Understanding cost drivers of software evolution: A quantitative and qualitative investigation of change effort in two evolving software systems

Authors

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.

Keywords Change effort; Software evolution cost; Quantitative and qualitative methods

1 Introduction

Software systems must adapt to continuously changing environments (Belady and Lehman 1976). 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, (Lientz 1983; Banker et al. 1993; Krishnan et al. 2000; Bhatt et al. 2006). Other studies have examined how system factors, such as structural attributes of source code, relate to the ease of changing software (Kemerer 1995; Munson and Elbaum 1998; Hayes et al. 2004). A third class of studies has focused on human factors and probed individual cognitive processes of developers attempting to comprehend and change software (Détienne and Bott 2002b).

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

Authors
Institution**,** Address
e-mail:

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 (Benestad et al. 2008). 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* (Graves and Mockus 1998), effects of *change experience* (Mockus and Weiss 2000), *tool effects* (Atkins et al. 2002), and effects of *refactorings* (Geppert et al. 2005). Other closely related studies have found effects of *structural attributes of changed components* (Evanco 2001; Polo et al. 2001; Arisholm 2006). *Subjectively assessed complexity* and the *size increase* are other factors found to be important (Jørgensen 1995b; Niessink and van Vliet 1998).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 (Jørgensen 1995b; Niessink and van Vliet 1997; Niessink and van Vliet 1998) 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 (Niessink and van Vliet 1998; Eick et al. 2001; Evanco 2001). 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 (Graves and Mockus 1998), 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 (Soloway et al.), and of different control styles in object-oriented designs (Arisholm and Sjøberg 2004). 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 (Jørgensen 1995b). Atkins *et al.* found that less effort was required when developers used a tool that supported changes to parallel versions of the system (Atkins et al. 2002). 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 (Schneidewind 2001). Iterative and agile processes take a different viewpoint, recommending that changes to requirements should be considered useful (Beck 1999). 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 (Briand and Wüst 2002). Eick *et al.* found that the history of code changes was more responsible for problems than measurable aspects of code complexity (Eick et al. 2001). On the other hand, Niessink and van Vliet showed that change effort correlated with size of the changed components (Niessink and van Vliet 1997). Likewise Arisholm found a relationship between structural attributes of affected Java classes, and change effort (Arisholm 2006). 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. (Polo et al. 2001; Xu et al. 2005). Most studies used one category for corrective changes and one or more categories for non-corrective changes, e.g., perfective and adaptive changes (Swanson 1976). Some researchers (Briand and Basili 1992; Reformat and Wu 2003) used fine-grained categories for corrective changes, similar to those proposed by Chillarege *et al.* (Chillarege et al. 1992). 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 (Curtis et al. 1989). Mockus and Weiss used historical change management data to measure developers' experience objectively (Mockus and Weiss 2000), while Jørgensen used subjective measures of skill and experience (Jørgensen 1995b). 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

Fig.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 (Benestad et al. 2008). 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.

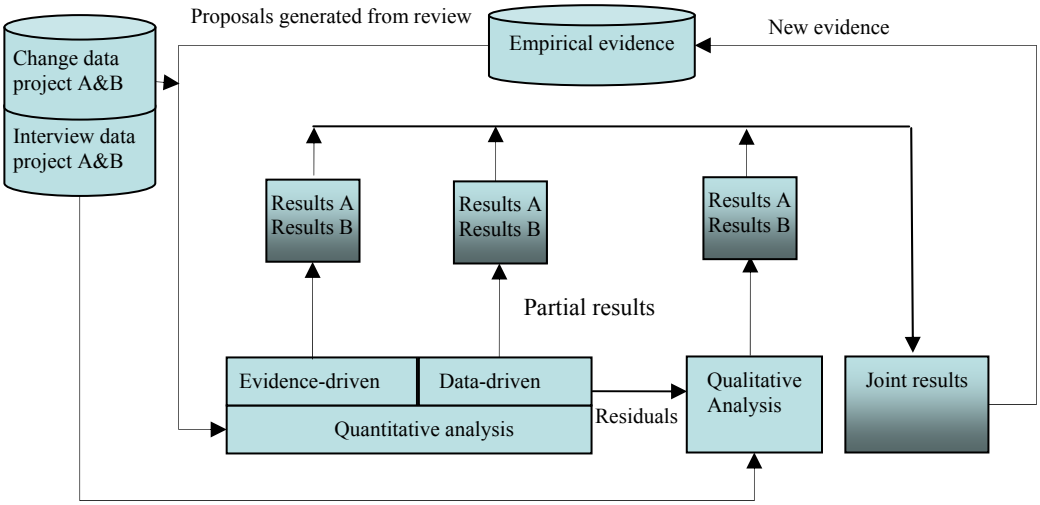

Fig. 1 Overview of analyses

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 (Yin 2003). 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., (Atkins et al. 2002).

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 (Benestad et al. 2005). 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 (RCN 2008). 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 (NSB 2008) 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. Fig. 2 and Fig. 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.

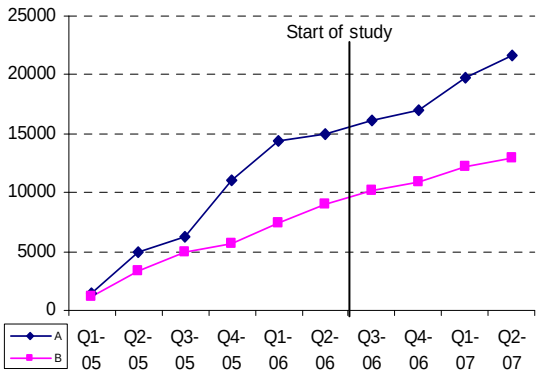

Fig. 2 Accumulated number of check-ins

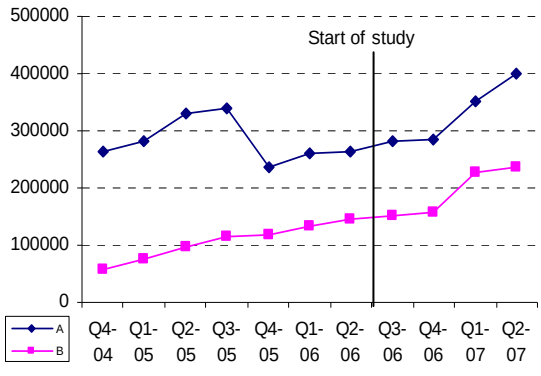

Fig. 3 System size, in lines of code

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 (IBM 2008)	CVS (GNU 2008)
Change tracker	Jira (Atlassian 2008)	Jira (Atlassian 2008)
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

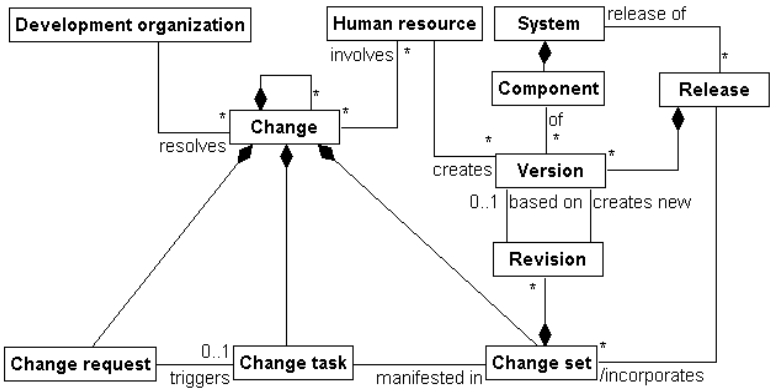

Fig. 4 Key terms and concepts

This study's perspective is that software evolution is organized around the *change task*. A conceptual model for change-based studies is given in Fig. 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.

The measures used as explanatory variables in quantitative models of change effort captured factors pertaining to the entities of the model shown in Fig. 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 (*).

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 reference
		delRefs	symbols (. ->)
Component version	Structural attrib.:		
	Size	*avgSize**	-Average/weighted (by *segments*) size of
		cpSize	changed components
	Coupling	*avgRefs*	-Average/weighted (by *segments*) number of
		cpRefs	references to members of imported components
	Control flow	*avgCC*	-Average/weighted (by *segments*) number of
		cpCC	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

Summary statistics and correlations for the measures are provided in (Benestad et al. 2009).

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 (Schneidewind 2001), the number of modifications to change requests correlated with fault proneness. In (Niessink and van Vliet 1998), 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 (Jørgensen 1995b; Graves and Mockus 1998; Niessink and van Vliet 1998; Evanco 1999). 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 (Briand and Basili 1992; Atkins et al. 2002; Arisholm 2006). 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 (Hunt and McIlroy 1975) 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 (Graves and Mockus 1998). 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 (Moløkken-Østvold et al. 2008), 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 (Fluri and Gall 2006). 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 (Briand and Basili 1992; Jørgensen 1995b; Graves and Mockus 1998; Polo et al. 2001; Atkins et al. 2002). Corrective, adaptive or perfective change types, as suggested by Swanson (Swanson 1976), 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 (Jørgensen 1995a; Graves and Mockus 1998). This does not contradict results that have shown that the mean effort for corrective changes is lower than for other change types (Polo et al. 2001), because corrective changes tend to have smaller change set size (Purushothaman and Perry 2005).

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 (Benestad 2008). 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 of code relevant to the change may affect comprehension effort involved in a change task. (Rajaraman and Lyu 1992; Etzkorn et al. 1999). Many change-based studies have investigated whether the size of changed modules (*avgSize**) correlate with change effort (Jørgensen 1995b; Niessink and van Vliet 1997; Niessink and van Vliet 1998; Arisholm 2006; Fluri and Gall 2006). Arisholm showed that size and certain other structural properties of the

changed source components were correlated with change effort (Arisholm 2006). 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 (Arisholm 2006).

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 (Eick et al. 2001). 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., (Graves et al. 2000). 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 (Atkins et al. 2002). 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 (Sackman et al. 1968; DeMarco and Lister 1985). 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

(Mockus and Weiss 2000). 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 (Myers et al. 2001). 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 (Ott and Longnecker 2001).

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) (Conte et al. 1986).

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 (Woods 2008), 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:

$$\log(\text{ceffort}) = ß_0 + ß_1\text{crTracks} + ß_2\text{components} + ß_3\text{systExp} + ß_4\text{avgSize} + ß_5\text{isCorrective} + ß_6\text{isA} \quad \textbf{(M1)}$$

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

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:

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

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

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

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.

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 8% 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 13% and 8% 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% 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*.

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

	Cross project constant model M4	Cross project w. project indicator M1	Cross project w. interactions M2	Project A M3 (standardized coefficients in parentheses)	Project B M3 (standardized coefficients in parentheses)
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

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 (Benestad et al. 2009) 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 Fig. 5 and Fig. 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 (Conte et al. 1986), the model fit was relatively poor.

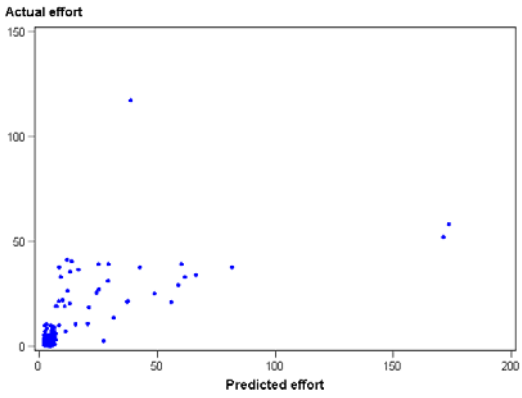

Fig. 5 Predicted vs. actual effort, project A

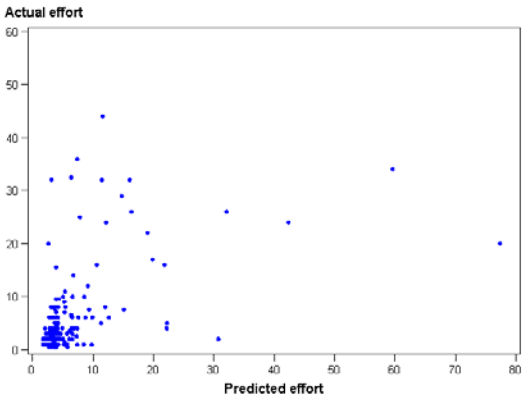

Fig. 6 Predicted vs. actual effort, project B

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*) represented a significant independent variable in the models. 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*) prior to the coding phase consistently contributed to change effort, and can therefore be useful for estimating effort in later phases of the change process. The qualitative analysis investigates the result in more depth, aiming at actions that could reduce the impact of change request volatility.

In project A, corrective changes required less effort than non-corrective changes. The data indicates the same effect in project B, although not statistically significant. The indicated effect is opposite to that of earlier studies. A possible explanation is that the processes involved in corrective vs. non-corrective changes are indeed different, but the direction of the difference depends on the developers, system and change tasks of a given project. 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., (Pinches and Mingo 1973) and (Briand and Wüst 2001b).
- Exhaustive search among variable subsets to identify the best models, described by (Miller 2002).
- A cross-validated measure of model fit (*MMRECross*) as a selection criterion (Stone 1974; Shin and Goel 2000).
- Decision trees to identify interaction effects and non-continuous effects (Briand and Wüst 2001a)

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* (Jolliffe 2002) 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) (Christensen 1996), 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 (Briand and Wüst 2001a).

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	isCorrective
Entity	*Component version*	Change set	Change set:	*Change set*	*Change request*	*Human resource*	Component 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 are in bold

Model	Variable	Coefficient (standardized coeff. in parenthesis)	MMREcr. MdMREcr.	Pred(25) Pred(50)	Pearson Spearman correl.
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 Fig. 7 and Fig. 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 Fig. 7 and Fig. 8) were analyzed in depth.

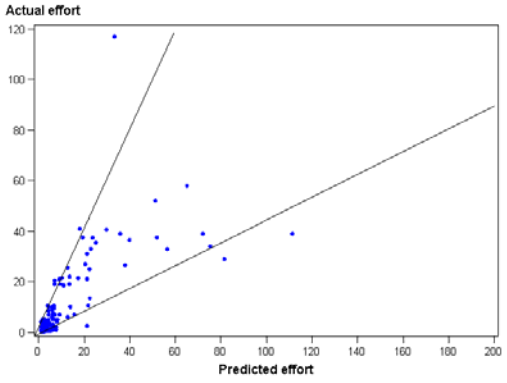

Fig. 7 Predicted vs. actual effort, project A

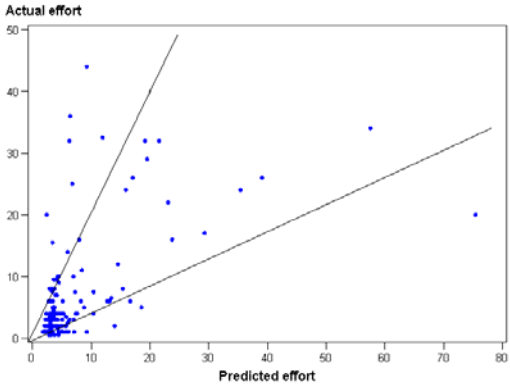

Fig. 8 Predicted vs. actual effort, project B

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 (Benestad et al. 2009).

Table 7 Summary of factors from qualitative analysis

Category	Code	Description of code	Occurrences in underestimated/over estimated 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 (von Mayrhauser and Vans 1995). 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 (Koenemann and Robertson 1991). 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 (Dzidek et al. 2008), or dynamic code measurement (e.g., by executing each user scenario), as proposed in (Arisholm et al. 2004).

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 (Détienne and Bott 2002a). 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: Reusable mechanisms solve recurring needs in the system. Typically, formalized design patterns (Gamma et al. 1995) can be used directly or as part of such code. In the investigated projects, examples 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 reusable code had to be created. According to the developers, it was challenging to create this code, for two reasons. First, the code had to be carefully designed for reusability. Second, when the purpose was to hide peculiarities of specific technologies, these needed to be well understood by the developer.

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. Did the factors identified from earlier change-based studies consistently affect change effort? 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 below.

2. How accurate were change effort models built from change management data? 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. What was the added value of using a larger number of candidate measures in the models? 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. Did change set dispersion affect change effort, beyond what could be explained by size alone? The principal component analysis showed that measures of change set dispersion captured a factor different from pure size. The measure *components* consistently and strongly contributed to change effort in the quantitative models: The standardized coefficients were 0.76 and 0.51.

5. What explained the effect of change set dispersion on change effort, e.g., how was dispersion related to the comprehension activity? 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. Was the effect of change set dispersion stronger when several languages or technologies were involved in changes? 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. Under which circumstances did change request volatility have the largest effect on change effort? 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 8% 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. Which structural properties of source code had the largest effect on change effort? 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. 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. What kind of changes required most effort? 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. Which particular skill shortages had the largest effect on change effort? A moderate effect of developers' experience was identified in project B. A 16% 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 consequences of these results from the perspective of software engineering, the 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. (Briand et al. 2003; TPTP 2008).
- 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 measures retrieved early in the change process were significant contributors to change effort.

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 (Jørgensen 1995b; Niessink and van Vliet 1997), 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 (Cohn 2006), 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:

$$\log(\text{ceffort}) = 9.25 + 0.13 * \text{relativeEffortEstimate} + 0.14 * \text{filetypes}$$

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 (TPTP 2008).
- 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* (Kitchenham et al. 2004). 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 (Kitchenham et al. 2002). 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., (Seaman 1999), empirical researchers have developed and used quantitative methods to a larger extent (Perry et al. 2000). 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 (Birk et al. 2002) or Agile Retrospectives (Derby and Larsen 2006).

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 (Fenton 1994). 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 increased the likelihood that one or more of the significant effects occurred due to chance, rather than to a true underlying effect. It would have been possible to perform multiple testing adjustment in this analysis, using procedures such as Bonferroni correction. However, due to the explorative nature of this part of the analysis, aiming at identifying additional relationships in the data, we considered such adjustment to be overly conservative. This risk of identifying shotgun correlations was lower in the evidence-driven analysis, because this analysis 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 (Mockus 2000). 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, consequences and further work

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.

Acknowledgements 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

Arisholm E (2006) Empirical assessment of the impact of structural properties on the changeability of object-oriented software. *Information and Software Technology* 48(11):1046-1055

Arisholm E, Briand LC&Føyen A (2004) Dynamic coupling measurement for object-oriented software. *IEEE Transactions on Software Engineering* 30(8):491-506

Arisholm E&Sjøberg DIK (2004) Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. *IEEE Transactions on Software Engineering* 30(8):521-534

Atkins DL, Ball T, Graves TL&Mockus A (2002) Using version control data to evaluate the impact of software tools: A case study of the version editor. *IEEE Transactions on Software Engineering* 28(7):625-637

Atlassian, Jira bug and issue tracker. Available at: http://www.atlassian.com/software/jira/. Accessed Sep 26 2008

Banker RD, Datar SM, Kemerer CF&Zweig D (1993) Software complexity and maintenance costs. *Communications of the ACM* 36(11):81-94

Beck K (1999) Embracing change with extreme programming. *Computer* 32(10):70-77

Belady LA&Lehman MM (1976) A model of large program development. *IBM Systems Journal* 15(3):225-252

Benestad HC (2008) Technical report 12-2008: Assessing the reliability of developers' classification of change tasks: A field experiment, Simula Research Laboratory

Benestad HC, Anda B&Arisholm E (2009) Technical report 02-2009: An investigation of change effort in two evolving software systems, Simula Research Laboratory

Benestad HC, Anda BC&Arisholm E (2008) Technical report 10-2008: A systematic review of empirical software engineering studies that analyze individual changes, Simula Research Laboratory

Benestad HC, Arisholm E&Sjøberg D (2005) How to recruit professionals as subjects in software engineering experiments. *Information Systems Research in Scandinavia (IRIS), Kristiansand, Norway*

Bhatt P, Shroff G, Anantaram C&Misra AK (2006) An influence model for factors in outsourced software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice* 18(6):385-423

Birk A, Dingsøyr T&Stålhane T (2002) Postmortem: Never leave a project without it. *IEEE Software* 19(3):43-45

Briand LC&Basili VR (1992) A classification procedure for the effective management of changes during the maintenance process. In: *Proceedings of 1992 Conference on Software Maintenance*, IEEE Computer Society Press: Los Alamitos CA, pp 328-336

Briand LC, Labiche Y&Miao Y (2003) Towards the reverse engineering of UML sequence diagrams. In: *Proceedings of 10th Working Conference on Reverse Engineering, WCRE 2003*, pp 57-66

Briand LC&Wüst J (2001a) The impact of design properties on development cost in object-oriented systems. *IEEE Transactions on Software Engineering* 27(11):963-986

Briand LC&Wüst J (2001b) Integrating scenario-based and measurement-based software product assessment. *The Journal of Systems & Software* 59(1):3-22

Briand LC&Wüst J (2002) Empirical studies of quality models in object-oriented systems. *Advances in Computers* 59(1):97-166

Chillarege R, Bhandari IS, Chaar JK, Halliday MJ, Moebus DS, Ray BK&Wong MY (1992) Orthogonal defect classification-a concept for in-process measurements. *Software Engineering, IEEE Transactions on* 18(11):943-956

Christensen R (1996) Principal component regression. In: *Analysis of variance, design and regression*. 446-451

Cohn M (2006) *Agile estimating and planning*. Pearson Education, Inc. Boston, MA

Conte SD, Dunsmore HE&Shen VY (1986) *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc. Redwood City, CA, USA

Curtis B, Sheppard SB, Kruesi-Bailey E, Bailey J&Boehm-Davis DA (1989) Experimental evaluation of software documentation formats. *Journal of Systems and Software* 9(2):167-207

DeMarco T&Lister T (1985) Programmer performance and the effects of the workplace. In: *Proceedings of Proceedings of the 8th international conference on Software engineering*, pp 268-272

Derby E&Larsen D (2006) *Agile retrospectives: Making good teams great*. Raleigh, NC: Pragmatic Bookshelf

Détienne F&Bott F (2002a) Influence of the task. In: *Software design - cognitive aspects*. Springer-Verlag, London.pp 105-110

Détienne F&Bott F (2002b) *Software design - cognitive aspects*. Springer-Verlag London

Dzidek WJ, Arisholm E&Briand LC (2008) A realistic empirical evaluation of the costs and benefits of UML in software maintenance. *IEEE Transactions on Software Engineering* 34(3):407-432

Eick SG, Graves TL, Karr AF, Marron JS&Mockus A (2001) Does code decay? Assessing the evidence from change management data. *IEEE Transactions on Software Engineering* 27(1):1-12

Etzkorn L, Bansiya J&Davis C (1999) Design and code complexity metrics for OO classes. *Journal of Object-Oriented Programming* 12(1):35-40

Evanco WM (1999) Analyzing change effort in software during development. In: *Proceedings of 6th International Symposium on Software Metrics (METRICS99)*, pp 179-188

Evanco WM (2001) Prediction models for software fault correction effort. In: *Proceedings of 5th European Conference on Software Maintenance and Reengineering*, IEEE Computer Society Press: Los Alamitos CA, pp 114-120

Fenton N (1994) Software measurement: A necessary scientific basis. *IEEE Transactions on Software Engineering* 20(3):199-205

Fluri B&Gall HC (2006) Classifying change types for qualifying change couplings. In: *Proceedings of 14th International Conference on Program Comprehension (ICPC)*, Athens, Greece, pp 35-45

Gamma E, Helm R, Johnson R&Vlissides J (1995) *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley

Geppert B, Mockus A&Rößler F (2005) Refactoring for changeability: A way to go? In: *Proceedings of 11th International Symposium on Software Metrics*, IEEE Computer Society Press: Los Alamitos CA, pp

GNU, Concurrent Versions System. Available at: http://www.nongnu.org/cvs/. Accessed Sep 26 2008

Graves TL, Karr AF, Marron JS&Siy H (2000) Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering* 26(7):653-661

Graves TL&Mockus A (1998) Inferring change effort from configuration management databases. In: *Proceedings of 5th International Symposium on Software Metrics*, pp 267–273

Hayes JH, Patel SC&Zhao L (2004) A metrics-based software maintenance effort model. In: *Proceedings of 8th European Conference on Software Maintenance and Reengineering*, IEEE Computer Society Press: Los Alamitos CA, pp 254-258

Hunt JW&McIlroy MD (1975) An algorithm for differential file comparison. Computing Science Technical Report 41, Bell Laboratories

IBM, Rational ClearCase LT. Available at: http://www-306.ibm.com/software/awdtools/clearcase/cclt/. Accessed Sep 26 2008

Jolliffe IT (2002) *Principal component analysis*. Springer-Verlag New York

Jørgensen M (1995a) An empirical study of software maintenance tasks. *Journal of Software Maintenance: Research and Practice* 7(1):27-48

Jørgensen M (1995b) Experience with the accuracy of software maintenance task effort prediction models. *IEEE Transactions on Software Engineering* 21(8):674-681

Kemerer C (1995) Software complexity and software maintenance: A survey of empirical research. *Annals of Software Engineering* 1(1):1-22

Kitchenham BA, Dybå T&Jørgensen M (2004) Evidence-based software engineering. In: *Proceedings of 26th International Conference on Software Engineering (ICSE)*, Edinburgh, Scotland, IEEE Computer Society, pp 273-281

Kitchenham BA, Pleeger SL, Pickard LM, Jones PW, Hoaglin DC, El Emam K&Rosenberg J (2002) Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering* 12(4):1106-1125

Koenemann J&Robertson SP (1991) Expert problem solving strategies for program comprehension. In: *Proceedings of SIGCHI conference on Human factors in computing systems: Reaching through technology*, pp 125-130

Krishnan MS, Kriebel CH, Kekre S&Mukhopadhyay. T (2000) An empirical analysis of productivity and quality in software products. *Management Science* 46(6):745-759

Lientz BP (1983) Issues in software maintenance. *ACM Computing Surveys* 15(3):271-278

Miller A (2002) Generating all subsets. In: *Subset selection in regression*. 48-52

Mockus A (2000) Missing data in software engineering. In: *Guide to advanced empirical software engineering*. 185-200

Mockus A&Weiss DM (2000) Predicting risk of software changes. *Bell Labs Technical Journal* 5(2):169-180

Moløkken-Østvold K, Haugen NC&Benestad HC (2008) Using planning poker for combining expert estimates in software projects. *Accepted for publication in Journal of Systems and Software*

Munson JC&Elbaum SG (1998) Code churn: A measure for estimating the impact of code change. In: *Proceedings of 14th International Conference on Software Maintenance*, IEEE Computer Society Press: Los Alamitos CA, pp 24-31

Myers RH, Montgomery DC&Vining GG (2001) The generalized linear model. In: *Generalized linear models with applications in engineering and the sciences*. Wiley Series in Probability and Statistics, 4-6

Niessink F&van Vliet H (1997) Predicting maintenance effort with function points. In: *Proceedings of 1997 International Conference on Software Maintenance*, IEEE Computer Society Press: Los Alamitos CA, pp 32-39

Niessink F&van Vliet H (1998) Two case studies in measuring software maintenance effort. In: *Proceedings of 14th International Conference on Software Maintenance*, IEEE Computer Society Press: Los Alamitos CA, pp 76–85

NSB, Norwegian State Railways. Available at: http://www.nsb.no/about_nsb/. Accessed Sep 26 2008

Ott RL&Longnecker M (2001) Inferences in multiple regression. In: *Statistical methods and data analysis*. Duxbury, 646-657

Perry DE, Porter AA&Votta LG (2000) Empirical studies of software engineering: A roadmap. In: *Proceedings of Conference on The Future of Software Engineering*, pp 345-355

Pinches GE&Mingo KA (1973) A multivariate analysis of industrial bond ratings. *Journal of Finance* 28(1):1-18

Polo M, Piattini M&Ruiz F (2001) Using code metrics to predict maintenance of legacy programs: A case study. In: *Proceedings of 2001 International Conference on Software Maintenance*, IEEE Computer Society Press: Los Alamitos CA, pp 202-208

Purushothaman R&Perry DE (2005) Toward understanding the rhetoric of small source code changes. *IEEE Transactions on Software Engineering* 31(6):511-526

Rajaraman C&Lyu MR (1992) Reliability and maintainability related software coupling metrics in C++ programs. In: *Proceedings of Third International Symposium on Software Reliability Engineering*, pp 303-311

RCN, Research Council of Norway, My RCN Web. Available at: https://www.forskningsradet.no/mittNettstedWeb/common/security/login.jsp?setLocale=en. Accessed Sep 26 2008

Reformat M&Wu V (2003) Analysis of software maintenance data using multi-technique approach. In: *Proceedings of 15th International Conference on Tools with Artificial Intelligence*, IEEE Computer Society Press: Los Alamitos CA, pp 53-59

Sackman H, Erikson WJ&Grant EE (1968) Exploratory experimental studies comparing online and offline programming performance. *Communications of the ACM* 11(1):3-11

Schneidewind NF (2001) Investigation of the risk to software reliability and maintainability of requirements changes. In: *Proceedings of 2001 International Conference on Software Maintenance*, IEEE Computer Society Press: Los Alamitos CA, pp 127-136

Seaman CB (1999) Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering* 25(4):557-572

Shin M&Goel AL (2000) Empirical data modeling in software engineering using radial basis functions. *IEEE Transactions on Software Engineering* 26(6):567-576

Soloway E, Pinto J&Letovsky S Designing documentation to compensate for delocalized plans. *Communications of the ACM* 31(11):1259–1267

Stone M (1974) Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society* 36(2):111-133

Swanson EB (1976) The dimensions of maintenance. In: *Proceedings of 2nd International Conference on Software Engineering*, San Francisco, California, United States, IEEE Computer Society Press: Los Alamitos CA, pp 492-497

TPTP, Eclipse Test&Performance Tools Platform Project. Available at: http://www.eclipse.org/tptp/home/documents/tutorials/profilingtool/profilingexample_32.html. Accessed Feb 02 2009

von Mayrhauser A&Vans AM (1995) Program comprehension during software maintenance and evolution. *Computer* 28(8):44-55

Woods D, Transana - Qualitative analysis software for video and audio data. Developed at the University of Wisconsin-Madison Center for Education Research. Available at: http://www.transana.org/. Accessed Sep 26 2008

Xu B, Yang M, Liang H&Zhu H (2005) Maximizing customer satisfaction in maintenance of software product family. In: *Proceedings of 18th Canadian Conference on Electrical and Computer Engineering*, IEEE Computer Society Press: Los Alamitos CA, pp 1320-1323

Yin RK (2003) Designing case studies. In: *Case study research: Design and methods*. Sage Publications:Thousand Oaks, CA, 19-53