investigated, using any mix of data sources and of qualitative and quantitative evidence. The ambition for Goal 1 was to investigate factors that affect change effort in the context of real software development. The case study method was chosen because we wanted to consider the full complexity of factors that could affect change effort in such a context.

Case studies are recommended to be designed to confirm, refute, or in some way modify *theories*. A theory makes it possible to understand fundamental and long-lived mechanisms in a domain. Results from a single case study become useful in other context through an improved theory. Therefore, theory is a mechanism for generalizing from a case study. Unfortunately, the theoretic foundation for many topics in software engineering is weak, including the topics investigated in this thesis.

Systematic reviews use a rigorous methodology to ensure a fair evaluation and interpretation of all research relevant to a phenomenon [8]. The design and propositions for the case study were generated on the basis of a systematic review of change-based studies. The results aimed to refine the existing knowledge on the phenomenon of ease of change. Hence, basing the case study on a systematic review addresses some of the design challenges caused by the lack of relevant scientific theories.

In summary, the thesis (i) attempts to identify causal links between change attributes and change effort and (ii) proposes and evaluates methods for measuring productivity trends during software evolution using a methodology that:

- Bases the propositions and generalization on a systematic review of similar studies.
- Investigates multiple cases (two and three for Goal 1 and 2, respectively).
- Uses software change repositories and developer interviews as sources of evidence.
- Analyzes data quantitatively and qualitatively.

3.2 Study Procedures

An overview of the study procedures is shown in Figure 2. The systematic review established a conceptual basis for change-based studies, and summarized evidence relevant to Goal 1. Data from change management systems (CMS) and data from monthly developer interviews in two commercial software development organizations were used for the analysis. The joint results contribute to the existing empirical evidence on factors that affect the change effort.

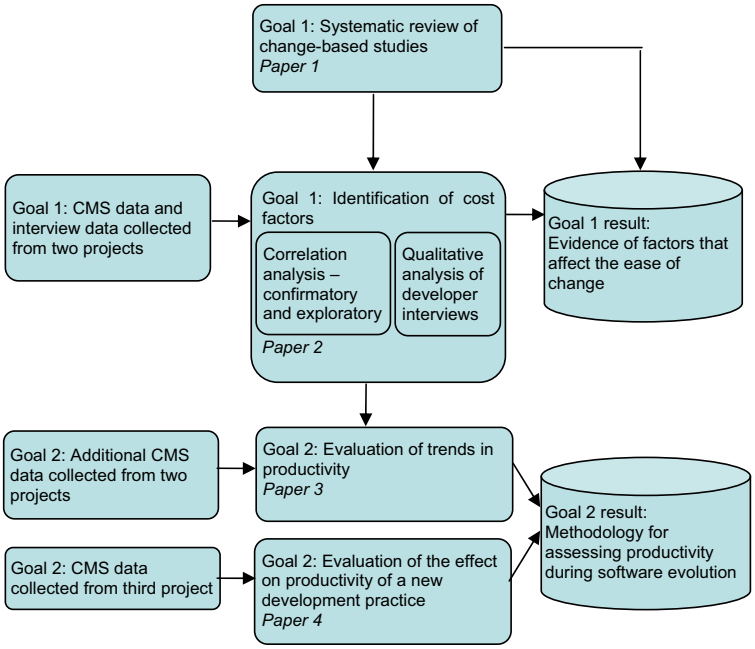


Figure 2. Study procedures

The proposed productivity measures for Goal 2 were utilized in ongoing projects in three commercial software development organizations. In two of the projects, the specific goal was to assess the trend in productivity between two time periods. These assessments reused the quantitative models constructed for Goal 1, but required equivalent data to be collected for the contrast period. The method developed for Goal 2 consists of unified definitions of the proposed productivity indicators, a set of evaluation procedures, and demonstrations of the feasibility of using the method in ongoing software projects.

3.3 Case Studies and Data Collection

The collaboration with two commercial software development organizations, MT and RCN, was initiated by the author. The organizations conformed to the following criteria for participation:

- Use of object-oriented development technology.
- Development process organized around change requests, and frequent releases.
- Planned development for at least 12 months ahead.
- Access to data in change management systems, and developers' agreement to participate in interviews.

Apart from these criteria, the companies were recruited by convenience. A local database of 83 organizations that had collaborated with our research group in the past was consulted to identify candidate organizations. Eight organizations were contacted, but only MT and RCN conformed to the above requirements. The organizations were motivated by prospects for improving their development practices on the basis of empirical evaluation. In addition, they considered it beneficial for their company profile to participate in research activities. It was crucial for the planned analysis that developers (i) recorded change effort expended for completing the change tasks, and that they (ii) tracked the relationships between source code changes and the associated change request. To strengthen the organizations' commitment to data collection, the collaboration agreement included compensation for the required data collection effort. In total, the companies were compensated for 59 work hours.

The analysis for Goal 2 used additional data about change tasks from a third organization, *FK*, which already collaborated with our research group.

The investigated organizations developed bespoke software on behalf of agencies in the public sector. Scrum principles [39] were followed for project management, and changes were for the most part completed under time-and-material contracts. Staffing was stable throughout the period, and most of the developers were classified as senior developers by their employer. The systems consisted of between 200 000 and 500 000 lines of code and had been in production for 2 to 5 years when data collection started. All organizations used Java-based technology. RCN used a stack of technologies that included modeling tools, code generators, a workflow engine, and a Java Enterprise Edition application server. MT used some C++ code for hardware-near functionality, while FK used a standard Java platform.

Data was collected from three sources. The quantitative analysis for both goals used data from change management systems, i.e. change trackers and version control systems. Developer interviews were conducted on a monthly basis, and focused on recently completed change tasks. Field experiments of half-day durations were organized on three occasions. The first of these experiments prepared for the main study by investigating the reliability of developers' classifications of changes [40]. In the two last sessions, the developers benchmarked a set of change tasks as part of data collection for Goal 2.

Table 1. Summary of data collection for the thesis

	RCN	MT	FK
Duration of data collection (months)	24	18	3
Number of changes analyzed quantitatively	273	228	34
Total effort for changes analyzed quantitatively (hours)	2590	1349	321
Number of changes discussed in interviews	120	65	n.a.
Field experiment sessions	3*3 hours	3*3 hours	n.a.
Total direct cost for data collection (USD)	4500	6000	0

4 Summary of Results

This section describes the key results, organized per goal and research paper. The two first papers addressed research questions related to Goal 1, while the third and fourth paper addressed research questions related to Goal 2.

4.1 Goal 1 – Identify factors that affect change effort

4.1.1 Paper 1

The research question for the systematic review summarized in Paper 1 was:

Which overall measurement goals have been set in change-based studies, and which attributes were measured to achieve these goals?

The research question was answered by establishing a framework for measurement and analysis in change-based studies. Thirty-four reported studies conformed to the inclusion criteria for the review and the framework was applied to summarize the results and contributions. Figure 3 shows a model of the information that was extracted and synthesized. The right part of Figure 3 describes the established framework. It consists of a hierarchy of study goals, a conceptual model with interrelated concepts important in change-based studies, and a related set of changes attributes. The framework was synthesized from the extracted data about studies and measures, described in the left part of Figure 3. Paper 1 describes in detail the methodology that was used to extract and synthesize this information.

As illustrated in Figure 3, an operationalization of a change attribute in a particular study corresponds to a *change measure*, typically used as response variables and explanatory variables in statistical analyses. Quantitative evidence from the studies was collected by extracting results from such statistical analyses.

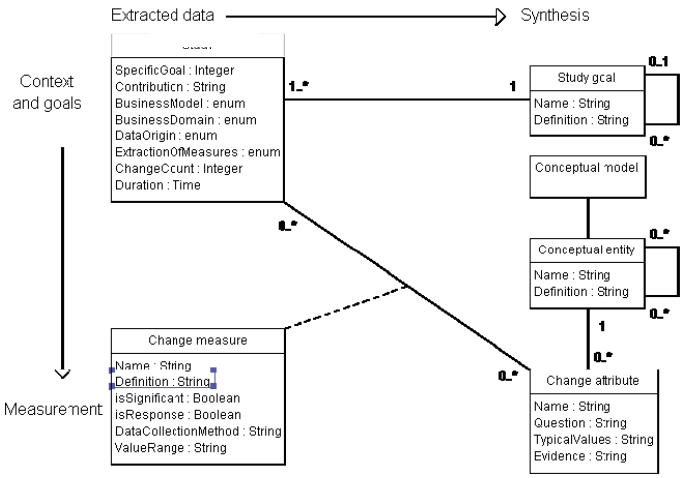
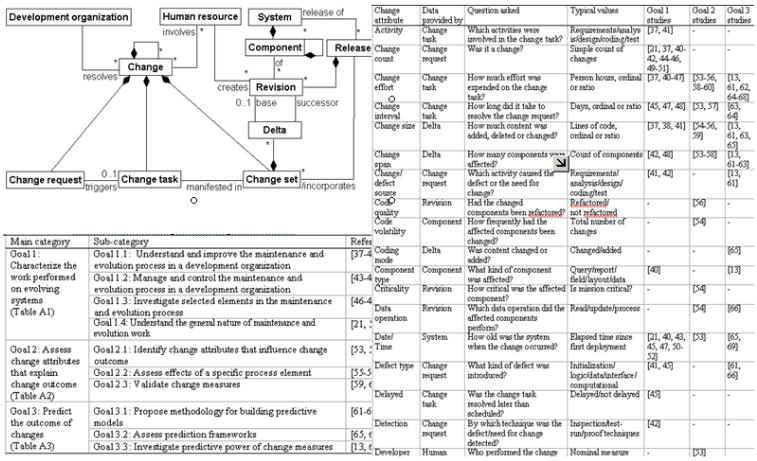


Figure 3. Model of information extracted and synthesized in the literature review



Change attribute	Data provided by	Question asked	Typical values	Goal 1 studies	Goal 2 studies	Goal 3 studies
Activity	Change task	Which activities were involved in the change task?	Requirements/analysis/design/coding/test	[37, 41]	-	-
Change count	Change request	Was it a change?	Simple count of changes	[21, 37, 40-42, 44-46, 49-51]	-	-
Change effort	Change task	How much effort was expended on the change task?	Person hours, ordinal or ratio	[37, 40-47]	[53-56, 58-60]	[13, 61, 62, 64-66]
Change interval	Change task	How long did it take to resolve the change request?	Days, ordinal or ratio	[45, 47, 48]	[53, 57]	[63, 64]
Change size	Delta	How much content was added, deleted or changed?	Lines of code, ordinal or ratio	[37, 38, 41]	[54-56, 59]	[13, 61, 63, 65]
Change span	Delta	How many components were affected?	Count of components	[42, 48]	[53-58]	[13, 61-63]
Change/defect source	Change request	Which activity caused the defect or the need for change?	Requirements/analysis/design/coding/test	[41, 42]	-	[13, 61]
Code quality	Revision	Had the changed components been refactored?	Refactored/not refactored	-	[56]	-
Code volatility	Component	How frequently had the affected components been changed?	Total number of changes	-	[54]	-
Coding mode	Delta	Was content changed or added?	Changed/added	-	-	[65]
Component type	Component	What kind of component was affected?	Query/report/field/layout/data	[48]	-	[13]
Criticality	Revision	How critical was the affected component?	Is mission critical?	-	[54]	-
Data operation	Revision	Which data operation did the affected components perform?	Read/update/process	-	[54]	[66]
Date/Time	System	How old was the system when the change occured?	Elapsed time since first deployment	[21, 40, 43, 45, 47, 50-52]	[53]	[65, 69]
Defect type	Change request	What kind of defect was introduced?	Initialization/logic/data/interface/computational	[41, 45]	-	[61, 66]
Delayed	Change task	Was the change task resolved later than scheduled?	Delayed/not delayed	[45]	-	-
Detection	Change request	By which technique was the defect/need for change detected?	Inspection/test-run/proof/techniques	[42]	-	-
Developer	Human	Who performed the change	Nominal measure	-	[53]	

Main category	Sub-category	Refer
Goal 1: Characterize the work performed on evolving systems (Table A1)	Goal 1.1: Understand and improve the maintenance and evolution process in a development organization	[37-4
	Goal 1.2: Manage and control the maintenance and evolution process in a development organization	[43-4
	Goal 1.3: Investigate selected elements in the maintenance and evolution process	[46-4
	Goal 1.4: Understand the general nature of maintenance and evolution work	[21,
Goal 2: Assess change attributes that explain change outcome (Table A2)	Goal 2.1: Identify change attributes that influence change outcome	[53,
	Goal 2.2: Assess effects of a specific process element	[55-5
	Goal 2.3: Validate change measures	[59,
Goal 3: Predict the outcome of changes (Table A3)	Goal 3.1: Propose methodology for building predictive models	[61-6
	Goal 3.2: Assess prediction frameworks	[65,
	Goal 3.3: Investigate predictive power of change measures	[13,

Figure 4. The resulting framework for change-based studies

Figure 4 shows an excerpt of the actual framework, as it appears in Paper 1. The review identified three broad categories of change-based studies, each with three or four sub-categories. The categorization schema is intended for use to search evidence on a particular topic, for example in the planning of new change-based studies, but it is not claimed to be definite or absolute.

In total, 41 change attributes were identified from the reviewed studies. Table 2 describes five change attributes with consistent effect on the ease of change (as in most cases captured by change effort) in the reviewed studies. These attributes were selected for

the evidence-driven analysis of factors conjectured to affect change effort (described in Paper 2).

Table 2. Summary of change attributes that have affected the ease of change

Change attribute	Question asked	Examples of measures
Maintenance type	What was the purpose of the change?	-*Fix/adapt/enhance* classified by developers or by heuristic search in change data
Change size and change dispersion	How much code was added, deleted or changed?	-Number of code lines in change set -Number of changed files in change set
Change request volatility	To what extent were change requests changed?	-Number of updates in change tracker -Number of words in change tracker
Change experience	How experienced was the developer in changing the system?	-Average number of previous commits by developers who committed in the change set, system wide or in components in the change set
Structural complexity of changed components	How many occurrences are there of a given programming construct?	-Size of affected files or classes -Measures of control-flow complexity, coupling, cohesion, inheritance of affected classes

4.1.2 Paper 2

The research question for Paper 2 was:

From the perspective of developers handling incoming change requests during software evolution, which factors affect the effort required to complete the change tasks?

The analysis consisted of (i) an evidence-driven part that investigated the effect of five pre-selected factors (see Table 2) in models of change effort, (ii) a data-driven part that used an extended set of 31 candidate variables to identify the variable subsets that optimized the cross-correlated model fit, and (iii) a qualitative part that used developer interviews to elicit complementary factors that affected change effort. To help in identifying the qualitative material that complemented the quantitative analysis, the analysis of developer interviews focused on changes that corresponded to large residuals in the quantitative models. Table 3a and Table 3b summarize the results from the quantitative and qualitative analysis, respectively. In the first column of these tables, factors from the evidence-driven, data-driven and qualitative analysis are prefixed with *ed*, *dd* and *qu*, respectively.

Table 3a. Summary of quantitative results from Paper 2

Factor	Proposition	Measured by	Result	Ref. to further results
ed_1	Change size and dispersion drive effort	Number of components in the change set	Supported	Explained by qu_2 Consistent with dd_1
ed_2	Change request volatility drives effort	Number of updates in change tracker prior to coding phase	Supported	Explained by qu_1
ed_3	Change experience reduces effort	Average number of previous commits by developers who committed changes	Weak supported in MT	Refined by qu_3
ed_4	Maintenance type affects effort	Corrective vs. non-corrective changes, using developers classification and text search	Supported in RCN	Refined by qu_4
ed_5	Structural complexity of changed components drives effort	Average number of lines of code in components in the change set	Not supported	Refined by qu_5
dd_1	Language heterogeneity of change drives effort	Number of unique file types in the change set	Supported in RCN	Explained by qu_2
dd_2	Structural complexity of the change drives effort	Number of control-flow statements in the change set	Supported in MT	-

Table 3b Summary of qualitative results from Paper 2

Factor	Relationships with change effort
qu_1	Clarifications of functional side effects generate effort throughout the change cycle
qu_2	Comprehending dispersed code is more difficult than comprehending localized code. Comprehension occurred mainly along execution paths of changed user scenarios.
qu_3	Experience had strong effect on effort in the few cases of unfamiliarity with relevant code
qu_4	Fixing *errors by omission* (caused by incomplete requirements) required more effort than did errors by commission and enhancive changes
qu_5	Comprehension occurred along user scenarios and execution paths, rather than within architectural units
qu_6	Frameworks or technologies sometimes had poor support for making the change, resulting in extra effort for workarounds. Poor debug support sometimes drove effort.
qu_7	Development and modification of reusable mechanisms were examples of *deep changes*, which required extensive effort

The number of components modified during the change task consistently contributed to change effort in the quantitative models (*ed_1*). Dispersion of change over several technologies added to the effect of dispersion over components (*dd_1*). Comprehension effort for dispersed code was an important contributor to change effort (*qu_2*). Measures of change dispersion were better predictors of change effort than were more fine-grained, LOC-based measures of change size.

In sum, multiple sources of evidence pointed to *dispersion* as a factor that causally affected change effort, beyond simple size effects. This interpretation is supported by existing evidence on the effect of discontinuities and delocalized plans from studies on text and program comprehension [41], and also by controlled experiments on the effect of centralized versus decentralized design styles [14].

The results indicate that effort involved in *comprehending dispersed code* is more important than the effort involved in carrying out modifications to some of that code, although the two activities were highly intertwined. The developers typically needed to comprehend code along the execution paths of affected user scenarios (*qu_2*). These paths were typically dispersed over many components. Consistent with this importance of comprehending functional cross-cuts of the source code, the measures of structural properties of individual architectural units did not contribute to change effort in the quantitative models (*ed_5*).

The number of updates to the initial change request prior to the coding phase consistently contributed to change effort in the quantitative models (*ed_2*). This result is perhaps surprising because it is reasonable to expect that more effort spent clarifying change requests, would mean fewer problems in the coding phase. A possible explanation is that a well considered and described initial change request eases subsequent phases for the change. Reversely, many modifications to the change request can indicate difficulty arriving at a sufficiently complete specification. The qualitative analysis showed a particularly large effect on change effort when the impacts on other user scenarios than the scenario originally addressed by the change request needed to be clarified throughout the change cycle (*qu_1*).

Paper 2 discusses in more detail additional results from the data-driven and the qualitative analysis. Those results are explorative in nature, and new propositions based on these results are subject to confirmatory analysis in further empirical studies.

4.2 Goal 2 - Improved Methods to Assess Trends in Productivity during Software Evolution

4.2.1 Paper 3

Paper 3 proposes methods to assess productivity trends during software evolution. The methods were demonstrated and evaluated in ongoing projects in two commercial software development organizations. The question asked for the empirical investigation was:

> *Did the productivity in the two projects change between the baseline period P0 (Jan-July 2007) and the subsequent period P1 (Jan-July 2008)?*

To answer this question, we proposed a conceptual definition of *productivity change*, which assumes that change effort for the same changes are compared between two time periods. Table 4 describes the four productivity indicators that operationalize this definition in alternative ways, the evaluation that accompanies the indicators, advantages and disadvantages of each indicator, and the measured productivity trend in the two projects.

Table 4. Summary of proposed productivity indicators

Indicator	Evaluated by (refer to Paper 3 for details)	Advantages (+) and disadvantages (-)	Result RCN	Result MT
Compare estimates for benchmark tasks	Statistical significance	+ Close approximation to the theoretical definition. - Estimates can be unreliable - Practical challenges, including measurement overhead	No change	Lower
Compare average change effort	Statistical significance Box plots Compare properties of changes	- Data collection is easy - Assumptions are easily violated - Validation is difficult	Higher	Lower
ANCOVA-model that adjusts for differences in change attributes	Statistical significance of effect-variable Inspection of residual plot	+ Validation is given from well-known statistical framework. - Models must be rebuilt when new data arrives	Higher	Lower
Compare actual with predicted effort	Statistical significance Stability of model structure	- Assumptions difficult to evaluate + Usable with any prediction framework + Easy to use once model is built	Higher	Lower

The first indicator in Table 4 most closely matches the proposed definition of productivity change. For practical reasons, effort estimates were used in place of actual change effort. A particular practical challenge with this indicator is to define benchmarking tasks that are representative of actual changes. The second indicator simply compares average change effort between two time periods, assuming that there are no systematic differences in the changes between the periods. The last two indicators use statistical models to control for differences in the properties of changes between the compared time periods. The models are required to have reasonable model fit, and to be stable across the time periods. The evaluation in the projects showed that these requirements are feasible.

For RCN, a gain in productivity was indicated, consistent with the project's intended and experienced effect of a major code restructuring initiative. For MT, a drop in productivity was indicated, consistent with a post-hoc explanation by the project manager that developers might have experienced less time-pressure in the second period, caused by reduced use of fixed-price maintenance contracts.

In summary, the empirical study showed that it was feasible to use the indicators even with a moderate number of change tasks, and with a moderate model fit for the statistical models required for two of the indicators.

4.2.2 Paper 4

The initial goal for the research in Paper 4 was to compare two effort estimation methods (collaborate estimation using planning poker, versus aggregation of individual estimates) with respect to estimation accuracy. Real change requests in the investigated project were randomly assigned to an estimation method, and the actual effort and estimation accuracy were analyzed. The results showed no differences in estimation accuracy but, perhaps surprisingly, large differences in average change effort.

We wanted to understand whether this apparent difference in productivity could be explained by variable size or complexity of changes in the two groups. The results showed that after controlling for the increased complexity of code changes with collaborative estimation, the estimation method did not affect change effort. Feedback from the project members suggested that design discussions during collaborative estimation helped in discovering ripple effects of changes. These discoveries, in turn, affected the extent of code changes, and eventually the change effort.

In effect, this study employed the second and third method from Table 4 to assess productivity, respectively with and without controlling for differences between changes.

The assessment illustrated the added value of using more than one of the proposed indicators, as each indicator contributes with specific insight. Also, the assessment showed that it is feasible to use the indicators even with only a few dozens change tasks.

5 Implications and Future Work

5.1 Implications for Practice

The implications for practice were (i) consolidated evidence on factors that affect change effort, and (ii) methods to assess trends in productivity during software evolution. Immediately, an implication of the evidence on *change dispersion* is that software should be designed so that code that needs to be comprehended and changed as part of future change tasks is localized rather than dispersed. There are two difficulties with this principle. First, design decisions must consider other, potentially higher prioritized quality goals than the ease of change. For example, requirements regarding distribution of runtime components often imply that source code must also be dispersed. There are often good reasons for separating user interface components from business logic components. Second, it is difficult to predict what will constitute future changes, and therefore difficult to identify clusters of code that need to be comprehended and changed together. Nevertheless, when evaluated from the perspective of change effort, the evidence suggests benefits with a higher degree of localization of code that is functionally cohesive.

Another tactic would be to improve development tools to reduce the effect of change dispersion. One possibility is that integrated development environments (IDE's) automatically construct change-friendly views of the code, on the basis of configuration management information or dynamic runtime-analysis. For example, it is conceptually simple to let a class browser or editor present the classes involved in the execution of a given user scenario. Code relevant to a given user scenario inside the classes could then be filtered or highlighted. A practical difficulty is that such tools are more difficult to create where they would be most useful - for physically distributed systems that use multiple implementation technologies. Nevertheless, with increasing runtime complexity of software systems, it is important that technologies are developed to reduce complexity as perceived by the maintainer.

The results on change request volatility suggest that more complete specifications of change impacts prior to the coding phase would have saved effort. A well-known practice that addresses this concern is change impact analysis [42]. To be effective, such analysis should address impacts on function and other external quality properties, instead of focusing only on the internal dependencies within source code. The interview analysis indicated that an underlying factor causing problems throughout the change process was

insufficient knowledge in the boundary between the software and the business domain. Specifically, problems arose when developers knew too little about the business domain and domain experts knew too little about the software. To produce more complete specifications early in the change process and hence avoid these problems, software development organizations should recognize the importance of domain knowledge, and use practices and principles that cultivate such knowledge. An example of such practices from agile development methods is *on-site customer* [43].

Frameworks have been developed to help the software industry base decisions on practices and technologies on objective criteria and empirical evidence [44, 45]. Difficulties in defining practical and trustworthy outcome measures, such as measures of productivity, have hampered the adoption of such methods. This research demonstrated that it is feasible to use data from change management systems to assess productivity during software evolution. Conceptually, it is straightforward to automate the calculation and validation of the indicators. With productivity assessment capabilities built into change management systems, the potentials are larger for quantitative evaluation of software projects.

5.2 Implications for Research

The contributions of this thesis with respect to research methodology were (i) a conceptual framework for change-based studies and (ii) a methodology that combine quantitative and qualitative analysis of changes. The conceptual framework for change-based studies can be used when designing new change-based studies. A common conceptual basis is important to aggregate evidence from such studies, ultimately enabling the development of theories of software evolution.

The thesis supports existing evidence suggesting that quantitative models can explain only some of the variability in change effort. Qualitative methods proved effective to complement and refine such models. In particular, the qualitative analysis provided more insight into causal relationships, into construct validity issues associated with the employed quantitative measures, and into additional factors not captured from change management data. Also, the quantitative and objective information on residual size reduced the subjectivity in interpreting the qualitative data. Because these are very general concerns in nearly all kinds of software engineering research, the experiences suggest that similar qualitative design elements would be effective in other situations. The strategy of

systematically focusing on model residuals is not limited to change effort models, but could be applied to any unit of analysis and any model response.

5.3 Future Work

5.3.1 Change-Based Measurement as a Basis for Lifecycle Optimization of Designs

Future work involves collaborating with one or more organizations involved in large-scale software evolution, to devise a plan-do-check-act methodology for continuous management and reduction of lifecycle costs. Such a methodology will identify opportunities for design improvements, establish the business case for such opportunities, evaluate the effects of improvement efforts, and feed knowledge back to the identification of new opportunities. A key topic of investigation is whether change-based analysis can constitute the cornerstone of such enterprise and lifecycle scoped assessment and improvement of software evolution. An outline for the methodology is:

- Identify typical change tasks performed in the organization. Base improvement proposals on experienced problems associated with those changes.
- Develop business cases, for example by comparing past development effort to estimates of development effort had the improvements already been completed.
- Implement the improvements, and evaluate changes in productivity using data from real changes.

Step 1 involves qualitative procedures similar to those described in Paper 2. Step 2 is similar to performing benchmarking by estimates, described in Paper 3. In step 3, the productivity indicators described in Paper 3 and Paper 4 can be used.

5.3.2 Semi-Controlled Studies on the Effect of Structural Properties on Change Effort

For Goal 1, the influence of structural properties of the software, as well as properties of people, practices and performed tasks were investigated. No significant effect of structural properties was identified in the analysis. It is counterintuitive that the internal properties of software have no effect on change effort. One explanation is that measures retrieved from architectural units, such as files and classes, did not capture a goal-driven comprehension process, which typically involved source code along execution paths of user scenarios. Alternatively, the investigated systems may have had a homogenous, overall good design which implied that the effect on change effort did not vary across change tasks. It is also possible that the measures of structural properties were too rudimentary and did not capture the aspects of complexity causing difficulties to the developers.

In a study that led to the eventual focus for this thesis, a methodology for assessing the maintainability of object oriented systems was proposed and demonstrated [12]. The study objects were four java-based systems (see, e.g., http://www.simula.no/des1) that conformed to the same requirements. Our research group currently conducts a follow-up study that assesses the outcome (including change effort) for real changes performed on these systems. With elaborate quantitative and qualitative data to represent candidate factors and outcomes, the proposed methodology for assessing the ease of change from structural attributes can be evaluated and improved.

5.3.3 Describing Strategies to Perform Change Tasks

The relatively poor model fit of change effort models justify more focus on qualitative investigations to identify factors that influence change effort. The coding scheme described in Paper 2 is a starting point for further qualitative analyses studying how developers spend effort to perform change tasks. One particular area would be to develop better taxonomies of problem-solving strategies used by developers. An important part of this work is to better understand how knowledge and skills affect such strategies and the resulting performance. Such taxonomies would be useful as a basis for investigating and ultimately recommending the most effective strategy for a given change task. Moreover, tools could explicitly support different strategies to performing change tasks. Finally, such taxonomies could improve model-based or judgment-based effort estimation, under the premise that the choice of strategy affects development effort, and that it is possible to decide on a strategy early in the change cycle. Existing research and models of program comprehension form a foundation for such taxonomies, but there is a need to develop them to consider real development situations that include unclear requirements, complex technological environments, and collaboration among developers.

6 Conclusion

This thesis focused on the change task as the basic constituent of software evolution. A systematic review and a series of empirical studies were conducted to (i) identify the factors that affect change effort and (ii) to propose and evaluate methods to assess productivity trends during software evolution. A novel approach to combining quantitative and qualitative analysis of change tasks was proposed and evaluated.

The results showed that change-based analysis was effective in eliciting factors that influenced change effort. Using data from change trackers and version control systems, it was possible to investigate the effect of a wide range of possible effort drivers, including those associated with developer experience, design properties of source code, the requirements process and the development technologies.

Because many individual and interpersonal processes do not leave traces in change management systems, only a moderate amount of the variability in change effort can normally be explained from change management data. Hence, additional sources of evidence are necessary for a more complete picture. In particular, the study demonstrated a method to make qualitative analysis more effective by focusing on changes corresponding to large residuals in the quantitative models.

One central result showed that the change effort is consistently affected by the dispersion of the changed and comprehended code, beyond simple size effects. To counteract the effect of dispersion, development tools could offer change-friendlier views of code that is somehow relevant to a change, for example by visualizing interactions involved in changed user scenarios. Such tools are likely to be particularly useful when the runtime components are physically distributed and uses several implementation technologies.

Volatile change requirements were found to affect change effort, although the importance of this depended on the causes for volatility. Although there are inherent advantages to accepting flexibility in the requirements process, development organizations should not uncritically embrace volatile requirements. The results indicate that effort would be saved with better developed knowledge in the boundary between the software and the business domain.

The thesis showed promising results for using change-based analysis to assess productivity trends during software evolution. The proposed method proved sufficiently

sensitive to detect trends in productivity in three software projects, even with a moderate number of data points. The procedures gave the software organizations useful insights into their own development processes, and they are conceptually simple to generalize and automate. The method could therefore accelerate the adoption of quantitative evaluation techniques in the software industry.

Future research will evaluate whether change-based analysis can constitute the cornerstone for lifecycle-long improvement process in the context of large scale software evolution. Further fundamental research on how structural properties of software affect change effort is needed, as is research on strategies used by developers to solve real world change tasks.

References for the Summary

[1] E. B. Swanson, "The Dimensions of Maintenance," in *2nd International Conference on Software Engineering*, San Francisco, California, United States, 1976, pp. 492-497.

[2] ISO/IEC, "Software Engineering — Product Quality — Part 1: Quality Model," 2001.

[3] B. Stroustrup, "The quote "Our Civilization Runs on Software" is widely contributed to Bjarne Stroustrup, the inventor of the C++ Programming Language."

[4] M. M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution," *Proceedings of the IEEE,* vol. 68, pp. 1060-1076, 1980.

[5] A. De Lucia, A. R. Fasolino, and E. Pompella, "A Decisional Framework for Legacy System Management," pp. 642-651.

[6] M. P. Papazoglou and D. Georgakopoulos, "Service-Oriented Computing," *Communications of the ACM,* vol. 46, pp. 25-28, 2003.

[7] K. Beck, "Embracing Change with Extreme Programming," *Computer,* vol. 32, pp. 70-77, 1999.

[8] B. A. Kitchenham, "Procedures for Performing Systematic Reviews," Keele University Technical report EBSE-2007-01, 2007.

[9] P. Oman and J. Hagemeister, "Construction and Testing of Polynomials Predicting Software Maintainability," *Journal of Systems and Software,* vol. 24, pp. 251-266, 1994.

[10] L. A. Belady and M. M. Lehman, "A Model of Large Program Development," *IBM Systems Journal,* vol. 15, pp. 225-252, 1976.

[11] L. Briand and J. Wuest, "Empirical Studies of Quality Models in Object-Oriented Systems," *Advances in Computers,* vol. 59, pp. 97-166, 2002.

[12] H. C. Benestad, B. Anda, and E. Arisholm, "Assessing Software Product Maintainability Based on Class-Level Structural Measures," in *Proceedings of the 7th International Conference on Product-focused Software Process Improvement (PROFES), edited by Jürgen Münch. Springer-Verlag*, 2006, pp. 94-111.

[13] D. I. K. Sjøberg, T. Dybå, and M. Jørgensen, "The Future of Empirical Methods in Software Engineering Research," 2007, pp. 358-378.

[14] E. Arisholm and D. I. K. Sjøberg, "Evaluating the Effect of a Delegated Versus Centralized Control Style on the Maintainability of Object-Oriented Software," *IEEE Transactions on Software Engineering,* vol. 30, pp. 521-534, 2004.

[15] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does Code Decay? Assessing the Evidence from Change Management Data," *IEEE Transactions on Software Engineering,* vol. 27, pp. 1-12, 2001.

[16] A. Mockus and D. M. Weiss, "Predicting Risk of Software Changes," *Bell Labs Technical Journal,* vol. 5, pp. 169-180, 2000.

[17] C. Wohlin and A. A. Andrews, "Assessing Project Success Using Subjective Evaluation Factors," *Software Quality Journal,* vol. 9, pp. 43-70, 2001.

[18] A. von Mayrhauser, A. M. Vans, and A. E. Howe, "Program Understanding Behaviour During Enhancement of Large-Scale Software," *Journal of Software Maintenance Research and Practice,* vol. 9, pp. 299-327, 1997.

[19] A. von Mayrhauser and A. M. Vans, "Program Understanding Behavior During Adaptation of Large Scalesoftware," in *6th International Workshop on Program Comprehension* 1998, pp. 164-172.

[20] A. von Mayrhauser and A. M. Vans, "Program Understanding Behavior During Debugging of Large Scale Software," in *Seventh workshop on empirical studies of programmers*, 1997, pp. 157-179.

[21] A. von Mayrhauser and A. M. Vans, "Program Understanding Needs During Corrective Maintenance of Large Scale Software," in *Computer Software and Applications Conference*, 1997, pp. 630-637.

[22] F. Détienne and F. Bott, *Software Design - Cognitive Aspects*. London: Springer-Verlag, 2002.

[23] T. L. Graves and A. Mockus, "Inferring Change Effort from Configuration Management Databases," in *5th International Symposium on Software Metrics*, 1998, pp. 267–273.

[24] F. Niessink and H. van Vliet, "Predicting Maintenance Effort with Function Points," in *1997 International Conference on Software Maintenance*, 1997, pp. 32-39.

[25] F. Niessink and H. van Vliet, "Two Case Studies in Measuring Software Maintenance Effort," in *14th International Conference on Software Maintenance*, 1998, pp. 76–85.

[26] M. Jørgensen, "Experience with the Accuracy of Software Maintenance Task Effort Prediction Models," *IEEE Transactions on Software Engineering,* vol. 21, pp. 674-681, 1995.

[27] M. Jørgensen, "An Empirical Study of Software Maintenance Tasks," *Journal of Software Maintenance: Research and Practice,* vol. 7, pp. 27-48, 1995.

[28] T. Graves and A. Mockus, "Identifying Productivity Drivers by Modeling Work Units Using Partial Data," *Technometrics,* vol. 43, pp. 168-179, 2001.

[29] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*: Benjamin-Cummings Publishing Co., Inc. Redwood City, CA, USA, 1986.

[30] G. L. Tonkay, "Productivity," in *Encyclopedia of Science & Technology*: McGraw-Hill, 2008.

[31] A. J. Albrecht and J. E. Gaffney Jr, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering,* vol. 9, pp. 639-648, 1983.

[32] T. DeMarco, "An Algorithm for Sizing Software Products," *ACM SIGMETRICS Performance Evaluation Review,* vol. 12, pp. 13-22, 1984.

[33] A. Abran and M. Maya, "A Sizing Measure for Adaptive Maintenance Work Products," in *International Conference on Software Maintenance*, Nice, France, 1995, pp. 286-294.

[34] J. F. Ramil and M. M. Lehman, "Defining and Applying Metrics in the Context of Continuing Software Evolution," in *Software Metrics Symposium*, London, 2001, pp. 199-209.

[35] N. E. Fenton and S. L. Pfleeger, "Measuring Productivity," in *Software Metrics, a Rigorous & Practical Approach*, 1997, pp. 412-425.

[36] B. Kitchenham and E. Mendes, "Software Productivity Measurement Using Multiple Size Measures," *IEEE Transactions on Software Engineering,* vol. 30, pp. 1023-1035, 2004.

[37] E. Arisholm and D. I. K. Sjøberg, "Towards a Framework for Empirical Assessment of Changeability Decay," *Journal of Systems and Software,* vol. 53, pp. 3-14, 2000.

[38] R. K. Yin, *Case Study Research: Design and Methods*: Sage Publications Inc, 2003.

[39] K. Schwaber, *Agile Project Management with Scrum*: Microsoft Press, 2004.

[40] H. C. Benestad, "Technical Report 12-2008: Assessing the Reliability of Developers' Classification of Change Tasks: A Field Experiment," Simula Research Laboratory2008.

[41] F. Détienne and F. Bott, "Discontinuities and Delocalized Plans," in *Software Design - Cognitive Aspects* London: Springer-Verlag, 2002, pp. 113-114.

[42] S. A. Bohner and R. S. Arnold, *Software Change Impact Analysis*: IEEE Computer Society Press, 1996.

[43] K. Beck, "On-Site Custimer," in *Extreme Programming Explained*: Addison-Wesley Reading, MA, 2000, pp. 60-61.

[44] V. R. Basili and H. D. Rombach, "The Tame Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering,* vol. 14, pp. 758-773, 1988.

[45] L. C. Briand, C. M. Differding, and H. D. Rombach, "Practical Guidelines for Measurement-Based Process Improvement," *Software Process Improvement and Practice,* vol. 2, pp. 253-280, 1996.

Paper 1:

Understanding Software Maintenance and Evolution by Analyzing Individual Changes: A Literature Review

Hans Christian Benestad, Bente Anda, Erik Arisholm

Submitted to the Journal of Software Maintenance and Evolution: Research and Practice

Abstract

Understanding, managing and reducing costs and risks inherent in change are key challenges of software maintenance and evolution, addressed in empirical studies with many different research approaches. *Change-based studies* analyze data that describes the individual changes made to software systems. This approach can be effective in order to discover cost and risk factors that are hidden at more aggregated levels. However, it is not trivial to derive appropriate measures of individual changes for specific measurement goals. The purpose of this review is to improve change-based studies by 1) summarizing how attributes of changes have been measured to reach specific study goals, and 2) describing current achievements and challenges, leading to a guide for future change-based studies. Thirty-four papers conformed to the inclusion criteria. Forty-three attributes of changes were identified, and classified according to a conceptual model developed for the purpose of this classification. The goal of each study was to either characterize the evolution process, to assess causal factors of cost and risk, or to predict costs and risks. Effective accumulation of knowledge across change-based studies requires precise definitions of attributes and measures of change. We recommend that new change-based studies base such definitions on the proposed conceptual model.

1 Introduction

Software systems that are used actively need to be changed continuously [1, 2]. Understanding, managing and reducing costs and risks of software maintenance and

evolution are important goals for both research and practice in software engineering. However, it is challenging to collect and analyze data in a manner that exposes the intrinsic features of software maintenance and evolution, and a number of different approaches have been taken in empirical investigations. A key differentiator between classes of software maintenance and evolution studies is the selection of entities and attributes to measure and analyze:

- Lehman's laws of software evolution were developed on the basis of measuring new and affected components in subsequent releases of a software system, c.f., [2, 3].
- Investigations into cost drivers during software maintenance and evolution have investigated the effects of project properties such as maintainer skills, team size, development practices, execution environment and documentation, c.f., [4-7].
- Measures of structural attributes of the system source code have been used to assess and compare the ease with which systems can be maintained and evolved, c.f., [8-10].

An alternative perspective is to view software maintenance and evolution as the aggregate of the individual changes that are made to a software system throughout its lifecycle. An individual change involves a change request, a change task and a set of modifications to the components of the system. With this perspective, software maintenance and evolution can be assessed from attributes that pertain to the individual changes. Such attributes are henceforth referred to as *change attributes*, the measures that operationalize the change attributes are referred to as *change measures*, and the studies that base the analysis on change attributes and change measures are referred to as *change-based studies.* Two examples of topics that can be addressed in a change-based study are:

- *Identify and understand factors that affect change effort during maintenance and evolution.* This knowledge would contribute to the understanding of software maintenance and evolution in general, because the total effort expended by developers to perform changes normally constitutes a substantial part of the total lifecycle cost. For a particular project, it is essential to know the factors that drive costs in order to make effective improvements to the process or product. For example, if system components that are particularly costly to change are identified, better decisions can be made regarding refactoring.
- *Measure performance trends during maintenance and evolutions.* Projects should be able to monitor and understand performance trends in order to plan evolution and take corrective actions if negative trends are observed.

A central challenge is to identify change attributes and change measures that are effective in order to perform such analyses. For example, in order to assess and compare changes with respect to the man-hours that were needed to perform them, it is necessary to characterize the changes in some way, e.g., by measuring their *size* and *complexity*. This paper addresses this challenge by performing a comprehensive literature review of change-based studies. Conducting a comprehensive literature review is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest [11]. The research question for the review is:

> *Which overall measurement goals have been set in change-based studies, and which attributes were measured to achieve these goals?*

The review summarizes change attributes that have been used in empirical investigations, and we propose a conceptual model for change-based studies that enables us to classify the attributes. We will argue that new change-based studies can benefit from using this model as a basis for definitions of change attributes and change measures. A classification scheme for study goals is developed, enabling new studies to identify the current state-of-research for a particular goal. To further guide new studies, we exemplify current achievements and challenges within each of the main study goals.

To sum up, the objective of this literature review is to facilitate more effective investigations into the costs and risks of software maintenance and evolution, whether they are conducted by empirical researchers or by practitioners who are implementing a measurement-based improvement program. The approach is to systematically summarize and appraise the state of the practice in change-based studies.

The remainder of this paper is organized as follows: Section 2 provides a summary of related work. Section 3 describes the review procedures, including the criteria for inclusion and exclusion of primary papers for the review. Section 4 describes the conceptual model for change-based studies. Sections 5 answers the research question, while Section 6 provides a guide for further studies. Section 7 discusses limitations to the review. Section 8 concludes.

2 Related Work

We are not aware of other attempts to provide a comprehensive review of change-based studies of software maintenance and evolution. Graves and Mockus summarized three of

their own studies that showed that *time of change*, *tool usage,* and *subsystem affected by change* affected change effort [12]. They also recommended that statistical models of change effort should control for *developer effects*, *change size* and *maintenance type*. Niessink listed six change attributes that affect change effort that have been identified in empirical work by other authors [13]. Of these, *maintenance type* and *change size* matched the change attributes identified by Graves and Mockus.

Kagdi *et al.* conducted a literature review of studies that have mined data from software repositories for the purpose of investigating changes to software components [14]. Their review focused on a particular approach to data collection (mining of software repositories) while this review focuses on a particular approach to analysis (change-based studies). The two reviews are complementary, because mining of software repositories is an appealing, though not always sufficient, approach to collecting the data required for change-based studies. We expect a new change-based study to benefit from consulting both reviews.

One contribution of this paper is a proposed conceptual model for change-based studies. Existing conceptual models that describe software maintenance and evolution [15-17] constituted a foundation for the model. Relationships between these models and our model are further described in Section 4.

3 Review Procedures

3.1 Criteria for Inclusion and Exclusion

The following top-level criterion for inclusion of papers was derived from the objective of the review that was stated above

> *Peer reviewed papers that report on case studies that assessed or predicted maintenance and evolution activities on the basis of properties of individual changes, in the context of managed development organizations.*

Assessment and prediction are two broad purposes of measurement [18]. They are highly interdependent and we chose to include studies that involved one or both purposes. We consider studies on commercial software development and studies on volunteer-based, open source development to be two main types of software engineering research. This review focused on the former kind of studies. An opportunity for further work is to apply the developed framework to studies on open source development, with the goal of revealing contrasts and similarities between the two types. The review targeted both

quantitative and qualitative studies. Candidate papers were identified using the following procedure:

1. Send queries based on the inclusion criterion to search engines using full-text search
2. Read identified papers to the extent necessary to determine whether they conformed to the criterion
3. Follow references to and from included papers; then repeat from step 2

Step 1 was piloted in several iterations in order to increase the sensitivity and precision of the search. A discussion of the tradeoffs between sensitivity and precision in the context of research on software engineering is provided by Dieste and Padua [19]. We arrived at the following search criterion for the first step, from which we derived search strings in the query languages that is supported by the selected search engines:

```
((<size | type | complexity> of [a] <change | modification | maintenance> [task | request])
OR
(<change | modification | maintenance> [ task | request ] <size | complexity | type>))
AND <project | projects> AND software
```

Angle brackets denote that exactly one of the enclosed terms is selected, square brackets denote that zero or one of the enclosed terms is selected, while parentheses clarify operator priority.

We used Google Scholar (http://scholar.google.com) and IEEExplore (http://ieeexplore.ieee.org) because full-text search was required to obtain reasonable sensitivity. The queries returned 446 results from Google Scholar and 169 results from IEEExplore on the 19 April 2007. In total, 261 papers remained after excluding papers on the basis of the title alone, i.e., non-software engineering work, definitely off topic, or not a peer reviewed paper. After merging the two sources, 230 papers remained. These underwent Steps 2 and 3 above. Sixty-two papers were judged as “included” or “excluded, but under some doubt”. These were re-examined by the second and third author, resulting in 33 included papers. Disagreements were resolved by discussion and by further clarifying and documenting the criteria for inclusion and exclusion. As a final quality assurance, the search criterion was applied to all papers from 27 leading software engineering journals and conference proceedings (1993 to 2007 volumes), see [20] for details of this source. One additional study was identified by this step, resulting in a total of 34 included papers. In summary, we have searched Google Scholar, IEEExplore, specifically selected journal and papers, and searched in references. We expect this to search to be reasonably

complete, although alternative sources exist. For example, we did not use the ACM digital library because the service did not feature advanced full-text search.

In order to convey the criteria for inclusion or exclusion more explicitly, the remainder of this section summarizes studies of software maintenance and evolution that were excluded, but were considered to lie on the boundaries of the criteria.

An influential body of research on software evolution has based analysis on software releases and the components, i.e., the system parts of some type and at some level of granularity, that were present in successive releases. Belady and Lehman [3] measured the number of components that were created or affected in successive releases of the same system. Using this study as a basis, they postulated the *law of continuing change*, the *law of increasing entropy*, and the *law of statistically smooth grow.* Kemerer and Slaughter [21] provided an overview of empirical studies that have followed this line of research. The studies that used another unit of analysis than the individual change, e.g., releases or components, were excluded from this review.

Based on an industrial survey on maintenance of application software, Lientz *et al.* quantified the amount of new development versus maintenance, and how work was distributed over types of maintenance [22]. This work has been influential in that it has drawn attention to later phases of the software lifecycle, and via the adoption of the change classification scheme of corrective, perfective and adaptive changes, originally described by Swanson [23], and frequently used as a change attribute in the body of research included in this review. This work is not included in the review, because it was based on a survey rather than a case study.

Measures of structural attributes (code metrics) have been conjectured to provide inexpensive and possibly early assessments and predictions of system qualities. Measures have normally been extracted from individual source code components, or from succeeding revisions of source code components. Briand and Wüst [24] provided an overview of empirical work on relationships between structural measures of object-oriented software, and process and product qualities. In order to identify erroneous components when building fault prediction models, some studies identified the components that were affected by a corrective change request, c.f., [25-27]. However, we did not consider these studies to be change-based, because the unit of analysis was the individual component.

Studies on the analysis of software defects have attempted to understand the causes and origins of defects. Generally, these studies have analyzed and extracted measures from individual components. Some of the studies collected data about corrective change tasks,

e.g., [28-30]. We chose to exclude studies that analyzed the causes of defects retrospectively, but to include studies that analyzed the change tasks that were performed to isolate or correct defects.

Research on cognitive aspects of software engineering has attempted to understand the mental processes that are involved in software engineering tasks. Some of these studies have been conducted in the context of change tasks that are performed during software maintenance and evolution, c.f., [31]. We chose to exclude these studies, because Détienne and Bott [32] have provided a comprehensive summary of this specialized line of research.

3.2 Extraction of Data

Goals, change attributes, and study context were described and classified by combining existing description frameworks with data-driven analysis similar to the constant comparison method of qualitative analysis [33]. In particular, for *measurement goals*, passages of relevant text were identified, condensed, and rephrased using terms consistent with the description template for measurement goals under the Goal Questions Metrics (GQM) paradigm [34]. The resulting measurement goals are listed in Tables A1, A2 and A3. The next step was to classify these instances of measurement goals into categories. We attempted to discriminate between lines of research, i.e. studies that have similar overall goals and take similar approaches to analysis. The process was driven by the first author of this paper. The second and third author proposed changes and clarifications where perceived necessary. These procedures resulted in the taxonomy listed in Table 1.

It was necessary to make a tradeoff with respect to the specificity of categories. If categories were too fine-grained, the schema could be over-fitted to particularities of the investigated studies. This would make it more difficult to reliably classify new papers according to the schema. If categories were too coarse-grained, important distinctions between lines of research could be lost, making the schema less useful. An example of a tradeoff is the categories *3.2 - assess prediction frameworks* and *3.3 - investigate predictive power of change measures* from Table 1. Prediction frameworks are normally assessed assuming one or more change measures, and vice versa. Still, because evaluation in the studies focused on either the effectiveness of the prediction frameworks or on the effectiveness of the measures, we considered the two categories of studies to contribute with two different kinds of results.

In order to describe and classify conceptual change attributes, we extracted information about each individual change measure that was used in the studies. Key information was

names, definitions, value ranges, and methods for data collection. This information was then compared and grouped with respect to the conceptual model in Figure 1, and with respect to a set of more detailed measurement questions, as listed in Table 2. The procedures for developing the conceptual model for change-based studies are described in Section 4.

For *study context*, we describe the business context, measurement procedures and extent of data collection. We identified two measures for each of these attributes by using information that was available in the reviewed papers. The results are shown in Table A4.

4 A Conceptual Model for Change-Based Studies

Our proposed conceptual model for change-based studies is depicted in Figure 1. The goals for the design of the model were 1) to create a minimal model that 2) facilitates the understanding and definition of entities, attributes and measures that were used in the reviewed body of research, while 3) maintaining compatibility with existing concepts that have been used to discuss software maintenance and evolution.

We developed and refined the model iteratively during the course of the review, in order to capture the change attributes that were used in the reviewed studies. Table 2 lists the relationships between these attributes and the entities in the model. Wherever possible we reused concepts from existing conceptual models of software maintenance. In particular, the entities *Development organization*, *Human resource*, *Change task*, *Change request*, *Component*, *System* and *Release*, some of them with different names, were reused from the proposed ontology of software maintenance by Kitchenham *et al.* [16]. Similar conceptual frameworks have been defined by Dias *et al.* [15] and Ruiz *et al.* [17]. We used terms in our model that were 1) commonly used in the reviewed body of research, 2) neutral with respect to specific technologies, practices or disciplines in software engineering, and 3) internally consistent. For example, we used the term *change task* for the entity that is named *maintenance task* in [16]. Compared to the existing frameworks, the entities *Change set*, *Revision* and *Delta* and their interrelationships were added, because they are necessary to describe and classify the change attributes that concern changes to the system components. The relationships between some of the reused entities were changed, in order to better represent the change-oriented perspective taken in this paper.

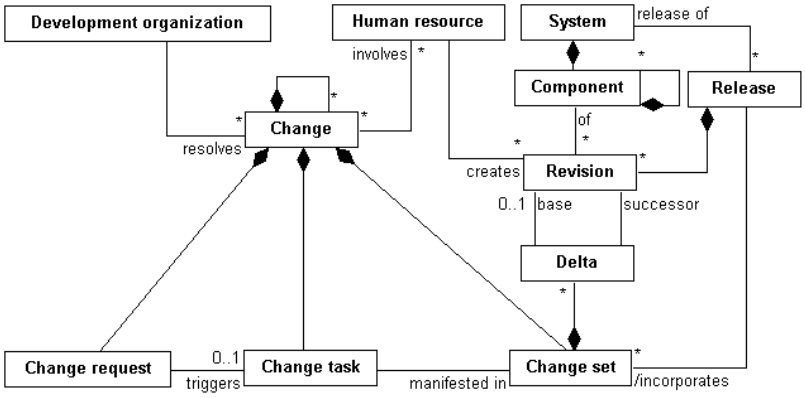


Figure 1. A conceptual model for change-based studies

Standard UML syntax is used in the diagram. A role multiplicity of 1 should be assumed when role multiplicity is not shown. Role names are assigned in one direction only, in order to avoid cluttering. For compositions, indicated by filled diamonds, the roles in the two directions can be read as *composed by* and *part of*.

The perspective adopted in this paper is that a *change task* constitutes the fundamental activity around which software maintenance and evolution is organized. A change task is a coherent and self-contained unit of work that is triggered by a *change request*. A change request describes the requirements for the change task. A change task is manifested in a corresponding *change set*. A change set consists of a set of *deltas*. A delta is the differences between two *revisions* of the same component. A component can, in principle, be any kind of work product that is considered to be part of the system, although the reviewed studies focused primarily on measurement of source code components. Components can form a hierarchy in where a large component can be composed by components of finer granularity. A system is deployed to its users through *releases*. A release is composed by a set of components, each in a specific revision. A release can also be described by the change sets or the corresponding change requests that the release incorporates.

It is convenient to use the term *change* as an aggregating term for the change task, the originating change request, and the resulting change set. Changes, in this sense, involve *human resources*, and are managed and resolved by a *development organization*. Large changes can be broken down into smaller changes that are more manageable to the development organization. In the reviewed studies, large changes are sometimes referred to

as *features*, although there are many possible underlying causes for large changes, as investigated in [35].

A change attribute is a property of a change task, of the originating change request, or of the resulting change set. A change attribute is sometimes derived from attributes of other entities in the conceptual model. For example, the sizes of all components that were involved in a change may be averaged, or otherwise combined, in order to form a change attribute that represents the size of changed components.

A *change outcome* is a change attribute that represents the primary focus of the study, e.g., *change effort*. A *change outcome measure* is the operationalization of a change outcome, and is typically used as the dependent variable in statistical analyses.

Many change measures can be extracted from *change management systems*, which are tools that manage the kind of information defined by our conceptual model. Such systems include change trackers and version control systems. In order to clarify the meaning of the entities in the conceptual model, the model entities were mapped to terminology used in the popular version control systems CVS and Subversion: A CVS *check-in* or *commit* creates a *delta*, i.e. a difference between the previous and the current revision of a component. In Subversion, a commit or an *atomic commit* guarantees that all *deltas* are either committed or rolled back. An atomic commit corresponds to a *change set* if it corresponds to the solution for one change request. CVS does not support change sets, but change sets can be deduced from metadata such as username and commit time. In CVS, and in our model, a *revision* refers to a reproducible state of a specific component. In Subversion, a revision can also refer to the state of the entire file system under version control. A development organization will promote one such Subversion revision to constitute a *release*. Both Subversion and CVS supports tags, which makes it easier to retrieve the exact contents of a release. In CVS and Subversion, a *file* refers to a *component*. The file system under version control corresponds to the *system*. Change management systems use terms such as *modification request (MR)*, *bug report*, *ticket*, *issue*, *software defect report* (SDR) and *problem report* (PR) to refer to a *change request*. A change task is sometimes referred to as an *activity* or a *task*. In the reviewed papers, the term *maintenance task* was often used for the same concept.

It is beyond the scope of this paper to provide operational definitions of all variations of change measures used in the reviewed studies. However, the conceptual model in Figure 1 can be utilized further in a specific measurement context to facilitate precise definitions of change measures. For example, the span of a change could be operationalized as “the

number of deltas that are part of a change set", while a measure of the size of affected components can be defined as "the arithmetic mean of lines of code in revisions affected by a change set". Such definitions can be expressed formally using the Object Constraint Language (OCL) [36].

5 Goals and Measured Change Attributes

By following the procedures described in Section 3.2, three main categories and 10 sub-categories of studies were identified, as shown in Table 1. Key properties of each individual study are listed in Tables A1, A2 and A3, in Appendix A.

Table 1. Goals and sub-goals for change-based studies.

Main category	Sub-category	References
Goal 1: Characterize the work performed on evolving systems (Table A1)	Goal 1.1: Understand and improve the maintenance and evolution process in a development organization	[37-42]
	Goal 1.2: Manage and control the maintenance and evolution process in a development organization	[43-45]
	Goal 1.3: Investigate selected elements in the maintenance and evolution process	[46-49]
	Goal 1.4: Understand the general nature of maintenance and evolution work	[21, 50-52]
Goal 2: Assess change attributes that explain change outcome (Table A2)	Goal 2.1: Identify change attributes that influence change outcome	[53, 54]
	Goal 2.2: Assess effects of a specific process element	[55-58]
	Goal 2.3: Validate change measures	[59, 60]
Goal 3: Predict the outcome of changes (Table A3)	Goal 3.1: Propose methodology for building predictive models	[61-64]
	Goal 3.2: Assess prediction frameworks	[65, 66]
	Goal 3.3: Investigate predictive power of change measures	[13, 67, 68]

Goal 2 and Goal 3 studies employed quantitative models that related independent change measures to the change outcome measure of interest. Goal 2 studies attempted to identify causal relationships for the purpose of understanding and assessment, while Goal 3 studies focused on correlations and predictions. Conversely, most Goal 1 studies used summary statistics to provide a bird's eye view of the work that was performed during maintenance and evolution. They focused on observing trends in the values for selected change attributes, rather than attempting to explain the observations.

Change attributes, typical questions and typical values used during data collection are shown in Table 2. All attributes can be regarded as attributes of a *change* in Figure 1. The second column indicates the entity in Figure 1 that provides the data for deriving a change attribute. For example, a measure of *change size* is usually obtained by aggregating data from the individual *deltas* in the change set.

Table 2. Change attributes used by the studies

Change attribute	Data provided by	Question asked	Typical values	Goal 1 studies	Goal 2 studies	Goal 3 studies
Activity	Change task	Which activities were involved in the change task?	Requirements/analysis/design/coding/test	[37, 41]	-	-
Change count	Change request	Was it a change?	Simple count of changes	[21, 37, 40-42, 44-46, 49-51]	-	-
Change effort	Change task	How much effort was expended on the change task?	Person hours, ordinal or ratio	[37, 40-47]	[53-56, 58-60]	[13, 61, 62, 64-68]
Change interval	Change task	How long did it take to resolve the change request?	Days, ordinal or ratio	[45, 47, 48]	[53, 57]	[63, 64]
Change size	Delta	How much content was added, deleted or changed?	Lines of code, ordinal or ratio	[37, 38, 41]	[54-56, 59]	[13, 61, 63, 65]
Change span	Delta	How many components were affected?	Count of components	[42, 48]	[53-58]	[13, 61-63]
Change/ defect source	Change request	Which activity caused the defect or the need for change?	Requirements/ analysis/design/ coding/test	[41, 42]	-	[13, 61]
Code quality	Revision	Had the changed components been refactored?	Refactored/ not refactored	-	[56]	-
Code volatility	Component	How frequently had the affected components been changed?	Total number of changes	-	[54]	-
Coding mode	Delta	Was content changed or added?	Changed/added	-	-	[65]
Component type	Component	What kind of component was affected?	Query/report/ field/layout/data	[40]	-	[13]
Criticality	Revision	How critical was the affected component?	Is mission critical?	-	[54]	-
Data operation	Revision	Which data operation did the affected components perform?	Read/update/process	-	[54]	[66]
Date/ Time	System	How old was the system when the change occurred?	Elapsed time since first deployment	[21, 40, 43, 45, 47, 50-52]	[53]	[65, 69]
Defect type	Change request	What kind of defect was introduced?	Initialization/ logic/data/interface/ computational	[41, 45]	-	[61, 66]
Delayed	Change task	Was the change task resolved later than scheduled?	Delayed/not delayed	[45]	-	-
Detection	Change request	By which technique was the defect/need for change detected?	Inspection/test-run/proof techniques	[42]	-	-
Developer id	Human resource	Who performed the change task?	Nominal measure	-	[53]	
Developer span	Human resource	How many developers were involved in performing the change task?	Number of people	-	[54, 57]	[63]
Documentation quality	Component	How well were the changed components documented?	Was documentation rewritten?	-	[58]	[13]
Execution resources	Delta	How much (added) computational resources were required by the change?	CPU-cycles, bytes of memory	-	[54]	-

Table 2. Continued

Change attribute	Data provided by	Question asked	Typical values	Goal 1 studies	Goal 2 studies	Goal 3 studies
Function points	Delta	How many logical units will be changed, added or deleted by the change?	Count of changed, added and deleted units, weighted by complexity	-	[60]	[67]
Location	Develop-ment org.	Where were human resources located physically?	Distributed/not distributed	-	[57]	-
Maintenance experience	Human resource	For how long had the developers performed software maintenance work?	Number of years	-	-	[65]
Maintenance type	Change request	What was the purpose of the change?	Fix/enhance/ adapt	[21, 37, 41-47, 49-51]	[53-55, 57]	[61, 63-65, 68]
Objective change experience	Human resource	How many changes had earlier been performed by the developer?	Number of previous check-ins in version control system	-	-	[63]
Origin	Change request	In what context or by which party was the change request made?	Internal test/ external users	[37, 46, 52]	-	-
Quality focus	Change request	Which system quality was improved by the change?	Functionality/ security/efficiency/reliability	[42, 45, 47, 52]	-	-
Request criticality	Change request	What would be the effect of not accepting the change request?	Minor/major inconvenience/stop	-	[54, 57]	[65, 68]
Requirements instability	Change request	To what extent were change requirements changed?	Number of requirement changes	-	[54]	[13]
Size	Revision	How large were the changed components?	Lines of code, number of components affected	-	[59]	[13, 63, 65, 67, 68]
Status	Change request	What is the current state of the change request?	New/accepted/ rejected/solved	[44, 45, 48, 52]	-	-
Structural attributes	Revision	What was the profile of the structural attributes of the changed components?	Count of structural elements (coupling, branching statements)	-	[59]	[13, 62]
Subjective complexity	Change task	How complex was the change perceived to be?	3-point ordinal scale	-	[54]	[13, 65-67]
Subjective experience	Human resource	How was experience with respect to the affected components perceived?	3-point ordinal scale	-	-	[65]
System experience	Human resource	For how long time had the developers been involved in developing or maintaining the system?	Number of years	-	-	[65]
System id	System	To which system or project did the change belong?	Nominal measure	[37, 42-44]	-	-
Team id	Develop-ment org	Which team was responsible for the change task?	Nominal measure	-	-	[67]
Technology	Component	Which technology was applied in the changed components?	3GL/4GL	-	-	[65]
Test effort	Change task	What was the test effort associated with the change?	Number of test runs	-	[54]	-
Tool use	Develop. org.	Which tool was involved in the change task?	Tool used/not used	-	[55]	-

Seventeen of the 43 change attributes were used by one study only. These 17 attributes were used by 10 different studies; hence we do not consider Table 2 to be over-influenced by any individual paper. The summaries in Sections 5.1 to 5.5 focus on goals and main contributions of each study. Section 6 provides a guide for future research on the basis of the current contributions within each main category.

5.1 Summary of Characterization Studies (Goal 1)

Goal 1 studies were split according to the sub-categories listed in Table 1. Goal 1.1 and Goal 1.2 studies are characterized by close involvement with the measured development organization. The measurement programs were planned in advance, e.g., following the GQM paradigm [34]. They are similar with respect to goals; however, we consider it to be an important discriminator whether or not the study aimed at developing management tools for monitoring and decision support in ongoing projects. Goal 1.2 studies focused on such tools, while Goal 1.1 studies had a longer-term goal of understanding and improving the maintenance and evolution process.

The four earliest Goal 1.1 studies are from the space domain, characterized by a long-lasting mutual commitment between the development organization and software engineering researchers. A certain amount of overhead for data collection was accepted in these environments. The studies appear to follow a tendency over time from studies for assessment and insight [41, 42], via studies for understanding and improved predictability [37], towards studies that took concrete actions in the form of process improvements [39]. Lam and Shankararaman [40] showed that these measurement goals were also feasible in projects that are managed less strictly. While the above studies focused on analyzing a comprehensive set of real changes, Bergin and Keating [38] used a benchmarking approach that evaluated the outcome of artificial changes that were designed to be representative of actual changes.

The Goal 1.2 studies were conducted within strictly managed development organizations. Arnold and Parker [44] involved management in setting threshold values on a set of selected indicators. This was an early attempt to use change measures to support decisions made by managers in a development organization. Likewise, Abran and Hguyenkim [43] focused on management decision support, and provided upfront and careful considerations about validity issues that pertain to change-based studies. Finally, Stark [45] suggested a rich set of indicators that provided answers to questions about the services provided by the development organization to its clients.

Goal 1.3 and Goal 1.4 studies collected data from change management systems, and attempted to provide insight into software maintenance and evolution that was generalizable beyond the immediate study context. Generalizability to other contexts was claimed on the basis of recurring characteristics of systems and development organizations.

Goal 1.3 studies investigated the effect or intrinsic properties of specific process elements. Ng [46] investigated change effort in the domain of Enterprise Resource Planning (ERP) implementation. The remaining three studies addressed three different process topics: the intrinsic properties of parallel changes [48], instability in requirements [47], and the intrinsic properties of small changes [49].

Goal 1.4 studies addressed the nature of the software evolution and maintenance process in general. Kemerer and Slaughter [21] categorized change logs that had been written by developers that maintained 23 systems within one development organization in order to identify patterns in the types of change that occurred during the investigated period of 20 years. Mohagheghi [52] analyzed a smaller set of change requests to answer specific questions about who requested changes, which quality aspects that were improved by the changes, time/phase at which the requests were created, and to what extent change requests were accepted by the development organization.

5.2 Change Attributes in Characterization Studies (Goal 1)

In summary, all Goal 1 studies attempted to characterize the work performed by development organizations. A predominant principle of measurement was to categorize changes according to selected characteristics. The proportion of changes that belonged to each category was compared to organizational standards, to other projects/systems, and between releases or time periods. Maintenance type, originally described by Swanson *et al*. [23], was the criterion for classification that was applied most frequently. In particular, the proportion of corrective change versus other types of change was frequently used as an indicator of quality, the assumption being that corrective work is a symptom of deficiencies in process or product. In most cases, observations and conclusions were based on descriptive statistics. In four studies, the statistical significance of proportions was investigated [21, 37, 51, 52]. Change effort, measured in person hours, was a key change measure for studies that focused on resource consumption. The number of changes was sometimes used as a surrogate measure when data on effort was not available. Some studies suggested using the average change effort per maintenance type as a rough prediction for the effort required to perform future change tasks of the same type.

5.3 Summary of Studies that Assess Change Attributes (Goal 2)

Goal 2 studies were split according to the goal sub-categories listed in Table 1. The studies used correlation analysis at different levels of complexity in order to identify relationships between change measures used as independent variables and the change outcome measure. An overview of change outcome measures is given in Section 5.5.

Goal 2.1 studies attempted to identify causal relationships between change attributes and change outcome, while Goal 2.2 studies investigated the effect of specific process elements. Graves and Mockus [53] controlled for variations due to maintenance type and change size, and showed that change effort increased with system age. They automated the extraction of change measures from change management systems in order to minimize measurement overhead. Schneidewind [54] used historical change requests to investigate correlations between change attributes and the presence of defects. Atkins *et al.* [55] showed that introducing a new tool to support the development of parallel versions of the same components had a positive effect on effort. Hersleb and Mockus [57] showed that decentralization prolonged the change interval. Rostkowycz, Rajlich *et al.* [58] showed that re-documenting a system reduced subsequent change effort, and demonstrated that the breakeven point for investment in re-documentation versus saved change effort was reached after 18 months.

Goal 2.3 studies attempted to find appropriate change measures of concepts that are commonly assumed to influence change outcome. Maya, Abran *et al.* [60] described how function point analysis could be adapted to the measurement of small functional enhancements. They tested whether the function point measure could predict change effort, and they observed a weak correlation in their study. Arisholm [59] showed that aggregation of certain measures of structural attributes of changed components could be used to assess the ease with which object-oriented systems could be changed.

5.4 Summary of Prediction Studies (Goal 3)

While Goal 2 studies attempted to identify change attributes that influence change outcome, the Goal 3 studies attempted to predict that outcome. These studies used various prediction frameworks in order to build development organization specific prediction models of change outcome. The studies can be split according to the sub-categories listed in Table 1.

Goal 3.1 studies investigated methods and processes for building prediction models. In [61], Briand and Basili suggested and validated a process for building predictive models

that classified corrective changes into different categories of effort. Evanco [62] used similar procedures to predict effort for isolating and fixing defects, and validated the prediction model by comparing the results with the actual outcomes in new projects. Xu *et al.* [64] employed decision tree techniques to predict the change interval. The predictions from the model were given to the clients to set their expectations, and the authors quantified the approach's effect on customer satisfaction. Mockus and Weiss [63] predicted the risk of system failures as a consequence of changes that were made to the system. They automated the statistical analysis required to build the models, and integrated the predictions into the change process that was used by the developers.

Goal 3.2 studies compared prediction frameworks with respect to their predictive power and the degree to which the frameworks exposed explanations for the predictions. In [65], Jørgensen assessed and compared neural networks, pattern recognition and regression models for predicting change effort. He concluded that models can assist experts in making predictions, especially when the models expose explanations for the predictions. In [66], Reformat and Wu compared Bayesian networks, IF-THEN rules and decision trees for predicting change effort on an ordinal scale. They concluded that the methods complemented each other, and suggested that practitioners should use multi-method analysis to obtain more confidence in the predictions.

Goal 3.3 studies attempted to identify change measures that could operationalize the conceptual change attribute of interest. Niessink and van Vliet [13] created and compared models for predicting change effort in two different development organizations. They suggested that the large difference in explanatory power between the organizations were due to the differences in the degree to which the development organizations applied a consistent change process. In [67], the same authors investigated variants of function point analysis to predict change effort. Although the regression models improved when the size of affected components was accounted for, the authors suggested that analogy-based predictions might be more appropriate for heterogeneous data sets. Using data on change requests and measures of system size from 55 banking systems, Polo *et al.* [68] attempted to build predictive models that could assist in the early determination of the value of maintenance contracts. Considerable predictive power was obtained from rudimentary measures, a finding that the authors contributed to the homogeneity of context (banking systems) and maturity of technology (Cobol).

5.5 Change Attributes in Assessment and Prediction Studies (Goal 2 and Goal 3)

Although Goal 2 and Goal 3 studies have very different goals, they are quite similar from the perspective of measurement, and they are therefore described together in this section.

The choice of dependent variable, i.e., the change outcome measure, is a key discriminator with respect to the focus and goal of a study. The dependent variables in the reviewed studies are derived from four change attributes:

Change effort. The number of person hours expended on performing the change task is used as a change outcome measure in studies on change attributes that may influence productivity, and in studies on the estimation of effort for change tasks. Twelve of 17 studies had these foci. In most cases, the measure was reported explicitly per change task by developers. Graves and Mockus proposed an algorithm that made it possible to infer change effort from more aggregated effort data [12]. This algorithm was put to use in, e.g., [55].

Change interval. While change effort is a measure of the internal cost of performing a change task, the time interval between receiving and resolving the change request can be a relevant dependent variable for stakeholders external to the development organization. This change measure was used in studies that focused on customer service and customer satisfaction [57, 64], where the measure could be extracted from information resident in change management systems.

Defects and failures. Historical data of defects and failures were used to identify change attributes that caused or correlated with defects and failures, to assess probabilities of defects or failures, and to assess the effect on defect proneness or failure proneness of a specific product improvement program. Such change measures are not straightforward to collect, because it can be difficult to establish a link from an observed defect or failure to the change that caused it. The two studies that have used this dependent variable analyze relatively large changes [54, 63].

Change attributes, typical questions and typical values used during data collection in Goal 2 and 3 studies are shown in Table 2. Measures of the change request, the change task and the deltas that are part of a change set occurred most frequently. Size, structure and age were the most frequently measured change attributes that used information from changed components and their revisions. Information about deltas, revisions and components that were involved in a change set could only be measured after the change had been made. For the prediction goals, such change measures needed to be predicted first. The degree of collaboration (developer span) was the most frequently measured change attribute that

used information about the human resources involved. No attribute of the development organization was used more than once.

6 Guide for Future Change-Based Studies

Change-based studies belong to the more general class of empirical software engineering studies, and are normally conducted with the characteristics of a case study. Methodological concerns for the more general class of studies can be expected to be relevant for change-based studies as well, c.f., [70-75]. A change-based study is appropriate when software evolution is organized around cohesive change tasks. Abran and Hguyenkim [43] assessed the adequacy of a planned change-based study by comparing the reported effort expended on individual changes, to the total effort expended by developers in the maintenance organization. These authors also piloted the feasibility of the planned data collection, by verifying that changes could be reliably classified into maintenance types. Such assessment and piloting reduces the risk of embarking on the wrong kind of study, and provides useful context information for the results of the main study.

Another important requirement for many change-based designs is that it is possible to group individual deltas into change sets associated with a change task or a change request. If the relationships are not explicitly tracked, the algorithm provided by German [76] can be used to recover change sets from individual deltas. Making use of naturally occurring data has the benefit over purpose-collected data that it reduces measurement overhead and enables researchers to collect more data over longer periods in time. This claim was evaluated and supported by analyzing information about data collection in the reviewed papers: The median duration of data collection was 48 and 21 months for naturally occurring and purpose-created data, respectively. The median number of analyzed changes was 1724 and 129 for the same two categories. More details about data collection in the reviewed studies are provided in Table A4.

Relevant literature for a planned change-based study can be identified by matching the study goal against the categories in Table 1. Tables A1 to A3 list individual goals, and can be consulted to identify studies that most closely match the goal of the planned study. Table 2 provides candidate change attributes to measure in the planned study, and points to earlier studies that are similar with respect to measurement. The listed relationships between change attributes in Table 2 and entities in Figure 1 can help to define and operationalize the measures.

To further guide future studies, it is necessary to somehow assess the current state-of-the-art and challenges within each group of studies. We chose to adopt the perspective of Perry *et al.* that empirical studies should go in the direction of being *causal*, *actionable* and *general* [73]. When there is a causal relationship, it is known why something happens. When the cause is actionable, outcomes can be changed and improved. Generality ensures that the study is useful to a wide range of situations. In the following, we present one paper for each of the three main goals in Table 1 that we consider to be most advanced with respect to these criteria.

Briand *et al* [39] (a Goal 1 study) attempted to identify causal links between inadequate organizational structures or maintenance practices, and problems in the maintenance phase. The actionable root causes of reported problems were expressed in statements like "*communication between users and maintainers, due in part to a lack of defined standards for writing change requirements*" and "*lack of experience and/or training with respect to the application domain*". A notable feature of this study is that it employed qualitative analysis to identify causal relationships. Furthermore, to be able to generalize case study results, it is recommended that proposals and hypothesis for such studies should be based on theories [77]. The results can then be used to refute, support or modify the theory in some way. The mentioned study contained elements of theory use, being based on a model that proposed causal links between flaws in overall maintenance process, the human errors that are committed, and the resulting software defects.

Atkins, Ball *et al* [55] (a Goal 2 study) quantified the effect on change effort and change interval of a particular development tool. They used an ANCOVA-type model that included a binary treatment variable (the tool was/was not used) and variables that controlled for factors such as size and type of change. The study showed that analysis of change-based field data can provide strong evidence of the causal effects of applying technologies and practices in software engineering. Although a controlled experiment would better handle threats to internal validity, concerning the existence of a cause-effect relationship, the power of Atkin, Ball et al's methodology was the use of real change data, representing exactly the constructs of interest to the organization. The authors were also able to control for individual developer differences, which was considered to be the most serious threat to internal validity.

Mockus and Weiss [63] (a Goal 3 study) integrated model-based risk prediction into the development process so that developers were alerted when risk threshold values were exceeded for new changes. They used candidate variables in their models that represented

proposed risk factors, such as size and type of the change. The study attempted to account for the effect of developers' experience by designing experience measures from version control data. The prediction models confirmed the correlations between some of the proposed factors, and the risks of failure. A particularly promising element of this work was the design of automated procedures to extract new data, update prediction models, perform predictions, and alert developers on the basis of dynamically adjusted threshold values.

The three mentioned change-based studies illustrate how empirical software engineering studies are capable of changing and improving the way development is done within specific software organizations. However, there are strong limitations to the generalizability of the results. It is the authors' opinion that a stronger foundation in common conceptual models and theories is required to overcome this shortage.

7 Limitations of this Study

The process by which papers were selected balanced the use of systematic, repeatable procedures with the intent to identify a comprehensive set of change-based studies. A more repeatable process could have been achieved by limiting searches to abstracts and titles only, by omitting traversal of literature references, and by excluding the Google Scholar search engine, which yielded low precision for paper retrieval. However, a more repeatable process may have failed to retrieve many of the included papers. Given that meeting the objective and answering the research questions of this study relied on identifying a broad set of change-based studies we chose to assign lower priority to repeatability. As a consequence, the procedures we followed did not fully comply with the procedures for systematic reviews that were suggested by Kitchenham *et al.* [11]. It is worth noting that the challenges experienced in attempting to follow systematic procedures stem from the lack of common conceptual frameworks. A common conceptual basis would clearly improve sensitivity and precision during the selection of papers.

The ease with which studies can be classified according to the categories in Table 1 was evaluated by letting two independent senior researchers in our research group attempt to classify six randomly sampled papers. The two researchers worked individually, and were instructed to complete the categorization for one paper before turning to the next. On average, the researchers used 7 minutes to classify one paper. The preparation material consisted of this paper's abstract, introduction, Table 1, and the descriptions of the

categories as they appear in Section 5. The results showed that the agreement between the baseline and evaluations was *fair* by normal standards [78]. An important source for disagreements was that some studies contained elements from different goals. For example, the study by Schneidewind [54] is described by the author to identify actionable risk factors at an early stage in the development cycle (indicating a causal Goal 2 study), but the methodology used resembles the methodology of a prediction study (by focusing on correlations). This situation resulted in different classifications between the researchers.

Despite the finding from the evaluation that some studies overlap several categories, we consider the schema to be a useful map of the goals for change-based studies. Also, the categories are assumed to be useful to help researchers clearly define their research goals, possibly covering more than one of the goal categories. It is future work to evaluate and improve the classification schema on the basis of new studies, and with respect to open sources studies, which were outside the scope of this review.

8 Conclusions and Further Work

Change-based studies assume that software maintenance and evolution is organized around change tasks that transform change requests into sets of modifications to the components of the system. This review of change-based studies has shown that specific study goals have been to characterize projects, to understand the factors that drive costs and risks during software maintenance and evolution, and to predict costs and risks. Change management systems constitute the primary source for extracting change measures. Several of the reviewed studies have demonstrated how measurement and analysis can be automated and integrated seamlessly into the maintenance and evolution process.

Although this review includes examples of successful measurement programs, it was difficult to determine whether and how insights into software maintenance and evolution could be transferred to situations beyond the immediate study context. We recommend that new change-based studies should base measurement on conceptual models and, eventually, theories. This observation may be seen as an instance of a general need for an improved theoretical basis for empirical software engineering research. In order to make progress along this line, we anchored this review in a minimal, empirically based, conceptual model with the intention of supporting change-based studies. We built the model by ensuring compatibility with existing ontologies of software maintenance, and by extracting and conceptualizing the change measures applied in 34 change-based studies from a period of

25 years. In future work, we will conduct a change-based multiple-case study with the aim of understanding more about the factors that drive costs of software maintenance and evolution. The results from this review constitute important elements of the study design. We believe that this review will be useful by other research and measurement programs, and will facilitate a more effective accumulation of knowledge from empirical studies of software maintenance and evolution.

Acknowledgements

We thank the Simula School of Research and Innovation for funding this work, the anonymous reviewers for important feedback, Simon Andresen for our discussions on conceptual models of software change, Aiko Fallas Yamashita for her comments that improved the readability of the paper, and Leon Moonen and Dietmar Pfahl for participating in the evaluation procedures.

Appendix A. Summary of Extracted Data

The three main classes of included studies are listed in Tables A1, A2 and A3. Within each class, the studies are listed in chronological order. In Tables A2 and A3, an asterisk (*) is used as an indication that the variable was statistically significant, at the level applied by the authors of the papers, in multivariate statistical models. Table A4 summarizes business context, measurement procedures and extent of data collection in the reviewed studies.

Table A1. Characterize the work performed on evolving systems (Goal 1)

Study	Category:Goal	Indicators and change attributes
Arnold and Parker [44]	1.2: Manage the maintenance process	Change count by: • Maintenance type (fix, enhance, restructure) • Status (solved requests vs. all requests), per maintenance type • Change effort (little/moderate vs. extensive) per maintenance type Measures were compared to local threshold values for several systems
Weiss and Basili [42]	1.1: Assess maintenance performance in ways that permit comparisons across systems	Change count by: • Defect source (req. specification, design, language, …) • Change effort (<1hr, <1day, >1day) • Quality focus (clarity, optimization, user services, unknown) • Maintenance type (change, fix non-clerical error, fix clerical error) • Change span (number and identity of changed components) • Detection (test runs, proof techniques, and more) • Change effort (design, code: <1hr, <1day, >1day, unknown) Measures were compared between projects/systems
Rombach, Ulery *et al*. [41]	1.1: Understand maintenance processes, in order to improve initial development and management of maintenance projects	Change count by: • Maintenance type (adapt, correct, enhance, other) • Change effort (<1hr, <1day, >1day) Average number of • Change size (source lines + modules added, changed and deleted) Compare development to maintenance with respect to proportion of • Change effort (<1hr, <1day, >1 day) per activity • Defect type (initialization, logic, interface, data, computational) • Defect source (specification, design, code, previous change)
Abran and Hguyenkim [43]	1.2: Analyze and manage maintenance effort	• Distribution of change effort by maintenance type (corrective, adaptive, perfective, user) by system and year • Average change effort, per maintenance type, system and time
Basili, Briand *al*. [37]	1.1: Improve understanding and predictability of software release effort	Distribution of change effort by • Activity (analysis, design, implementation, test, other) • Activity, for costliest projects/systems • Activity, compared between maintenance types (correct, enhance) • Maintenance type (adapt, correct, enhance, other) • Origin (user, tester) • Compare change count and change size (LOC), between origins (internal tester, user)
Stark [45]	1.2: Control customer satisfaction, maintenance cost and schedule	Time trend in proportions of • Delayed (delayed vs. not delayed) • Status (solved vs. not yet solved) • Status (rejected vs. not rejected) • Change interval used to close urgent requests • Change count and change effort by defect type/maintenance type/quality focus (computational, logic, input, data handling, output, interface, operations, performance, specification, improvement)

Table A1. Continued

Study	Category: Goal	Indicators and change attributes
Gefen and Schneberger [51]	1.4: Investigate the homogeneity of the maintenance phase, with respect to the amount of change	• Time trend in change count • Time trend in change count, by maintenance type (requirement change, programming related fix) • p-value and coefficient value in regression models of time vs. change count in time period, and per maintenance type • p-value in t-test of difference between time periods with respect to maintenance type (correct, adapt) and change count
Burch and Kung [50]	1.4: Understand time trends of changes	• Time trend in change count, by maintenance type (support, fix, enhance), using statistical models
Briand, Kim *et al.* [39]	1.1: Assess and improve quality and productivity of maintenance	• Qualitative summaries, based on interviews and questionnaires, of factors that influence maintenance performance (focused on product defects), related to development organization, process, product and people
Lam and Shankararaman [40]	1.1: Assess trends in maintenance performance	• Average change effort, by component type (domain specific) • Change count, by type and time period • Change count that resulted in defect, by time period
Kemerer and Slaughter [21]	1.4: Identify and understand the phases through which software systems evolve	• p-values and coefficient in regression model of time vs. change count • Degree to which certain maintenance types occur together over time, by using gamma analysis [79]. 31 sub-types of corrective, adaptive, enhancive and new changes were used
Ng [46]	1.3: Understand ERP maintenance effort	• Change effort and change count by origin (service provider, end-client) and maintenance type (fix, enhance, master data)
Perry, Siy *et al.*[48]	1.3: Understand parallelism in large-scale evolution	Change (at three levels of granularity) count by • Change interval (number of days) • Status (being worked on, not being worked on) • Change span (number of files) • Developer span (see Table 3)
Bergin and Keating [38]	1.1: Assess changeability of a software system	• Change size (percentage change to the software required by seven typical changes)
Mohagheghi, Conradi *et al.* [52]	1.4: Investigate the nature of change requests in a typical project	Proportions, and p-value for one-proportion tests of • Quality focus (functional vs. non-functional changes) • Origin (inside vs. outside development organization) • Time (before vs. after implementation and verification) • Status (accepted vs. not accepted), in total and per release
Nurmuliani and Zowghi [47]	1.3: Measure requirements volatility in a time-limited project	• Time trend in maintenance type (add, delete, modify requirement) • Time trend in quality focus • Change interval, by maintenance type and quality focus • Mean predicted change effort, by maintenance type and quality focus
Purushothaman and Perry [49]	1.3: Understand the nature of small code changes	• Change count by maintenance type (corrective, adaptive, perfective, inspect) compared between small and larger changes

Table A2. Assess change attributes that explain change outcome (Goal 2)

Study	Study goal and dependent variables (DV)	Independent variables of change request, change task, change set and delta	Independent variables of system, components or revisions	Independent variables of human resources/ organization
Maya, Abran *et al.* [60]	2.3: Propose and validate function points as measure of change size, for the purpose of productivity assessment and prediction DV: Change effort	Function points (fine granularity for complexity)	Not used	Not used
Graves and Mockus [53]	2.1: Identify change attributes that influence change effort and to find evidence of code decay DV: Change effort	Maintenance type* Change span (check-ins)* Change interval	Not used Time*	Developer id
Schneide-wind [54]	2.1 : Understand how change request attributes relate to process and product quality, and build quality prediction models DV: Change caused defect	Maintenance type Subjective complexity Change size * Change span (# requirements affected, modules affected) Change effort (code, test) Execution resources* Request criticality*	Criticality Code volatility Data operation	Developer span Requirements instability* Test effort
Atkins, Ball *et al.* [55]	2.2: Evaluate the impact of a tool (version editor) DV: Change effort, change interval, change caused defect	Maintenance type* Change size Change span (# check-ins)	Not used	Tool use* (version editor used)
Herbsleb and Mockus [57]	2.2: Evaluate the impact of project decentralization DV: Change interval	Maintenance type* Change span (check-ins, modules)* Request criticality*	Not used	Developer span* Time (date)* Location*
Rostkowycz, Rajlich *et al.* [58]	2.2: Assess the cost-benefit of re-documenting software components DV: Change effort	Change span	Documentation quality*	Time (date)*
Geppert, Mockus *et al.*[56]	2.2: Assess effect of refactoring DV: Defects, change effort, change size, change span	Not used	Code quality (affected code refactored)*	Not used
Arisholm [59]	2.3: Validate measures of structural attributes, adapted for changes, as indicators of changeability DV: Change effort	Change size	Structural attributes weighted by change size Export coupling* Class size*	Not used

Table A3. Predict the outcome of changes (Goal 3)

Study	Study goal and dependent variables (DV)	Independent variables of change request, change task, change set and delta	Independent variables of system, components or revisions	Independent variables of human resources/ organization
Briand and Basili [61]	3.1: Validate a proposed process for constructing customized prediction models of change effort, DV: Change effort	Maintenance type* Change source* Defect type* Change size Change span	Not used	Not used
Jørgensen [65]	3.2: Assess and compare modelling frameworks and change measures in predictive models DV: Change effort	Change size* Maintenance type* Subjective complexity* Coding mode* Request criticality	Technology (3GL/4GL) Age Size	System experience Maintenance experience
Niessink and van Vliet [67]	3.3: Assess feasibility of using function points to predict change effort DV: Change effort	Function points* Subjective complexity*	Size (LOC)*	Not used
Niessink and van Vliet [13]	3.3: Identify cost drivers that can be used in models for prediction change effort, in two development organizations DV: Change effort	Change size* Change span (screens, lists, components, db entities, db attributes, temporary programs)* Subjective complexity* Change source*	Size* Structural attributes (# GOTO's)* Component kind* Documentation quality*	Subjective experience * Team id* Requirement instability*
Mockus and Weiss [63]	3.1: Investigate attributes that influence failure-proneness Construct a usable failure-prediction model DV: Software failure as a consequence of change	Maintenance type* Change size* Change span (subsystems, modules, files, check-ins, sub-tasks)* Change interval*	Structural attributes (size of changed files)	Developer span (# developers) Objective change experience*
Evanco [62]	3.1: Develop and assess a prediction model for corrective changes DV: Change effort	Change span (subsystems, components, compilation units affected)*	Structural attributes (# parameters, cyclomatic complexity, # compilation units)*	Not used
Polo, Piattini *et al.*[68]	3.3: Early prediction of maintenance effort DV: High/low change effort	Maintenance type* Request criticality*	Size(LOC, modules)*	Not used
Reformat and Wu [66]	3.2: Assess AI techniques to construct predictive modes of corrective change effort, DV: Change effort	Defect type* Subjective complexity*	Data operation (accessing, computational)*	Not used
Xu, Yang *et al.* [64]	3.1: Manage customer satisfaction DV: Change interval	Maintenance type* Change effort*	Age (Task id, system id, version id)	Not used

Table A4. Business context, measurement procedures and extent of data collection

Category	Sub-category	Value	Value explanation	References
Business context	Business model	In-house Embedded	Embedded system developed for internal use	[37, 39, 41, 42, 44, 54, 61, 62, 65, 66]
		In-house IS	Information system developed for internal use	[13, 21, 43, 46, 60]
		Multi-client	System developed for multiple business clients	[40, 45, 47-49, 52, 53, 55-59, 63, 64, 68]
		Single-client	System developed for one business client	[38, 50, 51, 67]
	Business domain	Aero-space	NASA	[37, 39, 41, 42, 44, 54, 61, 62]
		Telecom	Switching, billing	[38, 48, 49, 52, 53, 55-57, 63, 65]
		Finance	Banking, insurance	[43, 58, 60, 67, 68]
		Government	-	[13, 46]
		Other	Retail, hotel management	[21, 40, 64]
		R&D	SW/research tools	[59, 66]
		Not reported	-	[45, 47, 50, 51]
Measurement procedures	Data origin	Natural	Measurements relied on footprints of change process	[13, 37, 39, 41-44, 54, 59, 61, 65, 66]
		Purpose	Data was created for the purpose of measurement	[21, 38, 40, 46-50, 52, 53, 55-57, 62, 63, 67, 68]
		Mixed	Combination of Natural and Purpose	[45, 51, 58, 60, 64]
	Extraction of measures	Expert	Expert resources required for measure extraction	[13, 21, 38, 44, 47, 54, 60, 66-68]
		Clerical	Non-expert resources required for measure extraction	[37, 39-43, 45, 51, 58, 61, 65]
		Automated	Measure extraction was automated	[46, 48-50, 52, 53, 55-57, 59, 62-64]
Extent of data collection	Change count	< 25 percentile	# changes <= 127	[13, 38, 41, 47, 54, 58, 59, 65]
		25 to 75 prcntl.	127 < # changes <= 2945	[37, 42-46, 50-53, 56, 60-62, 66, 67]
		75 to 95 prcntl.	2945 < # changes <= 20902	[48, 55, 57, 63, 64, 68]
		> 95 prcntl.	# changes > 20902	[21, 49]
		Not reported		[39, 40]
	Duration	< 25 percentile	# months <= 18	[13, 37, 59, 64, 65, 67, 68]
		25 to 75 prcntl.	18 < # months <= 60	[41-43, 45, 46, 51-53, 55, 57, 58, 60]
		75 to 95 prcntl.	60 < # months <=195	[48-50, 63]
		> 95 prcntl.	# months > 195	[21, 54]
		Not reported		[38-40, 44, 56, 61, 62, 66]

References

[1] K. Beck, "Embracing Change with Extreme Programming," *Computer,* vol. 32, pp. 70-77, 1999.

[2] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and Laws of Software Evolution - the Nineties View," in *4th International Symposium on Software Metrics*, 1997, pp. 20-32.

[3] L. A. Belady and M. M. Lehman, "A Model of Large Program Development," *IBM Systems Journal,* vol. 15, pp. 225-252, 1976.

[4] R. D. Banker, S. M. Datar, C. F. Kemerer, and D. Zweig, "Software Complexity and Maintenance Costs," *Communications of the ACM,* vol. 36, pp. 81-94, 1993.

[5] P. Bhatt, G. Shroff, C. Anantaram, and A. K. Misra, "An Influence Model for Factors in Outsourced Software Maintenance," *Journal of Software Maintenance and Evolution: Research and Practice,* vol. 18, pp. 385-423, 2006.

[6] M. S. Krishnan, C. H. Kriebel, S. Kekre, and T. Mukhopadhyay., "An Empirical Analysis of Productivity and Quality in Software Products," *Management Science,* vol. 46, pp. 745-759, 2000.

[7] B. P. Lientz, "Issues in Software Maintenance," *ACM Computing Surveys,* vol. 15, pp. 271-278, 1983.

[8] J. H. Hayes, S. C. Patel, and L. Zhao, "A Metrics-Based Software Maintenance Effort Model," in *8th European Conference on Software Maintenance and Reengineering*, 2004, pp. 254-258.

[9] C. Kemerer, "Software Complexity and Software Maintenance: A Survey of Empirical Research," *Annals of Software Engineering,* vol. 1, pp. 1-22, 1995.

[10] J. C. Munson and S. G. Elbaum, "Code Churn: A Measure for Estimating the Impact of Code Change," in *14th International Conference on Software Maintenance*, 1998, pp. 24-31.

[11] B. A. Kitchenham, "Procedures for Performing Systematic Reviews," Keele University Technical report EBSE-2007-01, 2007.

[12] T. L. Graves and A. Mockus, "Identifying Productivity Drivers by Modeling Work Units Using Partial Data," *Technometrics,* vol. 43, pp. 168-179, 2001.

[13] F. Niessink and H. van Vliet, "Two Case Studies in Measuring Software Maintenance Effort," in *14th International Conference on Software Maintenance*, 1998, pp. 76–85.

[14] H. Kagdi, M. Collard, and J. I. Maletic, "A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution," *Journal of Software Maintenance and Evolution: Research and Practice,* vol. 19, pp. 77-131, 2007.

[15] M. G. B. Dias, N. Anquetil, and K. M. de Oliveira, "Organizing the Knowledge Used in Software Maintenance," *Journal of Universal Computer Science,* vol. 9, pp. 641-658, 2003.

[16] B. A. Kitchenham, G. H. Travassos, A. von Mayrhauser, F. Niessink, N. F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, and H. Yang, "Towards an

Ontology of Software Maintenance," *Journal of Software Maintenance: Research and Practice,* vol. 11, pp. 365-389, 1999.

[17] F. Ruiz, A. Vizcaíno, M. Piattini, and F. García, "An Ontology for the Management of Software Maintenance Projects," *International Journal of Software Engineering and Knowledge Engineering,* vol. 14, pp. 323-349, 2004.

[18] N. Fenton, "Software Measurement: A Necessary Scientific Basis," *IEEE Transactions on Software Engineering,* vol. 20, pp. 199-205, 1994.

[19] O. Dieste and A. G. Padua, "Developing Search Strategies for Detecting Relevant Experiments for Systematic Reviews," in *1st International Symposium on Empirical Software Engineering and Measurement*, 2007, pp. 215-224.

[20] D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N. Liborg, and A. C. Rekdal, "A Survey of Controlled Experiments in Software Engineering," *IEEE Transactions on Software Engineering,* vol. 31, pp. 733-753, 2005.

[21] C. F. Kemerer and S. Slaughter, "An Empirical Approach to Studying Software Evolution," *IEEE Transactions on Software Engineering,* vol. 25, pp. 493-509, 1999.

[22] B. P. Lientz, E. B. Swanson, and G. E. Tompkins, "Characteristics of Application Software Maintenance," *Communications of the ACM,* vol. 21, pp. 466-471, 1978.

[23] E. B. Swanson, "The Dimensions of Maintenance," in *2nd International Conference on Software Engineering*, San Francisco, California, United States, 1976, pp. 492-497.

[24] L. C. Briand and J. Wüst, "Empirical Studies of Quality Models in Object-Oriented Systems," *Advances in Computers,* vol. 59, pp. 97-166, 2002.

[25] E. Arisholm and L. C. Briand, "Predicting Fault-Prone Components in a Java Legacy System," in *5th International Symposium on Empirical Software Engineering*, 2006, pp. 8-17.

[26] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting Fault Incidence Using Software Change History," *IEEE Transactions on Software Engineering,* vol. 26, pp. 653-661, 2000.

[27] M. Lindvall, "Monitoring and Measuring the Change-Prediction Process at Different Granularity Levels," *Software Process: Improvement and Practice,* vol. 4, pp. 3-10, 1998.

[28] V. R. Basili and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Communications of the ACM,* vol. 27, pp. 42-52, 1984.

[29] M. Leszak, D. E. Perry, and D. Stoll, "Classification and Evaluation of Defects in a Project Retrospective," *The Journal of Systems & Software,* vol. 61, pp. 173-187, 2002.

[30] D. E. Perry and C. S. Stieg, "Software Faults in Evolving a Large, Real-Time System: A Case Study," in *4th European Software Engineering Conference*, 1993, pp. 48-67.

[31] A. von Mayrhauser and A. M. Vans, "Program Comprehension During Software Maintenance and Evolution," *Computer,* vol. 28, pp. 44-55, 1995.

[32] F. Détienne and F. Bott, *Software Design - Cognitive Aspects*. London: Springer-Verlag, 2002.

[33] B. G. Glaser, "The Constant Comparative Method of Qualitative Analysis," *Social Problems,* vol. 12, pp. 436-445, 1965.

[34] V. R. Basili, G. Caldiera, and H. D. Rombach, "The Goal Question Metric Approach," in *Encyclopedia of Software Engineering*. vol. 1, 2002, pp. 578-583.

[35] A. Hindle, D. M. German, and R. Holt, "What Do Large Commits Tell Us?: A Taxonomical Study of Large Commits," in *International working conference on mining software repositories*, Leipzig, Germany, 2008, pp. 99-108.

[36] OMG, "OCL 2.0 Specification," in *http://www.omg.org/docs/ptc/03-10-14.pdf*, 2005.

[37] V. Basili, L. C. Briand, S. Condon, Y. M. Kim, W. L. Melo, and J. D. Valett, "Understanding and Predicting the Process of Software Maintenance Releases," in *18th International Conference on Software Engineering*, 1996, pp. 464–474.

[38] S. Bergin and J. Keating, "A Case Study on the Adaptive Maintenance of an Internet Application," *Journal of Software Maintenance and Evolution: Research and Practice,* vol. 15, pp. 245-264, 2003.

[39] L. C. Briand, Y. M. Kim, W. Melo, C. Seaman, and V. R. Basili, "Q-MOPP: Qualitative Evaluation of Maintenance Organizations, Processes and Products," *Journal of Software Maintenance: Research and Practice,* vol. 10, pp. 249-278, 1998.

[40] W. Lam and V. Shankararaman, "Managing Change in Software Development Using a Process Improvement Approach," in *24th Euromicro Conference*, 1998, pp. 779-786.

[41] H. D. Rombach, B. T. Ulery, and J. D. Valett, "Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL," *Journal of Systems and Software,* vol. 18, pp. 125-138, 1992.

[42] D. M. Weiss and V. R. Basili, "Evaluating Software Development by Analysis of Changes - Some Data from the Software Engineering Laboratory," *IEEE Transactions on Software Engineering,* vol. 11, pp. 157-168, 1985.

[43] A. Abran and H. Hguyenkim, "Measurement of the Maintenance Process from a Demand-Based Perspective," *Journal of Software Maintenance: Research and Practice,* vol. 5, pp. 63-90, 1993.

[44] R. S. Arnold and D. A. Parker, "The Dimensions of Healthy Maintenance," in *6th International Conference on Software engineering*, 1982, pp. 10-27.

[45] G. E. Stark, "Measurements for Managing Software Maintenance," in *1996 International Conference on Software Maintenance*, 1996, pp. 152-161.

[46] C. S. P. Ng, "A Decision Framework for Enterprise Resource Planning Maintenance and Upgrade: A Client Perspective," *Journal of Software Maintenance and Evolution: Research and Practice,* vol. 13, pp. 431-468, 2001.

[47] N. Nurmuliani and D. Zowghi, "Characterising Requirements Volatility: An Empirical Case Study," in *4th International Symposium on Empirical Software Engineering*, Noosa Heads, Australia, 2005, pp. 427-436.

[48] D. E. Perry, H. P. Siy, and L. G. Votta, "Parallel Changes in Large-Scale Software Development: An Observational Case Study," *ACM Transactions on Software Engineering and Methodology,* vol. 10, pp. 308-337, 2001.

[49] R. Purushothaman and D. E. Perry, "Toward Understanding the Rhetoric of Small Source Code Changes," *IEEE Transactions on Software Engineering,* vol. 31, pp. 511-526, 2005.

[50] E. Burch and H. Kung, "Modeling Software Maintenance Requests: A Case Study," in *1997 International Conference on Software Maintenance*, 1997, pp. 40-47.

[51] D. Gefen and S. L. Schneberger, "The Non-Homogeneous Maintenance Periods: A Case Study of Software Modifications," in *1996 International Conference on Software Maintenance*, 1996, pp. 134-141.

[52] P. Mohagheghi and R. Conradi, "An Empirical Study of Software Change: Origin, Acceptance Rate, and Functionality Vs. Quality Attributes," in *3rd International Symposium on Empirical Software Engineering*, 2004, pp. 7-16.

[53] T. L. Graves and A. Mockus, "Inferring Change Effort from Configuration Management Databases," in *5th International Symposium on Software Metrics*, 1998, pp. 267–273.

[54] N. F. Schneidewind, "Investigation of the Risk to Software Reliability and Maintainability of Requirements Changes," in *2001 International Conference on Software Maintenance*, 2001, pp. 127-136.

[55] D. L. Atkins, T. Ball, T. L. Graves, and A. Mockus, "Using Version Control Data to Evaluate the Impact of Software Tools: A Case Study of the Version Editor," *IEEE Transactions on Software Engineering,* vol. 28, pp. 625-637, 2002.

[56] B. Geppert, A. Mockus, and F. Rößler, "Refactoring for Changeability: A Way to Go?," in *11th International Symposium on Software Metrics*, 2005.

[57] J. D. Herbsleb and A. Mockus, "An Empirical Study of Speed and Communication in Globally Distributed Software Development," *IEEE Transactions on Software Engineering,* vol. 29, pp. 481-494, 2003.

[58] A. J. Rostkowycz, V. Rajlich, and A. Marcus, "A Case Study on the Long-Term Effects of Software Redocumentation," in *2004 International Conference on Software Maintenance* 2004, pp. 92-101.

[59] E. Arisholm, "Empirical Assessment of the Impact of Structural Properties on the Changeability of Object-Oriented Software," *Information and Software Technology,* vol. 48, pp. 1046-1055, 2006.

[60] M. Maya, A. Abran, and P. Bourque, "Measuring the Size of Small Functional Enhancements to Software," in *6th International Workshop on Software Metrics*, 1996.

[61] L. C. Briand and V. R. Basili, "A Classification Procedure for the Effective Management of Changes During the Maintenance Process," in *1992 Conference on Software Maintenance*, 1992, pp. 328-336.

[62] W. M. Evanco, "Prediction Models for Software Fault Correction Effort," in *5th European Conference on Software Maintenance and Reengineering*, 2001, pp. 114-120.

[63] A. Mockus and D. M. Weiss, "Predicting Risk of Software Changes," *Bell Labs Technical Journal,* vol. 5, pp. 169-180, 2000.

[64] B. Xu, M. Yang, H. Liang, and H. Zhu, "Maximizing Customer Satisfaction in Maintenance of Software Product Family," in *18th Canadian Conference on Electrical and Computer Engineering*, 2005, pp. 1320-1323.

[65] M. Jørgensen, "Experience with the Accuracy of Software Maintenance Task Effort Prediction Models," *IEEE Transactions on Software Engineering,* vol. 21, pp. 674-681, 1995.

[66] M. Reformat and V. Wu, "Analysis of Software Maintenance Data Using Multi-Technique Approach," in *15th International Conference on Tools with Artificial Intelligence*, 2003, pp. 53-59.

[67] F. Niessink and H. van Vliet, "Predicting Maintenance Effort with Function Points," in *1997 International Conference on Software Maintenance*, 1997, pp. 32-39.

[68] M. Polo, M. Piattini, and F. Ruiz, "Using Code Metrics to Predict Maintenance of Legacy Programs: A Case Study," in *2001 International Conference on Software Maintenance*, 2001, pp. 202-208.

[69] B. Xu, "Managing Customer Satisfaction in Maintenance of Software Product Family Via Id3," *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on,* vol. 3, 2005.

[70] B. A. Kitchenham, S. L. Pleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering,* vol. 12, pp. 1106-1125, 2002.

[71] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying Software Engineers: Data Collection Techniques for Software Field Studies," *Empirical Software Engineering,* vol. 10, pp. 311-341, 2005.

[72] A. Mockus, "Missing Data in Software Engineering," in *Guide to Advanced Empirical Software Engineering*, 2000, pp. 185-200.

[73] D. E. Perry, A. A. Porter, and L. G. Votta, "Empirical Studies of Software Engineering: A Roadmap," in *Conference on The Future of Software Engineering*, 2000, pp. 345-355.

[74] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering,* vol. 25, pp. 557-572, 1999.

[75] D. I. K. Sjøberg, T. Dybå, B. C. D. Anda, and J. E. Hannay, "Building Theories in Software Engineering," in *Guide to Advanced Empirical Software Engineering* London: Springer-Verlag, 2008, pp. 312-336.

[76] D. M. German, "An Empirical Study of Fine-Grained Software Modifications," *Empirical Software Engineering,* vol. 11, pp. 369-393, 2006.

[77] R. K. Yin, "Designing Case Studies," in *Case Study Research: Design and Methods*: Sage Publications:Thousand Oaks, CA, 2003, pp. 19-53.

[78] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics,* vol. 33, pp. 159-74, 1977.

[79] D. C. Pelz, "Innovation Complexity and the Sequence of Innovating Stages," *Science Communication,* vol. 6, pp. 261-291, 1985.

Paper 2:

Understanding Cost Drivers of Software Evolution: A Quantitative and Qualitative Investigation of Change Effort in Two Evolving Software Systems

Hans Christian Benestad, Bente Anda, Erik Arisholm

Submitted to the Journal of Empirical Software Engineering

Abstract

Making changes to software systems can prove costly and it remains a challenge to understand the factors that affect the costs of software evolution. This study sought to identify such factors by investigating the effort expended by developers to perform 336 change tasks in two different software organizations. We quantitatively analyzed data from version control systems and change trackers to identify factors that correlated with change effort. In-depth interviews with the developers about a subset of the change tasks further refined the analysis. Two central quantitative results found that dispersion of changed code and volatility of the requirements for the change task correlated with change effort. The analysis of the qualitative interviews pointed to two important, underlying cost drivers: Difficulties in comprehending dispersed code and difficulties in anticipating side effects of changes. This study demonstrates a novel method for combining qualitative and quantitative analysis to assess cost drivers of software evolution. Given our findings, we propose improvements to practices and development tools to manage and reduce the costs.

1 Introduction

Software systems must adapt to continuously changing environments [1]. With a greater understanding of the cost of software evolution, technologies and practices could be improved to act against typical cost drivers. Development organizations could also make more targeted process improvements and predict cost more accurately in their specific

context. Researchers have used varied approaches to understand the cost of software evolution. One class of studies has investigated project factors, such as maintainer skills, the size of teams, development practices, and documentation practices, [2-5]. Other studies have examined how system factors, such as structural attributes of source code, relate to the ease of changing software [6-8]. A third class of studies has focused on human factors and probed individual cognitive processes of developers attempting to comprehend and change software [9].

A premise set forth in this paper is that software evolution consists of change tasks that developers perform to resolve change requests, and that change effort, i.e., the effort expended to perform these tasks, is a meaningful measure of software evolution cost. Thus, by identifying the drivers of change effort, we can better understand the cost of software evolution.

Change effort might be affected by such factors as type of change, developer experience and task size. This study distinguishes between a confirmatory analysis testing the effect of factors important in earlier change-based studies, and an explorative analysis identifying factors that best explain change effort in the data at hand. This is also the first study we are aware of that combines quantitative and qualitative analysis of change tasks in a systematic manner. The purpose was to paint a rich picture of factors involved when developers spend effort to perform change tasks. Ultimately, our goal is to aggregate evidence from change-based studies into theories of software evolution.

The main contributions of this paper are threefold: First, from a *local perspective* the study results can improve practices in the two investigated projects. For example, the study identifies specific factors that were insufficiently accounted for when the projects estimated change effort. Second, from the *software engineering perspective*, it clarifies factors that drive cost of software evolution. For example, the study identifies commonly used design practices with an unfavorable effect on change effort. Third, from the *empirical software engineering perspective* the paper demonstrates a methodology of qualitative and quantitative analysis of software changes to assess factors that affect the cost of software evolution.

The remainder of this paper is organized as follows: Section 2 describes the design of the study, and includes a measurement model based on a literature review of empirical studies of software change. Sections 3 and 4 provide the results from the quantitative analysis, while Section 5 provides the results from the qualitative analysis. Section 6

summarizes the results of the analysis and discusses the consequences. Section 7 discusses threats to validity, and Section 8 concludes.

2 Design of the Study

2.1 Research Question

The study addresses the following overall research question:

> *From the perspective of developers handling incoming change requests during software evolution, which factors affect the effort required to complete the change tasks?*

In principle, a change can be viewed as a small project involving analysis, design, coding, testing and integration. The projects under study used lightweight development practices, and did not, for example, maintain the requirements or high-level design documents used for initial development. Most of the factors under study therefore pertain to coding-centric activities. Change trackers and version control systems were essential tools in order to maintain traceability and control of the evolving software. The regression models built for the quantitative analysis used data collected from such systems.

Because regression analysis essentially models statistical relationships between variables, evidence from such analysis is not sufficient to claim causal effects of the modeled factors. Also, there are many sources of unexplained variability in models of change effort, due to activities that leave no traces in change management systems. Examples of such activities can be informal discussions among developers, code comprehension activities and the maintenance of artifacts that are not fully traced in change management systems. To identify complementary factors affecting change effort, we therefore interviewed developers about effort expenditure for recently completed change tasks. Also, we relied on the interview data to reveal more about the involved causal effects.

2.2 Related Work and Open Issues

A systematic literature review performed by the authors identified 34 studies analyzing properties of change tasks and their outcome [10]. A significant and related research program in the area of change-based analysis was the *code decay project* based at Bell Labs, using change management data from the evolution of a large telecom switching system. Important findings were effects of the *type* and *size* of changes, a time-related

effect contributed to *code decay* [11], effects of *change experience* [12], *tool effects* [13], and effects of *refactorings* [14]. Other closely related studies have found effects of *structural attributes of changed components* [15-17]. *Subjectively assessed complexity* and the *size increase* are other factors found to be important [18, 19].Still, the evidence on factors that affect change effort is scattered, and it is unclear whether factors investigated in earlier change-based studies capture the most important cost drivers. The moderate or poor accuracy obtained in prediction models of change effort [18-20] indicate that important factors are not fully captured by quantitative data on changes. To attempt to clarify these issues, we established the comprehensive literature-based measurement model described in Section 2.6, wanting to answer:

1. Did the factors identified from earlier change-based studies consistently affect change effort?
2. How accurate were change effort models built from change management data?
3. What was the added value of using a larger number of candidate measures in the models?

Change-based studies have shown consistent correlations between change effort and *change set dispersion*, typically measured by the number of source code components affected by a change [16, 19, 21]. This recurring statistical correlation, also expected in this study, may simply capture an effect of *size.* Mockus and Graves found that measures of change set dispersion explained more variability than did counts of changed lines of code [11], indicating that dispersion might be a separate factor. This study explores the following questions about change set dispersion:

4. Did change set dispersion affect change effort, beyond what could be explained by size alone?
5. What explained the effect of change set dispersion on change effort, e.g., how was dispersion related to the comprehension activity?

These questions are closely related to research on the effect of delocalized plans [22], and of different control styles in object-oriented designs [23]. This research suggests that dispersed code hinders comprehension.

Some researchers have investigated the effects of technologies and tools on change effort. Jørgensen found that productivity was almost identical for changes to 3GL code versus changes to 4GL code [18]. Atkins *et al.* found that less effort was required when developers used a tool that supported changes to parallel versions of the system [13]. Apart

from these studies, the effects of using different languages and technologies have not received much focus in change-based studies. Given an effect of change dispersion in the quantitative models of change effort, we wanted to answer:

6. Was the effect of change set dispersion stronger when several languages or technologies were involved in changes?

Schneidewind focused on factors that can be assessed early in the change cycle, and found that the number of modifications to a proposed change was significantly correlated with fault proneness [24]. Iterative and agile processes take a different viewpoint, recommending that changes to requirements should be considered useful [25]. A relevant issue is therefore whether software organizations must differentiate between types of volatility in requirements. The study explores the following question:

7. Under which circumstances did change request volatility have the largest effect on change effort?

A large body of research exists on how structural attributes affect change activity [26]. Eick *et al.* found that the history of code changes was more responsible for problems than measurable aspects of code complexity [21]. On the other hand, Niessink and van Vliet showed that change effort correlated with size of the changed components [20]. Likewise Arisholm found a relationship between structural attributes of affected Java classes, and change effort [15]. We wanted to answer:

8. Which structural properties of source code had the largest effect on change effort?

Several studies have shown that change effort differs between types of changes, c.f. [17, 27]. Most studies used one category for corrective changes and one or more categories for non-corrective changes, e.g., perfective and adaptive changes [28]. Some researchers [29, 30] used fine-grained categories for corrective changes, similar to those proposed by Chillarege *et al.* [31]. In this study we wanted to use a bottom-up approach, generating categories for changes on the basis of the data at hand. We wanted to answer:

9. What kind of changes required most effort?

Differences in developer skills may potentially overshadow any other phenomenon in software development [32]. Mockus and Weiss used historical change management data to measure developers' experience objectively [12], while Jørgensen used subjective measures of skill and experience [18]. This study explores the question:

10. Which particular skill shortages had the largest effect on change effort?

Summarized answers to the questions are provided in Section 6. Most of the analyses for the questions above required that change management data was complemented with interview data.

2.3 Overview of Case Study Procedures

Figure 1 summarizes the case study procedures. Proposals for the case study were generated on the basis of empirical evidence from a systematic review of change-based studies [10]. Quantitative data to describe change tasks, including change effort, was extracted from change trackers and version control system in two software projects, henceforth labeled project A and B.

An evidence-driven analysis tested whether a small set of pre-selected measures contributed to change effort in statistical regression models. These measures captured cost factors important in earlier change-based studies. In the data-driven analysis, a wider set of factors and measures were input to statistical procedures designed to identify the models that best explained variations in change effort.

Roughly once a month, we interviewed the developers about recent change tasks and any circumstances making the task easier or more difficult. The interviews aimed to identify additional or more fundamental cost factors than those identified by the quantitative analysis. To achieve this goal, the analysis focused on the changes that had required considerably more or less effort than predicted from the regression models, i.e., the residuals were large.

The evidence from the different parts of the analysis was compared and integrated into a set of joint results. This constitutes the basis for discussing consequences from the three perspectives mentioned in the introduction.

With this design, we move towards a theory on software change effort that would be valuable both for researchers and practitioners within software engineering.

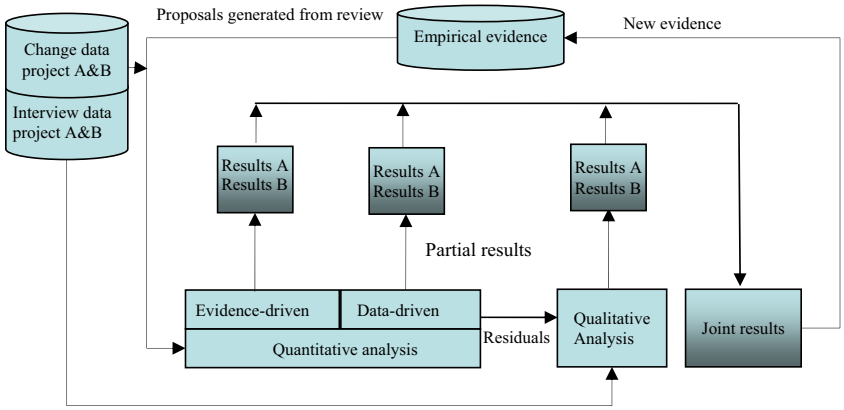


Figure 1. Overview of analyses

2.4 Generalization of Case Study Results

The case study paradigm is appropriate when investigating complex phenomena, especially when it is difficult to separate the investigated factors from their context [33]. In software development and software evolution, social and human factors interact with technological characteristics of the software. We chose the case study method because we wanted to consider the full complexity of factors affecting change effort in a realistic context.

A main concern with case studies is whether it is possible to generalize results beyond the immediate study context. Case study methodologists recommend that studies are designed to build or test *theories*. Theories can then explain, predict and manage the investigated phenomenon in some future situation, and are therefore useful to generalize from case studies. Because we are not aware of theories that are directly relevant to the research question, the proposals for this study were based on a systematic review of relevant empirical evidence. In other words, the systematic review of empirical evidence takes the place of theories in this study.

In particular, the evidence-driven analysis was essential to generalize from this study because it was designed to confirm, refute or modify the current empirically based knowledge about factors that correlate with or affect change effort. The role of the data-driven analysis was to discover additional relationships within the investigated projects, and to generate proposals for further confirmatory studies.

The qualitative analysis aimed at refining the quantitative results. For example, while regression analysis could show that more effort is expended when a particular programming language was used, interviews could reveal that developers used this

programming language for a particular type of task, say, to interface with hardware. This allows appropriate use of the study results in other contexts.

The results of this study are inevitably influenced by context factors pertaining to the development organizations in the data collection period. Understanding these factors makes it easier to judge the applicability of the results in a new context. By replicating the study across two development organizations, and comparing the results and the organizations, we were able to evaluate some of these context factors. Data was collected over a relatively short period of time. Although this was a pragmatic choice, analyzing data in a relatively narrow time span can make cost factors more clearly visible, see, e.g., [13].

2.5 Case Selection and Data Collection

We approached medium and large-sized software development organizations in the geographic area of our research group during 2006, using procedures that conformed to those described in [34]. The participants had to grant access to the planned sources for quantitative and qualitative data, to use object-oriented programming languages, to have planned development for at least 12 months ahead, and to use a well-defined change process that included some basic data collection procedures. The recruitment phase ended when we made agreements with two projects, henceforth named project A and project B.

Project A develops and maintains a Java-based system that handles the lifecycle of research grants for the Research Council of Norway. A publicly available web interface provides functionality for people in academia and industry to apply for research grants, and to report progress and financial status from ongoing projects [35]. Council officials use a Java client to review the research grant applications and reports. The system integrates with a number of other systems, such as a web publishing system. The consultancy company that we cooperated with was subcontracted by the Council to make improvements and add functionality to the system. For the most part, the contractor was paid per hour of development effort. Most change requests originated from the users at the Council. Roughly once a month, the development group agreed with user representatives and the product owner on changes for the next release.

Project B develops and maintains a Windows PocketPC system written in Java and C++. The system allows passengers on the Norwegian State Railways [36] to purchase tickets on-board, and offers electronic tickets and credit card payment. The system integrates with a back-end accounting system that is shared with other sales channels. The consultancy company that we cooperated with had been subcontracted by the Norwegian State

Railways to develop the system. Most change requests originated from the product owner and user representatives. The members of the development group prioritized and assigned development tasks directly in the change tracker, or as part of short and frequent meetings. New versions of the system were released roughly once a month. For the most part, the contractor was paid per hour of development effort.

Both projects were medium-sized with extensive change activity. Three to six developers were making code changes to the systems in each of the projects. Figure 2 and Figure 3 illustrate change activity and system size over a period of 30 months. Project A deployed the first version of their system in Q1 2003, while project B deployed the system in Q1 2005. The apparent dip in system size for project A around Q3 in 2005 was due to a major reorganization of the software that included a change in the technology platform. According to the developers, this change eased further development, and they perceived the project to be in a relatively healthy state during the period of measurement.

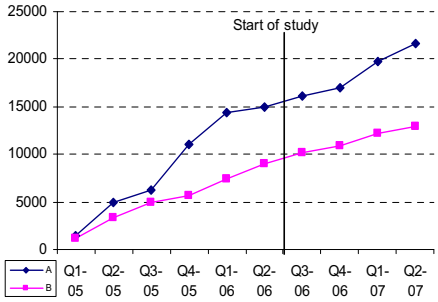


Figure 2. Accumulated number of check-ins

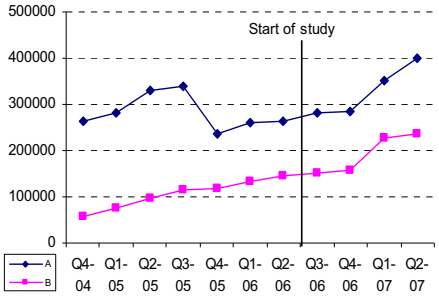


Figure 3. System size, in lines of code

Table 1. Key information about collected data

	Project A	Project B
Number of analyzed changes	136	200
Total effort of analyzed changes	1425 hours	1115 hours
Changes discussed in interviews	120	65
Period for data collection	Jan 2007-Jul 2007	Aug 2006 – Jul 2007
Version control system	IBM Rational Clearcase LT [37]	CVS [38]
Change tracker	Jira [39]	Jira [39]
Total duration of interviews	20 hours	10 hours
Total time charged for data collection	18 hours	14 hours

It was crucial for the analysis that changes to source components could be traced to change requests, and that data on change effort was available. The developers recorded the identifier of the change request on every check-in to the version control system. During and after each change task, the effort expended on detailed design, coding, unit testing and integration was recorded in the change tracker. Interviews were conducted on a monthly basis, discussing each change according to the interview guide shown in Appendix A.

The interview sessions allowed us to remind the developers to accurately report code changes and change effort according to the agreed procedures (question 3 in the interview guide). To further increase commitment to data collection, the companies could charge their normal hourly rate for data collection time. In sum, we believe these steps resulted in accurate and reliable quantitative data, although some measurement noise is inherent to this kind of data.

Prior to the analysis, four and six data points were removed from project A and B, respectively, because they corresponded to continuously ongoing maintenance activities, rather than independent and cohesive tasks.

2.6 Measurement Model

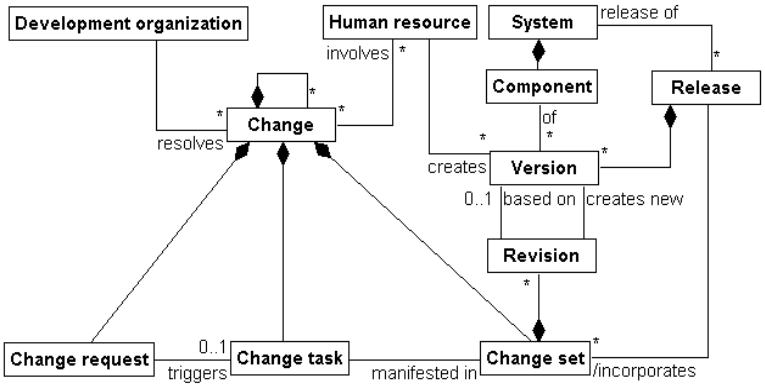


Figure 4. Key terms and concepts

This study's perspective is that software evolution is organized around the *change task*. A conceptual model for change-based studies is given in Figure 4. A *change task* is a cohesive and self-contained unit of work triggered by a *change request*. In these projects, a change task consists of detailed design, coding, unit testing and integration. A change task is manifested in a corresponding *change set*. A change set consists of *revisions*, each of which creates a new *version* of a *component* of the *system*. The new version can be based

on a pre-existing version of the component, or it can be the first version of an entirely new component.

A system is deployed to its users through *releases*. A release is built from particular versions of the components of the system. A release can also be described by the change sets or corresponding change requests that it incorporates. The term *change* aggregates the change task, the originating change request, and the resulting change set. Changes involve *human resources*, and are managed and resolved by the *development organization*. Changes can be hierarchical, because large changes may be broken down into smaller changes that are more manageable for the development organizations.

Table 2. Summary of measures

Entity	Factor	Measure	Explanation of measure
Change task	Change effort	*ceffort*	Time expended to design, code, test, and integrate change, tracked by developers Used as response variable in the study.
Change request	Change request volatility	*crTracks**	-Change tracks for CR before first check-in
		crWords	-Words in CR before first check-in
		crInitWords	-Words in original CR
		crWait	-Calendar time before first check-in
	Change type	*isCorrective**	-Classification + text scanning
Change set	Change set size	*components**	-Changed components
		addLoc	-Measures collected by
		chLoc	parsing side-by-side
		delLoc	output (-y)
		newLoc	of unix/linux *diff*
		segments	-diff –y v2 v1 | cut –c65 | tr –d '\n' | wc –w
	Change set complexity	*addCC*	Parse output of *diff* to measure the number of
		delCC	structural elements added and deleted.
		addRefs	Measures control-flow statements and
		delRefs	reference symbols (. ->)
Component version	Structural attrib.: Size	*avgSize**	-Average/weighted (by *segments*) size of
		cpSize	changed components
		avgRefs	-Average/weighted (by *segments*) number of
	Coupling	*cpRefs*	references to members of imported
		avgCC	components
	Control flow	*cpCC*	-Average/weighted (by *segments*) number of control flow statements
Component	Language heterogeneity	*filetypes*	-Unique file types that were changed
	Specific technology	*hasCpp (A)*	-Change concerns C++ code
		hasWorkflow (B)	-Change concerns the workflow engine
	Code volatility	*avgRevs*	-Average number of earlier revisions
Human resource and Revision	Change experience	*systExp**	-Avg. previous check-ins by developers
		techExp	-Avg. previous check-ins on same file types
		packExp	-Avg. previous check-ins in same package
		compExp	-Avg. previous check-ins in same components
		devspan	-Number of developers participating in change
Development organization	Project identity	*isA**	1 if change belongs to project A 0 if change belongs to project B

The measures used as explanatory variables in quantitative models of change effort captured factors pertaining to the entities of the model shown in Figure 4. Table 2 provides a summary of the relationships between entities, factors and measures. For each factor, we select one primary measure and zero or more alternative measures. The primary measures are used as explanatory variables in models for the evidence-driven analysis. These models are a reference point allowing us to assess the added value of the data-driven analysis, where we build optimized, project-specific models using all the described measures as candidate variables. We preferred primary measures that were likely to be robust to variations in measurement context, that have been used and validated in previous empirical studies, and that were measurable or assessable at an early stage in the change cycle. Measures are written in *italics*, while primary measures are marked with an additional asterisk (*). Summary statistics and correlations for the measures are provided in [40].

2.6.1 Change Request Volatility

Modifications or additions that the developers or other stakeholders make to the original change request, the change request volatility, can indicate uncertainty or other problems in envisioning the change incorporated into the system. Such problems could propagate to the coding phase and affect change effort. In [24], the number of modifications to change requests correlated with fault proneness. In [19], the number of new requirements to change requests loaded on a principal component that correlated with change effort. A straightforward measure of change request volatility is the number of modifications to the original change request, as recorded in the change tracker (*crTracks*).* Related, candidate measures include the number of words in the original change request (*crInitWords*), the number of words in all modifications to the change requests (*crWords*), and the elapsed time from when a stakeholder created the change request until a developer started the change task (*crWait*).

2.6.2 Change Set Size

The *change set size* reflects the differences between the current and preceding versions of changed source components. The intuitive notion that this affects change effort is verified by previous studies [11, 18, 19, 41]. Other studies have shown that after controlling for change type or structural complexity of changed components, discussed below, change set size is not necessarily a significant factor [13, 15, 29]. A coarse-grained measure of change set size is the number of source components that were changed during the change task (*components*).* Finer granularity measures use text difference algorithms [42] to measure

the number of lines of code (LOC) that were added (*addLoc*), deleted (*delLoc*) and changed (*chLoc*). Added code in existing components can be differentiated from code in newly created components (*newLoc*). Comments and whitespace were removed before computing these measures.

We selected a coarse-grained measure of change set size because there is evidence that these perform equally well or better than LOC-based measures [11]. LOC counts are less meaningful in technologically heterogeneous environments, and when tools that generate code automatically are used. Furthermore, LOC counts may become high for conceptually trivial changes, such as when program variables or methods are renamed. For estimation of change effort, it is probably easier to estimate the number of components to change than the number of lines of code to change. An alternative, medium-grained measure counts the number of disjointed places in the existing code where changes were made (*segments*).

2.6.3 Change Set Complexity

If the structural complexity of the change set is high, e.g., if there are many changes to the control-flow, an increase in change effort beyond the effect of change set size could be expected. Except for one study in the authors' research group [43], we are not aware of any studies investigating this effect of change set complexity on change effort. Fluri and Gall showed that measures of edits to the abstract syntax trees of individual components predict ripple effects better than measures of textual differences [44]. We constructed two measures to capture the number of added control-flow statements and added references to members of external components, *addCC* and *addRefs*. Corresponding measures were constructed for deleted control-flow statements and deleted references to members of external components, *delCC* and *delRefs*. Because these are likely to correlate with measures of change set size, and they are experimental in nature, we only used these measures in the data-driven analysis.

2.6.4 Change Type

Changes can be described according to their origin, importance, quality focus, and other criteria. In change-based studies, the *change type* has been important in order to understand change effort [11, 13, 17, 18, 29]. Corrective, adaptive or perfective change types, as suggested by Swanson [28], was the most commonly used classification schema. A recurring result from existing change-based studies is that corrective changes are more time consuming than other types of change, after controlling for change set size [11, 45]. This does not contradict results that have shown that the mean effort for corrective changes

is lower than for other change types [17], because corrective changes tend to have smaller change set size [46].

Corrective and non-corrective changes (*isCorrective**) are the primary measure of classification in the analysis. This decision was based on the results from a field experiment in one of the projects, which showed that developers' classification into fine-grained change types was unreliable [47]. To further increase reliability of the measures, we combined the categorizations performed by the developers with textual search for words like "bug", "fails" and "crash" (in the native language) in change request descriptions.

2.6.5 Structural Attributes of Changed Components

The structural attributes code relevant to the change may affect comprehension effort involved in a change task. [48, 49]. Many change-based studies have investigated whether the size of changed modules (*avgSize**) correlate with change effort [15, 18-20, 44]. Arisholm showed that size and certain other structural properties of the changed source components were correlated with change effort [15]. We constructed alternative measures of control flow complexity and coupling in the changed components. The first measure takes the average number of control-flow statements (*avgCC*) in the changed components, while the second takes the average number of references to members of imported components, of each changed component (*avgRefs*). Variations of the measures were constructed by weighting the measures by the relative amount of change in each component (*cpSize*, *cpCC* and *cpRefs*), as proposed in [15].

2.6.6 Code Volatility

While many components rarely change, some are involved in a large proportion of the change tasks. We propose that the *code volatility* or change proneness will affect change effort, and that change prone components require *less* effort, simply because developers are more experienced with changing these components. Conversely, changes to infrequently changed components represent unfamiliarity, and may also indicate more fundamental changes. Higher code volatility could also result in *increased* change effort, because frequently changed modules may experience code decay [21]. However, in the investigated projects, components believed to have decayed due to frequent changes were re-factored, and we therefore expected this effect to be limited. The number of historical revisions, averaged over all changed components (*avgRevisions*), captures code volatility of changed components. Several researchers have used volatility of individual components as a

predictor of failure proneness, see e.g., [50]. However, we are not aware of studies that have investigated the relationships between code volatility and change effort. Due to this lack of existing empirical evidence we only used this measure in the data-driven analysis.

2.6.7 Language Heterogeneity

Language heterogeneity refers to the number of different programming languages involved in a change. Using many languages may increase change effort, because it sets higher demands on developer skills and integration challenges may arise. One simple way to measure language heterogeneity is to count the number of unique file name extensions among the changed components (*filetypes*). For example, changing one java-file and one properties-file would give a count of two. We are not aware of studies that have investigated how language heterogeneity affects change effort. Due to the lack of existing empirical evidence we only used this measure in the data-driven analysis.

2.6.8 Specific Technology

Use of a specific technology can affect change effort. For example, Atkins *et al*. showed that when developers used a tool that supported evolution of system variants, change effort was significantly reduced [13]. In project B, functionality interfacing with hardware was written in C++. We propose that changes that involve C++ will be more expensive to change than other code, which was predominantly written in Java. One rationale is that more specialized knowledge is required to develop code that interfaces to hardware. An effect of the lower abstraction level in C++ as compared to Java would work in the same direction. The binary measure *hasCpp* evaluates to true if any of the changed components were written in C++. Project A used a Java-based workflow engine as an important part of the technological basis. Although the project assumed that they benefited from the high abstraction level of this technology, we wanted to investigate whether the changes involving the workflow engine were different with respect to change effort. The binary measure *hasWorkflow* evaluates to true if any of the changed components were based on the technology of the Java-based workflow engine.

2.6.9 Change Experience

Experiments have shown that there can be large productivity differences between individual developers [51, 52]. Because we were not allowed to assess individuals, we used measures of *change experience* to assess one important source of individual differences. A basic measure is the total number of previous check-ins by the developer who performed the change (*systExp**). Other measures include the average number of earlier check-ins of

the changed components (*compExp*), packages (*packExp*) or technologies (*techExp*). If several developers were involved in the change, the averages of the measures were used, weighted by the number of components changed by each developer. Similar measures were used in [12]. In that study, the coarsest-grained measure (*systExp*) significantly affected the response variable capturing failure proneness, while the other measures did not.

2.7 Analysis of Quantitative Data

2.7.1 Statistical Procedures

Change effort was used as the response variable for all statistical models. The measures discussed in Section 2.6 were used as candidate explanatory variables. The regression model framework was Generalized Linear Models (GLM) with a *gamma* response variable distribution (sometimes called the error structure) and a *log* link-function, see [53]. One reason to assume gamma-distributed responses was that the effort data distribution has a natural lower bound of zero and was right-skewed with a long right tail. A *log* link function ensures that predicted values are always positive, which is appropriate for wait-time data. The size of effect of a specific explanatory variable x_n is assessed by the proportional change in expected change effort that results from a change to x_n. Because a *log* link-function is used, the proportional change in expected change effort becomes:

$$\frac{ceffort(x_1=C_1..x_{n-1}=C_{n-1},x_n=C_n+1)}{ceffort(x_1=C_1..x_{n-1}=C_{n-1},x_n=C_n)} = \frac{e^{ß_0+ß_1C_1+..+ß_{n-1}C_{n-1}+ß_n(C_n+1)}}{e^{ß_0+ß_1C_1+..+ß_{n-1}C_{n-1}+ß_nC_n}} = e^{ß_n}$$

Cross-project models were constructed to identify effects that were present in both projects, and to formally test for project differences. Project-specific models were constructed to identify effects specific to each project, and to quantify those effects.

The p-values, sign and magnitude of the coefficients are inspected to interpret the models. The significance level is set to 0.05. This means that for a variable to be assessed as significant, the probability that the variable has no impact must be less than 5%. It is difficult to interpret coefficients when there is a high degree of multicollinearity between the explanatory variables. In the evidence-driven analysis we attempted to reduce multicollinearity by selecting primary measures designed to capture independent factors. In the data-driven analysis, the results from a principal component analysis identified orthogonal factors in the data sets. The actual amount of multicollinearity in the fitted models was measured by the variance inflation factor (VIF). If the VIF is 1, there is no multicollinearity. If VIF is very large, such as 10 or more, multicollinearity is a serious problem according to existing rules-of-thumb [54].

2.7.2 Measures of Model Fit

We chose the cross-validated mean and median magnitude of relative error to assess the fit of models. The basis for these measures is the magnitude of relative error (MRE) which is the absolute value of the difference between the actual and the predicted effort, divided by the actual effort. The measures were calculated by *n-fold cross-validation*. With this procedure, the variable subset was fitted in n iterations on n-1 data points. In each iteration, the fitted model predicted the last data point. The mean MRE forms *MMREcross*, while the median of the values forms *MDMREcross*. The cross-validated measures are more realistic measures of the predictive ability of regression models than measures not based on cross-validated predictions. This was particularly important during the data-driven analysis, where models were selected on the basis of the MMREcross-measure.

Another measure to assess model fit is the percentage of data points with an MRE of less than a particular threshold value. *PRED(0.25)* and *PRED(0.50)* measure the percentages of the data points that have a MRE of less than 0.25 and 0.50, respectively. The Pearson and Spearman correlations between actual and predicted effort are also provided.

As a reference point to assess the model performance, we calculated the measures of model fit for the constant model, i.e. the model that uses a constant value as predictor for all data points. A commonly used criteria for accepting a model as "good" is a value of less than 0.25 for MMRE or MdMRE, and higher than 0.75 from Pred(25) [55].

2.8 Collection and Analysis of Qualitative Data

We prepared for interviews by studying data about each change request in the change trackers and version control systems, and attempted to understand how the changed code fulfilled the changes. Appendix A shows the interview guide. The interviews focused on phenomena that developers perceived to have affected change effort.

The changes with the largest magnitude of relative error (MRE) from the data-driven analysis were selected for in depth analysis. We limited the analysis to data points with an MRE of more than 0.5 for underestimated changes and more than 1.3 for overestimated changes. These limits were set somewhat arbitrarily.

The interviews were transcribed and analyzed in the tool Transana [56], which allows navigation between transcripts and audio data. This made it feasible to re-listen to the original voice recordings throughout the analysis. The interviews were coded in two phases. In phase 1, immediately after each interview session, the interviews were transcribed and coded according to a scheme that evolved as more data became available.

In phase 2, when the quantitative models had been constructed, we selected changes to be analyzed in depth. The focus was narrowed to categories and codes that suggested a relationship with change effort. Finally, the exact naming and meaning of codes and categories was reconsolidated to make them more straightforward and easy to understand. The coding schema that resulted from this process is described in Section 5.

3 Evidence-Driven Analysis

3.1 Models Fitted in Evidence-Driven Analysis

Cross-project models were constructed to identify effects in both projects, and to formally test for project differences:

$$\log(\text{ceffor t}) = ß_0 + ß_1\text{crTracks} + ß_2\text{components} + ß_3\text{systExp} + ß_4\text{avgSize} + ß_5\text{isCorrecti ve} + \quad \textbf{(M1)}$$

$$\log(\text{ceffort}) = ß_0 + ß_1\text{crTracks} + ß_2\text{components} + ß_3\text{systExp} + ß_4\text{avgSize} + ß_5\text{isCorrective} + ß_6\text{crTracks} \cdot ß_7\text{components} * \text{isA} + ß_8\text{systExp} * \text{isA} + ß_9\text{avgSize} * \text{isA} + ß_{10}\text{isCorrective} * \text{isA} + ß_{11}\text{isA} \quad \textbf{(M2)}$$

The model M1 includes one explanatory variable for each of the primary measures. It also includes a project indicator (*isA*) allowing for a constant multiplicative between the projects. Model 2 adds interaction terms between the project indicator and each of the primary measures, allowing for different coefficients for each factor in each project. Two project specific models were also fitted, one for each of the two data sets:

$$\log(\text{ceffor t}) = ß_0 + ß_1\text{crTracks} + ß_2\text{components} + ß_3\text{systExp} + ß_4\text{avgSize} + ß_5\text{isCorrecti} \quad \textbf{(M3)}$$

The constant models were used as yardsticks for the assessment of model fit:

$$\log(\text{ceffor t}) = ß_0 + ß_1\text{isA} \quad \textbf{(M4)}$$

3.2 Results from Evidence-Driven Analysis

Key information about coefficients in the fitted models is provided in Table 3. A p-value lower than 0.05* (the chosen significance level), 0.01** and 0.001*** are indicated with one, two and three asterisks, respectively.

Table 3. Coefficient values, significance and model fit in evidence-driven analysis

	Cross project, M4	Cross project, M1	Cross project, M2	Project A, M3 (w. standardized coefficients)	Project B, M3 (w. standardized coefficients)
Intercept (β_0)	9.91***	9.17***	9.30***	9.44***	9.30***
crTracks	.	0.075**	0.076**	0.08* (0.18)	0.076** (0.26)
components	.	0.098***	0.12***	0.076*** (0.76)	0.12*** (0.51)
systExp	.	-0.000039	-0.00018**	0.000026 (0.0719)	-0.00018** (-0.23)
avgSize	.	-0.000033	-0.000061	-0.000011 (-0.0082)	-0.000061 (-0.038)
isCorrective	.	-0.28*	-0.11	-0.78*** (-0.38)	-0.11 (-0.050)
isA	0.63***	0.18	0.14	.	.
*crTracks*isA*	.	.	0.0044	.	.
*components*isA*	.	.	-0.043	.	.
*systExp*isA*	.	.	0.00020**	.	.
*avgSize*isA*	.	.	0.000051	.	.
*isCorrective*isA*	.	.	-0.67*	.	.
MMREcross	3.29	1.52	1.5192	1.86	1.32
MdMREcross	1.43	0.69	0.6786	0.72	0.60
Pred(25)	0.095	0.20	0.23	0.21	0.25
Pred(50)	0.24	0.36	0.40	0.35	0.43
Pearson corr.	0.20	0.53	0.63	0.64	0.51
Spearman corr.	0.091	0.59	0.59	0.66	0.56

Solving M4 for *ceffort*, and dividing by 3600 (because the underlying measurement unit is *seconds*) gives an expected change effort of 5.6 hours for project B. The intercept is higher (statistically significant) by 0.63 in project A, which gives an expected change effort of 10.5 hours. The significant interaction terms in M2 indicate that *isCorrective* and *systExp* are project specific effects. The project specific models M3 show:

- The variable *crTracks* had a significant effect on change effort in all models. A 7% increase in change effort could be expected for each additional track in the change tracker. This size of effect was similar in the two projects.
- The variable *components* had a significant effect on change effort in the models from both projects. When one additional component was changed, a 12.9% and 7% increase in effort could be expected in project A and B, respectively.
- In project A, corrective changes were expected to require slightly less than half the effort compared to that required by non-corrective changes ($e^{-0.780}$=46%), after controlling for differences in other variables.
- In project B, *systExp* was significantly related to change effort. It was expected to decrease by 16.2% for every 1000th check-in performed by a developer. In project A, the effect was small and statistically insignificant.
- The estimated coefficients for *avgSize* indicate that change effort was slightly lower when large components are changed, but the effects are very small and statistically insignificant.

- The standardized regression coefficients show that relative to the statistical variability of each variable, *components* had the largest effect on change effort. For example, one standard deviation change in *components* had double (project B) and quadruple effect (project A) than did one standard deviation change in *crTracks*.

The variance inflation factor was less than 1.34 for all the coefficients in all models. The principal component analysis in Section 4.2.1 and the correlations reported in [40] further confirmed that multicollinearity was not a threat to the above interpretation of the coefficients.

Plots of actual versus predicted change effort of projects A and B are provided in Figure 5 and Figure 6, respectively. MdMREcross was down from 1.43 for the constant model to between 0.60 and 0.72 for the rest of the models. However, judged by commonly used standard [55], the model fit was relatively poor.

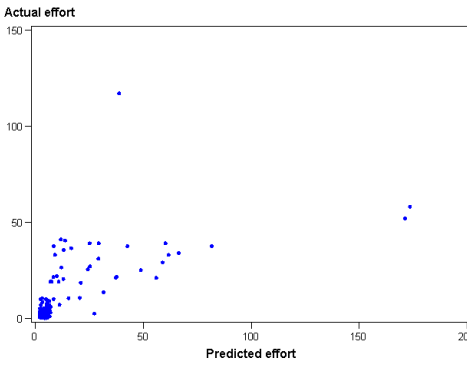


Figure 5. Predicted vs. actual effort, project A

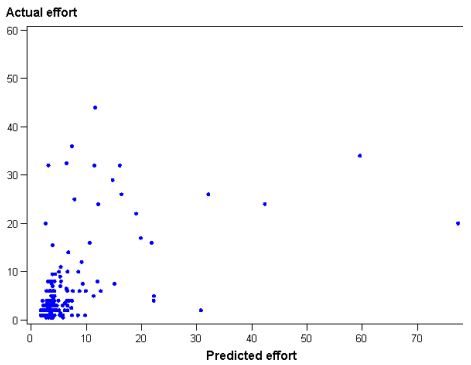


Figure 6. Predicted vs. actual effort, project B

3.3 Discussion of Evidence-Driven Analysis

It is interesting from a practical perspective that a relatively coarse grained, easily collectable and early assessable measure of change set size (*components*) performed well as a predictor of change effort. Code changes dispersed among many components could possibly require more effort than changing the same number of lines in fewer components. The data-driven analysis and the qualitative analysis investigate this topic in more depth.

The number of updates to change requests (*crTracks*) can be automatically retrieved in an early phase of the change process, and can therefore be useful for effort estimation. The qualitative analysis investigates the result in more depth, aiming at actions that could reduce the impact of change request volatility.

Corrective changes required less effort than non-corrective changes, although the difference was statistically insignificant in project B. The direction of this effect is opposite

to that of earlier studies. A possible explanation is that the tasks and processes involved in corrective vs. non-corrective changes are indeed different, but the direction of the difference is situation-dependent. A negative coefficient for *isCorrective* indicates that it is relatively easy to correct defects compared to making other types of changes. We consider this to be a favorable situation where it is important to quickly correct defects or where defects are associated with undesirable noise.

The measure of system experience, *systExp,* was statistically significant for project B, but not for project A. One problem with *systExp* as a measure of system experience is that it may be confounded with system decay: The favorable effects of more experienced developers can be counteracted by an effect of system decay, because *systExp* and system decay may be inversely related to the underlying factor of time.

We did not obtain any significant effect of the size of changed components. There are several possible explanations for this. First, because larger components probably are more change-prone, due to the effect of size, developers will have more experience in changing these components. Second, the class or the file is not necessarily the natural unit for code comprehension during change tasks, as discussed in the qualitative analysis in Section 5.

4 Data-Driven Analysis

In the data-driven analysis we explored relationships that were not originally proposed, assessed factors that have a weaker foundation in theory and empirical evidence, and evaluated the predictive power of alternative measures of the same underlying factor.

4.1 Procedures for Data-Driven Analysis

The measures from Table 2 were used as candidate variables in the statistical procedures described below. The goal was to identify the models that explained the most possible change effort variability, under the constraint that each model variable captured relatively orthogonal cost factors. We used:

- Principal component analysis (PCA) to identify candidate variable subsets, consisting of uncorrelated or moderately correlated variables. Selecting among variables on the basis of a PCA is a common approach, see, e.g., [57] and [58].
- Exhaustive search among variable subsets to identify the best models, described by [59].
- A cross-validated measure of model fit (*MMRECross*) as a selection criterion [60, 61].
- Decision trees to identify interaction effects and non-continuous effects [62]

4.1.1 Identification of Main Effects

The structure of the correlations between the candidate variables was analyzed by principal component analysis (PCA). Each *principal component* (PC) resulting from a PCA is a linear combination of the original variables, constructed so that the first PC explains the maximum of the variance in the data set, while each of the next PC's explains the maximum of the variance that remains, under the constraint that the PC is orthogonal to all the previously constructed PC's. The *loading* of each variable in PC indicates the degree to which it is associated with that PC. In order to interpret a PC, we inspected the variables that loaded higher than 0.5, after the *varimax rotation* [63] had been applied. The results from the analysis are provided in Section 4.2.1.

The results from the PCA were used to construct all possible subsets of candidate variables that contained exactly one variable from each PC. This constraint prevents high multicollinearity in the models, making them easier to interpret. For each of the constructed variable subsets, regression models of change effort were fitted. The models with the lowest cross-validated MMRE (*MMREcross*) in the two projects were selected as the best.

We also performed a principal component regression (PCR) [64], which is an alternative approach for data-driven analysis. With this approach, the linear combinations that define each principal component produce new variables used in the regression in place of the original variables. The new variables are uncorrelated, which completely eliminates the problem of interpreting the coefficients of correlated regression variables. This comes at the cost that it can be difficult to interpret the meaning of the regression variables. Because information from all variables is used in the regression, the approach can yield models that are well fitted to the data.

The best models resulting from the PCR were compared to the models obtained from using a single variable as a representative for a principal component. We preferred to use the latter models for interpretation, but only if multicollinearity in those models was acceptable (measured by the variance inflation factor) and if model performance was similar to or better than the PCR models.

4.1.2 Identification of Decision Tree Rules

The goal of this step was to identify possible interaction effects and effects applying only to parts of the value ranges for the explanatory variables. We used a hybrid regression

technique that combines the explorative nature of decision trees with the formality of statistical regression [62].

A decision tree splits the data set at an optimal value for one of the explanatory variables. The split is performed so that the significance of the difference between the two splits is maximized. This step is performed recursively on the splits, until a stop criterion is reached. The stop criterion was that a leaf node should contain no less than 15 data points.

For use in GLM regression, a binary indicator variable was created for each of the leaf nodes in the resulting decision tree. Since this procedure partitions the dataset, every change task had the value 1 for one of the indicator variables, and 0 for the rest. Candidate variable subsets were generated from all possible combinations of the indicator variables and the main effects. The models with the lowest *MMREcross* were selected as the best.

4.2 Results from Data-Driven Analysis

4.2.1 Factors Identified by PCA

The summary of results from the principal component analyses for project A and B are shown in Table 4 and Table 5, respectively.

Table 4. Summary of principal component analysis, project A

PC	PC1A	PC2A	PC3A	PC4A	PC5A	PC6A	PC7A	PC8A
Load > 0.5 after varimax rotation	avgSize avgRefs avgCC cpRefs cpCC cpSize	hasWorkflow addCC addRefs newLoc components filetypes devspan	delLoc delCC delRefs crWait	addLoc chLoc segments	crWords crInitWords crTracks	systExp techExp packExp	avgRevs	isCorrecti ve
Entity	*Component version*	Change set	Change set:	*Change set*	*Change request*	*Human resource*	Compone nt version	*Change request*
Factor	*Size*	Dispersion	Rework	*Size*	*Volatility*	*Change experience*	Code volatility	*Change type*

Table 5. Summary of principal component analysis, project B

PC	PC1B	PC2B	PC3B	PC4B	PC5B	PC6B	PC7B
Load > 0.5 after varimax rotation	addLoc delLoc chLoc segments addCC delCC addRefs delRefs	avgSize avgRefs avgCC avgRevs cpRefs cpCC cpSize	components filetypes devspan packExp hasCpp	crWords crInitWords crTracks crWait	systExp techExp	newLoc components	isCorrective
Entity	*Change set*	*Component version*	Change set	*Change request*	*Human resource*	Change set	*Change request*
Factor	*Size*	*Size*	Dispersion	*Volatility*	*Change experience*	Design mismatch	*Change type*

We made the following observations about the match between the conceptual measurement model and the PCA:

- The factors in italics match factors described in Section 2.6. The collected measures for these factors are consistent with the measurement model, and capture five orthogonal factors in the data set: *Change set size, Component version size*, *Change request volatility*, *Change experience* and *Change type*.
- PC1A and PC2B show that the suggested measures for control-flow and coupling belong to the same principal component as the LOC-based measures of size. The underlying factor captured by all these measures is the size of changed components.
- Likewise, PC1B shows that the suggested measures of change set complexity belong to the same principal component as the LOC-based measures of change set size, in project B.
- PC2A and PC3B contain measures that capture the dispersion of changed code over components, types of components and developers. We label this dimension *change set dispersion*. It is interesting that this captures a factor that is orthogonal to change set size.
- PC3A contains measures of removed code. This principal component captures the *amount of rework*, apparently distinguishable from the concept of change set size in project A.
- In project A, the measure of code volatility belongs to a distinct principal component (PC7A), while in project B, it belongs to the principal component that captures size (PC2B). The latter result indicates that large components are more prone to change, simply due to size.
- PC6B contains a measure of lines of code in new components, and the change set dispersion. One possible interpretation is that these measures capture the degree of mismatch between the current design and the design required by the change.

These observations are accounted for when the models are interpreted, in Sections 4.3 and 6.

4.2.2 Regression Models for the Data-Driven Analysis

The models resulting from the procedures described in 4.1 are shown in Table 6.

Table 6. Coefficient values, significance and model fit in data-driven analysis, discussed results in bold

Model	Variable	Coefficient (standardized coeff. in parenthesis)	MMREcr MdMREcr	Pred(25) Pred(50)	Pearson Spearman corr.
Project A Main effects	Intercept	9.06***	1.52	0.23	0.58
	crWords	0.00187** (0.25)	0.63	0.40	0.72
	filetypes	**0.279*** (0.72)**			
	chLoc	0.005111** (0.31)			
	isCorrective	-0.503* (-0.25)			
Project B Main effects	Intercept	9.06***	1.12	0.24	0.46
	crTracks	0.0879***	0.60	0.42	0.58
	addCC	**0.00949****			
	components	0.1027***			
	systExp	-0.000161**			
Project A with decision tree rules	Intercept	9.64***	1.37	0.24	0.70
	crWords	0.00109* (0.14)	0.57	0.46	0.77
	filetypes	**0.178***** (0.46)			
	isCorrective	-0.376* (-0.18)			
	filetypes=1&*crWords*<24	**-1.145***** (-0.36)			
	filetypes=1&*crWords*>23&*chLoc* < 2	**-0.831***** (-0.28)			
	filetypes=1&*crWords*>23&*chLoc*>=2	**-0.653**** (-0.22)			
	filetypes>=3&*chLoc*>= 48	**0.963***** (0.32)			
Project B with decision tree rules	Intercept	9.15***	1.12	0.22	0.59
	crTracks	0.0839***	0.62	0.40	0.54
	components	0.0798***			
	systExp	-0.000153**			
	addCC>=23	**0.7877****			
Project A PCR	PC2A	0.9686***	1.71	0.24	0.53
	PC3A	0.2252*	0.66	0.42	0.78
	PC4A	0.4058***			
	PC5A	0.3492***			
Project B PCR	PC1B	0.3529***	1.33	0.275	0.39
	PC2B	-0.1659*	0.55	0.48	0.59
	PC3B	0.2640***			
	PC4B	0.4928***			
	PC5B	-0.2143***			
	PC6B	-0.1682***			
	PC7B	1.4008*			

For project A, the results show that:

- The indicator of change type *isCorrective* recurred from the evidence-driven analysis
- The measure *filetypes*, capturing language heterogeneity, had a strong effect. Change effort is expected to increase by around 30 % with one additional file type changed.
- The number of change lines of code, *chLoc*, also entered the model. An increase of 30 % can be expected when around 50 additional lines of code were changed.
- Three of the decision tree rules handle cases where only one *filetype* is affected. The coefficients show that change effort is particularly low in such cases, beyond the continuous effect of the variable. Fifty of the 136 changes were covered by these rules.

- The last rule indicates a particularly strong effect of changes that span three or more languages and at the same time involve a large change set (48 or more code lines changed). The coefficient shows that 2.6 times more effort can be expected for such changes.

For project B, the results show that:

- Compared with the results from the evidence-based analysis, the data-driven analyses identified the additional factor *addCC* (row 2 in Table 6). This measure was intended to capture structural complexity of the change set, but the PCA showed that *addCC* captures change set size in this data set. The expected change effort increases by 10% when *addCC* increases by 10.
- Allowing for decision tree rules (row 4 in Table 6), a simple binary rule replaced a continuous effect of *addCC*: The expected change effort doubles if 23 or more control-flow statements are added. This rule applies to 12% of the changes.

The models that combined regression with decision rules performed better than the models from principal component regression, shown in the two last rows of Table 6. The variance inflation factor was lower than 1.88 for all the coefficients in the models. This verifies that multicollinearity is not a problem for the interpretability of the coefficients.

4.3 Discussion of Data-Driven Analysis

In project A, fewer *filetypes* involved in a change strongly contributed to reduced change effort. A particularly favorable effect occurred when a change involved only one file type. Because such changes often can be identified before the coding phase, this result can be useful to improve change effort estimates.

In project B, *addCC* and *components* had significant effects on change effort. The PCA showed that these measures captured orthogonal factors in the data set. We conclude that change set dispersion affected change effort, beyond the effect of LOC-based size. For effort prediction purposes, the simple decision rule (*addCC*>=23) indicates that even a very coarse grained estimate of change set size is useful.

For project A, the data-driven analysis resulted in models that had better model fit than those from the evidence-based analysis. This was mainly due to the measure of language heterogeneity. For project B, the model fit did not improve, as the primary measures already seemed to capture the important factors. The total amount of explained change effort variability was moderate.

The plots in Figure 7 and Figure 8 show MRE boundaries for overestimated and underestimated changes. The changes that fell outside the area formed by these lines received particular attention during the qualitative analysis. In total, 32 underestimated changes and 16 overestimated changes (those with MRE limits of 0.5 for underestimated changes and 1.3 for overestimated changes, see Figure 7 and 8) were analyzed in depth.

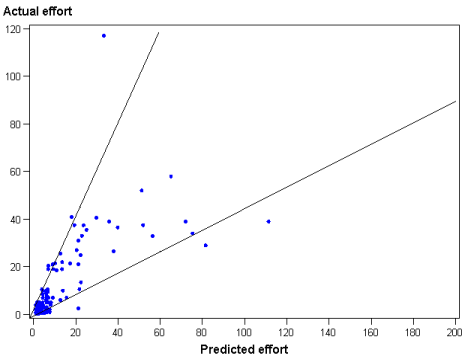


Figure 7. Predicted vs. actual effort, project A

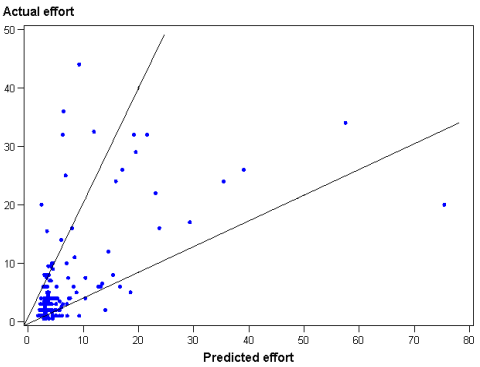


Figure 8. Predicted vs. actual effort, project B

5 Results from the Qualitative Analysis

Table 7 provides a summary of the results from the qualitative analysis of 44 of the 48 selected changes. Four changes were excluded from the analysis because the interviews showed that code changes had not been properly tracked.

The three first columns in Table 7 define the coding schema resulting from the coding process. Each code captures a factor that was perceived by the interviewees to drive or save effort. For example, *T0* could drive effort if the developer was unfamiliar with a relevant technology, and save effort if the developer had particularly good knowledge about the technology.

The rightmost column shows the number of times a code was used in underestimated and overestimated changes, respectively. The numbers can be interpreted as the degree of presence of a phenomenon in the projects, but we do not consider evidence from exceptional cases to be any less valid or important than frequent cases. Consequently, no statistical analyses of the qualitative results are provided. More detailed results from the qualitative analysis can be found in [40].

Table 7. Summary of factors from qualitative analysis

Category	Code	Description of code	Occurr. in underest./ overest. changes
Understanding requirements	R1	Clarification of change request was needed/not needed	9/2
Identifying and understanding relevant code	U1	It was difficult/easy to understand the relevant source code	7/1
	U2	It was difficult/easy to identify the relevant system states	3/3
	U3	The developer was unfamiliar/familiar with relevant source code	3/2
Learning relevant technologies and resolving technology issues	T0	Developer was unfamiliar/familiar with the relevant technology	3/0
	T1	The features of the technology did not/did suite the task	1/2
	T2	Technology had/did not have defects that affected the task	4/0
	T3	Technology had limited/good debugging support	5/0
Designing and applying changes to source code	D1	Change required deep/shallow understanding of user scenario	0/9
	D2	The needed mechanisms were not/were in place	13/2
	D3	Changes were made to many/very few parts of the code	0/8
Verifying change	V1	It was necessary/not necessary to establish test conditions	2/1
Cause of change (analyzed for all changes)	C1	Error by omission – failed to handle a system state	11/5
	C2	Error by commission – erroneous handling of a system state	1/3
	C3	Improve existing functionality – within current system scope	4/9
	C4	Planned expansion of functionality – extend the system scope	6/5

Many of the codes and categories coincide with concepts studied within the field of software comprehension. For example, Von Mayrhauser and Vans suggested lists of activities involved in change tasks that largely conform to our categories [65]. In our case, a separate category was justified for technology properties. Also, the design activity was difficult to distinguish from the coding activity; hence we used a common category. We chose to use a common coding schema for all types of changes, and let the cause of change be part of the coding schema.

5.1 Understanding Requirements

R1. For nine of the underestimated changes, the developers mentioned that the need to clarify requirements resulted in increased change effort. For two of the overestimated changes, they mentioned that a concise and complete specification made it easier to perform the change. This supports the results from quantitative analysis, which showed a consistent relationship between the number of updates to the original change request, and change effort. For the nine underestimated changes, the requirement clarifications were only partially documented in the change tracker. This explains the large residuals for these changes. The need to clarify requirements occurred more frequently in project A than in project B. However, six of nine underestimated changes for project B were fixes of errors

due to missed requirements, see Section 5.6. Hence, incomplete requirements had an unfavorable effect in both projects.

In some cases, the developers said that the user representatives deliberately failed to provide complete specifications, in particular for changes that concerned the look and feel of the user interface. However, the strongest effect on effort occurred when unanticipated side effects of a change needed to be clarified during detailed design and coding. In most cases, this meant that existing functionality was somehow impacted by the change, but that the developer was uncertain how to deal with these impacts.

5.2 Identifying and Understanding Relevant Source Code

A substantial portion of the total change effort can be comprehension effort. Koenemann and Robertson suggested that the comprehension process involves code of direct, intermediate and strategic relevance [66]. Directly relevant is code that has to be modified. Code that is perceived to interact with directly relevant code has intermediate relevance. Strategic code acts as a pointer towards other relevant parts of the code.

U1: Typically, the change requests were described by referencing a *user scenario*, i.e. a sequence of interactions between the user and the system, and by requesting a change to that scenario. For seven of the underestimated changes, the developers claimed considerable time was spent understanding relevant, *intermediate* code when it was dispersed among many files. The dispersion of *changed* code had a strong and consistent effect on change effort in the quantitative models. The time developers spend to *comprehend dispersed code* might be a more fundamental factor that in many cases explains the apparent effect of *making dispersed changes*.

The effort involved in comprehending code along the lines of user scenarios can also explain why the measures of structural attributes of changed components did not have an effect on change effort in the quantitative models. First, only directly affected components were captured by these measures, even though the structural attributes of intermediate code were likely to be important. Second, the measures capture the structural attributes of architectural units rather than of user scenarios. This suggests that it would be useful to collect measures of structural attributes along the execution path of the changed user scenarios. These measures could be based on models such as UML sequence diagrams, which would also aid in comprehension [67], or dynamic code measurement (e.g., by executing each user scenario), as proposed in [68].

U2: For three of the underestimated changes, the developers expressed that it was difficult to identify and understand the system states relevant to the change task. One developer stated: "All the states that need to be handled in the GUI make the code mind-blowing." This indicates that the perceived code complexity is caused by a complex underlying state model. It also suggests that in order to understand the code from the functional view discussed above, it is a prerequisite that the underlying state model is understood. An obvious proposal is to make it easier to understand the most complex underlying state models, e.g., by the use of diagramming techniques such as UML state diagrams.

U3: The degree of familiarity with relevant code was said to have affected change effort in five cases. The quantitative results for change experience showed that relatively little of the variations in change effort can be explained by familiarity with the systems. The qualitative analysis showed that experience was indeed important in both projects, in the few extreme cases when it was either very high or very low.

5.3 Learning Relevant Technologies and Resolving Technology Issues

T0. Lack of familiarity with relevant technology was perceived to increase change effort for three of the changes. The measure of the effect of technology experience (*techexp*) was not significant in the quantitative analysis. One possible explanation is that familiarity with the involved technology affected change effort in the relatively few cases where the familiarity was particularly low or high.

*T1, T2, T3***:** The degree of match between the actual and required features of the development tools and technologies was considered important in 12 cases. If the functionality required by the change task was provided out of the box, the technology was considered to save effort. Reversely, if the technology was incompatible with the change task, or had defects, considerable effort was required to create workarounds. Unsatisfactory facilities for debugging were considered to increase change effort in five cases.

5.4 Designing and Applying Changes to Source Code

D1: Empirical studies have shown that the nature of a given task determines the comprehension process [69]. Indeed, the interview data showed that the developers associated a certain degree of superficiality or *shallowness* with a change task. A change was perceived as *shallow* when the developer assumed that it was not necessary to understand the details of the code involved in the changed user scenario. Typically, shallow changes were performed by textual search in intermediate code to identify the

direct code to change. Examples of shallow changes were those that concerned the appearance in the user interface, user messages, logging behaviour and simple refactoring. Deep changes, on the other hand, required full comprehension of the code involved in the changed user scenario. The comprehension activities described in the previous section are therefore primarily relevant for deep changes.

D2: We use the term *mechanism* for code that implements a solution to a recurring need in the system. Typically, formalized design patterns [70] can be used directly or as part of a mechanism. In the investigated projects, examples of mechanisms are handling of runtime exceptions and transfer of data between the physical and logical layers of the system. In 13 cases, the change was perceived to be particularly challenging because a required mechanism had to be constructed. According to the developers, creating these mechanisms was challenging for two reasons: First, the mechanism had to be carefully designed for reusability. Second, when the purpose of mechanisms was to hide peculiarities of specific technologies, these needed to be well understood by the developer of the mechanism.

D3: The developers expressed that eight of the overestimated changes were easy to perform because they were concentrated in one or few parts in the code. This observation supports the results for *change set dispersion* from the quantitative analysis, and suggests a particularly strong effect for the most localized changes. However, this explanation is contradicted by data from 50 other change tasks that affected only one segment of the code without resulting in particularly low change effort. An alternative explanation is that the developers *perceived* the change to be particularly local because the code of intermediate relevance was not dispersed among many components, as elaborated in Section 5.2

5.5 Verifying Change

V1: The effort expended to test the developers' own code changes was discussed in the interviews. For a large majority of the changes, the developers found it quite easy to verify that the change was correctly coded. In two cases, verification was perceived to be difficult because the change task affected time-dependent behavior simulated in the test environment. In project A, some extra time was needed to generate and execute the system on the target mobile platform. In project B, extra time was needed when the technology necessitated deployment on a dedicated test server.

5.6 Cause of Change

The cause of each change, i.e. the events that triggered the change request, was discussed in the interviews. Based on this, we classified all changes according to the codes shown in the last row of Table 7. In order to better understand the results for change type from the quantitative analysis, we measured the agreement between the automated classification into change types, and the classification from qualitative analysis. Sufficient data was available for 87 and 61 changes, for project A and B, respectively. When mapping C1 and C2 to corrective change, and C3 and C4 to non-corrective change, the agreement was good (Cohen's kappa=0.64) for project A, but less than what could be expected by pure chance (Cohen's kappa=-0.038) for project B. This result shows that the automated classification for project B did not appropriately reflect real differences in change type, which can explain why there was no effect of change type in the quantitative models. From the qualitative analysis of project B, it can be seen that six out of nine of the underestimated changes were fixes of *error by omission*. A typical reason for such an error was not recognizing a side effect of a change. We conclude that for project B, fixes of errors by omission were associated with underestimated changes. In line with the conclusion in Section 5.1, we recommend practices that help to identify side effects of change requirements, because they are likely to reduce occurrences of errors by omission.

6 Joint Results and Discussion

The results from the different parts of the analysis are summarized as answers to the questions posed in Section 2.2:

1. Overall, the selected variables proved to be useful predictors in models of change effort. A notable exception was variables capturing structural properties of affected code, which could partly be explained by item 8 and item 9 below.
2. The explained variability was quite poor (best *MdMREcross* was 0.57) in the quantitative models. The qualitative analysis focusing on change tasks that corresponded to large model residuals was therefore justified.
3. In project A, the model fit substantially improved when a larger number of candidate variables were used (MdMREcross was reduced from 0.72 to 0.57). Improvement was due to the use of one additional variable, capturing language heterogeneity (see item 6 below).

4. The principal component analysis showed that measures of change set dispersion captured a factor different from pure size. This consistently and strongly contributed to change effort in the quantitative models: The standardized coefficients were 0.76 and 0.51.
5. The qualitative analysis suggested that the developers' effort to comprehend highly dispersed code was a more fundamental factor than the effort involved in making dispersed changes. However, comprehending and modifying code seemed to be closely intertwined processes, and therefore difficult to separate.
6. Language heterogeneity substantially contributed to change effort, as one additional affected language implies 30% more effort. A plausible explanation is that the effect of dispersion (see item 4 and 5) was amplified when comprehended and modified code spanned multiple technologies and languages.
7. Change request volatility, measured by updates in the change tracker, consistently contributed to change effort in the quantitative models. One additional update in the change tracker implied a 7% increase in change effort. The qualitative analysis showed that when change request volatility was due to difficulties in anticipating functional side effects of a change, the effect was particularly large. A possible underlying cause for these difficulties was insufficient knowledge in the interface between the software and the business domain.
8. The qualitative analysis showed that change effort was affected by code properties along the changed user scenarios. In particular, the complexity of the underlying state model of the user scenario was important, as was the dispersion of code that implemented the changed user scenario, as described in items 4 and 5. The developers' focus on functional cross-cuts can explain why structural attributes of architectural units, such as files and classes, proved inefficient in explaining change effort variability.
9. In project A, corrective changes required only 46% of the effort compared with non-corrective changes, after accounting for other factors. No significant difference was found for project B. The qualitative analysis for both projects showed that a sub-class of corrective changes (fixes of errors by omission) required additional effort. This analysis also showed that certain other characteristics of the change task, such as the need for *innovation*, was an important factor that is difficult to capture from change management data.
10. A moderate effect of developers' experience was identified in project B. A 16.2% decrease in change effort could be expected for every 1000th check-in. The qualitative

analysis showed that familiarity with the changed functional and technological areas was indeed important in both projects, in particular in the extreme cases when the familiarity was either very high or very low. This effect of experience was not appropriately captured by the quantitative models.

In the following, we discuss the consequences of these results from the perspective of software engineering, the local projects, and that of research methods within empirical software engineering.

6.1 Consequences for Software Engineering

Earlier change-based studies have assumed that measures such as *components*, or number of check-ins for a change task, can be considered coarse-granularity measures of size. An alternative interpretation is that such measures capture *delocalization* or *dispersion*. Controlled experiments and research into the cognitive processes of programmers have demonstrated difficulties in comprehending and changing dispersed code. An important contribution of this study is that it found clear evidence of the effect of dispersion in a real project setting with real change tasks. More refined results, and related consequences, are:

- Comprehension typically occurred along functional cross-cuts of the system. Hence, to mitigate the effect of dispersion, tools should have the capability of presenting change-friendlier views of the system based on such functional cross-cuts. Automatic generation of sequence diagrams is one possible implementation, c.f. [71, 72].
- The results indicate that the effect of dispersion depends on the heterogeneity of the involved components, and cannot be fully captured by a simple count of components. It seems particularly important that tools aimed at mitigating the effect of dispersion are able to handle technological heterogeneous environments.
- The results point to design practices that minimize dispersion for future change tasks. A recommended practice could be that functionally cohesive code should be localized rather than dispersed. However, the concern about change effort should be balanced against other concerns, such as potentials for reuse and constraints set by the physical architecture.
- Comprehending and changing dispersed code seemed to be intertwined processes. Hence, measures of affected components retrieved from version control systems can be expected to capture the phenomenon of dispersion reasonably well, though not perfectly.

If estimates of dispersion are used as input to prediction models, estimates of components to inspect can be just as effective as estimates of components to change.

Earlier change-based studies have shown a relationship between the number of modifications to change requests, and change effort. The confirmatory analysis in this study consistently supported the results. From the perspective of effort estimation, it is useful insight that such early retrievable measures have been consistently effective predictors.

Software organizations need to make trade-offs between enforcing well-defined upfront requirements and allowing for the flexibility of evolving requirements. This study contributes with the insight that volatility has the most serious effect on change effort when it is caused by lack of knowledge in the interface between software and business domain. In consequence, organizations should try to cultivate such knowledge, to avoid inefficient iterations towards the final requirements. Other kinds of volatility, such as refining a user interface based on customer feedback, have inherent advantages and do not seem to have severe effects. We believe that such results provide important insights to the on-going debates on plan-driven versus agile development principles.

Due to the wide prediction intervals implied by the relatively poor model fit obtained in this and similar studies [18, 20], it seems infeasible to build models that are sufficiently accurate to be accepted as a black-box method for estimating individual change tasks. Model-based estimates may still play a role to support projects in planning releases during software evolution, where the primary interest is in the aggregate of change effort estimates. A reasonable starting point for creating organization specific models is to use measures of change request volatility, developers' experience, type of change, and dispersion.

6.2 Consequences for the Investigated Projects

In project A, effort estimation was a team activity performed on a regular basis as part of release planning. To judge the potential for more accurate effort estimates, we calculated the accuracy of the current estimation process, on the basis of effort estimates and actual effort for the 107 change tasks where this data was available. The effort estimates were given in units of relative size, see [73], and were scaled according to the factor that minimized MdMRE. The resulting MMRE and MdMRE was 1.47 and 0.54, respectively. Even though these values roughly correspond to the accuracy of the models from the data-driven analysis, we did not recommend replacing judgement-based estimates with model-

based estimates, for two reasons. First, change set size or change set dispersion would have to be subjectively assessed to obtain the required input measures. This would likely decrease the model accuracy, and preclude fully automated procedures. Second, the team estimation of change tasks was perceived to be important to share knowledge, to build team spirit in the project, and to constitute an initial step of design for a solution to the change request.

To assess whether insight obtained from our analysis was already accounted for by the developers, we fitted regression models that included the significant quantitative factors and the developers' estimate as explanatory variables. Measures of change request volatility, change set dispersion and change type became statistically insignificant, indicating that these factors were already sufficiently accounted for by the subjective estimates. The number of different technologies involved, on the other hand, had a significant effect on actual effort. The model was:

log(ceffort)= 9.25 + 0.13*relativeEffortEstimate + 0.14*filetypes

We recommended that the developers put more emphasis on language heterogeneity when they made effort estimates. On the basis of the qualitative analysis we also advised more awareness of the effect of particularly strong familiarity or lack of familiarity with code of intermediate and direct relevance. On the basis of the results, we were also able to give the following recommendations:

- To reduce the most severe effects of change request volatility, actions should be taken to cultivate knowledge in the interface between the software and business domains. However, change request volatility should be accepted when solutions are iteratively optimized on the basis of immediate feedback, such as in the case of GUI design.
- Identify the user scenarios that are most frequently changed, and that involve many components and languages. Look for opportunities to refactor these, aiming at reducing the dispersion.
- Evaluate tools that make it easier to trace and understand the code involved in user scenarios. For example, emerging tools for dynamic code analysis for the Eclipse platform might have some of the desired qualities [72].
- Document the underlying state models in areas where those models are particularly complex

6.3 Consequences for Empirical Software Engineering

This study included a number of design elements that we believe constitute a step forward for change-based studies:

Foundation in a systematic review. The use of systematic reviews in software engineering was suggested as an important element of *evidence-driven software engineering* [74]. The factors and measures for the quantitative analysis were selected on the basis of a systematic literature review of earlier change-based studies. Systematic reviews are particularly useful when study proposals cannot be derived from established theories. Currently, this is the situation for most topics investigated within the empirical software engineering community.

Combined confirmatory and explorative analysis. Strong conclusions can only be drawn from confirmatory studies, while explorative studies are important to generate hypothesis and guide further research [75]. The evidence-driven analysis largely confirmed existing evidence. The data-driven analysis explored and identified additional factors to be investigated in future confirmatory studies.

Procedures for performing data-driven analysis. The data-driven analysis combined known sub-strategies for variable selection into an overall procedure for selecting the models, based on well-defined criteria. This was shown to perform better than a more traditional approach based on principal component regression. It is future work to attempt to improve this approach by, e.g., using alternative prediction frameworks.

Qualitative analysis to explain large model residuals. Even though the role of qualitative methods in this field has long been recognized, see e.g., [76], empirical researchers have developed and used quantitative methods to a larger extent [77]. Because we used *the individual change* as a common unit of analysis, and *change effort* as the dependent variable, we were able to tightly integrate the quantitative analysis of data from version control systems and change trackers with the qualitative analyses of developer interviews. This method also focuses the more expensive qualitative analysis on the most interesting data. This can be particularly important for practitioners who use lightweight empirical methods to evaluate their own practices such as Postmortem analysis [78] or Agile Retrospectives [79].

7 Threats to Validity

Construct validity. Quantitative measures were based on data from version control systems and change trackers. Such data will not perfectly capture the factors of interest. For example, change request volatility may not be fully documented in the change tracker. In this and other cases, we were able to use the qualitative data to compensate for these threats to construct validity. There were also threats to construct validity in the qualitative coding schema. We attempted to mitigate this by reconsolidating the coding schema to reflect commonly used concepts within our field.

Code complexity cannot be fully captured by one or a few measures [80]. To judge, in a meaningful and repeatable manner, whether a piece of code is "more complex than" another piece of code, very specific criteria must be defined. Therefore, there were obvious construct validity threats in the measurement of complexity of *change sets* and *changed components*. As indicated from the qualitative analysis, the apparent insignificance of code complexity could be due to problems with operationalizing the concept. For change experience, it is obviously a simplification to associate one check-in with one unit of experience. Moreover, averaging experience measures over developers does not perfectly capture the concept of joint experience. Measurement noise due to unreliable collection of change effort data could also have affected the results, although random noise would normally weaken the conclusions rather than incorrectly strengthening them.

In sum, it is likely that some of the unexplained variability in the quantitative models was due to the inability to fully capture the intended factors by measures retrieved from version controls systems.

Internal validity. Internal validity refers to the degree to which causal relationships can be claimed. Issues of internal validity are important when the context, tasks and procedures for allocating study units to groups cannot be controlled, which is the case with data that occurs naturally in software development projects. Qualitative data from developer interviews was useful to evaluate such threats. For example, the qualitative analysis suggested that a more fundamental, causal factor than the effect of dispersion of changed code was the effect of dispersion of *intermediate* code that needed to be comprehended.

Another threat to internal validity was the possibility of shotgun correlations. In the data-driven analysis, a large number of factors and measures were tested. This increases the likelihood that one or more of the significant effects occurred due to chance, rather than to a true underlying effect. This risk was lower in the evidence-driven analysis, because this

investigated the effect of a small set of factors and measures selected on the basis of existing empirical evidence.

A third type of threat to internal validity was the potential bias introduced by missing data points in the data set, see [81]. For project A, change effort was not recorded for around 10% of the actual changes that were performed. For project B, it was not recorded for 25% of the changes. Most of the missing data points were due to challenges with establishing the routines to track change effort and code changes. Because the data points that we did collect from the initial periods can be considered randomly selected, we do not expect the missing data points to constitute a serious threat to internal validity.

The use of interviews introduced the possibility of researcher bias, consciously or unconsciously skewing the investigation to conform to the competencies, opinions, values or interests of the involved researchers. Although such threats apply to quantitative research as well, they can be particularly difficult to assess handle when subjectivity is involved. Imperfect memory, lack of trust or other communication barriers between the interviewer and the interviewee may also introduce biases. We believe that the strict focus on relatively small, cohesive tasks recently performed by the interviewee helped to mitigate such biases. To mitigate communication barriers, the interviewer made extensive efforts to be prepared for the interviews, and data from the version control systems and change trackers was readily available during the interviews to help the developers recollect details.

External validity. The ability to generalize results beyond the study context is one of the key concerns with case studies. Section 2.4 described the design elements introduced to interpret the results in a wider context. We believe that the lack of relevant theories on which to base the study proposals is a major obstacle to generalizing the results. In this situation, we chose to base the study proposals on a comprehensive review of earlier empirical studies with similar research questions.

8 Conclusion

Software engineering practices can be improved if they address factors that have been shown empirically to affect developers' effort during software evolution. In this study, we identified such factors by analyzing data about changes in two software organizations. Regression models were constructed to identify factors that correlated with change effort,

and developer interviews explored additional factors at play when the developers expended effort to perform change tasks. Two central results were:

- Change request volatility had a consistent effect on effort in the quantitative models. The effect was particularly large when volatility resulted from difficulties in anticipating side effects of a change. Such difficulties also resulted in errors by omission, which in turn were particularly expensive to correct.
- The dispersion of modified code also had a large and consistent effect on change effort in the quantitative models, beyond the effect of size alone. The qualitative analysis indicated that the dispersion of *comprehended* code was a more fundamental factor.

Because these results are also consistent with results from earlier empirical studies, we suggest that these (admittedly quite course-grained) factors should be considered when attempting to improve software engineering practices.

The specific analyses of the two projects provided additional and more fine-grained results. In one project, changes that concerned only one language required considerably less effort. The analysis of estimation accuracy indicated that this factor was not sufficiently accounted for when developers made their estimates. This exemplifies how projects can benefit from analyzing data from their version control systems and change trackers to improve their estimation practices.

One important direction for further work is to investigate further the causal relationships occurring when developers perform change tasks. Interviewing developers about recent changes was an effective method for making tentative suggestions about such relationships. However, studies that control possibly confounding factors should be conducted before firm conclusions are drawn. It is also necessary to paint a richer picture of how context factors, such as size and type of the system, influence change effort. Ultimately, the empirical results could be aggregated into a *theory on software change effort*, which would define invariant knowledge about software evolution, and be immediately useful for practitioners within the field.

Acknowledgement

We are indebted to the managers and developers at Esito and Know IT who provided us with high quality empirical data. The research was funded by the Simula School of Research and Innovation.

Appendix A

Interview Guide

Part 1. (Only in first interview with each developers **-** Information about the purpose of the research. Agree on procedures, confidentiality voluntariness, audio-recording).

Question: Can you describe your work and your role in the project?

Part 2. Project context (factors intrinsic to the time period covered by the changes under discussion)

How would you describe the project and your work in the last time period? Did any particular event require special focus in the period?

For each change (CR-nnnn, CR-nnnn, CR-nnnn….,)

Part 3. Measurement control (change effort and name of changed components shown to the interviewee)

Are change effort and code changes correctly registered?

Part 4. Change request characteristics (change tracker information shown on screen to support discussion)

Can you describe the change from the viewpoint of the user? Why was the change needed?

Part 5. General cost factors

Can you roughly indicate how the X hours were distributed on different activities?

Part 6. Properties of relevant code (output from *windiff* showed on screen to support the discussions)

Can you summarize the changes that you made to the components?

What can you say about the code that was relevant for the change? Was it easy or difficult to understand and make changes to the code?

Part 7. Stability

Did you go through several iterations before you reached the final solution? If so, why?

Did anything not go as expected?

How did you proceed to test the change?

Go to Part 3 for next change

Part 8. Concluding remarks

Do you think this interview covered your activities during the last period?

References

[1] L. A. Belady and M. M. Lehman, "A Model of Large Program Development," *IBM Systems Journal,* vol. 15, pp. 225-252, 1976.

[2] R. D. Banker, S. M. Datar, C. F. Kemerer, and D. Zweig, "Software Complexity and Maintenance Costs," *Communications of the ACM,* vol. 36, pp. 81-94, 1993.

[3] P. Bhatt, G. Shroff, C. Anantaram, and A. K. Misra, "An Influence Model for Factors in Outsourced Software Maintenance," *Journal of Software Maintenance and Evolution: Research and Practice,* vol. 18, pp. 385-423, 2006.

[4] M. S. Krishnan, C. H. Kriebel, S. Kekre, and T. Mukhopadhyay., "An Empirical Analysis of Productivity and Quality in Software Products," *Management Science,* vol. 46, pp. 745-759, 2000.

[5] B. P. Lientz, "Issues in Software Maintenance," *ACM Computing Surveys,* vol. 15, pp. 271-278, 1983.

[6] J. H. Hayes, S. C. Patel, and L. Zhao, "A Metrics-Based Software Maintenance Effort Model," in *8th European Conference on Software Maintenance and Reengineering*, 2004, pp. 254-258.

[7] C. Kemerer, "Software Complexity and Software Maintenance: A Survey of Empirical Research," *Annals of Software Engineering,* vol. 1, pp. 1-22, 1995.

[8] J. C. Munson and S. G. Elbaum, "Code Churn: A Measure for Estimating the Impact of Code Change," in *14th International Conference on Software Maintenance*, 1998, pp. 24-31.

[9] F. Détienne and F. Bott, *Software Design - Cognitive Aspects*. London: Springer-Verlag, 2002.

[10] H. C. Benestad, B. C. Anda, and E. Arisholm, "Technical Report 10-2008: A Systematic Review of Empirical Software Engineering Studies That Analyze Individual Changes," Simula Research Laboratory, 2008.

[11] T. L. Graves and A. Mockus, "Inferring Change Effort from Configuration Management Databases," in *5th International Symposium on Software Metrics*, 1998, pp. 267–273.

[12] A. Mockus and D. M. Weiss, "Predicting Risk of Software Changes," *Bell Labs Technical Journal,* vol. 5, pp. 169-180, 2000.

[13] D. L. Atkins, T. Ball, T. L. Graves, and A. Mockus, "Using Version Control Data to Evaluate the Impact of Software Tools: A Case Study of the Version Editor," *IEEE Transactions on Software Engineering,* vol. 28, pp. 625-637, 2002.

[14] B. Geppert, A. Mockus, and F. Rößler, "Refactoring for Changeability: A Way to Go?," in *11th International Symposium on Software Metrics*, 2005.

[15] E. Arisholm, "Empirical Assessment of the Impact of Structural Properties on the Changeability of Object-Oriented Software," *Information and Software Technology,* vol. 48, pp. 1046-1055, 2006.

[16] W. M. Evanco, "Prediction Models for Software Fault Correction Effort," in *5th European Conference on Software Maintenance and Reengineering*, 2001, pp. 114-120.

[17] M. Polo, M. Piattini, and F. Ruiz, "Using Code Metrics to Predict Maintenance of Legacy Programs: A Case Study," in *2001 International Conference on Software Maintenance*, 2001, pp. 202-208.

[18] M. Jørgensen, "Experience with the Accuracy of Software Maintenance Task Effort Prediction Models," *IEEE Transactions on Software Engineering,* vol. 21, pp. 674-681, 1995.

[19] F. Niessink and H. van Vliet, "Two Case Studies in Measuring Software Maintenance Effort," in *14th International Conference on Software Maintenance*, 1998, pp. 76–85.

[20] F. Niessink and H. van Vliet, "Predicting Maintenance Effort with Function Points," in *1997 International Conference on Software Maintenance*, 1997, pp. 32-39.

[21] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does Code Decay? Assessing the Evidence from Change Management Data," *IEEE Transactions on Software Engineering,* vol. 27, pp. 1-12, 2001.

[22] E. Soloway, J. Pinto, and S. Letovsky, "Designing Documentation to Compensate for Delocalized Plans," *Communications of the ACM,* vol. 31, pp. 1259–1267.

[23] E. Arisholm and D. I. K. Sjøberg, "Evaluating the Effect of a Delegated Versus Centralized Control Style on the Maintainability of Object-Oriented Software," *IEEE Transactions on Software Engineering,* vol. 30, pp. 521-534, 2004.

[24] N. F. Schneidewind, "Investigation of the Risk to Software Reliability and Maintainability of Requirements Changes," in *2001 International Conference on Software Maintenance*, 2001, pp. 127-136.

[25] K. Beck, "Embracing Change with Extreme Programming," *Computer,* vol. 32, pp. 70-77, 1999.

[26] L. C. Briand and J. Wüst, "Empirical Studies of Quality Models in Object-Oriented Systems," *Advances in Computers,* vol. 59, pp. 97-166, 2002.

[27] B. Xu, M. Yang, H. Liang, and H. Zhu, "Maximizing Customer Satisfaction in Maintenance of Software Product Family," in *18th Canadian Conference on Electrical and Computer Engineering*, 2005, pp. 1320-1323.

[28] E. B. Swanson, "The Dimensions of Maintenance," in *2nd International Conference on Software Engineering*, San Francisco, California, United States, 1976, pp. 492-497.

[29] L. C. Briand and V. R. Basili, "A Classification Procedure for the Effective Management of Changes During the Maintenance Process," in *1992 Conference on Software Maintenance*, 1992, pp. 328-336.

[30] M. Reformat and V. Wu, "Analysis of Software Maintenance Data Using Multi-Technique Approach," in *15th International Conference on Tools with Artificial Intelligence*, 2003, pp. 53-59.

[31] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M. Y. Wong, "Orthogonal Defect Classification-a Concept for in-Process

Measurements," *Software Engineering, IEEE Transactions on,* vol. 18, pp. 943-956, 1992.

[32] B. Curtis, S. B. Sheppard, E. Kruesi-Bailey, J. Bailey, and D. A. Boehm-Davis, "Experimental Evaluation of Software Documentation Formats," *Journal of Systems and Software,* vol. 9, pp. 167-207, 1989.

[33] R. K. Yin, "Designing Case Studies," in *Case Study Research: Design and Methods*: Sage Publications:Thousand Oaks, CA, 2003, pp. 19-53.

[34] H. C. Benestad, E. Arisholm, and D. Sjøberg, "How to Recruit Professionals as Subjects in Software Engineering Experiments," *Information Systems Research in Scandinavia (IRIS), Kristiansand, Norway,* 2005.

[35] RCN, 2008, Research Council of Norway, My RCN Web. http://www.forskningsradet.no

[36] NSB, 2008, Norwegian State Railways, http://www.nsb.no

[37] IBM, 2008, IBM Rational ClearCase, http://www-01.ibm.com/software/rational

[38] GNU, 2008, Concurrent Versions System, http://www.nongnu.org/cvs

[39] Atlassian, 2008, Jira bug and issue tracker, http://www.atlassian.com

[40] H. C. Benestad, B. Anda, and E. Arisholm, "Technical Report 02-2009: An Investigation of Change Effort in Two Evolving Software Systems," Simula Research Laboratory Technical report 01/2009, 2009.

[41] W. M. Evanco, "Analyzing Change Effort in Software During Development," in *6th International Symposium on Software Metrics (METRICS99)*, 1999, pp. 179-188.

[42] J. W. Hunt and M. D. McIlroy, "An Algorithm for Differential File Comparison," in *Computing Science Technical Report 41, Bell Laboratories*, 1975.

[43] K. Moløkken-Østvold, N. C. Haugen, and H. C. Benestad, "Using Planning Poker for Combining Expert Estimates in Software Projects," *Accepted for publication in Journal of Systems and Software,* 2008.

[44] B. Fluri and H. C. Gall, "Classifying Change Types for Qualifying Change Couplings," in *14th International Conference on Program Comprehension (ICPC)*, Athens, Greece, 2006, pp. 35-45.

[45] M. Jørgensen, "An Empirical Study of Software Maintenance Tasks," *Journal of Software Maintenance: Research and Practice,* vol. 7, pp. 27-48, 1995.

[46] R. Purushothaman and D. E. Perry, "Toward Understanding the Rhetoric of Small Source Code Changes," *IEEE Transactions on Software Engineering,* vol. 31, pp. 511-526, 2005.

[47] H. C. Benestad, "Technical Report 12-2008: Assessing the Reliability of Developers' Classification of Change Tasks: A Field Experiment," Simula Research Laboratory2008.

[48] L. Etzkorn, J. Bansiya, and C. Davis, "Design and Code Complexity Metrics for Oo Classes," *Journal of Object-Oriented Programming,* vol. 12, pp. 35-40, 1999.

[49] C. Rajaraman and M. R. Lyu, "Reliability and Maintainability Related Software Coupling Metrics in C++ Programs," in *Third International Symposium on Software Reliability Engineering*, 1992, pp. 303-311.

[50] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting Fault Incidence Using Software Change History," *IEEE Transactions on Software Engineering,* vol. 26, pp. 653-661, 2000.

[51] T. DeMarco and T. Lister, "Programmer Performance and the Effects of the Workplace," in *Proceedings of the 8th international conference on Software engineering*, 1985, pp. 268-272.

[52] H. Sackman, W. J. Erikson, and E. E. Grant, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," *Communications of the ACM,* vol. 11, pp. 3-11, 1968.

[53] R. H. Myers, D. C. Montgomery, and G. G. Vining, "The Generalized Linear Model," in *Generalized Linear Models with Applications in Engineering and the Sciences*: Wiley Series in Probability and Statistics, 2001, pp. 4-6.

[54] R. L. Ott and M. Longnecker, "Inferences in Multiple Regression," in *Statistical Methods and Data Analysis*: Duxbury, 2001, pp. 646-657.

[55] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*: Benjamin-Cummings Publishing Co., Inc. Redwood City, CA, USA, 1986.

[56] D. Woods, 2008, Transana - Qualitative analysis software for video and audio data. Developed at the University of Wisconsin-Madison Center for Education Research

[57] G. E. Pinches and K. A. Mingo, "A Multivariate Analysis of Industrial Bond Ratings," *Journal of Finance,* vol. 28, pp. 1-18, 1973.

[58] L. C. Briand and J. Wüst, "Integrating Scenario-Based and Measurement-Based Software Product Assessment," *The Journal of Systems & Software,* vol. 59, pp. 3-22, 2001.

[59] A. Miller, "Generating All Subsets," in *Subset Selection in Regression*, 2002, pp. 48-52.

[60] M. Shin and A. L. Goel, "Empirical Data Modeling in Software Engineering Using Radial Basis Functions," *IEEE Transactions on Software Engineering,* vol. 26, pp. 567-576, 2000.

[61] M. Stone, "Cross-Validatory Choice and Assessment of Statistical Predictions," *Journal of the Royal Statistical Society,* vol. 36, pp. 111-133, 1974.

[62] L. C. Briand and J. Wüst, "The Impact of Design Properties on Development Cost in Object-Oriented Systems," *IEEE Transactions on Software Engineering,* vol. 27, pp. 963-986, 2001.

[63] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. New York: Springer-Verlag, 2002.

[64] R. Christensen, "Principal Component Regression," in *Analysis of Variance, Design and Regression*, 1996, pp. 446-451.

[65] A. von Mayrhauser and A. M. Vans, "Program Comprehension During Software Maintenance and Evolution," *Computer,* vol. 28, pp. 44-55, 1995.

[66] J. Koenemann and S. P. Robertson, "Expert Problem Solving Strategies for Program Comprehension," in *SIGCHI conference on Human factors in computing systems: Reaching through technology*, 1991, pp. 125-130.

[67] W. J. Dzidek, E. Arisholm, and L. C. Briand, "A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance," *IEEE Transactions on Software Engineering,* vol. 34, pp. 407-432, 2008.

[68] E. Arisholm, L. C. Briand, and A. Føyen, "Dynamic Coupling Measurement for Object-Oriented Software," *IEEE Transactions on Software Engineering,* vol. 30, pp. 491-506, 2004.

[69] F. Détienne and F. Bott, "Influence of the Task," in *Software Design - Cognitive Aspects* London: Springer-Verlag, 2002, pp. 105-110.

[70] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley, 1995.

[71] L. C. Briand, Y. Labiche, and Y. Miao, "Towards the Reverse Engineering of UML Sequence Diagrams," in *10th Working Conference on Reverse Engineering, WCRE 2003*, 2003, pp. 57-66.

[72] TPTP, 2008, Eclipse Test&Performance Tools Platform Project, http://www.eclipse.org/tptp

[73] M. Cohn, *Agile Estimating and Planning*: Pearson Education, Inc. Boston, MA, 2006.

[74] B. A. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-Based Software Engineering," in *26th International Conference on Software Engineering (ICSE)*, Edinburgh, Scotland, 2004, pp. 273-281.

[75] B. A. Kitchenham, S. L. Pleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering,* vol. 12, pp. 1106-1125, 2002.

[76] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering,* vol. 25, pp. 557-572, 1999.

[77] D. E. Perry, A. A. Porter, and L. G. Votta, "Empirical Studies of Software Engineering: A Roadmap," in *Conference on The Future of Software Engineering*, 2000, pp. 345-355.

[78] A. Birk, T. Dingsøyr, and T. Stålhane, "Postmortem: Never Leave a Project without It," *IEEE Software,* vol. 19, pp. 43-45, 2002.

[79] E. Derby and D. Larsen, *Agile Retrospectives: Making Good Teams Great*: Raleigh, NC: Pragmatic Bookshelf, 2006.

[80] N. Fenton, "Software Measurement: A Necessary Scientific Basis," *IEEE Transactions on Software Engineering,* vol. 20, pp. 199-205, 1994.

[81] A. Mockus, "Missing Data in Software Engineering," in *Guide to Advanced Empirical Software Engineering*, 2000, pp. 185-200.

Paper 3:

Are We More Productive Now? Analyzing Change Tasks to Assess Productivity Trends During Software Evolution

Hans Christian Benestad, Bente Anda, Erik Arisholm

Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering

Abstract

Organizations that maintain and evolve software would benefit from being able to measure productivity in an easy and reliable way. This could allow them to determine if new or improved practices are needed, and to evaluate improvement efforts. We propose and evaluate indicators of productivity trends that are based on the premise that productivity during software evolution is closely related to the effort required to complete change tasks. Three indicators use data about change tasks from change management systems, while a fourth compares effort estimates of benchmarking tasks. We evaluated the indicators using data from 18 months of evolution in two commercial software projects. The productivity trend in the two projects had opposite directions according to the indicators. The evaluation showed that productivity trends can be quantified with little measurement overhead. We expect the methodology to be a step towards making quantitative self-assessment practices feasible even in low ceremony projects.

1 Introduction

1.1 Background

The productivity of a software organization that maintains and evolves software can decrease over time due to factors like code decay [1] and difficulties in preserving and developing the required expertise [2]. Refactoring [3] and collaborative programming [4]

are practices that can counteract negative trends. A development organization might have expectations and gut feelings about the total effect of such factors and accept a moderate decrease in productivity as the system grows bigger and more complex. However, with the ability to quantify changes in productivity with reasonable accuracy, organizations could make informed decisions about the need for improvement actions. The effects of new software practices are context dependent, and so it would be useful to subsequently evaluate whether the negative trend was broken.

The overall aim for the collaboration between our research group and two commercial software projects (henceforth referred to as MT and RCN) was to understand and manage evolution costs for object-oriented software. This paper was motivated by the need to answer the following practical question in a reliable way:

> *Did the productivity in the two projects change between the baseline period P0 (Jan-July 2007) and the subsequent period P1 (Jan-July 2008)?*

The project RCN performed a major restructuring of their system during the fall of 2007. It was important to evaluate whether the project benefitted as expected from the restructuring effort. The project MT added a substantial set of new features since the start of *P0* and queried whether actions that could ease further development were needed. The methodology used to answer this question was designed to become part of the projects' periodic self-assessments, and aimed to be a practical methodology in other contexts as well.

1.2 Approaches to Measuring Productivity

In a business or industrial context, productivity refers to the ratio of output production to input effort [5]. In software engineering processes, inputs and outputs are multidimensional and often difficult to measure. In most cases, development effort measured in man-hours is a reasonable measure of input effort. In their book on software measurement, Fenton and Pfleeger [6] discussed measures of productivity based on the following definition of software productivity:

$$\text{productivity} = \frac{\text{size}}{\text{effort}} \qquad \textbf{(1)}$$

Measures of developed size include *lines of code*, *affected components* [7], *function points* [8-10] and *specification weight metrics* [11]. By plotting the productivity measure, say, every month, projects can examine trends in productivity. Ramil and Lehman used a

statistical test (CUSUM) to detect statistically significant changes over time [12]. The same authors proposed to model development effort as a function of size:

$$\text{effort} = \beta_0 + \beta_1 \cdot \text{size} . \quad (2)$$

They suggested collecting data on effort and size periodically, e.g., monthly, and to interpret changes in the regression coefficients as changes in *evolvability*. *Number of changed modules* was proposed as a measure of size. The main problem with these approaches is to define a size measure that is both meaningful and easy to collect. This is particularly difficult when software is changed rather than developed from scratch.

An alternative approach, corresponding to this paper's proposal, is to focus on the *completed change task* as the fundamental unit of output production. A change task is the development activity that transforms a change request into a set of modifications to the source components of the system. When software evolution is organized around a queue of change requests, the completed change task is a more intuitive measure of output production than traditional size measures, because it has more direct value to complete a change task than to produce another *n* lines of code. A corresponding input measure is the development effort required to complete the change task, referred to as *change effort*.

Several authors compared average change effort between time periods to assess trends in the maintenance process [13-15]. Variations of this indicator include average change effort per maintenance type (e.g., corrective, adaptive or enhancive maintenance). One of the proposed indicators uses direct analysis of change effort. However, characteristics of change tasks may change over time, so focusing solely on change effort might give an incomplete picture of productivity trends.

Arisholm and Sjøberg argued that *changeability* may be evaluated with respect to the same change task, and defined that changeability had *decayed* with respect to a given change task *c* if the effort to complete *c* (including the consequential change propagation) increased between two points in time [16]. We consider *productivity* to be closely related to *changeability*, and we will adapt their definition of *changeability decay* to *productivity change*.

In practice, comparing the same change tasks over time is not straightforward, because change tasks rarely re-occur. To overcome this practical difficulty, developers could perform a set of "representative" tasks in periodic *benchmarking* sessions. One of the proposed indicators is based on benchmarking identical change tasks. For practical

reasons, the tasks are only estimated (in terms of expected change effort) but are not completed by the developers.

An alternative to benchmarking sessions is using naturally occurring data about change tasks and adjusting for differences between them when assessing trends in productivity. Graves and Mockus retrieved data on 2794 change tasks completed over 45 months from the version control system for a large telecommunication system [17]. A regression model with the following structure was fitted on this data:

$$\text{Change effort} = f(\text{developer}, \text{type}, \text{size}, \text{date}) \quad (3)$$

The resulting regression coefficient for *date* was used to assess whether there was a time trend in the effort required to complete change tasks, while controlling for variations in other variables. One of our proposed indicators is an adaption of this approach.

A conceptually appealing way to think about productivity change is to compare change effort for a set of completed change tasks to the hypothetical change effort *had the same changes been completed at an earlier point in time*. One indicator operationalizes this approach by comparing change effort for completed change tasks to the corresponding effort estimates from statistical models. This is inspired by Kitchenham and Mendes' approach to measuring the productivity of finalized projects by comparing actual project effort to model-based effort estimates [18].

The contribution of this paper is i) to define the indicators within a framework that allows for a common and straightforward interpretation, and ii) to evaluate the validity of the indicators in the context of two commercial software projects. The evaluation procedures are important, because the validity of the indicators depends on the data at hand.

The remainder of this paper is structured as follows: Section 2 describes the design of the study, Section 3 presents the results and the evaluation of the indicators and Section 4 discusses the potential for using the indicators. Section 5 concludes.

2 Design of the Study

2.1 Context for Data Collection

The overall goal of the research collaboration with the projects RCN and MT was to better understand lifecycle development costs for object-oriented software. The projects' incentive for participating was the prospect of improving development practices by participating in empirical studies.

The system developed by MT is owned by a public transport operator, and enables passengers to purchase tickets on-board. The system developed by RCN is owned by the Research Council of Norway, and is used by applicants and officials at the council to manage the lifecycle of research grants. MT is mostly written in Java, but uses C++ for low-level control of hardware. RCN is based on Java-technology, and uses a workflow engine, a JEE application server, and a UML-based code generation tool. Both projects use management principles from Scrum [19]. Incoming change requests are scheduled for the monthly releases by the development group and the product owner. Typically, 10-20 percent of the development effort was expended on corrective change tasks. The projects worked under time-and-material contracts, although fixed-price contracts were used in some cases. The staffing in the projects was almost completely stable in the measurement period.

Project RCN had planned for a major restructuring in their system during the summer and early fall of 2007 (between *P0* and *P1*), and was interested in evaluating whether the system was easier to maintain after this effort. Project MT added a substantial set of new features over the two preceding years and needed to know if actions easing further development were now needed.

Data collection is described in more detail below and is summarized in Table 1.

Table 1. Summary of data collection

	RCN	MT
Period *P0*	Jan 01 2007 - Jun 30 2007	Aug 30 2006 - Jun 30 2007
Period *P1*	Jan 30 2008 - Jun 30 2008	Jan 30 2008 - Jun 30 2008
Change tasks in *P0/P1*	136/137	200/28
Total change effort in *P0/P1*	1425/1165 hours	1115/234 hours
Benchmarking sessions	Mar 12 2007, Apr 14 2008	Mar 12 2007, Apr 14 2008
Benchmark tasks	16	16
Developers	4 (3 in benchmark)	4

2.2 Data on Real Change Tasks

The first three proposed indicators use data about change tasks completed in the two periods under comparison. It was crucial for the planned analysis that data on change effort was recorded by the developers, and that source code changes could be traced back to the originating change request. Although procedures that would fulfil these requirements were already defined by the projects, we offered an economic compensation for extra effort required to follow the procedures consistently.

We retrieved data about the completed change tasks from the projects' change trackers and version control systems by the end of the baseline period (P0) and by the end of the

second period (*P1*). From this data, we constructed measures of change tasks that covered requirements, developers' experience, size and complexity of the change task and affected components, and the type of task (corrective vs. non-corrective). The following measures are used in the definitions of the productivity indicators in this paper:

- *crTracks* and *crWords* are the number of updates and words for the change request in the change tracker. They attempt to capture the volatility of requirements for a change task.
- *components* is the number of source components modified as part of a change task. It attempts to capture the dispersion of the change task.
- i*sCorrective* is 1 if the developers had classified the change task as corrective, or if the description for the change task in the change tracker contained strings such as *bug*, *fail* and *crash*. In all other cases, the value of *isCorrective* is 0.
- *addCC* is the number of control flow statements added to the system as part of a change task. It attempts to capture the control-flow complexity of the change task.
- *systExp* is the number of earlier version control check-ins by the developer of a change task.
- *chLoc* is the number of code lines that are modified in the change task.

A complete description of measures that were hypothesized to affect or correlate with change effort is provided in [20].

2.3 Data on Benchmark Tasks

The fourth indicator compares developers' effort estimates for benchmark change tasks between two *benchmarking sessions*. The 16 benchmark tasks for each project were collaboratively designed by the first author of this paper and the project managers. The project manager's role was to ensure that the benchmark tasks were representative of real change tasks. This meant that the change tasks should not be perceived as artificial by the developers, and they should cross-cut the main architectural units and functional areas of the systems.

The sessions were organized approximately in the midst of *P0* and *P1*. All developers in the two projects participated, except for one who joined RCN during *P0*. We provided the developers with the same material and instructions in the two sessions. The developers worked independently, and had access to their normal development environment. They were instructed to identify and record affected methods and classes before they recorded the estimate of most likely effort for a benchmark task. They also recorded estimates of

uncertainty, the time spent to estimate each task, and an assessment of their knowledge about the task. Because our interest was in the productivity of the *project*, the developers were instructed to assume a normal assignment of tasks to developers in the project, rather than estimating on one's own behalf.

2.4 Design of Productivity Indicators

We introduce the term *productivity ratio* (PR) to capture the change in productivity between period *P0* and a subsequent period *P1*.

The productivity ratio with respect to a single change task *c* is the ratio between the effort required to complete *c* in *P1* and the effort required to complete *c* in *P0*:

$$\mathrm{PR}(c) = \frac{\mathrm{effort}(c, P1)}{\mathrm{effort}(c, P0)} \quad (4)$$

The productivity ratio with respect to a set of change tasks *C* is defined as the set of individual values for *PR(c)*:

$$\mathrm{PR}(C) = \{ \frac{\mathrm{effort}(c, P1)}{\mathrm{effort}(c, P0)} \mid c \in C \} \quad (5)$$

The central tendency of values in *PR(C)*, *CPR(C),* is a useful single-valued statistic to assess the typical productivity ratio for change tasks in *C*:

$$\mathrm{CPR}(C) = \mathrm{central}\{\frac{\mathrm{effort}(c, P1)}{\mathrm{effort}(c, P0)} \mid c \in C\} \quad (6)$$

The purpose of the above definition is to link practical indicators to a common theoretical definition of productivity change. This enables us to define scale-free, comparable indicators with a straightforward interpretation. For example, a value of 1.2 indicates a 20% increase in effort from *P0* to *P1* to complete the same change tasks. A value of 1 indicates no change in productivity, whereas a value of 0.75 indicates that only 75% of the effort in *P0* is required in *P1*. Formal definitions of the indicators are provided in Section 2.4.1 to 2.4.4.

2.4.1 Simple Comparison of Change Effort

The first indicator requires collecting only change effort data. A straightforward way to compare two series of unpaired effort data is to compare their arithmetic means:

$$\mathrm{ICPR}_1 = \frac{\mathrm{mean}(\mathrm{effort}(c1) \mid c1 \in P1)}{\mathrm{mean}(\mathrm{effort}(c0) \mid c0 \in P0)} \quad (7)$$

The Wilcoxon rank-sum test determines whether there is a statistically significant difference in change effort values between *P0* and *P1*. One interpretation of this test is that

it assesses whether the median of all possible differences between change effort in *P0* and *P1* is different from 0:

$$HL = \text{median}(\text{effort}(c1) - \text{effort}(c0) \mid c1 \in P1, c0 \in P0) \quad (8)$$

This statistic, known as the Hodges-Lehmann estimate of the difference between values in two data sets, can be used to complement $ICPR_1$. The actual value for this statistic is provided with the evaluation of $ICPR_1$, in Section 3.1.

$ICPR_1$ assumes that the change tasks in *P0* and *P1* are comparable, i.e. that there are no systematic differences in the properties of the change tasks between the periods. We checked this assumption by using descriptive statistics and statistical tests to compare measures that we assumed (and verified) to be correlated with change effort in the projects (see Section 3.2). These measures were defined in Section 2.2.

2.4.2 Controlled Comparison of Change Effort

$ICPR_2$ also compares change effort between *P0* and *P1*, but uses a statistical model to control for differences in properties of the change tasks between the periods. Regression models with the following structure for respectively RCN and MT are used:

$$\log(\text{effort}) = \beta_0 + \beta_1 \cdot \text{crWords} + \beta_2 \cdot \text{chLoc} + \beta_3 \cdot \text{filetypes} + \beta_4 \cdot \text{isCorr} + \beta_5 \cdot \text{inP1}. \quad (9)$$

$$\log(\text{effort}) = \beta_0 + \beta_1 \cdot \text{crTracks} + \beta_2 \cdot \text{addCC} + \beta_3 \cdot \text{components} + \beta_4 \cdot \text{systExp} + \beta_5 \cdot \text{inP1}. \quad (10)$$

The models (9) and (10) are project specific models that we found best explained variability in change effort, c.f. [20]. The dependent variable *effort* is the reported change effort for a change task. The variable *inP1* is 1 if the change task *c* was completed in *P1* and is zero otherwise. The other variables were explained in Section 2.2. When all explanatory variables except *inP1* are held constant, which would be the case if one applies the model on the same change tasks but in the two, different time periods *P0* and *P1*, the ratio between change effort in *P1* and *P0* becomes

$$ICPR_2 = \frac{\text{effort}(Var1..Var4, inP1 = 1)}{\text{effort}(Var1..Var4, inP1 = 0)} = \frac{e^{ß0+ß1\cdot Var1+ß2\cdot Var2+ß3\cdot Var3+ß4\cdot Var4+ß5\cdot 1}}{e^{ß0+ß1\cdot Var1+ß2\cdot Var2+ß3\cdot Var3+ß4\cdot Var4+ß5\cdot 0}} = e^{\beta 5}. \quad (11)$$

Hence, the value of the indicator can be obtained by looking at the regression coefficient for *inP1*, β5. Furthermore, the p-value for β5 is used to assess whether β5 is significantly different from 0, i.e. that the indicator is different from 1 (e^0=1).

Corresponding project specific models must be constructed to apply the indicator in other contexts. The statistical framework used was Generalized Linear Models assuming Gamma-distributed responses (change effort) and a *log* link-function.

2.4.3 Comparison between Actual and Hypothetical Change Effort

$ICPR_3$ compares change effort for tasks in *P1* with the hypothetical change effort had the same tasks been performed in *P0*. These hypothetical change effort values are generated with a project-specific prediction model built on data from change tasks in *P0.* The model structure is identical to (9) and (10), but without the variable *inP1.*

Having generated this paired data on change effort, the definition (6) can be used directly to define $ICPR_3$. To avoid over-influence of outliers, the median is used as a measure of central tendency.

$$ICPR_3 = \text{median}\{\frac{\text{effort}(c)}{\text{predictedEffort}(c)} \mid c \in Pl\} \quad (12)$$

A two-sided sign test is used to assess whether actual change effort is higher (or lower) than the hypothetical change effort in more cases than expected from chance. This corresponds to testing whether the indicator is statistically different from 1.

2.4.4 Benchmarking

$ICPR_4$ compares developers' estimates for 16 benchmark change tasks between *P0* and *P1*. Assuming the developers' estimation accuracy does not change between the periods, a systematic change in the estimates for the same change tasks would mean that the productivity with respect to these change tasks had changed. Effort estimates made by developers *D* for benchmarking tasks $\mathbf{C_b}$ in periods *P1* and *P0* therefore give rise to the following indicator:

$$ICPR_4 = \text{median}\{\frac{\text{estEffort}(Pl, d, c)}{\text{estEffort}(P0, d, c)} \mid c \in C_b, d \in D\} \quad (13)$$

A two-sided sign test determines whether estimates in *P0* were higher (or lower) than the estimates in *P1* in more cases than expected from chance. This corresponds to testing whether the indicator is statistically different from 1.

Controlled studies show that judgement-based estimates can be unreliable, i.e. that there can be large random variations in estimates by the same developer [21]. Collecting more estimates reduces the threat implied by random variation. The available time for the benchmarking session allowed us to collect 48 (RCN – three developers) and 64 (MT – four developers) pairs of estimates.

One source of change in estimation accuracy over time is that developers may become more experienced, and hence provide more realistic estimates. For project RCN, it was possible to evaluate this threat by comparing the estimation bias for *actual changes* between the periods. For project MT, we did not have enough data about estimated change effort for real change tasks, and we could not evaluate this threat.

Other sources of change in estimation accuracy between the sessions are the context for the estimation, the exact instructions and procedures, and the mental state of the developers. While impossible to control perfectly, we attempted to make the two benchmarking sessions as identical as possible, using the same, precise instructions and material. The developers were led to a consistent (bottom-up) approach by our instructions to identify and record affected parts of the system before they made each estimate.

Estimates made in P1 could be influenced by estimates in P0 if developers remembered their previous estimates. After the session in P1, the feedback from all developers was that they did not remember their estimates or any of the tasks.

An alternative benchmarking approach is comparing change effort for benchmark tasks that were actually completed by the developers. Although intuitively appealing, the analysis would still have to control for random variation in change effort, outcomes beyond change effort, representativeness of change tasks, and also possible learning effects between benchmarking sessions.

In certain situations, it would even be possible to compare change effort for change tasks that recur naturally during maintenance and evolution (e.g., adding a new data provider to a price aggregation service). Most of the threats mentioned above would have to be considered in this case, as well. We did not have the opportunities to use these indicators in our study.

2.5 Accounting for Changes in Quality

Productivity analysis could be misleading if it does not control for other outcomes of change tasks, such as the change task's effect on system qualities. For example, if more time pressure is put on developers, change effort could decrease at the expense of correctness. We limit this validation to a comparison of the amount of corrective and non-corrective work between the periods. The evaluation assumes that the change task that introduced a fault was completed within the same period as the task that corrected the fault. Due to the short release-cycle and half-year leap between the end of *P0* and the start of *P1*,

we are confident that change tasks in *P0* did not trigger fault corrections in *P1*, a situation that would have precluded this evaluation.

3 Results and Validation

The indicator values with associated p-values are given in Table 2.

Table 2. Results for the indicators

	RCN		MT	
Indicator	Value	p-value	Value	p-value
$ICPR_1$	0.81	0.92	1.50	0.21
$ICPR_2$	0.90	0.44	1.50	0.054
$ICPR_3$	0.78	<0.0001	1.18	0.85
$ICPR_4$	1.00	0.52	1.33	0.0448

For project RCN, the analysis of real change tasks indicate that productivity increased, since between 10 and 22% less effort was required to complete change tasks in *P1*. $ICPR_4$ indicates no change in productivity between the periods. The project had refactored the system throughout the fall of 2008 as planned. Overall, the indicators are consistent with the expectation that the refactoring initiative would be effective. Furthermore, the subjective judgment by the developers was that the goal of the refactoring was met, and that change tasks were indeed easier to perform in *P1*.

For project MT, the analysis of real change tasks ($ICPR_1$, $ICPR_2$ and $ICPR_3$) indicate a drop in productivity, with somewhere between 18 and 50% more effort to complete changes in *P1* compared with *P0*. The indicator that uses benchmarking data ($ICPR_4$) supports this estimate, being almost exactly in the middle of this range. The project manager in MT proposed post-hoc explanations as to why productivity might have decreased. During *P0*, project MT performed most of the changes under fixed-price contracts. In *P1*, most of the changes were completed under time-and material contracts. The project manager indicated that the developers may have experienced more time pressure in *P0*.

As discussed in Section 2.5, the indicators only consider trends in change effort, and not trends in other important outcome variables that might confound the results, e.g., positive or negative trends in quality of the delivered changes. To assess the validity of our indicators with respect to such confounding effects, we compared the amount of corrective versus non-corrective work in the periods. For MT, the percentage of total effort spent on corrective work dropped from 35.6% to 17.1% between the periods. A plausible explanation is that the developers, due to less time pressure, expended more time in *P1*

ensuring that the change tasks were correctly implemented. So even though the productivity indicators suggest a drop, the correctness of changes was also higher. For RCN, the percentage of the total effort spent on corrective work increased from 9.7% to 15%, suggesting that increased productivity was at the expense of slightly lesser quality.

3.1 Validation of $ICPR_1$

The distribution of change effort in the two periods is shown in Figure 1 (RCN) and Figure 2 (MT). The square boxes include the mid 50% of the data points. A log scale is used on the y-axis, with units in hours. Triangles show outliers in the data set.

For RCN, the plots for the two periods are very similar. The Hodges-Lehmann estimate of difference between two data sets (8) is 0, and the associated statistical test does not indicate a difference between the two periods. For MT, the plots show a trend towards higher change effort values in *P1*. The Hodges-Lehmann estimate is plus one hour in *P1*, and the statistical test showed that the probability is 0.21 that this result was obtained by pure chance.

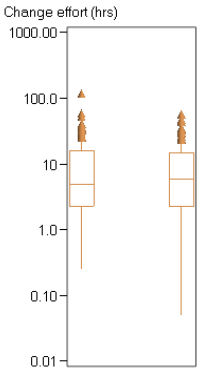


Figure 1. Change effort in RCN, *P0* (left) vs. *P1*

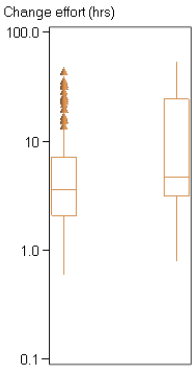


Figure 2. Change effort in MT, *P0* (left) vs. *P1*

If there were systematic differences in the properties of the change tasks between the periods, $ICPR_1$ can be misleading. This was assessed by comparing values for variables that capture certain important properties. The results are shown in Table 3 and Table 4. The Wilcoxon rank-sum test determined whether changes in these variables were statistically significant. In the case of *isCorrective*, the Fischer's exact test determined whether the proportion of corrective change tasks was significantly different in the two periods.

For RCN, *chLoc* significantly increased between the periods, while there were no statistically significant changes in the values of other variables. This indicates that larger changes were completed in *P1*, and that the indicated gain in productivity is a conservative estimate

For MT, *crTracks* significantly decreased between *P0* and *P1*, while *addCC* and *components* increased in the same period. This indicates that more complex changes were completed in *P1*, but that there was less uncertainty about requirements. Because these effects counteract, it cannot be determined whether the value for $ICPR_1$ is conservative. This motivates the use of $ICPR_2$ and $ICPR_3$, which explicitly control for changes in the mentioned variables.

Table 3. Properties of change tasks in RCN

Variable	*P0*	*P1*	p-value
chLoc (mean)	26	104	0.0004
crWords (mean)	107	88	0.89
filetypes (mean)	2.7	2.9	0.50
isCorrective (%)	38	39	0.90

Table 4. Properties of change tasks in MT

Variable	*P0*	*P1*	p-value
addCC (mean)	8.7	44	0.06
components (mean)	3.6	7	0.09
crTracks (mean)	4.8	2.5	<0.0001
systExp (mean)	1870	2140	0.43

3.2 Validation of $ICPR_2$

$ICPR_2$ is obtained by fitting a model of change effort on change task data from *P0* and *P1*. The model includes a binary variable representing period of change (*inP1*) to allow for a constant proportional difference in change effort between the two periods. The statistical significance of the difference can be observed directly from the p-value of that variable. The fitted regression expressions for RCN and MT were:

$$\log(\text{effort}) = 9.5 + 0.0018 \cdot \text{crWords} + 0.2258 \cdot \text{filetypes} + 0.00073 \cdot \text{changed} - 0.79 \cdot \text{isCorrective} - 0.10 \cdot \text{inP1}. \quad (14)$$

$$\log(\text{effort}) = 9.1 + 0.088 \cdot \text{crTracks} + 0.0041 \cdot \text{addCC} + 0.098 \cdot \text{components} - 0.00013 \cdot \text{systExp} + 0.40 \cdot \text{inP1}. \quad (15)$$

The p-value for *inP1* is low (0.054) for MT and high (0.44) for RCN. All the other model variables have p-values lower than 0.05. For MT, the interpretation is that when these model variables are held constant, change effort increases by 50% ($e^{0.40}$=1.50). A plot of deviance residuals in Figure 3 and Figure 4 is used to assess whether the modelling

framework (GLM with gamma distributed change effort and log link function) was appropriate. If the deviance residuals increase with higher outcomes (overdispersion) the computed p-values would be misleading. The plots show no sign of overdispersion. This validation increases the confidence in this indicator for project MT. For project RCN, the statistical significance is too weak to allow confidence in this indicator alone.

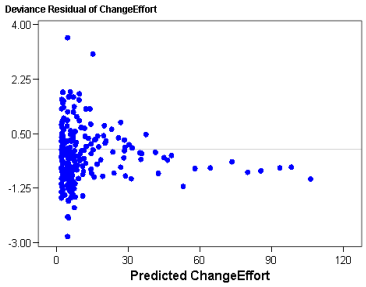


Figure 3. Residual plot for RCN model (14)

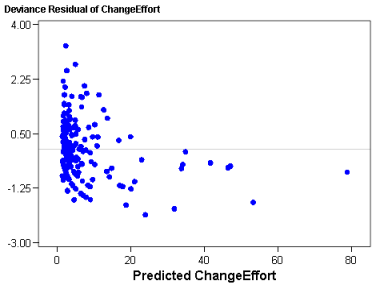


Figure 4. Residual plot for MT model (15)

3.3 Validation of $ICPR_3$

$ICPR_3$ compares change effort in *P1* with the model-based estimates for the same change tasks had they been completed in *P0*. The model was fitted on data from *P0*. Figure 5 shows that actual change effort tends to be higher than estimated effort for MT, while the tendency is opposite for RCN. For RCN, the low p-value shows that that actual change effort is systematically lower than the model-based estimates. For project MT, the high p-value means that actual effort was not systematically higher.

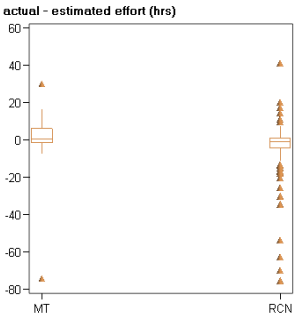


Figure 5. Model estimates subtracted from actual effort

If the variable subset is overfitted to data from P0, the model-based estimates using data from P1 can be misleading. To evaluate the stability of the model structure, we compared the model residuals in the *P0* model with those in a new model fitted on data from *P1* (using the same variable subset). For MT, the model residuals were systematically larger (Wilcoxon rank-sum test, p=0.0048). There was no such trend for RCN (Wilcoxon rank-sum test, p=0.78), indicating a more stable model structure.

Another possible problem with $ICPR_3$ is that model estimates can degenerate for variable values poorly covered by the original data set. Inspection of the distributions for the independent variables showed that there was a potential problem with the variable *chLoc*, also indicated by the large difference in mean, shown in Table 3. We re-calculated $ICPR_3$ after removing the 10 data points that were poorly covered by the original model, but this did not affect the value of the indicator.

In summary, the validation for $ICPR_3$ gives us high confidence in the result for project RCN, due to high statistical significance, and evidence of a stable underlying model structure. For project MT, the opposite conclusion applies.

3.4 Validation of $ICPR_4$

$ICPR_4$ is obtained by comparing the estimates that were made in the benchmarking sessions in *P0* and *P1*. Figure 6 shows that for project MT, the estimates tended to be higher in *P1* than in *P0*. For project RCN, there was no apparent difference.

A two-sided sign determines whether the differences are positive or negative in more cases than could be expected by pure chance. For project MT, the low p-value shows that estimates in *P1* are systematically higher than estimates in *P0*. For project RCN, the high p-value means that estimates in *P1* were not systematically different from in *P0*.

A change in estimation accuracy constitutes a threat to the validity of $ICPR_4$. For example, if developers tended to underestimate changes in *P0*, experience may have taught them to provide more relaxed estimates in *P1*. Because this would apply to real change tasks as well, we evaluated this threat by comparing estimation accuracy for real changes between the periods. The required data for this computation (developers' estimates and actual change effort) was only available for RCN. Figure 7 shows a difference in estimation bias between the periods (Wilcoxon rank-sum test, p=0.086).

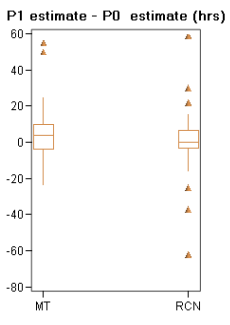


Figure 6. Differences in estimates

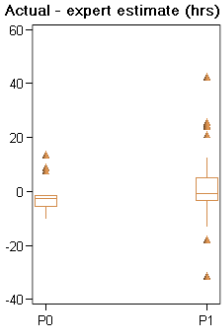


Figure 7. RCN: Estimates subtracted from actual effort

Changes tended to be overestimated in *P0* and underestimated in *P1*. Hence, the developers became more optimistic, indicating that $ICPR_4$ can be biased towards a higher value. This agrees with the results for the other indicators.

In summary, the benchmarking sessions supported the results from data on real change tasks. An additional result from the benchmarking session was that uncertainty estimates consistently increased between the periods in both projects. The developers explained this result by claiming they were more realistic in their assessments of uncertainty.

4 Discussion

The described approach to measuring productivity of software processes has some notable features compared with earlier work in this area. First, rather than searching for generally valid indicators of productivity, we believe it is more realistic to devise such indicators within more limited scopes. Our indicators target situations of software evolution where comparable change tasks are performed during two time intervals that are subject to the assessment. Second, rather than attempting to assess general validity, we believe it is more prudent to integrate validation procedures with the indicators. Third, our indicators are flexible within the defined scope, in that the structure of the underlying change effort models can vary in different contexts.

In a given project context, it may not be obvious which indicator will work best. Our experience is that additional insight was gained about the projects from using and assessing several indicators. The three first indicators require that data on change effort from individual change tasks is available. The advantage of $ICPR_1$ is that data on change effort is the *only* requirement for data collection. The caveat is that additional quantitative data is needed to assess the validity of the indicator. If this data is not available, a development organization may choose to be more pragmatic, and make qualitative judgments about potential differences in the properties of change tasks between the periods.

$ICPR_2$ and $ICPR_3$ require projects to collect data about factors that affect change effort, and that statistical models of change effort are established. To do this, it is essential to track relationships between change requests and code changes committed to the version control system. An advantage of $ICPR_3$ is that any type of prediction framework can be used to establish the initial model. For example, data mining techniques such as decision trees or neural networks might be just as appropriate as multiple regression. Once the model is established, spreadsheets can be used to generate the estimates, construct the indicator and perform the associated statistical test.

$ICPR_2$ relies on a statistical regression model fitted on data from the periods under consideration. This approach better accounts for underlying changes in the cost drivers between the periods, than does $ICPR_3$. In organizations with a homogenous process and a large amount of change data, the methodology developed by Graves and Mockus could be used to construct the regression model [17]. With their approach, data on development effort need only be available on a more aggregated level (e.g., monthly), and relationships between change requests and code commits need not be explicitly tracked.

$ICPR_4$ most closely approximates the hypothetical measure of comparing change effort for identical change tasks. However, it can be difficult to design benchmarking tasks that resemble real change tasks, and to evaluate whether changes in estimation accuracy have affected the results. If the benchmarking sessions are organized frequently, developers' recollection of earlier estimates would constitute a validity threat.

As part of our analysis, we developed a collection of scripts to retrieve data, construct basic measures and indicators, and produce data and graphics for the evaluation. This means that it is straightforward and inexpensive to continue to use the indicators in the studied projects. It is conceptually straightforward to streamline the scripts so that they can be used with other data sources and statistical packages.

5 Conclusions

We conducted a field study in two software organizations to measure productivity changes between two time periods. Our perspective was that productivity during software evolution is closely related to the effort required to complete change tasks. Three of the indicators used the same data from real change tasks, but different methods to control for differences in the properties of the change tasks. The fourth indicator compared estimated change effort for a set of benchmarking tasks designed to be representative of real change tasks.

The indicators suggested that productivity trends had opposite directions in the two projects. It is interesting that these findings are consistent with major changes and events in the two projects: Between the measured periods, the project with the indicated higher productivity performed a reorganization of their system with the goal of simplifying further maintenance and evolution. The project with indicated lower productivity had changed from fixed-price maintenance contracts to time-and-material contracts, which may have relaxed the time pressure on developers.

The paper makes a contribution towards the longer term goal of using methods and automated tools to assess trends in productivity during software evolution. We believe such methods and tools are important for software projects to assess and optimize development practices.

Acknowledgement

We thank Esito AS and KnowIT Objectnet for providing us with high quality empirical data, and the Simula School of Research and Innovation for funding the research.

References

[1] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does Code Decay? Assessing the Evidence from Change Management Data," *IEEE Transactions on Software Engineering,* vol. 27, pp. 1-12, 2001.

[2] T. DeMarco and T. Lister, "Human Capital," in *Peopleware. Productive Projects and Teams* New York: Dorset House Publishing, 1999, pp. 202-208.

[3] T. Mens and T. Tourwé, "A Survey of Software Refactoring," *IEEE Transactions on Software Engineering,* vol. 30, pp. 126-139, 2004.

[4] T. Dybå, E. Arisholm, D. I. K. Sjøberg, J. E. Hannay, and F. Shull, "Are Two Heads Better Than One? On the Effectiveness of Pair Programming," *IEEE Software,* vol. 24, pp. 12-15, 2007.

[5] G. L. Tonkay, "Productivity," in *Encyclopedia of Science & Technology*: McGraw-Hill, 2008.

[6] N. E. Fenton and S. L. Pfleeger, "Measuring Productivity," in *Software Metrics, a Rigorous & Practical Approach*, 1997, pp. 412-425.

[7] J. F. Ramil and M. M. Lehman, "Cost Estimation and Evolvability Monitoring for Software Evolution Processes," in *Workshop on Empirical Studies of Software Maintenance*, San Jose, CA, USA, 2000.

[8] A. Abran and M. Maya, "A Sizing Measure for Adaptive Maintenance Work Products," in *International Conference on Software Maintenance*, Nice, France, 1995, pp. 286-294.

[9] A. J. Albrecht and J. E. Gaffney Jr, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering,* vol. 9, pp. 639-648, 1983.

[10] M. Maya, A. Abran, and P. Bourque, "Measuring the Size of Small Functional Enhancements to Software," in *6th International Workshop on Software Metrics*, 1996.

[11] T. DeMarco, "An Algorithm for Sizing Software Products," *ACM SIGMETRICS Performance Evaluation Review,* vol. 12, pp. 13-22, 1984.

[12] J. F. Ramil and M. M. Lehman, "Defining and Applying Metrics in the Context of Continuing Software Evolution," in *Software Metrics Symposium*, London, 2001, pp. 199-209.

[13] A. Abran and H. Hguyenkim, "Measurement of the Maintenance Process from a Demand-Based Perspective," *Journal of Software Maintenance: Research and Practice,* vol. 5, pp. 63-90, 1993.

[14] H. D. Rombach, B. T. Ulery, and J. D. Valett, "Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL," *Journal of Systems and Software,* vol. 18, pp. 125-138, 1992.

[15] G. E. Stark, "Measurements for Managing Software Maintenance," in *1996 International Conference on Software Maintenance*, 1996, pp. 152-161.

[16] E. Arisholm and D. I. K. Sjøberg, "Towards a Framework for Empirical Assessment of Changeability Decay," *Journal of Systems and Software,* vol. 53, pp. 3-14, 2000.

[17] T. L. Graves and A. Mockus, "Inferring Change Effort from Configuration Management Databases," in *5th International Symposium on Software Metrics*, 1998, pp. 267–273.

[18] B. Kitchenham and E. Mendes, "Software Productivity Measurement Using Multiple Size Measures," *IEEE Transactions on Software Engineering,* vol. 30, pp. 1023-1035, 2004.

[19] K. Schwaber, "Scrum Development Process," in *10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications*, Austin, Texas, USA, 1995, pp. 117-134.

[20] H. C. Benestad, B. Anda, and E. Arisholm, "Technical Report 02-2009: An Investigation of Change Effort in Two Evolving Software Systems," Simula Research Laboratory Technical report 01/2009, 2009.

[21] S. Grimstad and M. Jørgensen, "Inconsistency of Expert Judgment-Based Estimates of Software Development Effort," *Journal of Systems and Software,* vol. 80, pp. 1770-1777, 2007.

Paper 4:

Using Planning Poker for Combining Expert Estimates in Software Projects

Kjetil Moløkken-Østvold, Nils Christian Haugen, Hans Christian Benestad

Journal of Systems and Software, Vol. 81, Issue 12, pp. 2106-2117, 2008.

Abstract

When producing estimates in software projects, expert opinions are frequently combined. However, it is poorly understood whether, when, and how to combine expert estimates. In order to study the effects of a combination technique called planning poker, the technique was introduced in a software project for half of the tasks. The tasks estimated with planning poker provided: 1) group consensus estimates that were less optimistic than the statistical combination (mean) of individual estimates for the same tasks, and 2) group consensus estimates that were more accurate than the statistical combination of individual estimates for the same tasks. For tasks in the same project, individual experts who estimated a set of control tasks achieved estimation accuracy similar to that achieved by estimators who estimated tasks using planning poker. Moreover, for both planning poker and the control group, measures of the median estimation bias indicated that both groups had unbiased estimates, because the typical estimated task was perfectly on target. A code analysis revealed that for tasks estimated with planning poker, more effort was expended due to the complexity of the changes to be made, possibly caused by the information provided in group discussions.

1 Introduction

In the software industry, various techniques are used to combine estimates. One of the most recent additions is *planning poker*, introduced by Grenning in 2002 [1] and also described in a popular textbook on agile estimating and planning by Mike Cohn [2]. There

exist few empirical studies on the combining of estimates in software engineering, but there are some indications that combination may reduce the bias towards optimism in estimates [3].

However, plenty of evidence from other research communities details the possible hazards of group processes [4]. For example, some recent papers published in psychology and forecasting emphasize the problems of decision-making in groups [5, 6]. Bueler et al. found that a) both individual and group predictions had an optimistic bias, b) group discussion increased individual biases, and c) this increase of bias in groups was mediated by the participants' focus on factors that promote the successful completion of tasks [6]. Scott Armstrong states that he has been unable to obtain evidence that supports the use of face-to-face groups in decision making [5].

These recent observations are in line with many previous classic studies on decision making in groups; individuals are inherently optimistic and this optimistic bias is increased by group interaction [4, 7, 8].

In contrast to the foregoing, our studies on software expert estimates have found that individuals are, in general, biased towards optimism, but that this bias can actually be *reduced* by group discussions [3].

The explanation for this apparent disparity may be that there are differences between a typical software estimation task and the tasks studied in other research areas. First, other studies frequently use ad hoc groups (e.g. [6]), whereas software estimation usually involves professionals who are accustomed to collaborating with each other and are motivated to perform in a professional manner [3]. Second, other studies tend to use tasks of which kind the participants might have little experience [4], whereas in software projects the participants are usually experienced. Third, another oft-reported problem is that laboratory studies tend to investigate hypothetical task and/or outcomes [4], and not real executed tasks with a recorded outcome.

Table 1. Overview of some common combination techniques

Method	Structure	Anonymity	Interaction	Overhead
Delphi	Heavy	Yes	No	Major
Wideband Delphi	Moderate	Limited	Limited	Moderate
Planning Poker	Light	No	Yes	Limited
Unstructured groups	Light	No	Yes	Limited
Statistical groups	Light	Yes	No	Limited
Decision markets	Heavy	Yes	No	Moderate

It might be that estimating software projects is a type of task that it is, for some reason, sensible to discuss in groups. However, it might also be that previous studies in software engineering have had methodological shortcomings.

The purpose of this study was to 1) explore the group processes that may occur when planning poker is used to estimate tasks, and 2) compare planning poker estimates with existing individual estimation methods. Section 2 introduces group estimation. In Sections 3 and 4 respectively, research questions and methods are described. The results are presented in Section 5 and discussed in Section 6. Section 7 concludes.

2 Combining Estimates in Groups

Group estimation has not been widely studied in a software engineering context. In fact, a recent review [3] of the leading journals in the systems and software engineering field did not find a single paper that described empirical studies of group estimates in an industrial context. Since that review, at least two studies have been published, one of which compared individual expert estimates (combined in statistical groups) with an unstructured group estimation method [3] and the other of which compared unstructured group estimates with planning poker [9]. In addition, the combining of estimates has been studied in student tasks [10, 11].

Various techniques can be used to combine estimates. A simplified overview of six of the most common techniques, including what we perceive to be central properties, is displayed in Table 1.

Structure describes the level of formality, amount of learning requirements, and degree of rigidity associated with the technique. *Anonymity* describes whether the estimators are anonymous to each other. *Interaction* describes whether, and if so to what extent, the estimators interact with each other. *Overhead* describes the typical extra amount of effort spent on estimating each project or task.

Perhaps the most well-known technique for combination is the Delphi technique [12], which was devised by the RAND corporation in the 1950s [13]. The Delphi technique does not involve face-to-face discussions, but anonymous expert interaction through several iterations, supervised by a moderator until a majority position is attained. In addition to *anonymity,* the method needs to include *iterations*, *controlled feedback* and *statistical aggregation of responses* for it to be implemented properly [13].

It is claimed that even though the technique has been used widely, actual scientific studies of the techniques' merits are sparse and often conducted inappropriately [13, 14]. However, even though reviews advise caution, there is evidence that the Delphi technique outperforms statistical groups and unstructured interacting groups [13] and that it is a sound method for harnessing the opinions of a diverse group [14]. However, there is no conclusive evidence that Delphi outperforms other structured group combination techniques.

We have found no empirical research on the accuracy of Delphi in a software engineering context. However, it is frequently recommended in papers on software management, e.g. [15].

The Wideband Delphi technique is a modification of the Delphi technique and includes more group interaction than Delphi [16]. As in the Delphi technique, there is a moderator, who supervises the process and collects estimates. However, in this approach the experts meet for group discussions both prior to, and during, the estimation iterations.

The Wideband Delphi technique is very similar to the Nominal Group technique, which is also know as the estimate-talk-estimate technique [17]. Due to its similarities to the Wideband Delphi technique, the Nominal Group technique is neither presented nor discussed in this paper.

Wideband Delphi has been proposed as an estimation method in books [16], and papers on software metrics [18] and software process improvement [19]. To the best of our knowledge, the Wideband Delphi technique has not been studied empirically.

The planning poker technique is relatively new. It is a lightweight technique, with face-to-face interaction and discussions. In short, the steps of the technique, as originally described by Grenning, are: "*The customer reads a story. There is a discussion clarifying the story as necessary. Each programmer writes their estimate on a note card without discussing their estimate. Anticipation builds. Once all programmers have written their estimate, turn over all the cards. If there is agreement, great, no discussion is necessary, record the estimate and move on to the next story. If there is disagreement in the estimates, the team can then discuss their different estimates and try to get to consensus* [1]". By *story*, Grenning means a *user story.* A user story is a software system requirement that is formulated as one or two sentences in the everyday language of the user. The technique, and how it was adopted to the project studied, will be described in greater detail in Section 4.

Being a relatively new technique, planning poker has, as far as we are aware, been the subject of only one published empirical study [9]. In that study, planning poker was compared to unstructured group estimation. It was found that for familiar tasks, the planning poker technique produced more accurate estimates than unstructured combination, whereas the opposite was found for unfamiliar tasks.

Unstructured group combination is, as the name implies, basically discussions with a group decision being made at the end. Depending on needs, individuals can derive their own estimates before the discussion.

A review of the literature on forecasting [20] suggests that unstructured groups were, on average, outperformed by Delphi-groups. However, the review also found that there are tasks for which unstructured groups are better suited. In some situations, it is possible that an unstructured group can outperform a Delphi group if the motivation of, and information sharing among, the participants is adequate [20].

In a previous study on software estimation [3], we found that group estimates made after an unstructured discussion were less optimistic and more realistic than individual estimates derived prior to the group discussion and combined in a statistical group. The main sources of this decrease in optimism seemed to be the identification of additional activities and an awareness that activities may be more complicated than was initially thought.

Note that that study used an unstructured technique that involved prior individual estimates. Often, companies use an unstructured combination where experts meet to provide consensus estimates, without having previously made their individual estimates. This latter procedure is perhaps more susceptible to peer-pressure than when individual estimates have been derived initially.

In a statistical group, there is no interaction between the group members. They are a group only in the sense that their individual estimates are combined statistically.

When considering how to combine estimates given by several individuals into an estimate, well-known statistical methods can be used. Computing the mean or median of the different individual estimates will give us one estimate that is based on multiple estimates.

Jørgensen claims that taking a simple average often works as the best method for combining estimates [21].

A decision market is a technique for combining opinions that can also be used in estimation. Hanson provides the following definition: *"Decision markets are (markets)*

designed primarily for the purpose of using the information in market values to make decisions [22]*".*

A decision market can be set up like a stock market, with decisions being substituted for stocks. Traders are invited to invest money in the alternative, represented by stocks (decisions), that they think will be the eventual outcome. A trader holding a stock (decision) that becomes the actual outcome receives a fixed amount of money, prize or similar. Through the dynamics of a market, this results in higher stock (decision) prices for the alternatives that most people think will be the outcome, which creates a likelihood distribution for the different outcomes.

According to Surowiecki, such a market is wise because it aggregates the opinions of traders. A market may be especially powerful if the traders are diverse in their backgrounds, independent of each other, and have local knowledge [23].

Inspired by Surowiecki, Berndt, Jones et al. advocate the use of decision markets in software effort estimation [24]. They stress that by allowing all project stakeholders to participate in the decision market, one ensures diversity in the input to the estimation process and aggregates the knowledge from all the project stakeholders. According to Berndt, Jones et al., another positive feature of decision markets is that the different traders can apply whatever estimation technique they like, thus enabling a combination of different estimation techniques.

A decision market is, as is Delphi, a way of aggregating different opinions without face-to-face meetings. Like Delphi, a decision market seeks to preempt the social and political problems caused by the use of interacting groups, while at the same time utilizing the increase in knowledge that using groups offers. An important factor is that the participants can receive (continuous) feedback on their own opinion compared to others.

The main difference between Delphi and decision markets is the way in which the knowledge and opinions of the group members are aggregated.

We have not managed to find any empirical research on the use of decision markets for software estimates. However, a recent paper by Berndt, Jones et al. describes an ongoing study [24].

Studies on the combining of estimates for student tasks have shown some positive effects, both when combining estimates statistically [11] and in face-to-face discussions [10].

To summarize, the strategy of combining estimates for groups in general, and for software estimation in particular, is far from understood.

In addition, some studies, e.g., by Buehler et al., specify some limitations that may reduce the strategy's applicability to real life problems, mainly that the groups studied consisted of individuals who were unfamiliar with each other [6].

It is also important to note that findings may vary in their applicability as task characteristics, motivational factors, social relations, and communication structure differ. In general, when reviewing studies on group processes, it is also important to differentiate between studies in which the participants cannot influence the outcome and those in which they can.

3 Research Questions

It is possible that typical software estimation tasks are suitable for group combination, such as planning poker. In an estimation process, there may be several experts who contribute different project experiences and knowledge. Such experiences can be shared more easily in a face-to-face group, as with planning poker, than through a moderator, as with the Delphi technique. In addition, face-to-face interaction may make the participants more committed to the decisions.

We wanted to further explore whether, as found in a previous study on group estimation [3], optimism could be reduced by group discussions. From this, we derived the following research question:

> ***RQ1:*** *Are group consensus estimates less optimistic than the statistical combination of individual expert estimates?*

We define optimism of estimates in the relative sense, and irrespective of accuracy. We deem one estimate to be more optimistic than another if and only if it states that it will take less time to complete a task than the other estimate. I.e., an estimate of 4 hours is more optimistic than an estimate of 5 hours.

Any observation of reduced optimism would indicate a *choice-shift*, defined by Zuber, Crott et al. as the difference between the arithmetic average (mean) of individual decisions and the group consensus decision [25]. Such an observation of reduced optimism would be contrary to that which is typically reported from other research areas, where the choice-shift is generally in the direction of increased risk willingness and optimism [6].

It is important to note that our study does not merely confirm or undermine the results of the previous study on group estimation [3]. It also generates a new result, because the

subjects used planning poker rather than the unstructured discussion used in the previous study.

Even if a reduction in optimism were observed, it would not necessarily guarantee that estimates would be more accurate, because some or all of the individual estimates might already be biased towards pessimism. Thus, we also wanted to investigate whether group consensus estimates made after a discussion are more accurate than mean individual estimates. This concern generated the following research question:

RQ2: *Are group consensus estimates more accurate than the statistical combination of individual expert estimates?*

The previous empirical study on planning poker compared planning poker with an unstructured method for combining estimates in groups [9]. Therefore, we wanted to compare the estimates that were derived by using planning poker to a series of estimates that were derived by individual experts, and not subject to subsequent group discussions. This generated the following research question:

RQ3: *Are group consensus estimates more accurate than the existing individual estimation method?*

In addition to possibly influencing estimation accuracy, the introduction of group estimation might lead to changes in how the developers work. Such changes might include differences in the amount of total effort spent on the estimation phase, or effort spent on restructuring code during implementation. The final research question to explore is therefore:

RQ4: *Does the introduction of a group technique for estimation affect other aspects of the developers' work, when compared to the individual estimation method?*

The sizes of changes have been shown to be fundamental in explaining change effort variations; see, e.g., [26]. It is therefore necessary to explore:

RQ4A. *Are there differences between the planning poker tasks and the control group tasks that are related to the size of the changes?*

A larger change size for the planning poker tasks may, at least partly, explain any differences between the groups in actual effort, and provide an intermediate link for the causal analysis.

The other subquestion to investigate is:

RQ4B. *Are there differences between the planning poker tasks and the control group tasks that are related to the complexity of the changes?*

Conclusions drawn from the analysis for question 4B are tentative, because only a subset of possible factors was investigated. However, the analysis can provide partial evidence and insight that can be useful in a wider causal analysis in combination with the other study results. Thus, it is of interest to assess any difference in effort after controlling for possible differences in change size and complexity.

Research questions 1 and 2 concerns *intragroup* differences, while research questions 3, 4A and 4B concerns *intergroup* differences.

4 Research Method

The research method was designed to address some of the issues pertaining to validity that arose in our previous studies [3, 9].

The main limitation of the previous study of individual estimates (combined in statistical groups) followed by unstructured group combination was that the groups of professionals did not themselves implement the project they estimated [3]. This was done by a separate team; thus, the estimators did not estimate their own work. Therefore, from the perspective of the estimators in that study, there was a hypothetical outcome. However, as the project was actually implemented, it was possible to discern estimates that were clearly optimistic or pessimistic.

The previous study on planning poker was limited to one team [9]. Another limitation was an unknown effect of increased system experience, because there was no randomization of tasks to the different methods (unstructured group estimation vs. planning poker). In addition, the study compared two different group estimation methods (planning poker vs. unstructured groups), and did not compare group estimation with individual estimation.

In the study reported herein, we wanted a design that could both measure any shift in choices among the planning poker tasks and compare planning poker with a control group of tasks estimated with the existing individual estimation method. In addition, it was important that the design allowed for the comparison of estimates with the actual effort for all tasks.

We also wanted to perform an analysis of code following the completion of the tasks, to explore any possible differences related to the size or complexity of the changes. Finally,

we wanted to conduct face-to-face interviews with the participants, in order to further explore and explain possible findings.

A simplified overview of the study design is presented in Table 2.

Table 2. Study overview

	Planning poker	Control group
A	Tasks requested by client entered into the task tracking system and given an initial estimate	
B	Tasks assigned randomly to planning poker or control group	
C	Initial estimate discarded	Initial estimate kept
D	Task presented to team	N/A
E	Task discussed briefly	N/A
F	Individual estimates derived	N/A
G	Individual estimates revealed	N/A
H	Estimates discussed	N/A
I	Consensus estimate derived by group	Initial estimate used as estimate
J	Task performed	
K	Actual effort recorded	
L	Source-code analyzed	
M	Participants interviewed	

4.1 The Company and Project Studied

The company studied is a medium-size Norwegian software company that delivers custom-made solutions to various private and public clients. The project team studied had been working for a large public client for several months at the start of the study and was using Scrum (http://www.controlchaos.com/) as the project methodology. The project team estimates the tasks to be performed in the upcoming *sprint* (Scrum terminology for the next period). In the team studied, this happens once each fortnight, and about 15-20 tasks are selected for each sprint. From four to six team members participate in each sprint, depending on the demands of the tasks.

All project participants in the study were guaranteed anonymity and assured that no results regarding performance could be traced.

4.2 The Estimation Methods Studied

The existing estimation method of the project was that the tasks were estimated individually by the team member responsible for the part of the system that would be

affected by the change. Changes requested by the client were analyzed, estimated and recorded (Step A; Table 2) in their task tracking system, Jira (http://www.atlassian.com/software/jira/). The tasks were of varying size and character, ranging from two-hour bug fixes to three-day analyses of larger changes, and were relatively well defined in the task-tracking system, which used text and screenshots. All members of the team participated in estimating tasks. This was done individually and estimates were not revealed to other team members. This method was used in the control group in our study.

For each sprint, half of the tasks were to be re-estimated with a variation of planning poker, while the initial estimate was retained for the other half (Step C). The tasks were assigned randomly to either the planning poker or the control group (Step B).

Before the study, the team was given an introduction to planning poker by the company's chief scientist. The estimation method for the planning poker condition was employed with the following steps in sequence for each task:

- The task was presented to the team (Step D) by the developer who registered the task in the task tracking system. The initial estimate was not revealed to other team members and was discarded (Step C).
- The task was discussed briefly by the team (Step E), to ensure that everybody had the same interpretation before estimates were made.
- The team members then estimated, individually, the most likely effort needed to perform the task specified (Step F). The estimate was given in work-hours.
- All team members revealed their estimates simultaneously (Step G):
  - If any estimates were larger than 18 hours, there was a brief discussion of how to break the task down into subtasks. A full day of work was deemed to be six hours. From previous experience, the team felt that estimates larger than 18 hours (i.e. three days) were less accurate. Therefore, they decided to split tasks above this size.
  - Those with the lowest and highest estimates had to justify their estimates. A brief debate followed (Step H). The debate was led by the company's chief scientist for the first sprint; thereafter, the team worked on its own.
  - If a consensus was reached on an estimate, this was recorded (Step I) and the team moved on to the next task (Step C). If no consensus could be reached, the members revised their estimates and participated in a new individual estimation round for that particular task.

- After the task had been performed, the developer who performed the task recorded the actual effort expended, together with his or her initials (Step K).

Note that this method differs somewhat from the description given by Grenning [1]. For example, planning poker was, in our case, used for task estimation, and not estimation of user stories [2] or features (for which use it is most commonly recommended).

After all the tasks had been performed, the code for solutions in both study groups was analysed (Step L) and the participants were interviewed to get their opinions (Step M).

4.3 Calculation of Estimation Accuracy

To calculate estimation accuracy, we employed the BRE (Balanced Relative Error), because it is a more balanced measure than the MRE [27]. It is calculated as:

$$BRE = \frac{|x - y|}{\min(x, y)}, \text{ x=actual and y=estimate}$$

In order to measure whether there was a bias towards optimism or pessimism, the BREbias was calculated, because this measures both the size *and* the direction of the estimation error:

$$BREbias = \frac{(x - y)}{\min(x, y)}, \text{ x=actual and y=estimate}$$

To measure the size of any difference in mean values, we used Cohen's size of effect measure (d) [28], where

$$d = \frac{\overline{sample1} - \overline{sample2}}{pooledStdDev}$$

Here, pooledStDev denotes the *pooled standard deviation,* a method for assessing the true standard deviation of different samples.

4.4 Code analysis

Checkins to the code repository, Subversion (http://subversion.tigris.org), were tagged by the developers with a task identifier. Hence, we were able to retrieve the exact state of the application before and after each task, and quantitative measures of changes could be extracted. By studying the changes to the application code associated with the individual tasks, we were better prepared to investigate and discuss whether, and how, the estimation method may have influenced how effort was spent for each task.

More specifically, the code analysis sought to compare the amount and complexity of code that was added, changed, and removed during the tasks. This allowed a more focused root cause analysis of possible differences in change effort. Research question 4A was investigated by inspecting mean and median values for change size and performing a Kruskal-Wallis test [29] on the medians.

For both questions, regression models of change effort versus measures of change size and change complexity were used to gain insight into factors that explain change effort. When applying these models and measures, we used the same statistical framework as, and similar procedures to, those discussed and used by Graves and Mockus [26], who performed similar analyses of change effort. In brief, the framework utilizes Generalized Linear Models using change effort (measured in work-hours) as a dependent variable, measures of possibly influential factors as covariates, and a log link. It assumes Poisson distributed errors, and allows a free scale parameter to adjust for possible overdispersion, *c.f.* [30].

Although artifacts such as binaries, design models, and build scripts were present in the code repository, only *source code* was considered for this analysis. Source code included files for Java, Java Server Pages, the eXtensible Stylesheet Language, XML, and XML Schema Definitions.

Added, deleted and changed lines were measured by processing the side-by-side (-y option) output of the standard Linux program diff, c.f. [31, 32]. Frequently used measure of change size is the sum of these measures (SIZE1); see e.g., [33]. Graves and Mockus [26] evaluated several size measures and found that the number of file check-ins to the code repository (SIZE2) best explained change effort. We constructed a third measure, the number of changed segments of code (SIZE3). A changed segment of code is a set of consecutive lines of code, where all lines were either added, changed or deleted. The measures from the code repository were extracted automatically. Their definitions are provided in Table 3.

Table 3. Change measures

ADD	Number of source code lines added
CH	Number of source code lines changed
DEL	Number of source code lines deleted
SIZE1	ADD + CH + DEL
SIZE2	Number of file revisions checked in (deltas)
SIZE3	Number of changed segments of code

ACS	Number of control-flow statements added
DCS	Number of control-flow statements deleted
AOR	Number of out-of-class references added
DOR	Number of out-of-class referencesdeleted
SZAFF	Mean number of source code lines in affected modules
isControl	Binary variable representing group membership. Value set to 1 if task was in control group, 0 if task estimated by PP

In order to select the most appropriate size measure among the three candidates, the deviance of the three regression models of the type described above was compared. Lower deviance values indicate a better fit between model and the actual data [30]. A model based on SIZE3 and a mathematical intercept value provided the best fit and SIZE3 was selected as the size measure to use for the analysis; see Table 4.

Table 4. Model fit with alternative size measures

Variables	Coefficient	p-value	Deviance
Intercept	2.05	<0.0001	188
SIZE1	0.000738	0.0487	
Intercept	2.03	<0.0001	180
SIZE2	0.0203	0.0177	
Intercept	1.95	<0.0001	154
SIZE3	0.00793	0.0009	

We used three types of measures of change complexity. These were hypothesized to explain possible change effort variations: Measures of the size of the affected code, similar to SZAFF, have been found by other researchers to affect change effort significantly [34], [35].

Measures of the type of change, similar to ADD, DEL, CH were used by, e.g., Jørgensen [33]. Measures of additions and deletions of structural attributes (ACS, DCS, AOR, DOR) are less common, but have been investigated at the file level by Fluri and Gall [36]. An out-of-class reference means that the measured class uses a method or attribute in another class. A control-flow statement changes the sequential flow of control. Hence, the measures are similar to the concepts of import coupling [37] and cyclomatic complexity [38], but adapted to measuring complexity *change* at the task level.

4.5 Interviews

All project participants were interviewed individually on a range of issues. These interviews sought to a) uncover background information regarding project priorities, b) ask specific questions regarding the planning poker technique, and c) determine the

participants' perception of differences between the planning poker technique and the individual estimation technique.

The interviews were performed in person and followed a structured questionnaire.

5 Results

In total, 55 tasks were estimated and implemented, of which 24 were estimated with planning poker and 29 with the existing individual estimation method. Two tasks were deleted from the dataset due to suspicion of faulty registration in the database. A brief summary of important data is presented in Table 5. The first column represents the initial estimates (Step A; Table 2), the second the statistical combination of individual estimates for the planning poker tasks (Step F), the third the final estimates (consensus estimate for the planning poker tasks, individual estimate for the control group, Step I), the fourth column the actual effort (Step J), the fifth the estimation accuracy of planning poker tasks measured against the statistical combination of individual estimates, the sixth estimation accuracy against final estimates, and the final column contains estimation bias.

Table 5. Key results

		Initial estimate	Statistical comb. (hrs)	Estimate (hrs)	Actual effort (hrs)	BRE statistical comb.	BRE	BREbias
Planning poker (n=24)	Mean	6.6	6.3	7.1	10.4	0.94	0.82	0.33
	Median	5.0	6.0	6.0	8.0	0.56	0.50	0.00
Control group (n=29)	Mean	5.3		5.3	6.1		0.78	-0.04
	Median	4.0		4.0	4.0		0.33	0.00

The first two research questions relate only to the 24 tasks that were estimated with planning poker. The third research question compares accuracy results from the 24 tasks estimated by using planning poker with the 29 tasks estimated by using the existing individual estimation method. The final research question examines possible differences in change size and complexity.

5.1 RQ1: Are group consensus estimates less optimistic than the statistical combination of individual expert estimates?

For the 24 tasks that were estimated using planning poker, the mean of the statistical combination of individual estimates before group discussion was 6.3 hours (median 6.0 hours). After group discussion, the mean consensus estimate was 7.1 hours (median 6.0 hours). An analysis was performed with a paired t-test, as suggested in similar research on

choice shift [39]. Since the research question suggests a direction of effect (group discussion reduces optimism), the paired t-test was one-sided. We provide the actual p-values, as suggested by Wonnacott and Wonnacott [29], instead of predefining a significance level for rejection. The results are displayed in Table 6. The analysis of possible choice shift on the estimates yielded a p-value of 0.04 and an effect size of 0.16.

Table 6. RQ1 results

N	24
Statistical combination (mean hours)	6.3
Group consensus (mean hours)	7.1
Difference	0.8
PooledStDev	4.6
p-value	0.04
Size of effect (d)	0.16

5.2 RQ2: Are group consensus estimates more accurate than the statistical combination of individual expert estimates?

On the basis of the estimates, we calculated the estimation accuracy measured in BRE. The mean BRE of the statistical combination of individual estimates before group discussion was 0.94 (median 0.56). The mean BRE of the consensus estimates after group discussion was 0.82 (median 0.50). The calculation of the statistics followed the same procedure as for the previous research question. A summary is presented in Table 7.

The analysis of a possible difference in accuracy between the statistical combination of individual estimates and the group consensus estimates yielded a p-value of 0.07 and an effect-size of 0.11.

Table 7. RQ2 results

N	24
BRE Statistical combination (mean)	0.94
BRE Group consensus (mean)	0.82
Difference	0.12
Pooled StDev	1.02
p-value	0.07
Size of effect (d)	0.11

5.3 RQ3: Are group consensus estimates more accurate than the existing individual estimation method?

The mean BRE of the tasks completed using the existing individual estimation method was 0.78 (median 0.33), compared to a mean BRE of 0.82 (median 0.50) for the planning poker tasks. For the statistical test of difference between the two groups, a Kruskal-Wallis test [29] was performed on the medians. A summary for both groups is found in Table 8.

Table 8. RQ4 results

BRE planning poker group (mean, n=24)	0.82
BRE control group (mean, n=29)	0.78
Difference	0.04
Pooled StDev	1.22
p-value (Kruskal Wallis)	0.77
Size of effect (d)	0.03

The analysis of difference between the two study groups yielded a p-value of 0.77, and a size of effect of 0.03. Regarding any difference in estimation bias, the BREbias values are presented in Figure 1 (see also Table 5).

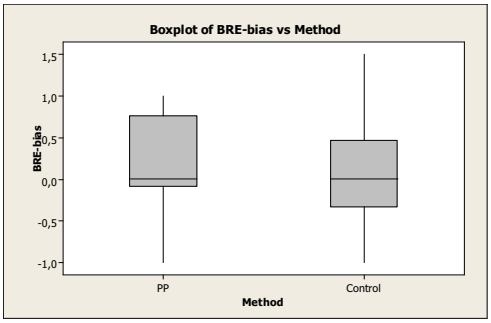


Figure 1. BREbias

An interesting finding emerges. The median BREbias for both the planning poker group and the control group is 0.00, which indicates that the typical case is estimated perfectly on target.

However, the mean BREbias of the planning poker tasks was 0.33, compared to -0.04 for the control group (see also Table 5).

5.4 RQ4: Does the introduction of a group technique for estimation affect other aspects of the developers' work, when compared to the individual estimation method?

Thirty-four of the 53 tasks studied involved changing code. For seven of the 34 tasks that involved code changes, the developers did not tag the associated code repository checkins. Interviews revealed that this could sometimes happen for minor changes. Hence, the analysis below is based on 27 valid data points. An overview of the results of the code analysis is presented in Table 9.

Table 9. Mean values of key measures compared

Measure	Mean value control	Mean value PP
Estimate	7.5	7.6
Actual	8.1	12.8
ADD	246	326
CH	23.5	36.0
DEL	66.5	50.9
SIZE1	336	413
SIZE2	17.0	12.3
SIZE3	45.3	39.9
ACS	21.6	14.4
DCS	14.1	6.8
AOR	209	206
DOR	65	50
SZAFF	224	151

The mean actual effort for both groups is somewhat higher for this subset than for the complete set of tasks, which is presented in Table 5. This is not surprising, because tasks that involve changes to the code are usually larger than other tasks. The mean actual effort for the planning poker tasks involving code change is 12.8 hours, compared to 10.4 hours for the complete planning poker subset. Similarly, the mean actual effort of the control group is 8.1 hours for the tasks involving code, compared with 6.1 hours for all control tasks.

Research question 4A asked**:** *Are there differences between the planning poker tasks and the control group tasks related to the size of the changes?*

The control group has a higher mean value for the SIZE3 measure than the planning poker group (see Table 9). Given this, it appears that the control group tasks took less time *and* were larger than the planning poker tasks. However, this difference in size is not statistically significant using a Kruskal-Wallis test (p=0.59). The individual data points of SIZE3 vs. change effort are depicted in Figure 2.

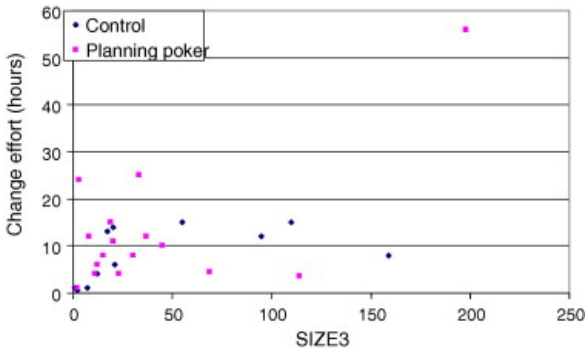


Figure 2. Change effort and size

In order to further explore research question 4A, we fitted two regression models of the type discussed in Section 4.4, one that included isControl (the group indicator, refer to Table 3) only, and one that added the size variable, SIZE3. The results are summarized in Table 10.

Table 10. Models of change effort, without and with size measure included as covariate

Variable	Coefficient	p-value	Deviance
Intercept	2.55	<0.0001	210
isControl	-0.449	0.25	
Intercept	2.12	<0.0001	139
SIZE3	0.00795	0.0004	
isControl	-0.482	0.10	

As can be seen, when accounting for size (the second model), the difference in change effort between the groups become clearer (the p-value of the isControl variable decreases), and is statistically significant at the 0.1 level. In other words, there is initial evidence that, after controlling for size, effort expended on tasks estimated by planning poker is greater than effort expended on tasks in the control group.

Returning to the summary statistics in Table 9, we observe that planning poker tasks involved more changes (CH) and fewer deletions (DEL) to existing code. These factors can account for the observed difference in change effort.

Research Question 4B asked: *Are there differences between the planning poker tasks and the control group tasks related to the complexity of the changes*?

To analyze possible differences in complexity, we entered all complexity measures into the model presented in Section 4.4., and applied a variable selection method called backward elimination to attempt to identify factors that explain change effort variations. The results are presented in the top half of Table 11.

Measures of changed lines (CH), added control statements (ACS), and deleted out-of-class references (DOR) contribute positively to change effort. The measure of deleted lines (DEL) contributes negatively to change effort. When entering these measures of change complexity into the model, the estimation method (isControl) was no longer related significantly to change effort.

Thus, there is no evidence that effort expended on tasks estimated by planning poker is greater than effort expended on tasks in the control group after controlling for change complexity.

Note that the measures selected by the backward elimination procedure may have been influenced by correlations between the measures. ACS is correlated heavily with AOR and DOR is correlated heavily with DCS; hence, it is likely that AOR and DCS will provide almost as good explanatory power as ACS and DOR. We confirmed this by refitting a model forcing in the AOR and DCS variables; see the results in the bottom half of Table 11.

Table 11. Models for change complexity

Variable	Coefficient	p-value	Deviance
Intercept	1.71	<0.0001	76
DEL	-0.0242	<0.0001	
CH	0.0193	<0.0001	
ACS	0.0184	0.016	
DOR	0.0166	0.0005	
Intercept	1.79	<0.0001	92
DEL	-0.017	0.0003	
CH	0.0198	<0.0001	
AOR	0.00136	0.0081	
DCS	0.0493	0.0018	

Given the above, we answer Research Question 4B positively: there are differences in change complexity between the planning poker tasks and the control tasks,

These differences in complexity might explain differences in effort. The underlying reasons for differences in complexity will be explored in the next section.

A final observation is that there is no evidence that the observed bias between the groups with respect to the mean size of affected modules resulted in differences in change effort.

5.5 Results from the participant interviews

The interviews provided information that will be used in the discussion section to try to explain the results presented above. The interviews focused on a) background information regarding project priorities, b) specific questions regarding planning poker, and c) the participants' perception of differences between the planning poker technique and the individual estimation technique.

The participants were asked to rate how they perceived the priorities of the project used in this study on a five-point Likert scale (1=very important, 2=important, 3=of medium importance, 4=somewhat important, 5=not important). The respondents were free to use their personal interpretation of parameters such as quality and functionality. The results are displayed in Table 12.

Table 12. Perceived importance of project parameters

Parameter	Mean	Median	*StDev*
Customer satisfaction	1.8	2.0	0.8
Functionality	2.0	2.0	0.6
Quality	2.2	2.0	0.8
Schedule	2.7	2.5	0.8
Effort	3.0	2.5	1.3

There are several reasons for why people change their opinion about the estimate of a task after group discussion [4]. Some of the more common reasons are a) pressure (direct or perceived) from seniors, b) new information revealed, or c) a desire for consensus.

The participants were asked to rate how much these reasons affected their estimates on a five-point Likert-scale (1=influence in all tasks, 2=influence in most tasks, 3=influence in about half of the tasks, 4=influence in some tasks, 5=influence in none of the tasks). The results are displayed in Table 13.

Table 13. Perceived influence when changing opinion

Parameter	Mean	Median	StDev
New information	1.8	2.0	0.4
Pressure from seniors	2.7	2.0	1.2
Desire for consensus	3.3	3.5	0.8

The participants were interviewed regarding possible differences induced by the planning poker technique when compared to the individual estimation method (control group). They were asked to rate their perception of whether, and if so how, several aspects of their work was influenced. This included both effort spent in various phases and suitability. They rated these aspects on a five-point Likert-scale (1=much more, 2=more, 3=similar, 4=less, 5=much less). The results are summarized in Table 14.

Table 14. Perceived differences of planning poker compared to control group tasks

Property	Rating (mean)	Median	StDev
Suitability for identifying task challenges	1.5	1.5	0.5
Suitability for identifying subtasks	1.7	1.5	0.8
Effort spent on estimation	1.8	2.0	0.4
Motivation to follow estimates	2.0	2.0	0.6
Estimation accuracy	2.2	2.0	0.8
Effort spent on analysis and design	2.8	2.5	1.0
Effort spent on refactoring of code	3.0	3.0	0.6
Effort spent on clarifying tasks during implementation (i.e. not including the estimation phase)	3.2	3.0	0.8

6 Discussion

In general, small differences in estimation accuracy were found between the groups, whether the comparison was between a statistical combination of individual estimates and group consensus estimates for the planning poker tasks (choice shift), or between planning poker tasks and the control group.

Interestingly, there appeared to be a difference between the planning poker tasks and the control tasks that was related to change size and change complexity.

6.1 RQ1: Are group consensus estimates less optimistic than the statistical combination of individual expert estimates?

When we looked in isolation at the tasks estimated with planning poker, the results indicated a slight shift in choices that showed a reduction of optimism after group discussion. For these tasks, there was an initial individual bias towards optimism, as in other studies [6]. However, in our study, this optimism was not increased by group discussion. Rather, we found the opposite, that optimism was reduced, as in a previous

study on software estimation [3]. However, note that the effect must be considered small (Cohen's d<0.20).

The more common reasons for a change of opinion regarding the estimate of a task following group discussion have already been noted. The results of the interviews presented in Section 5.5 show that the respondents rated new information as the most important reason for changing their minds (mean response 1.8, and standard deviation of 0.4). However, pressure (2.7) and desire for consensus (3.3) were also rated as important for changes of opinion for about half of the tasks.

6.2 RQ2: Are group consensus estimates more accurate than the statistical combination of individual expert estimates?

There were indications of a slight shift towards increased accuracy when comparing consensus estimates with the statistical combination of individual estimates. This comes as a direct function of an initial bias towards optimism in the individual estimates. When, as found with respect to RQ1, this optimism was reduced, accuracy increased. However, the size of effect is considered small (Cohen's d<0.20).

When exploring differences between a group's consensus estimate and the statistical combining of individual estimates, a relevant property is the initial (dis)agreement of the estimates for each task. This (dis)agreement can be measured by the standard deviation (StDev). For the 24 tasks estimated with planning poker, the StDev of the individual estimates varied from 0.00 to 6.65, and the median and mean StDev were 2.23 and 2.20 respectively.

However, initial agreement on the part of the estimators (reflected in a low standard deviation) did not entail more accurate group consensus estimates. A correlation test of standard deviation (StDev) of individual estimates, and the accuracy (BRE) of the group estimates resulted in a Pearson correlation of -0.54 and a p-value of 0.80.

6.3 RQ3: Are group consensus estimates more accurate than the existing individual estimation method?

The planning poker and control group had fairly similar estimation accuracy.

Regarding the observed difference in the *direction* of inaccuracy between the planning poker tasks and the control tasks, as seen by the difference in mean BREbias, one possible explanation is that some of the planning poker tasks were, purely by chance, more difficult to complete. Even though the tasks were assigned at random to the two study groups, anomalies may appear, especially in datasets of this size.

The mean of the *initial estimates,* i.e. estimates made before the assignment of the estimation procedure, was 5.3 hours (median 4 hours) for the control group, compared to 6.6 hours (median 5 hours) for the tasks estimated using planning poker. So, it is possible that the planning poker tasks were a bit more difficult, because they had initial estimates that were somewhat larger.

Comparison of the actual effort of the tasks in the two study groups yields an interesting observation. The average size in the control group tasks was 6.1 hours (median 4 hours), compared to 10.4 hours (median 8 hours) in the planning poker tasks. While the median *initial estimates* were 25% larger in the planning poker group, the difference in median *actual effort* was 100%. Even though it is possible that initially, the tasks in the planning poker tasks carried a somewhat larger workload, this cannot account for the observed difference in actual effort between the groups.

The observations of differences in actual effort between planning poker and control tasks was unexpected, and, as stated in Sections 1 and 2, there is very little research on the combining of estimates for comparison. At this stage, we can only speculate about a set of interacting causes that may explain our observations:

1. Group discussion identifies subtasks and complexity. Previous studies have shown that groups are able to identify more tasks than individuals [3]. When the group discusses a task (as when estimating using planning poker), they are likely to look at it from different angles, especially if they have diverse backgrounds [23]. They may offer different perspectives on a task and identify different subproblems.

Software engineering textbooks [16, 40] and papers [15, 41, 42] frequently mention forgotten tasks as major obstacles to successful estimation by experts. Several estimators who discuss the same task will identify at least as many subtasks as any single estimator alone. It might be that this happened for the tasks that were estimated using planning poker.

As seen from the results of the interviews presented in Section 5.5, the participants perceived that planning poker influenced their work most with respect to identifying subtasks and challenges. They also thought that they spent more time estimating and that they were more motivated to match their estimates. Even though estimation accuracy did not increase when using planning poker, the participants believed that it did. The three areas in which the participants did not perceive any changes were related to effort spent on analysis and design, refactoring, and clarifications.

The code analysis revealed that more effort was spent on performing complex changes in the planning poker tasks. This might have been induced by the group discussion.

2. Anchor-effect from individual estimates. Even if the participants identified more subtasks and complexity during group discussion, it is possible that they were not able to make sufficient adjustments on the basis of this new information when seeking consensus estimates. Even though the group decision exhibited decreased optimism when compared to the statistical combining of individual estimates, this was, for some tasks, not sufficient, and they were underestimated.

It is probable that the initial individual estimates acted as anchors [43] when group consensus was sought. Participants were frequently willing to increase their estimates somewhat, e.g. by about one hour, but it seldom happened that consensus estimates deviated substantially from the statistical combination. As seen from results of the interviews, presented in subsection 5.5, it was important for the participants to reach a consensus.

3. Priority of scope over effort and schedule. If we assume that more task work and greater complexity was identified during planning poker discussions (as follows from explanation 1), and the participants have their original estimates (and group mean) as anchors (as follows from explanation 2), this may lead to underestimated tasks.

When a task is estimated in a group (as with planning poker), and then handed to an individual, that individual must address all the aspects of the work discussed by the group when implementing the solution. By contrast, individual programmers who estimate and plan alone (as in the control group) have a more limited range of work aspects to address.

When underestimated tasks are encountered, the implementation will be affected, according to how priority is assigned to scope, effort, schedule, etc. A recent study found that software professionals gave priority to “project scope” when defining project success [44]. The professionals in that study, independent of their role in the company, stated that scope was more important than cost (effort) or time (schedule) when asked to state their priorities. We have also recently conducted a study in Norway, where it was found that (lack of) estimation accuracy did not affect perceived project success [45].

As seen from the results of the interviews presented in Section 5.5, effort and schedule were perceived as least important by the participants, while functionality and customer satisfaction were perceived as most important.

Thus, if more work is identified during discussion, programmers may feel inclined to expend more effort in order to implement it. It might just be that for some of the planning

poker tasks, work such as the restructuring of code uncovered in our analysis caused some overruns.

6.4 RQ4: Does the introduction of a group technique for estimation affect other aspects of the developers' work, when compared to the individual estimation method?

Differences in effort between groups were amplified when controlling for *change size*: More effort was expended on the planning poker tasks *and* the sizes of these tasks were smaller than in the control group.

Differences with respect to the *complexity* of the tasks can explain the difference. Changes made in the tasks estimated by planning poker were more complex, as manifested in measures of the changed code.

This observation must be seen together with the respondents' claim that planning poker was more suitable for identifying subtasks and challenges. It is possible that the planning poker method itself influenced the way the developers translated the change requests into working code.

6.5 Study Validity

In their framework for analysing the accuracy of software estimation [46], Grimstad and Jørgensen describe several factors that can have a major impact on the measured estimation error. Their top-level categories are: 1) estimation ability factors, 2) estimation complexity factors, and 3) measurement process factors.

When discussing the internal validity of our comparison of the planning poker and control groups (RQ3 and RQ4), many of the factors in the framework do not cause concern, because they are similar for both groups. Examples of these are: a) the project manager's ability to control costs, b) client and subcontractor performance, c) completeness and certainty of the information upon which the estimates were based, d) project priorities, e) project member skill, f) inherent complexity of project execution, g) experience with similar tasks, h) experience of the system under consideration, i) flexibility in product and process execution, j) terminology and measures, and k) the recording of data.

These factors were similar in both groups and did not have any effect, because all tasks in the study were taken from the same project, with, e.g., the same client, participants, and prioritizations.

In addition, the *isolation strategy* used was randomization, which is the most powerful strategy. This approach addresses concerns such *as skill in the selection of estimation approach*, because this was assigned randomly. However, as described previously, even randomization is no guarantee that the samples will have similar properties with respect to all factors. As seen, the *sizes* of the initial estimates of the tasks were not entirely similar in the groups, even though the variations were small.

Perhaps the most challenging issue concerns one of the estimation ability factors; namely, *skill in the use of estimation approach* [46]. Since the estimators were more familiar with their existing individual estimation method (control group), it might be that their skill in employing this was superior to their skill in using planning poker. However, we do not believe that this factor had any major impact, because planning poker is a straightforward and easy-to-use approach that should not require a steep learning curve. We performed an analysis of the estimation accuracy of the planning poker tasks to determine whether there was any learning. It was found that the estimation accuracy was similar for the planning poker tasks throughout the entire study.

The internal validity of the choice shift (RQ1 and RQ2) research questions is relatively unproblematic, because it involved several estimates for the same task.

Regarding external validity, only one project team was studied. Therefore, several factors must be considered when generalizing. In particular, factors such as team motivation [4] and team composition [23] will probably have a large impact on the results. For example, a team that lacks diversity and motivation may increase an optimistic bias instead of reducing it. Most important perhaps, is that this study was on task estimation, which has properties other than user-story estimation (for which planning poker is recommended), project estimation, and factors related to bidding.

Finally, regarding generalization, it is important to note that the tasks studied here were relatively small and were to be performed in an agile project environment. At this time, we have no opportunity to assess the merits of using planning poker in other project environments.

A general concern is that the study had a relative small sample size with respect to statistical analysis. The source-code data, especially, contained few data points with large variances; hence the analysis is sensitive to the values of small groups of data points. In particular, one data point has a large influence on the mean change effort of planning poker tasks. However, removing this data point does not change the observation that change effort is greater, by median or mean value, for planning poker tasks. In addition, when controlling for size, this data point can no longer be considered an outlier. We therefore included the data point in our analysis.

Given the above reservations, this study must be interpreted with care and used primarily in combination with previous studies on group estimation (presented in Section 2) as a stepping stone for further research.

7 Conclusions

Previous reviews of the literature and experiments have concluded that it does not seem to be very important which of a set of structured methods for combining estimates is used in order to achieve accuracy [47]. The Delphi technique is probably the best studied example, and though it has been found to outperform unstructured groups, there is no evidence that it outperforms other structured techniques [13]. There are also general findings to the effect that group performance is increased when motivation exists [4] and that group goals can increase productivity [6].

On the surface, planning poker has several properties that, in theory, should make it suitable for estimation; for example, the possibility of combining knowledge from diverse sources [23], the use of iterative techniques, and the fact that estimates are revealed simultaneously in order to reduce the impact of social comparison [4].

Considering a summary of our findings and combining them with previous studies, we may conclude tentatively that planning poker reduces optimism when compared to the statistical combining of individual estimates and is also, in some cases, more accurate than the unstructured combining of estimates in a group.

In this study, the set of control tasks in the same project were estimated by individual experts with accuracy similar to that of the estimates of the tasks when using planning poker. Moreover, for both the planning poker and control groups, the median estimation bias indicated that both groups had fairly unbiased estimates. In addition, as seen in our study, group discussion (facilitated by planning poker) may have certain positive side effects that, at this stage, we cannot fully explain. An interesting issue, derived from the analysis of code, is whether the use of planning poker leads to an increased focus on the quality of the code.

Equally important as findings from the quantitative data, the project team seemed to receive the planning poker technique very well. They found that the technique was useful for discussing implementation strategies for each task and that it provided a better overview of what each developer was working on. Given that it is difficult to measure the full effect of the knowledge sharing aspect of planning poker, we cannot provide any empirical results on whether the benefit exceeds the work and effort it takes to conduct this technique compared to individual estimating. However, the team decided to implement the planning poker technique for all forthcoming tasks in the project.

Future studies might seek to complement these findings by investigating projects with different constraints regarding team size and client, and using planning poker for estimating user stories.

In addition, we should investigate how planning poker can be combined with complementary techniques for tracking time/cost, i.e., having developers report progress daily (at the stand-up meeting), having all tasks posted on the wall to visualize the total work load for the sprint, and/or using a burn-down chart to visualize progress.

It is also important to compare planning poker with more structured techniques, such as Delphi, and to investigate whether planning poker affects other technical or social aspects, such as the quality of the code or the accountability of the team.

Acknowledgement

We thank all the subjects and the management of the studied company for providing data, and Magne Jørgensen, Stein Grimstad, Mike Cohn, Amund Tveit, Kristian Marius Furulund, and Chris Wright for valuable comments. This research was funded by the Research Council of Norway under the project INCO.

References

[1] J. Grenning, "Planning Poker or How to Avoid Analysis Paralysis While Release Planning," 2002.

[2] M. Cohn, *Agile Estimating and Planning*: Addison-Wesley, 2005.

[3] K. Moløkken-Østvold and M. Jørgensen, "Group Processes in Software Effort Estimation," *Empirical Software Engineering,* vol. 9, pp. 315-334, 2004.

[4] R. Brown, *Group Processes*, 2nd ed.: Blackwell Publishers, 2000.

[5] J. S. Armstrong, "How to Make Better Forecasts and Decisions: Avoid Face-to-Face Meetings," *The International Journal of Applied Forecasting,* vol. Fall 2006, pp. 3-15, 2006.

[6] R. Buehler, D. Messervey, and D. Griffin, "Collaborative Planning and Prediction: Does Group Discussion Affect Optimistic Biases in Time Estimation?," *Organizational Behaviour and Human Decision Processes,* vol. 97, pp. 47-63, 2005.

[7] E. Aronson, T. D. Wilson, and R. M. Akert, *Social Psychology*, 3rd ed.: Addison-Wesley Educational Publishers Inc., 1999.

[8] R. L. Atkinson, R. C. Atkinson, E. E. Smith, D. J. Bem, and S. Nolen-Hoeksema, *Hilgard's Introduction to Psychology*, 12th ed. Orlando: Harcourt Brace College Publishers, 1996.

[9] N. C. Haugen, "An Empirical Study of Using Planning Poker for User Story Estimation," in *Agile 2006 Conference (Agile'06)*, 2006.

[10] U. Passing and M. Shepperd, "An Experiment on Software Project Size and Effort Estimation," in *2003 International Symposium on Empirical Software Engineering (ISESE 2003)*, Frascati - Monte Porzio Catone (RM), ITALY, 2003, pp. 120-129.

[11] M. Höst and C. Wohlin, "An Experimental Study of Individual Subjective Effort Estimationsand Combinations of the Estimates," in *(20th) International Conference on Software Engineering*, Kyoto, Japan, 199, pp. 332-339.

[12] O. Helmer, *Social Technology*. New York: Basic Books, 1966.

[13] G. Rowe and G. Wright, "The Delphi Technique as a Forecasting Tool: Issues and Analysis," *International Journal of Forecasting,* pp. 353-375, 1999.

[14] C. Powell, "The Delphi Technique: Myths and Realities," *Journal of Advanced Nursing,* vol. 41, pp. 376-382, 2003.

[15] D. Fairley, "Making Accurate Estimates," *IEEE Software,* vol. 19, pp. 61-63, 2002.

[16] B. Boehm, *Software Engineering Economics*: Prentice Hall PTR, 1981.

[17] A. H. Van de Ven and A. L. Delbecq, "Nominal Versus Interacting Group Processes for Committee Decision Making Effectiveness," *Academy of Management Journal,* vol. 14, pp. 203-212, 1971.

[18] N. E. Fenton, *Software Metrics*. London: Thompson Computer Press, 1995.

[19] W. S. Humphrey, *Managing the Software Process*: Addison-Wesley Publishing Company, Inc., 1990.

[20] G. Rowe and G. Wright, "Expert Opinions in Forecasting: The Role of the Delphi Technique," in *Principles of Forecasting*, J. S. Armstrong, Ed. Boston: Kluwer Academic Publishers, 2001.

[21] M. Jørgensen, "Practical Guidelines for Expert-Judgment-Based Software Effort Estimation," *IEEE Software,* vol. 22, p. 57, 2005.

[22] R. Hanson, "Decision Markets," *IEEE intelligent systems,* vol. 14, p. 16, 1999.

[23] J. Surowiecki, *The Wisdom of Crowds* Doubleday, 2004.

[24] D. J. Berndt, J. L. Jones, and D. Finch, "Milestone Markets: Software Cost Estimation through Market Trading," in *39th Hawaii International Conference on System Science*, Hawaii, 2006.

[25] J. A. Zuber, H. W. Crott, and J. Werner, "Choice Shift and Group Polarization: An Analysis of the Status of Arguments and Social Decision Schemes," *Journal of Personality and Social Psychology,* vol. 62, pp. 50-61, 1992.

[26] T. L. Graves and A. Mockus, "Inferring Change Effort from Configuration Management Databases," *Proceedings of the 5th International Symposium on Software Metrics,* pp. 267–272, 1998.

[27] Y. Miyazaki, A. Takanou, H. Nozaki, N. Nakagawa, and K. Okada, "Method to Estimate Parameter Values in Software Prediction Models.," *Information and Software Technology,* vol. 33, pp. 239-243, 1991.

[28] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. New York: Academic Press, Inc., 1969.

[29] T. H. Wonnacott and R. J. Wonnacott, *Introductory Statistics*, 5th ed.: John Wiley & Sons, Inc., 1990.

[30] R. H. Myers, D. C. Montgomery, and G. G. Vining, *Generalized Linear Models with Applications in Engineering and the Sciences*: Wiley Series in Probability and Statistics, 2002.

[31] J. W. Hunt and M. D. McIlroy, "An Algorithm for Differential File Comparison," *Computing Science Technical Report, Bell Laboratories,* vol. 41, 1976.

[32] D. MacKenzie, P. Eggert, and R. Stallman, "Comparing and Merging Files with Gnu Diff and Patch," Network Theory Ltd., 2003.

[33] M. Jørgensen, "Experience with the Accuracy of Software Maintenance Task Effort Prediction Models," *IEEE Transactions on Software Engineering,* vol. 21, pp. 674-681, 1995.

[34] F. Niessink and H. van Vliet, "Two Case Studies in Measuring Software Maintenance Effort," *International Conference on Software Maintenance,* pp. 76–85, 1998.

[35] E. Arisholm, "Empirical Assessment of the Impact of Structural Properties on the Changeability of Object-Oriented Software," *Information and Software Technology,* 2006.

[36] B. Fluri and H. Gall, "Classifying Change Types for Qualifying Change Couplings," *Proceedings of 14th IEEE International Conference on Program Comprehension, Athens, Greece, Jun,* pp. 14-16, 2006.

[37] L. C. Briand, J. W. Daly, and J. K. Wust, "A Unified Framework for Coupling Measurement in Object-Oriented Systems," *IEEE Transactions on Software Engineering,* vol. 25, pp. 91-121, 1999.

[38] McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering,* vol. SE-2, pp. 308-320, 1976.

[39] R. C. Liden, S. J. Wayne, R. T. Sparrowe, M. L. Kraimer, T. A. Judge, and T. M. Franz, "Management of Poor Performance: A Comparison of Manager, Group Member, and Group Disciplinary Decisions," *Journal of Applied Psychology,* vol. 84, pp. 835-850, 1999.

[40] B. Kitchenham, *Software Metrics: Measurement for Software Process Improvement*. Oxford: NCC Blackwell, 1996.

[41] B. Boehm, "Software Engineering Economics," *IEEE Transactions on Software Engineering,* vol. 10, pp. 4-21, 1984.

[42] R. T. Hughes, "Expert Judgment as an Estimating Method," *Information and Software Technology,* pp. 67-75, 1996.

[43] M. Jørgensen and D. I. K. Sjøberg, "The Impact of Customer Expectation on Software Development Effort Estimates," *International Journal of Project Management,* vol. 22, pp. 317-325, 2004.

[44] N. Agarwal and U. Rathod, "Defining "Success" For Software Projects: An Exploratory Revelation," *International Journal of Project Management,* vol. 24, pp. 358-370, 2006.

[45] M. K. Furulund and K. Moløkken-Østvold, "The Role of Effort and Schedule in Assessing Software Project Success - an Empirical Study," *To be submitted,* 2007.

[46] S. Grimstad and M. Jørgensen, "A Framework for the Analysis of Software Cost," in *ISESE 2006*, Rio de Janeiro, Brazil, 2006, pp. 58–65.

[47] G. W. Fischer, "When Oracles Fail--a Comparison of Four Procedures for Aggregating Subjective Probability Forecasts.," *Organizational Behaviour and Human Performance,* vol. 28, pp. 96-110, Aug. 1981 1981.