A systematic review of empirical software engineering studies that analyze individual changes

Hans Christian Benestad, Bente Anda and Erik Arisholm
Simula Research Laboratory/University of Oslo

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

1. INTRODUCTION

Software systems that are used actively need to be changed continuously [1, 2]. Understanding, managing and reducing costs and risks of software maintenance and evolution are important goals for both research and practice in software engineering. However, it is challenging to collect and analyze data in a manner that exposes the intrinsic features of software maintenance and evolution, and a number of different approaches have been taken in empirical investigations. A key differentiator between classes of software maintenance and evolution studies is the selection of entities and attributes to measure and analyze:

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

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

- *Identify and understand factors that affect change effort during maintenance and evolution.* This knowledge would contribute to the understanding of software

maintenance and evolution in general, because the total effort expended by developers to perform changes normally constitutes a substantial part of the total lifecycle cost. For a particular project, it is essential to know the factors that drive costs in order to make effective improvements to the process or product. For example, if system components that are particularly costly to change are identified, better decisions can be made regarding refactoring.

- *Measure performance trends during maintenance and evolutions.* Projects should be able to monitor and understand performance trends in order to plan evolution and take corrective actions if negative trends are observed.

A central challenge is to identify change attributes and change measures that are effective in order to perform such analyses. For example, in order to assess and compare changes with respect to the man-hours that was needed to perform them, it is necessary to characterize the changes in some way, e.g., by measuring their *size* and *complexity*. This paper addresses this challenge by performing a comprehensive literature review of change-based studies. Conducting a comprehensive literature review is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest [11]. This review describes the change attributes that have been used in empirical investigations, and we propose a conceptual model for change-based studies that enables us to classify them. We will argue that future change-based studies can benefit from using this model as a basis for classifications and definitions of change attributes and change measures.

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

RQ1. *Which overall measurement goals have been set in change-based studies, and which attributes were measured to achieve these goals?*

RQ2. *How can change-based studies be improved over the current state of practice?*

The remainder of this paper is organized as follows: Section 2 provides a summary of related work. Section 3 describes the review procedures, including the criteria for inclusion and exclusion of primary papers for the review. Section 4 describes the conceptual model for change-based studies. Sections 5 and 6 answer RQ1 and RQ2, respectively. Section 7 discusses limitations to the review. Section 8 concludes.

2. RELATED WORK

We are not aware of other attempts to provide a comprehensive review of change-based studies of software maintenance and evolution. Graves and Mockus summarized three of their own studies that showed that *time of change*, *tool usage*, and *subsystem affected by change* affected change effort [12]. They also recommended that statistical models of change effort should control for *developer effects*, *change size* and *maintenance type*. Niessink listed six change attributes that affect change effort that have been identified in empirical work by other authors [13]. Of these, *maintenance type* and *change size* matched the change attributes identified by Graves and Mockus.

Kagdi *et al.* conducted a literature review of studies that have mined data from software repositories for the purpose of investigating changes to software components [14]. Their perspective is complementary to ours, because automated extraction of data from software repositories can be an attractive *method* for obtaining certain change measures.

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

3. REVIEW PROCEDURES

3.1. Criteria for inclusion and exclusion

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

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

Assessment and prediction are two broad purposes of measurement [18]. They are highly interdependent and we chose to include studies that involved one or both purposes. Due to our primary interest in the management of costs and risks of software maintenance and evolution, we focused on studies that have been conducted within managed development organizations, and chose to exclude investigations on distributed, volunteer based development, commonly used in open source software development. Our review targeted both quantitative and qualitative studies. Candidate papers were identified using the following procedure:

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

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

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

We used Google Scholar (http://scholar.google.com) and IEEExplore (http://ieeexplore.ieee.org) because full-text search was required to obtain reasonable sensitivity. The queries returned 446 results from Google Scholar and 169 results from IEEExplore on the 19 April 2007. In total, 261 papers remained after excluding papers on the basis of the title alone, i.e., non-software engineering work, definitely off topic, or not a peer reviewed paper. After merging the two sources, 230 papers remained. These underwent Steps

2 and 3 above. Sixty-two papers were judged as “included” or “excluded, but under some doubt”. These were re-examined by the second and third author, resulting in 33 included papers. Disagreements were resolved by discussion and by further clarifying and documenting the criteria for inclusion and exclusion. As a final quality assurance, the search criterion was applied to all papers from 27 leading software engineering journals and conference proceedings (1993 to 2007 volumes), see [20] for details of this source. One additional study was identified by this step, resulting in a total of 34 included papers.

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

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

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

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

Studies on the analysis of software defects have attempted to understand the causes and origins of defects. Generally, these studies have analyzed and extracted measures from individual components. Some of the studies collected data about corrective change tasks, e.g., [28-30]. We chose to exclude studies that analyzed the causes of defects retrospectively, but to include studies that analyzed the change tasks that were performed to isolate or correct defects.

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

3.2. Extraction of data

Goals, change attributes, and study context (RQ1) were described and classified by combining existing description frameworks with data-driven analysis similar to the constant comparison method of qualitative analysis [33]. In particular, for *measurement goals*, passages of relevant text were identified, condensed, and rephrased using terms consistent with the description template for measurement goals under the Goal Questions Metrics (GQM) paradigm [34]. These procedures resulted in the taxonomy listed in Table 1. In order to describe and classify conceptual change attributes, we extracted information about the concrete change measures that were used in the studies. Key information was names, definitions, value ranges, and methods for data collection. This information was then compared and grouped with respect to the conceptual model in Figure 1, and with respect to a set of more detailed measurement questions, as listed in Tables 2 and 3. The procedures for developing the conceptual model for change-based studies are described in Section 4.

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

Our approach to assessing the quality of change-based studies (RQ2) was to assess the studies in light of recurring validity issues, as described by Shadish *et al.* [35]. In order to identify areas for improvement for change-based studies, we focused on those validity issues that we judged to be particularly relevant in the context of such studies.

4. A CONCEPTUAL MODEL FOR CHANGE-BASED STUDIES

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

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

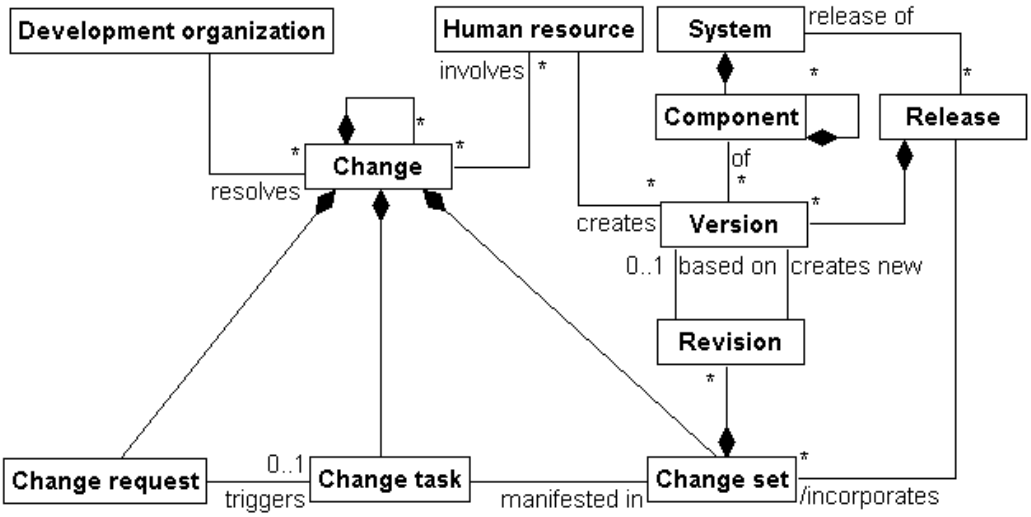

Figure 1. A conceptual model for change-based studies

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

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

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

A change attribute is a property of a change task, of the originating change request, or of the resulting change set. A change attribute can also be derived from attributes of other entities in the conceptual model. For example, the sizes of all components that were involved in a change may be averaged, or otherwise combined, in order to form a change attribute that represent the size of changed components. Change measures can be extracted from *change management systems*, which are tools that manage the kind of information that is defined by our conceptual model. Such systems include tools that are used to manage and track change requests and change tasks, and tools that are used to support controlled change of the system components. A *change outcome* is a change attribute that represents the primary quality focus of the study, e.g., *change effort*. A *change outcome measure* is the operationalization of a change outcome, and is typically used as the dependent variable in statistical analyses.

It is beyond the scope of this paper to provide operational definitions of all variations of specific change measures used in the reviewed body of research. However, the conceptual model in Figure 1 can be utilized further in a specific measurement context to facilitate precise definitions of change measures. For example, the span of a change could be operationalized as "the number of revisions that are part of a change set", while a measure of the size of affected components can be defined as "the arithmetic mean of lines of code in versions that revisions in the change set are based on". Such definitions can be expressed formally using the Object Constraint Language (OCL) [36].

5. GOALS AND MEASURED CHANGE ATTRIBUTES (RQ1)

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

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

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

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

5.1. Summary of characterization studies (Goal 1)

Goal 1 studies were split according to the sub-categories listed in Table 1. Goal 1.1 and Goal 1.2 studies are characterized by close involvement with the measured development organization. The measurement programs were planned in advance, e.g., following the GQM paradigm [34]. They are similar with respect to goals, the difference being that Goal 1.1

studies had the overall goal of improving the maintenance and evolution process, while Goal 1.2 studies focused on improving management control in ongoing projects.

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

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

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

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

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

5.2. Change attributes in characterization studies (Goal 1)

Change attributes, typical questions and typical values used during data collection in Goal 1 studies are shown in Table 2. The leftmost column indicates the part of the conceptual model in Figure 1 that normally provides the data for change measures derived from the listed change attributes.

Table 2: Change attributes that were measured in Goal 1 studies, ordered by number of studies

Entity providing information	Change attribute	Question asked	Typical values	#
Change	Maintenance	What was the purpose of the	Fix/enhance/adapt	12

request	type	change?		
Change request	Change count	Was it a change?	Simple count of changes	11
Change request	Time (period)	When did the change occur?	Date, year, time since first deployment	8
Change task	Change effort	How much effort was expended on the change task?	Person hours, ordinal or ratio	9
System	System name	To which system or project did the change belong?	Nominal measure	4
Change request	Quality focus	Which system quality was improved by the change?	Functionality/security/ efficiency/reliability	4
Change request	Status	What is the current state of the change request?	New/accepted/rejected/ solved	4
Change request	Origin	In what context or by which party was the change request made?	Internal test/external users	3
Change task	Change interval	How long did it take to resolve the change request?	Days, ordinal or ratio	3
Revision	Change size	How much content was added, deleted or changed?	Lines of code, ordinal or ratio	3
Change task	Activity	Which activities were involved in the change task?	Requirements/analysis/ design/coding/test	2
Change request	Change/defect source	Which activity caused the defect or the need for change?	Requirements/analysis/ design/coding/test	2
Revision	Change span	How many components were affected?	Count of components	2
Change request	Defect type	What kind of coding error was committed?	Initialization/logic/data/ interface/computational	2
Change request	Artifact type	What kind of component was affected?	Query/report/field/ layout/data	1
Change request	Detection	By which technique was the defect/need for change detected?	Inspection/test-run/ proof techniques	1
Change task	Delayed	Was the change task resolved later than scheduled?	Delayed/not delayed	1

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

5.3. Summary of studies that assess change attributes (Goal 2)

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

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

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

5.4. Summary of prediction studies (Goal 3)

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

Goal 3.1 studies investigated methods and processes for building prediction models. In [61], Briand and Basili suggested and validated a process for building predictive models that classified corrective changes into different categories of effort. Evanco [62] used similar procedures to predict effort for isolating and fixing defects, and validated the prediction model by comparing the results with the actual outcomes in new projects. Xu *et al.* [64] employed decision tree techniques to predict the change interval. The predictions from the model were given to the clients to set their expectations, and the authors quantified the approach's effect on customer satisfaction. Mockus and Weiss [63] predicted the risk of system failures as a consequence of changes that were made to the system. They automated the statistical analysis required to build the models, and integrated the predictions into the change process that was used by the developers.

Goal 3.2 studies compared prediction frameworks with respect to their predictive power and the degree to which the frameworks exposed explanations for the predictions. In [65], Jørgensen assessed and compared neural networks, pattern recognition and regression models for predicting change effort. He concluded that models can assist experts in making predictions, especially when the models expose explanations for the predictions. In [66], Reformat and Wu compared Bayesan networks, IF-THEN rules and decision trees for

predicting change effort on an ordinal scale. They concluded that the methods complemented each other, and suggested that practitioners should use multi-method analysis to obtain more confidence in the predictions.

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

5.5. Change attributes in assessment and prediction studies (Goal 2 and Goal 3)

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

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

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

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

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

Change attributes, typical questions and typical values used during data collection in Goal 2 and 3 studies are shown in Table 3. The leftmost column indicates the parts of the conceptual model in Figure 1 that provide data for deriving change measures from the listed change attributes. Measures of the change request, the change task and the revisions that are part of a change set occurred most frequently. Size, structure and age were the most frequently measured change attributes that used information from changed components and their

versions. Information about revisions, versions and components that were involved in a change set could only be measured after the change had been made. For the prediction goals, such change measures needed to be predicted first. The degree of collaboration (developer span) was the most frequently measured change attribute that used information about the human resources involved. No attribute of the development organization was used more than once.

Table 3. Change attributes measured in Goal 2 and 3 studies, ordered by category

Entity providing data	Change attribute	Question asked	Typical values/scale	#
Change request	Maintenance type	What was the purpose of the change?	Fix/enhance/adapt	9
Change request	Criticality	What would be the effect of not accepting the change request?	Minor/major inconvenience/stop	4
Change request	Change/defect source	Which activity caused the defect or the need for change?	Requirements/analysis/ design/coding	2
Change request	Defect type	What kind of coding error was committed?	Init/logic/interface/data	2
Change request	Requirements instability	To what extent were change requirements changed?	Number of requirement changes	2
Change request	Quality focus	Which system quality was improved by the change?	Security/efficiency/ reliability	1
Change task	Change effort	How much effort was expended on performing the change task?	Person hours, ordinal or ratio	15
Change task	Change interval	How long did it take to resolve the change request?	Days, ordinal or ratio	5
Change task	Subjective complexity	How complex was the change perceived to be?	3-point ordinal scale	3
Change task	Test effort	What was the test effort associated with the change?	# test runs	1
Revision	Change span	How many components were affected?	# components changed	9
Revision	Change size	How much content was changed?	Added + deleted LOC	7
Revision	Function points	How many logical units will be changed, added or deleted by the change?	Count of changed, added and deleted units, weighted by complexity	2
Revision	Coding mode	Was content changed or added?	Changed/added	1
Revision	Execution resources	How much (added) computational resources were required by the change?	CPU-cycles, bytes of memory	1
Version	Size	How large were the changed components?	Lines of code, number of components affected	4
Version	Structural attributes	What was the profile of the structural attributes of the changed components?	Count of structural elements (coupling, branching statements)	3

Table 3. Continued

Version	Data operation	Which data operation did the affected components perform?	Read/update/process	2
Version	Criticality	How critical was the affected component?	Is mission critical?	1
Version	Code quality	Had the changed components	Refactored/not	1

		been refactored?	refactored	
Component	Age	How old were the changed components?	Years since deployment, version number, date	3
Component	Documentation quality	How well was the changed components documented?	Was documentation rewritten?	1
Component	Code volatility	How frequently had the affected components been changed?	Total number of changes	1
Component	Component kind	What kind of component was affected?	Batch/online program	1
Component	Component id	Which specific components were changed?	Class or subsystem name	1
Component	Technology	Which technology was applied in the changed components?	3GL/4GL	1
Human resource	Developer span	How many developers were involved in performing the change task?	Number of people	3
Human resource	System experience	For how long time had the developers been involved in developing or maintaining the system?	Number of years	1
Human resource	Developer id	Who performed the change task?	Nominal measure	1
Human resource	Maintenance experience	For how long had the developers performed software maintenance work?	Number of years	1
Human resource	Objective change experience	How many changes had earlier been performed by the developers on affected components?	Number of previous check-ins in version control system	1
Human resource	Subjective experience	How was experience with respect to the affected components perceived?	3-point ordinal scale	1
Development organization	Team id	Which team was responsible for the change task?	Nominal measure	1
Development organization	Location	Where were human resources located physically?	Distributed/not distributed	1
Development organization	Tool use	Which tool was involved in the change task?	Tool used/not used	1

6. VALIDITY ISSUES AND IMPROVEMENTS TO CHANGE-BASED STUDIES (RQ2)

Study validity refers to the truth of propositions about correlations (conclusion validity) and causal relationships (internal validity) between variables within the studied context, whether these variables captured the intended aspects of real worlds concepts (construct validity), and whether results are applicable beyond the studied context (external validity), c.f., [35]. The following more specific validity issues were investigated in order to identify possible improvements to change-based studies:

1. Did the measured entities and change attributes adequately represent the main phenomenon or quality under study (construct validity)?

An assumption that underlies most change-based studies is that software maintenance and evolution can be viewed as the aggregation of changes applied to the evolving system. However, if change effort constitutes a small proportion of the total cost of maintenance, the validity of this assumption is questionable. Furthermore, if study questions are about the effects of factors at the project level, such as principles of project management or contract regimes, a change-based study is not necessarily appropriate. Abran and Hguyenkim [43] stated and handled the former threat explicitly, by comparing change effort data to activity-based effort reports. This procedure verified the appropriateness of using change measures to characterize the activities of the development organization. We recommend comparing the effort expended on individual changes to the total cost of maintenance, and using this proportion as a rough indicator of the relevance of a change-based study.

2. Was a clear rationale applied when deriving change measures from change attributes (construct validity)?

This question concerns construct validity of the individual change measures. An illustrative example is the use of the term *change size*. Investigators should make clear whether the change measure is supposed to capture the amount of affected code in the change set, or whether it is supposed to capture the amount of work involved in resolving a change request. Only four of the reviewed studies discussed such issues [48, 49, 52, 59]. A plausible explanation is the lack of use of conceptual frameworks from which to derive change measures. The extracted change attributes that are listed in Tables 2 and 3 and that refer to the conceptual model in Figure 1 are intended to be a framework that is useful for deriving change measures in new change-based studies.

3. Were all change attributes that could contribute to observed outcome considered (internal validity)?

It is challenging to identify causal relationships in change-based studies, due to the presence of and interactions between the multitudes of change attributes that might influence change outcome. Even with a coarse grouping of change attributes, as in Tables A2 and A3, very few studies considered change measures from every group. We recommend that new studies consider a broader set of change measures. The summaries in Tables 2 and 3 are intended to be helpful in this respect. In addition, accumulation of knowledge of causal relationships through continuously improved conceptual frameworks and theories would support new studies in the selection of appropriate change attributes and corresponding operational change measures.

4. *Did the study have sufficient power to detect relationships in the population (conclusion validity)?*

In statistical hypothesis testing, power is defined as the probability of discovering a relationship that exists in the population. The power of an analysis increases as the number of data points increases. Table A4 summarizes the extent of data collection in the reviewed studies. One factor that might have affected the extent of data collection in the studies is whether data was created by the change process that the development organization applied, i.e., naturally created, or created for the purpose of measurement. The median numbers of analyzed changes in the reviewed studies were 1724 and 129 for naturally created and purpose-created data, respectively. The median durations of data collection were 48 and 21 months for the same categories. These observations support the intuitive idea that relying on data that occurs naturally not only reduces organizational overhead, but also facilitates prolonged and more extensive measurement programs. We therefore recommend that investigators look for naturally created data that can be used as alternatives to purpose-created data.

5. Did the study collect appropriate kinds of data for the research questions (internal validity)?
The ability to discover causal relationships also depends on the type of data that was collected, and the method of analysis. In the reviewed studies, the data collected and analysis are primarily quantitative, and qualitative methods are used to a limited degree. Briand *et al.* [39] elicited root causes for changes by interviewing developers and by inspecting change artifacts. Nurmuliani and Williams [47] employed qualitative methods, such as interviews and observations, in order to extract quantitative change measures. Indeed, many studies relied on interpretations of qualitative data. For example, reading change requests for the purpose of classifying changes is a simple use of qualitative procedures. The study by Lam and Shankararaman [40] went one step further by creating a system specific hierarchy of types of changes, based on the changes that actually occurred in the system. In studies that attempt to explain why and how effects occur, the use of systematic qualitative procedure can be appropriate. The review shows that there is a potential for change-based studies to utilize such procedures.

6. Could change measures be reliably collected (conclusion validity)?
Some of the change measures defined in Table 3 (*criticality*, *subjective experience*, *subjective complexity* and *maintenance type*) depend on human judgement. The potential unreliability inherent in such change measures can be a threat to conclusion validity, because it may weaken or strengthen an observed effect beyond the true effect. Abran and Hguyenkim [43] used a pilot study to verify that change data could be classified reliably. Remedies for unreliability include improved training and use of change measures that are adapted to the local context. Graves and Mockus [53] applied techniques to automatically deduce the *maintenance type* from information in change management systems. Such techniques improve reliability and reduce measurement overhead, but may introduce other validity issues.

7. Were the study results generalizable to situations beyond the studied context (external validity)?
Case studies rely on analytic generalization, which means that the investigator attempts to determine whether or not results are applicable to other contexts through the application of *theory* [69]. A case study can confirm or refute an existing theory, or indicate that the theory needs to be modified in some way. A strengthened or modified theory can subsequently be applied in other contexts in order to make predictions or explain observed phenomena. Kemerer and Slaughter [21] intended to study and develop a theory of the process of software evolution. No other study discusses theory in the sense described above. This finding is not surprising, because it is known that the use of theory in software engineering research is weak [70]. The result is that many of the studies provide results that are useful within the studied context, while their applicability to other contexts is more questionable. A basic practice to improve generalizability is to select and report context attributes that may have affected the results. Whenever possible, the rationale for selecting specific context attributes should also be reported.

7. LIMITATIONS

The process by which papers where selected balanced the use of systematic, repeatable procedures with the intent to identify a comprehensive set of change-based studies. A more repeatable process could have been achieved by limiting searches to abstracts and titles only, by omitting traversal of literature references, and by excluding the Google Scholar search

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

The elicitation of measurement goals, change attributes and study contexts (RQ1) was based partly on the coding of qualitative information; hence, decisions regarding coding that were made on the basis of subjective judgment could have influenced the results. The use of existing description frameworks mitigated this effect to some extent, and contributed to a relatively straightforward coding process.

The validity issues that were investigated in the quality assessment (RQ2) were identified by the judgment of the authors. They should therefore not be taken as comprehensive. However, we do believe that key issues for change-based studies are addressed.

8. CONCLUSIONS AND FURTHER WORK

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

Although this review includes examples of successful measurement programs, it was difficult to determine whether and how insights into software maintenance and evolution could be transferred to situations beyond the immediate study context. On the basis of our discussion on generalizability and other selected study qualities, we recommend that new change-based studies should base measurement on conceptual models and, eventually, theories. This observation may be seen as an instance of a general need for an improved theoretical basis for empirical software engineering research. In order to make progress along this line, we anchored this review in a minimal, empirically based, conceptual model with the intention of supporting change-based studies. We built the model by ensuring compatibility with existing ontologies of software maintenance, and by extracting and conceptualizing the change measures applied in 34 change-based studies from a period of 25 years. In future work, we will conduct a change-based multiple-case study with the aim of understanding more about the factors that drive costs of software maintenance and evolution. The results from this review constitute important elements of the study design. We believe that this review will be useful by other research and measurement programs, and will facilitate a more effective accumulation of knowledge from empirical studies of software maintenance and evolution.

ACKNOWLEDGEMENTS

We thank Simon Andresen for our discussions on conceptual models of software change, the anonymous reviewers for useful feedback, Chris Wright for proofreading the paper, and Aiko Fallas Yamashita for her comments that improved the readability of this paper.

APPENDIX A. SUMMARY OF EXTRACTED DATA

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

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

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

Table A1. Continued.

Study	Study goal	Indicators and change attributes
Gefen and Schneberger [51]	Investigate the homogeneity of the maintenance phase, with respect to the amount of change	• Time trend in change count • Time trend in change count, by maintenance type (requirement change, programming related fix) • p-value and coefficient value in regression models of time vs. change count in time period, and per maintenance type • p-value in t-test of difference between time periods with respect to maintenance type (correct, adapt) and change count
Burch and Kung [50]	Understand time trends of changes	• Time trend in change count, by maintenance type (support, fix, enhance), using statistical models
Briand, Kim *et al.* [39]	Assess and improve quality and productivity of maintenance	• Qualitative summaries, based on interviews and questionnaires, of factors that influence maintenance performance (focused on product defects), related to development organization, process, product and people
Lam and Shankararaman [40]	Assess trends in maintenance performance	• Average change effort, by artifact type (domain specific) • Change count, by type and time period • Change count that resulted in defect, by time period
Kemerer and Slaughter [21]	Identify and understand the phases through which software systems evolve	• p-values and coefficient in regression model of time vs. change count • Degree to which certain maintenance types occur together over time, by using gamma analysis [71]. 31 sub-types of corrective, adaptive, enhancive and new changes were used
Ng [46]	Understand ERP maintenance effort	• Change effort and change count by origin (service provider, end-client) and maintenance type (fix, enhance, master data)
Perry, Siy *et al.*[48]	Understand parallelism in large-scale evolution	Change (at three levels of granularity) count by • Change interval (number of days) • Status (being worked on, not being worked on) • Change span (number of files) • Developer span (see Table 3)
Bergin and Keating [38]	Assess changeability of a software system	• Change size (percentage change to the software required by seven typical changes)
Mohagheghi, Conradi *et al.* [52]	Investigate the nature of change requests in a typical project	Proportions, and p-value for one-proportion tests of • Quality focus (functional vs. non-functional changes) • Origin (inside vs. outside development organization) • Time (before vs. after implementation and verification) • Status (accepted vs. not accepted), in total and per release
Nurmuliani and Zowghi [47]	Measure requirements volatility in a time-limited project	• Time trend in maintenance type (add, delete, modify requirement) • Time trend in quality focus • Change interval, by maintenance type and quality focus • Mean predicted change effort, by maintenance type and quality focus
Purushothaman and Perry [49]	Understand the nature of small code changes	• Change count by maintenance type (corrective, adaptive, perfective, inspect) compared between small and larger changes

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

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

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

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

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

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

REFERENCES

1. Beck K. Embracing change with extreme programming. *Computer* 1999; **32**(10):70-77.
2. Lehman MM, Ramil JF, Wernick PD, Perry DE, and Turski WM. Metrics and laws of software evolution - the nineties view. *Proceedings of the 4th International Symposium on Software Metrics.* IEEE Computer Society Press: Los Alamitos CA, 1997; 20-32.
3. Belady LA and Lehman MM. A model of large program development. *IBM Systems Journal* 1976; **15**(3):225-252.
4. Banker RD, Datar SM, Kemerer CF, and Zweig D. Software complexity and maintenance costs. *Communications of the ACM* 1993; **36**(11):81-94.
5. Bhatt P, Shroff G, Anantaram C, and Misra AK. An influence model for factors in outsourced software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice* 2006; **18**(6):385-423.
6. Krishnan MS, Kriebel CH, Kekre S, and Mukhopadhyay. T. An empirical analysis of productivity and quality in software products. *Management Science* 2000; **46**(6):745-759.
7. Lientz BP. Issues in software maintenance. *ACM Computing Surveys* 1983; **15**(3):271-278.
8. Hayes JH, Patel SC, and Zhao L. A metrics-based software maintenance effort model. *Proceedings of the 8th European Conference on Software Maintenance and Reengineering.* IEEE Computer Society Press: Los Alamitos CA, 2004; 254-258.
9. Kemerer C. Software complexity and software maintenance: A survey of empirical research. *Annals of Software Engineering* 1995; **1**(1):1-22.
10. Munson JC and Elbaum SG. Code churn: A measure for estimating the impact of code change. *Proceedings of the 14th International Conference on Software Maintenance.* IEEE Computer Society Press: Los Alamitos CA, 1998; 24-31.
11. Kitchenham BA. Procedures for performing systematic reviews. *Technical report EBSE-2007-01*, Keele University, 2007.
12. Graves TL and Mockus A. Identifying productivity drivers by modeling work units using partial data. *Technometrics* 2001; **43**(2):168-179.
13. Niessink F and van Vliet H. Two case studies in measuring software maintenance effort. *Proceedings of the 14th International Conference on Software Maintenance.* IEEE Computer Society Press: Los Alamitos CA, 1998; 76–85.
14. Kagdi H, Collard M, and Maletic JI. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice* 2007; **19**(2):77-131.
15. Dias MGB, Anquetil N, and de Oliveira KM. Organizing the knowledge used in software maintenance. *Journal of Universal Computer Science* 2003; **9**(7):641-658.
16. Kitchenham BA, Travassos GH, von Mayrhauser A, Niessink F, Schneidewind NF, Singer J, Takada S, Vehvilainen R, and Yang H. Towards an ontology of software maintenance. *Journal of Software Maintenance: Research and Practice* 1999; **11**(6):365-389.
17. Ruiz F, Vizcaíno A, Piattini M, and García F. An ontology for the management of software maintenance projects. *International Journal of Software Engineering and Knowledge Engineering* 2004; **14**(3):323-349.
18. Fenton N. Software measurement: A necessary scientific basis. *IEEE Transactions on Software Engineering* 1994; **20**(3):199-205.
19. Dieste O and Padua AG. Developing search strategies for detecting relevant experiments for systematic reviews. *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement.* IEEE Computer Society Press: Los Alamitos CA, 2007; 215-224.
20. Sjøberg DIK, Hannay JE, Hansen O, Kampenes VB, Karahasanovic A, Liborg N, and Rekdal AC. A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering* 2005; **31**(9):733-753.
21. Kemerer CF and Slaughter S. An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering* 1999; **25**(4):493-509.
22. Lientz BP, Swanson EB, and Tompkins GE. Characteristics of application software maintenance. *Communications of the ACM* 1978; **21**(6):466-471.
23. Swanson EB. The dimensions of maintenance. *Proceedings of the 2nd International Conference on Software Engineering.* IEEE Computer Society Press: Los Alamitos CA, 1976; 492-497.
24. Briand LC and Wüst J. Empirical studies of quality models in object-oriented systems. *Advances in Computers* 2002; **59**(1):97-166.

25. Arisholm E and Briand LC. Predicting fault-prone components in a java legacy system. *Proceedings of the 5th International Symposium on Empirical Software Engineering.* IEEE Computer Society Press: Los Alamitos CA, 2006; 8-17.
26. Graves TL, Karr AF, Marron JS, and Siy H. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering* 2000; **26**(7):653-661.
27. Lindvall M. Monitoring and measuring the change-prediction process at different granularity levels. *Software Process: Improvement and Practice* 1998; **4**(1):3-10.
28. Basili VR and Perricone BT. Software errors and complexity: An empirical investigation. *Communications of the ACM* 1984; **27**(1):42-52.
29. Leszak M, Perry DE, and Stoll D. Classification and evaluation of defects in a project retrospective. *The Journal of Systems & Software* 2002; **61**(3):173-187.
30. Perry DE and Stieg CS. Software faults in evolving a large, real-time system: A case study. *Proceedings of the 4th European Software Engineering Conference.* Springer-Verlag:Berlin / Heidelberg, 1993; 48-67.
31. von Mayrhauser A and Vans AM. Program comprehension during software maintenance and evolution. *Computer* 1995; **28**(8):44-55.
32. Détienne F and Bott F, *Software design - cognitive aspects*: Springer-Verlag:London, 2002.
33. Glaser BG. The constant comparative method of qualitative analysis. *Social Problems* 1965; **12**(4):436-445.
34. Basili VR, Caldiera G, and Rombach HD. *Encyclopedia of software engineering.* 2002; 578-583.
35. Shadish WR, Cook TD, and Campbell DT. *Experimental and quasi-experimental designs.* Houghton Mifflin:Boston, 2002; 33-102.
36. OMG, "OCL 2.0 specification," in *http://www.omg.org/docs/ptc/03-10-14.pdf*, 2005.
37. Basili V, Briand LC, Condon S, Kim YM, Melo WL, and Valett JD. Understanding and predicting the process of software maintenance releases. *Proceedings of the 18th International Conference on Software Engineering.* IEEE Computer Society Press: Los Alamitos CA, 1996; 464–474.
38. Bergin S and Keating J. A case study on the adaptive maintenance of an internet application. *Journal of Software Maintenance and Evolution: Research and Practice* 2003; **15**(4):245-264.
39. Briand LC, Kim YM, Melo W, Seaman C, and Basili VR. Q-MOPP: Qualitative evaluation of maintenance organizations, processes and products. *Journal of Software Maintenance: Research and Practice* 1998; **10**(4):249-278.
40. Lam W and Shankararaman V. Managing change in software development using a process improvement approach. *Proceedings of the 24th Euromicro Conference.* IEEE Computer Society Press: Los Alamitos CA, 1998; 779-786.
41. Rombach HD, Ulery BT, and Valett JD. Toward full life cycle control: Adding maintenance measurement to the SEL. *Journal of Systems and Software* 1992; **18**(2):125-138.
42. Weiss DM and Basili VR. Evaluating software development by analysis of changes - some data from the software engineering laboratory. *IEEE Transactions on Software Engineering* 1985; **11**(2):157-168.
43. Abran A and Hguyenkim H. Measurement of the maintenance process from a demand-based perspective. *Journal of Software Maintenance: Research and Practice* 1993; **5**(2):63-90.
44. Arnold RS and Parker DA. The dimensions of healthy maintenance. *Proceedings of the 6th International Conference on Software engineering.* IEEE Computer Society Press: Los Alamitos CA, 1982; 10-27.
45. Stark GE. Measurements for managing software maintenance. *Proceedings of the 1996 International Conference on Software Maintenance.* IEEE Computer Society Press: Los Alamitos CA, 1996; 152-161.
46. Ng CSP. A decision framework for enterprise resource planning maintenance and upgrade: A client perspective. *Journal of Software Maintenance and Evolution: Research and Practice* 2001; **13**(6):431-468.
47. Nurmuliani N and Zowghi D. Characterising requirements volatility: An empirical case study. *Proceedings of the 4th International Symposium on Empirical Software Engineering.* IEEE Computer Society Press: Los Alamitos CA, 2005; 427-436.
48. Perry DE, Siy HP, and Votta LG. Parallel changes in large-scale software development: An observational case study. *ACM Transactions on Software Engineering and Methodology* 2001; **10**(3):308-337.
49. Purushothaman R and Perry DE. Toward understanding the rhetoric of small source code changes. *IEEE Transactions on Software Engineering* 2005; **31**(6):511-526.
50. Burch E and Kung H. Modeling software maintenance requests: A case study. *Proceedings of the 1997 International Conference on Software Maintenance.* IEEE Computer Society Press: Los Alamitos CA, 1997; 40-47.

51. Gefen D and Schneberger SL. The non-homogeneous maintenance periods: A case study of software modifications. *Proceedings of the 1996 International Conference on Software Maintenance.* IEEE Computer Society Press: Los Alamitos CA, 1996; 134-141.
52. Mohagheghi P and Conradi R. An empirical study of software change: Origin, acceptance rate, and functionality vs. Quality attributes. *Proceedings of the 3rd International Symposium on Empirical Software Engineering.* IEEE Computer Society Press: Los Alamitos CA, 2004; 7-16.
53. Graves TL and Mockus A. Inferring change effort from configuration management databases. *Proceedings of the 5th International Symposium on Software Metrics.* IEEE Computer Society Press: Los Alamitos CA, 1998; 267–273.
54. Schneidewind NF. Investigation of the risk to software reliability and maintainability of requirements changes. *Proceedings of the 2001 International Conference on Software Maintenance.* IEEE Computer Society Press: Los Alamitos CA, 2001; 127-136.
55. Atkins DL, Ball T, Graves TL, and Mockus A. Using version control data to evaluate the impact of software tools: A case study of the version editor. *IEEE Transactions on Software Engineering* 2002; **28**(7):625-637.
56. Geppert B, Mockus A, and Rößler F. Refactoring for changeability: A way to go? *Proceedings of the 11th International Symposium on Software Metrics.* IEEE Computer Society Press: Los Alamitos CA, 2005;
57. Herbsleb JD and Mockus A. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering* 2003; **29**(6):481-494.
58. Rostkowycz AJ, Rajlich V, and Marcus A. A case study on the long-term effects of software redocumentation. *Proceedings of the 2004 International Conference on Software Maintenance* IEEE Computer Society Press: Los Alamitos CA, 2004; 92-101.
59. Arisholm E. Empirical assessment of the impact of structural properties on the changeability of object-oriented software. *Information and Software Technology* 2006; **48**(11):1046-1055.
60. Maya M, Abran A, and Bourque P. Measuring the size of small functional enhancements to software. *Proceedings of the 6th International Workshop on Software Metrics.* University of Regensburg, 1996;
61. Briand LC and Basili VR. A classification procedure for the effective management of changes during the maintenance process. *Proceedings of the 1992 Conference on Software Maintenance.* IEEE Computer Society Press: Los Alamitos CA, 1992; 328-336.
62. Evanco WM. Prediction models for software fault correction effort. *Proceedings of the 5th European Conference on Software Maintenance and Reengineering.* IEEE Computer Society Press: Los Alamitos CA, 2001; 114-120.
63. Mockus A and Weiss DM. Predicting risk of software changes. *Bell Labs Technical Journal* 2000; **5**(2):169-180.
64. Xu B, Yang M, Liang H, and Zhu H. Maximizing customer satisfaction in maintenance of software product family. *Proceedings of the 18th Canadian Conference on Electrical and Computer Engineering.* IEEE Computer Society Press: Los Alamitos CA, 2005; 1320-1323.
65. Jørgensen M. Experience with the accuracy of software maintenance task effort prediction models. *IEEE Transactions on Software Engineering* 1995; **21**(8):674-681.
66. Reformat M and Wu V. Analysis of software maintenance data using multi-technique approach. *Proceedings of the 15th International Conference on Tools with Artificial Intelligence.* IEEE Computer Society Press: Los Alamitos CA, 2003; 53-59.
67. Niessink F and van Vliet H. Predicting maintenance effort with function points. *Proceedings of the 1997 International Conference on Software Maintenance.* IEEE Computer Society Press: Los Alamitos CA, 1997; 32-39.
68. Polo M, Piattini M, and Ruiz F. Using code metrics to predict maintenance of legacy programs: A case study. *Proceedings of the 2001 International Conference on Software Maintenance.* IEEE Computer Society Press: Los Alamitos CA, 2001; 202-208.
69. Yin RK. *Case study research: Design and methods.* Sage Publications:Thousand Oaks, CA, 2003; 19-53.
70. Hannay JE and Sjøberg D. A systematic review of theory use in software engineering experiments. *IEEE Transactions on Software Engineering* 2007; **33**(2):87-107.
71. Pelz DC. Innovation complexity and the sequence of innovating stages. *Science Communication* 1985; **6**(3):261-291.