Perceived Causes of Software Project Failures – An Analysis of their Relationships

Timo O.A. Lehtinen, Mika V. Mäntylä, Jari Vanhanen, Juha Itkonen, Casper Lassenius

Department of Computer Science and Engineering, School of Science, Aalto University

P.O. BOX 19210, FI-00076, Aalto, Finland

Tel. +358 40 775 2781 (Timo Lehtinen)

timo.o.lehtinen@aalto.fi, mika.mantyla@aalto.fi, jari.vanhanen@aalto.fi, juha.itkonen@aalto.fi, casper.lassenius@aalto.fi

pre-print accepted for publication in Information and Software Technology
http://www.journals.elsevier.com/information-and-software-technology/

Abstract

Context: Software project failures are common. Even though the reasons for failures have been widely studied, the analysis of their causal relationships is lacking. This creates an illusion that the causes of project failures are unrelated.

Objective: The aim of this study is to conduct in-depth analysis of software project failures in four software product companies in order to understand the causes of failures and their relationships. For each failure, we want to understand which causes, so called bridge causes, interconnect different process areas, and which causes were perceived as the most promising targets for process improvement.

Method: The causes of failures were detected by conducting root cause analysis. For each cause, we classified its type, process area, and interconnectedness to other causes. We quantitatively analyzed which type, process area, and interconnectedness categories (bridge, local) were common among the causes selected as the most feasible targets for process improvement activities. Finally, we qualitatively analyzed the bridge causes in order to find common denominators for the causal relationships interconnecting the process areas.

Results: For each failure, our method identified causal relationships diagrams including 130 to 185 causes each. All four cases were unique, albeit some similarities occurred. On average, 50% of the causes were bridge causes. Lack of cooperation, weak task backlog, and lack of software testing resources were common bridge causes. Bridge causes, and causes related to tasks, people, and methods were common among the causes perceived as the most feasible targets for process improvement. The causes related to the project environment were frequent, but seldom perceived as feasible targets for process improvement.

Conclusion: Prevention of a software project failure requires a case-specific analysis and controlling causes outside the process area where the failure surfaces. This calls for collaboration between the individuals and managers responsible for different process areas.

Key words: Root Cause Analysis, Cause and Effect Relationships, Software Project Failure, Multiple Case Study

1. Introduction

The discipline of software engineering (SE) was born in 1968 due to software project failures [1]. Preventing software project failures is the main objective of software process improvement (SPI) as it aims at lowering the costs of development work, shortening the time to market, and improving product quality [2]. When considering preventive measures, analyzing the causes of failures becomes important as they explain why the failures occur [3]. This requires understanding the causal relationships, i.e., the causes of failures and their effects. Analyzing the causal relationships between the causes helps develop effective and feasible software process improvement ideas [4, 5] as it allows extracting cause and effect relationships [6] that can then be used in process improvement.

While trying to understand why a failure occurs, it is important to analyze all relevant areas of work [7]. Prior studies of software project failures [5, 8-24] support this, as the causes of failures reported are spread over various areas including project management, requirements engineering, and implementation. Furthermore, prior studies indicate that the causes are interconnected [8, 25], i.e., they have causal relationships between one another. Thus, in order to control the causes of failures, it is important to understand their causal relationships.

Even though software project failures and their causes are widely studied [8-18], prior studies have failed to explain and present the causal relationships between the identified causes. In the prior studies, the common causes of failures are often presented as lists where the causes are isolated from one another. This unlikely reflects the real case in software companies [8, 25]. Surveys [8, 12-14, 16, 26] and interviews [10, 11, 17] have been commonly used to detect the causes of failures [27], but not to explain the causal relationships between the causes.

In contrast to prior studies, this study utilizes root cause analysis (RCA), allowing explicitly identifying the causal relationships between the identified causes of project failures. RCA is a structured investigation of a problem to detect the causes that need to be controlled [28]. RCA takes the problem as an input and provides a set of perceived causes with causal relationships as an output [4]. It aims to state what the causes of the problem are, where they occur and why they occur. This helps with software process improvement in various contexts [4, 5, 19, 21, 23, 24, 29-36], and across software organizations [31]. There are many RCA methods available. The method that we used to conduct RCA is called ARCA [4].

The definition for a software project failure is problematic, as the term “failure” is perceived to be vague and challenging to measure [27]. McLeod et al. [27] characterize a software project failure as a breakdown in the software project outcome covering a wide variety of definitions. The term “failure” is either directly related to the outcome of the development process or it is multi-dimensional covering technical, economic, behavioral, psychological, political, subjective, contested/negotiated, and temporal interpretations [27]. Ahmad et al. [37] claim that “it may be almost impossible to find agreement about whether a project succeeded or failed”. It has happened that the developers perceive the project as a total success and the other stakeholders perceive it as a dramatic failure [38]. Agarwal and Rathod [39] state that a success and a failure are related to the perceptions of project members. They conclude that the perceptions about a success or a failure are often related to fulfilling the project goals [39]. Similar claims are presented by Procaccino et al. [40]. In our terminology, a software project failure means *a recognizable failure to succeed in the cost, schedule, scope, or quality goals of the project*. The “recognizable” refers to a project failure perceived as severe enough to be prevented in the upcoming projects.

In this paper, we present perceived causal relationships between the causes of project failures in four software product companies. Additionally, we discuss which causes the company personnel perceived as the most promising targets for process improvement activities, and how these differ from the other causes. We extend our prior paper on the causes of the failures of software projects [41], which resulted in a general list of the separated causes of project failures. The rest of the paper is structured as follows: Section 2 introduces the theoretical background. We present the common causes of software project failures, the law of causality, and the relationship between RCA and software process improvement. Section 3 presents our research objective, research questions, as well as how the research data was collected and analyzed. Section 4 presents the case study results through the distributions of the causes of failures and their causal relationships that interconnect the process areas. Section 5 answers the research questions and discusses the most interesting findings and threats to their validity. Finally, Section 6 states the conclusions and proposes future work on this topic.

2. Theoretical Background

In this section, we first analyze the prior work to point out the common causes of software project failures. Second, we discuss the causality in software engineering. Then, we present a brief review of RCA that we used as a data collection method in our study. Finally, in Section 2.4 we point out the gaps in prior works.

2.1 Common Causes of Software Project Failures

In this section, we discuss the common causes of software project failures introduced in prior studies. We consider the following three questions: 1) What causes of software project failures are introduced, 2) Where in the development processes do the causes occur, and 3) What is the relationship between the causes? We base our reasoning on a review of software engineering outcome factors introduced by McLeod et al. [27]. The review covers a total of 177 empirical studies published in the years 1996 to 2006. Additionally, we supplement the review with otherwise missing, but relevant papers on software project failures we found using the Google Scholar and Scopus databases from 1998 to 2012.

Figure 1 summarizes the common causes of failures presented in the prior studies and Section 2.1.1 elaborates the prior work behind the figure further. The existing software engineering literature on software project failures indicates that the causes of failures are commonly caused by the project environment, tasks, methods, and people. The causes of failures occur in various processes, which include management, sales & requirements, and implementation. Furthermore, the failures are likely an effect of many interconnected causes having causal relationships to one another. However, while considering such relationships, it seems that there is a gap in the prior studies. Thus, it is difficult to conclude how the causes of failures are interconnected.

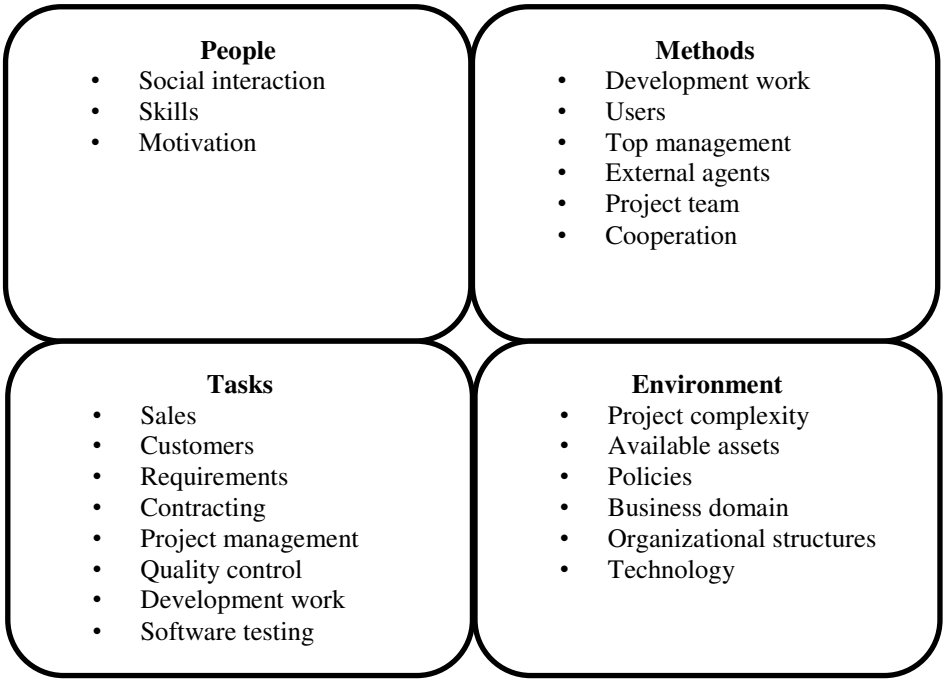

Figure 1. Summary of the common causes of software project failures

2.1.1 Causes of failures and affected process areas

McLeod et al. listed factors that affect the outcome of software systems development projects. These include factors related to project environment, people, methods, and tasks. Their findings resulted in a theory indicating that the development and deployment of software systems is a multidimensional process where people and technology are interconnected. [27]

The project environment characterizes the environmental conditions and organizational properties [27] that have an impact on the software project outcome. Moløkken-Østvold and Jørgensen [14] indicate that a chaotic environment is a common cause for software project overruns. In the case of software project failure, the project environment is commonly related to the project complexity [18, 42, 43], organizational factors [37], available assets [12, 37, 42-44], policies [43], structure [43], business domain [27, 37, 45] and technology [45].

The people related causes [27] cover social interaction [27, 42, 45, 46], skills [13, 42, 45, 46], and motivation [16, 47]. McLeod et al. [27] indicate that social interaction affects the outcome of software projects. Moløkken-Østvold and Jørgensen [14] claim that if too many people are involved, then the risks for software project overrun increases. Lorin [17] claims that stakeholder conflicts and breakdowns in communication are common causes of

failures. Egorova et al. [46] present that team spirit have a significant impact to the success of software projects. Verner et al. [16] present that if the staff is not rewarded for working long hours, their motivation decreases resulting in increasing risk for failures. Lack of skills and lack of subject matter experts are also introduced as the common causes of failures [13, 46]. That may explain why social interaction among different stakeholders is also crucial. The expertise of an individual could be insufficient [45] and thus he needs assistance from others.

Methods used in the project, i.e., the work practices used by the project members, are common causes for failures. McLeod et al. [27] present that understanding how people are conducting their work is necessary in order to improve the outcome of software projects. The causes of failures related to the methods may occur at every action conducted. These cover especially the actions of developers, users, top management, external agents, and project team [27].

The outcome of project tasks including the project scope, goals, resources, and technologies, have been introduced as the common factors of the software project outcome [27]. Verner et al. [18] supplement this list by introducing the tasks of contracting, financing, legal, and requirements. Furthermore, the list could be extended with the findings of Nasir and Sahibuddin [42] covering the project schedule, budget, project plan, progress reports, top management support, and assignment of roles and responsibilities. The project tasks are conducted in development processes. Considering the common sources of software project failures, McLeod et al. [27] recognized the processes of requirements determination, project management, user participation, user training, and change management. Furthermore, this list could be extended with the processes of sales [14], customers [11, 14, 48, 49], end users involvement [45], contracting [14, 18], risk management [8, 12, 16, 46], configuration management [12], quality control [46, 50, 51], software development [42, 49], software testing [51] and subcontractor management [14, 43].

2.1.2 Relationships between the causes of failures

The prior literature does not say much about how the causes of failures are interconnected. The common causal relationships of software project failures presented in the literature [8] are hypothetical and based on the authors' own experiences. McLeod et al. [27] summarizes that the factors of people and technology are interconnected through multidimensional processes. Similarly, Xiangnan et al. [25] present that software project failures are caused by internal and external causes being interconnected. Cerpa et al. [8] hypothesize that the causes of failures have causal relationships to one another and Lehtinen et al. [4] claim that the problems and events of software engineering are interconnected through causal relationships. However, none of the articles we identified had systematically studied the interconnections between the causes of failures.

2.2 Causality of Software Engineering Problems

Causality has been of interest to scientists and philosophers for centuries starting from Aristotle [52], Hume [53], and recently Pearl [54]. Causality refers to the relationship between two sequential and mutually exclusive events [55], i.e., the cause and its effect [6]. Such relationship is commonly known as a causal relationship. Linking all causal relationships together forms a *causal model* defined as "a complete specification of the causal relationships that govern a given domain" [56].

Focusing on causal relationships helps to structure the problem into its sub-causes making sense for its occurrence and solution [57]. This can be helpful in software process improvement [4, 5, 19, 21, 23, 24, 29-36]. Monteiro et al. [58] claims that software engineering processes are interconnected. The commonality of all software process models is that they describe a set of linked activities [59], e.g., the development work follows the specification work, which in turn follows the work of sales. Therefore, problems in one process area may cause problems in other areas, e.g., it is difficult to create test cases from insufficient requirements. Thus, it is reasonable to consider problem solving through the causes of the problems and related process areas. We conclude that:

- Software engineering problems are interrelated causally through software processes.
- It is valuable to study the causalities as it can help to prevent software project failures.

2.3 Root Cause Analysis and Software Process Improvement

Analyzing the causes of software development problems can help improve development work [60, 61]. RCA is a group work technique used to detect and analyze the causes of problems, providing a cause and effect structure by recursively identifying causes, constantly asking "why" [4].

Various software process improvement models, e.g., CMMI, ISO/IEC 12207, and Six Sigma [24], list RCA as a mandatory method for process optimization, and even agile methods recommend reflection meetings that can utilize RCA [19, 35, 62]. In general, RCA has been presented as a logical part of retrospectives [62, 63] and defect

prevention activities [5, 6, 20, 21, 24, 30, 31, 64]. In the defect prevention activities, RCA has been used to detect the causes of defects from various company processes, whereas in the retrospectives, RCA has been used to detect the causes of problems internal to the project team helping the team to improve their work practices.

Software process improvement aims to lower the costs of development work, shorten the time to market, and improve the product quality [2]. Due to the relative complexity of these three general problems, the number of potential problem causes may become extensive [30]. To avoid this problem, RCA has been applied to such relatively focused problems as a high number of a specific type of software defects [5, 19-22, 24, 30-32, 35, 64, 65].

2.4 Gaps in the Prior Work

A high number of causes of software project failures have been listed [27] and the need for understanding how the causes of failures are interconnected has been acknowledged [8]. The concept of a causal model has been defined as a complete specification of the causal relationships that govern a given domain [56]. It explains what happens, where, and why. A causal model of a software project failure would completely model the causal relationships affecting the failure. Simply providing lists of the causes of failures, as done in prior works [27], does not create a causal model as it separates the causes from each other. This paper focuses on the perceived causal relationships between the causes, taking a step towards building a causal model of software project failure.

3. Methodology

This section presents our research objective, research questions, as well as how the research data was collected and analyzed. The overall research approach is a multiple case study in four software product companies [66]. This approach was reasonable as we wanted to study real-world phenomena in a real-world context.

3.1 Research Objectives and Questions

Our research objective is *to reveal perceived causal relationships and interconnections between process areas, and evaluate their importance for analyzing software project failures and feasibility for process improvement*. To elaborate our research objective, we introduce three more detailed research questions:

RQ1: *Which process areas and cause types were frequently used to explain the software project failures?* We developed, evaluated, and applied the taxonomy of the perceived causes. We classified the causes by using two dimensions: process areas and cause types. The process area expresses where in the software process the cause occurs (see Figure 2), whereas the cause type describes what the cause is.

RQ2: *What causal relationships bridge the process areas?* We qualitatively studied the perceived causal relationships, for which the process areas of the effect was different from the one of the cause. We call such causes *bridge causes*, while the others are called *local causes* (see Figure 2).

RQ3: *Do the causes perceived as feasible targets for process improvement differ from the other detected causes, and if so, how?* After the RCA was conducted, the company people proposed and selected causes to be processed further in software process improvement activities. We call these *proposed causes* and *selected causes.* In terms of the cause types, process areas, and interconnectedness, we compared the proposed causes and selected causes with the other detected causes.

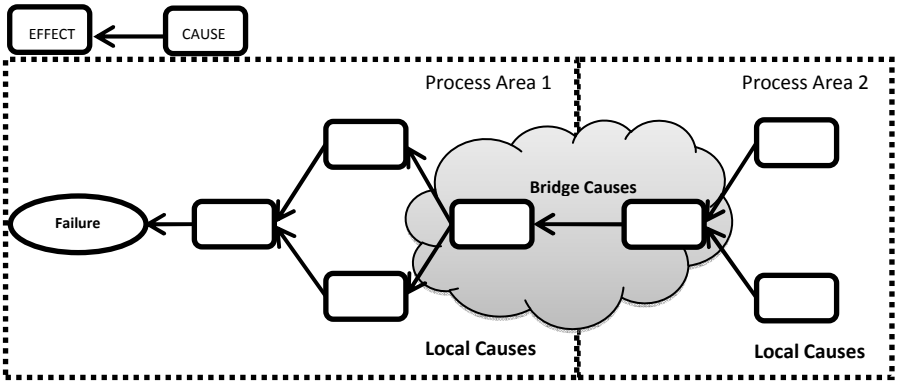

Figure 2. Terminology used in the study

3.2 Data Collection

We used the ARCA root cause analysis method [4] to identify the perceived causes of software project failures. The ARCA method implements RCA [28], and it includes four steps common for RCA methods [4]: the problem detection, root cause detection, corrective action innovation, and documentation of the results. In this section, we briefly introduce how the steps of the problem detection and root cause detection were conducted to explain how our research data was collected using RCA.

3.2.1 Problem detection

We first analyzed the project failures in each case company using a focus group with *key representatives*. The key representatives were senior managers, who had the power to make process changes in their companies. In each case, measurable evidence was used in the focus group to specify a target problem that has systematically caused a project failure, e.g., in the first case (Case Defects), it was shown that a high number of defects detected at the very end of the projects have systematically caused schedule overruns.

3.2.2 Root cause detection

Following the focus group session, we identified the causes for the selected failure in two phases: a preliminary cause collection, and a causal analysis workshop. The key representatives selected six to nine participants, who included company employees from various fields of expertise, as shown in Table 3. In the preliminary cause collection, the RCA facilitators (researchers and one key representative) sent an email to the case participants asking them to list at least five causes of the target problem. This forced the case participants to think about the problem and its causes in advance. Additionally, it helped the key representatives to select only the most important causes for further analysis in the causal analysis workshop. The email responses were handled anonymously.

The RCA facilitators organized the preliminary causes into a cause and effect diagram, as illustrated in Figure 3. Based upon the cause and effect diagram, the key representatives selected the most important cause entities to be processed in the causal analysis workshop. A cause entity includes a cause and its sub-causes, which together form an entity that is perceived as reasonable to process together (see the dark and light causes in Figure 3).

The causal analysis workshop was a time-boxed meeting of 120 minutes in which new causes were identified for each selected cause entity. The cause entities were processed one at a time. Each cause either deepened or widened a cause entity. Detecting new causes for a cause entity was done in three parts:

1. The case participants used five minutes to individually brainstorm new causes, writing them down on paper.
2. Each case participant presented the causes, and explained where they should be placed in the cause and effect diagram.
3. The case participants briefly discussed the cause entity's causes, trying to brainstorm more causes and to recognize whether a cause had a relationship to other causes.

After all the selected cause entities were processed, the related cause and effect diagram was analyzed as a whole. The RCA facilitators asked the case participants to point out essential causes and to discuss them.

The finalized cause and effect diagram was sent to the case participants of the causal analysis workshop. The participants were asked to select causes for which they thought that corrective actions should be developed. Then, the key representatives selected five to six causes of failures to be further processed, using the judgment and analysis of the causes proposed by the case participants.

3.2.3 Validity of findings

As the output of the ARCA method is based on the expert judgment of the case participants, we found it highly important to evaluate whether correct and accurate causes were detected. Triangulation of the data sources and the data collection methods increases the reliability of the results [66-68]. Before using the ARCA method, we conducted interviews [66] with the key representatives to detect the causes of failures they perceived important. We hypothesize that the causes the key representatives underlined in the interviews should also be recognized by the case participants using the ARCA method. We compared the results of the ARCA method with the perceptions of the key representatives on the causes of failures. At each case, the case participants detected and extended most of the causes underlined by the key representatives. This comparison is presented among the case study results at Section 4. Furthermore, after using the ARCA method, we used interviews and questionnaires [69] to evaluate whether correct and accurate causes of failures were detected. In each case, the case participants and key

representatives perceived that the causes detected explained the failure extensively. This validation is further described in [4], where the ARCA method was originally presented.

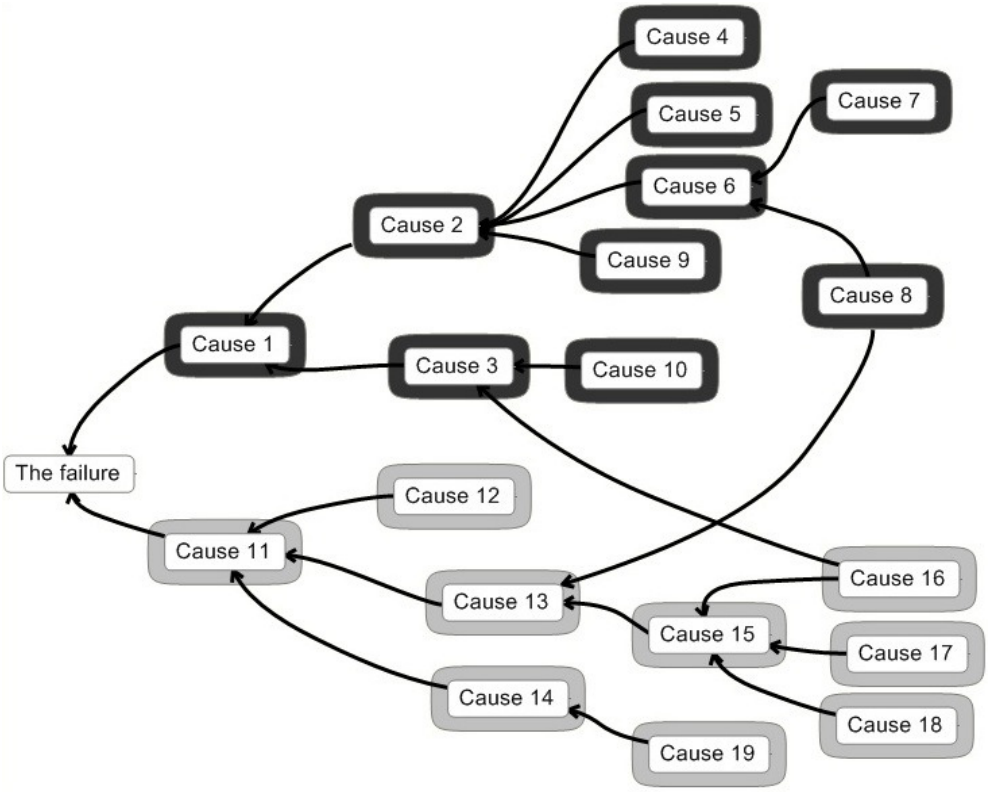

Figure 3. A cause and effect diagram of the ARCA method [4]

3.3 Data Analysis

In order to analyze the causes and their perceived causal relationships systematically, we developed and applied a detailed classification system. The classification system includes four dimensions for each cause: process area, type, interconnectedness, and feasibility for process improvement.

The development of the classification system was done iteratively. We started with a literature review to conclude what kind of the cause classification dimensions have been previously used in the software engineering domain [5, 9, 19-24, 30-32, 35, 64, 65]. We concluded that two dimensions are important: process areas and cause types. The process area expresses where the cause occurs [9, 19, 20, 22], whereas the cause type describes what the cause is, e.g., the product [9, 19, 22] and human resources [9, 20, 21, 23]. Following the literature review, we created preliminary categories for the cause types and their related process areas. We then combined the preliminary classifications with the grounded theory approach, classifying a sample of the causes, similar to the approach used in [70]. During this iteration, we modified the preliminary categories to create a final classification corresponding to the causes identified in our cases.

3.3.1 Process Areas

Table 1 introduces the classification dimension used to describe the process areas. In the classification system, *a process area* describes where the cause occurs, e.g., "sales & requirements" or "software testing." The process areas are similar to the ones found in software engineering process literature. If we compare the process areas with commonly recognized software processes such as RUP [71] or the waterfall model [72] we can see several similarities such as requirements engineering, implementation, software testing, and product release and deployment. However, there are also some differences. First, we have merged software design and implementation under the process area of implementation. It would not have been feasible to separate whether the technical problems of the product were due to poor design or implementation, because our data did not support such a

division. Similarly, the process area of software testing merges test design, execution and reporting. Another difference is that we have a process area called management that gathers causes such as insufficient decision making at the top management. Such issues cannot be placed under the process area of project management [73]. Thus, the process area of the management was needed to enable a descriptive and honest presentation of the causes. The Unknown process area includes causes that cannot be classified into any specific process area, such as "laziness."

Table 1

The process area categories expressing where the cause occurs

Process Area	General characterization of the detected causes	Concrete examples of detected causes
Management (MA)	Company support and the way the project stakeholders are managed and allocated to tasks.	*The quality of the product is of low priority in the company.* *Lack of managing the projects and their related interactions.*
Sales & Requirements (S&R)	Requirements and input from customers.	*Too many change requests from the customers.* *It is assumed that a developer understands an ambiguous specification.*
Implementation (IM)	The design and implementation of features including defect fixing.	*The features are implemented without caring about quality.* *Too much unreported error handling.*
Software Testing (ST)	Test design, execution, and reporting.	*Defects are not reported immediately.* *Tests are not conducted against proper requirements.*
Release & Deployment (PD)	Releasing and deploying the product.	*Product installation is experienced to be difficult.* *Developers do not configure the features they have implemented.*
Unknown (UN)	Causes that cannot be focused on any specific process area.	*Laziness.* *Gratuitous work is done.*

3.3.2 Cause Types

Table 2 presents the classification used to describe the types of causes. These are used to describe what the cause is, e.g., "lack of instructions & experience" or "lack of monitoring." The *cause type* characterizes the causes of failures on a general level, i.e., "People," "Tasks," "Methods," and "Environment." The cause types are based on the classification dimensions introduced in prior work [5, 20, 30]. We wanted to extend the cause types to provide more details on the causes of failures and thus we added the *sub-types* under each cause type [41]. Considering the reliability of this extended classification dimension, similar factors affecting the software project outcome has been introduced by McLeod et al. [27].

3.3.3 Dimension of Interconnectedness

We qualitatively analyzed the bridge causes, i.e., the causes that directly interconnected the process areas, as shown in Figure 2. We did this by selecting the causal relationships for which the cause and effect were in different process areas. We grouped the selected causal relationships according to the process areas, e.g., management & implementation. For each group, we explored the causal relationships by looking up the causes from the original cause and effect diagram created in the case. Thereafter, we concluded and concretized the whole path of causes and effects from the original cause and effect diagram related to the bridge causes. For example, in the first company case (Figure 4), the values of the company managers (a local cause) was used to explain why the quality was ignored during the task prioritization (a bridge cause), which was used to explain the low priority of the defect fixing (a bridge cause), which was used to explain the wrong task priorities perceived in the implementation (a local cause).

3.3.4 Feasibility of the Causes for Process Improvement

We quantitatively studied the causes that were perceived as feasible targets for process improvement. During the cause classification, we registered whether a cause was selected or proposed for further processing. The causal analysis workshop, see Section 3.2.2, resulted in *detected causes* with cause and effect structures. After the causal analysis workshop, the case attendees were asked to propose which causes they believed needed to be solved first, i.e., *proposed causes*. The key representatives, i.e., senior managers having the power to make process changes in their companies, selected five to six causes that were later processed further by developing corrective actions. These causes are called *selected causes*. Furthermore, the selected causes were explained by their *sub-causes*, which were processed among the selected causes.

During the analysis, we divided the perceived feasibility for process improvement into three categories. The highest importance is assigned to the *selected causes* that were further processed by developing corrective actions for them. The selected causes represent the senior managers' perceptions on the causes of failures feasible for

process improvement. The second highest importance is related to the *proposed causes*. They represent the participants' perceptions about which causes are feasible for process improvement. The third category consists of the causes detected but neither proposed nor selected for process improvement.

We quantitatively analyzed how the causes from these three categories differed. We compared the distributions for process areas, see Table 1, and types, see Table 2. Additionally, we compared the share of bridge causes with the share of other detected causes to understand the perceived importance of bridge causes for process improvement.

Table 2

The cause types expressing what the cause of the failure is

Type / Sub-type	**General characterization of the detected causes**	**Concrete examples of detected causes**
People (P)	This cause type includes the people related causes	
Instructions & Experience	Missing or inaccurate documentation and lack of individual experience.	*Lack of instructions when and how to verify.* *No knowledge on how many files have to be configured.*
Values & responsibilities	Poor attitude and lack of taking responsibility.	*People do not care if the number of bugs increases.* *We have a problem in our organization culture.*
Cooperation	Inactive, inaccurate, or missing communication.	*The requirements were not reviewed together enough.* *Miscommunication between the developers and testers.*
Company Policies	Not following the company policies.	*Features are marked as "done" without testing.* *New issues are not registered.*
Tasks (T)	This cause type includes the task related causes	
Task Output	Low quality task output.	*Requirements are insufficient.* *Management work is insufficient.*
Task Difficulty	The task requires too much effort, or time, or it is highly challenging.	*Design and execution of standard tests is too difficult.* *It is difficult to create a comprehensive specification.*
Task Priority	Missing, wrong, or too low task priority.	*The amount of features is more important than the quality.* *The priority of defect fixing is too low.*
Methods (M)	This cause type includes the methodological causes	
Work Practices	Missing or inadequate work practices.	*Unit testing was insufficient throughout the project.* *Implementation is done directly in the test environment.*
Process	The process model is missing, unclear, vague, too heavy, or inadequate.	*The process for software testing is missing.* *The product versions are developed in parallel for too long.*
Monitoring	Lack of monitoring.	*An opaque view of the product quality during the development work.* *Installations are usually scattered and nobody knows which one is at use.*
Environment (E)	This cause type includes the environment related causes	
Existing Product	Complex or badly implemented existing product.	*The structure of the product has decayed during the past.* *Nobody knows the whole product as it is an extremely large system.*
Resources & Schedules	Wrong resources and schedules.	*Lack of time in software testing.* *Lack of time to report defects specific enough.*
Tools	Missing or insufficient tools.	*The requirement template does not force to define constraints.* *The version control system does not support customization.*
Customers & Users	Customers' and users' expectations and need.	*The customers' desires are not analyzed and prioritized quickly.* *Importance for the customers is not well defined.*

4. Case Study Results

This section introduces our industrial cases and provides the results of each case followed by a cross case analysis. The cases and their background are the same than in our prior work [4]. In the cross case analysis, we first present the distribution of the causes, followed by pictorial descriptions of the causal relationships bridging process areas as well as the types of other causes in each process area.

Table 3 summarizes the company cases. Due to confidentiality and to make it easier to follow the discussion for the readers, the names of the case companies have been replaced by pseudonyms that reflect the main causes of failures selected for analysis in the case in question. At the table, the qualitative data is based on the results from the interviews and the quantitative data is based on the questionnaires introduced in [4]. The table summarizes how the key representatives characterized the failure and how the case participants evaluated it. The similarities of the cases made them more comparable, whereas the dissimilarities consolidated the case study results in different case

contexts. In each case, the software project failure was found to be highly complex and difficult to prevent. Similarly, in each case, the impact of the failure was experienced as relatively high. Instead, the failure itself and the effort the company had employed to try to prevent it varied between the cases and between the opinions of the case attendees. Additionally, there were also differences in the roles of the case participants.

Table 3

Summary of the case contexts [4]

	Case Defects	**Case Quality**	**Case Complicated**	**Case Isolated**
Case company	Software product company with 100 employees	Software product company with 450 employees	Software product company with 100 employees	Software product company with 110 employees
Project Failure	Fixing and verifying defects delays the project schedules	Blocker type defects are detected in the product after release	New product installation and updating are challenging tasks	Issues' lead time is sometimes intolerably long
Roles of the case participants	Project managers, quality managers, developers, sales personnel, N = 9	Mostly developers, N = 9	Project managers, testers, developers, N = 7	Project managers, testers, developers, sales personnel, N = 6
Failure characteristics	*"Extremely costly and complex"*	*"Not very costly, but very complex"*	*"High impact on customer relationships and complex"*	*"Extremely costly and complex"*
Difficulty of preventing the analyzed failure *	Average = 5.3 Standard deviation = 1.1	Average = 5.6 Standard deviation = 0.8	Average = 5.4 Standard deviation = 1.3	Average = 5.5 Standard deviation = 1.0
Impact of the analyzed failure *	Average = 5.8 Standard deviation = 1.1	Average = 5.0 Standard deviation = 1.3	Average = 5.6 Standard deviation = 0.9	Average = 5.9 Standard deviation = 0.9

*Scale: 1=very low; 2, 3, 4=neutral; 5, 6, 7=very high

4.1 Case Defects

4.1.1 The Failure and the Background

Case Defects is a medium-sized international software product company with approximately 100 employees. The average size of the project organization is about seven people. The main product is a large and complex software system, released twice a year, consisting of a major and a minor release.

The failure selected for deeper analysis in the case was that the product releases are often delayed due to a high number of software defects detected at the end of the development projects. The main goal of the company was to understand why defect fixing takes an intolerably long time and why the fixes are not verified quickly. The key representatives underlined that the company has continuously tried to prevent this failure during recent years. Additionally, they assumed that the failure is extremely complex and costly for the company. They claimed that the main causes of the failure are that the technical blocks in the software are too large and that employees' attitudes are not professional enough to develop high-quality software. Additionally, they assumed that increasing discipline among the developers and releasing the software in shorter cycles would help prevent the failure.

4.1.2 The Causes and Causal Relationships

In Case Defects, the causes of failure were related to all process areas as can be seen in Table 4. Software testing included the highest number of causes (37.6%), but management (24.0%), implementation (23.0%), and sales & requirements (11.6%) also included a high number of causes. Furthermore, the causes related to management (6.1%), implementation (10.8%), and software testing (11.5%) were most often proposed as targets for process improvement. The selected causes were related to the values & responsibility and task output of management, the task priorities of implementation, and the task output of software testing. Our results indicate that Case Defects was highly focused on management, implementation and software testing, as all of these process areas included a high number of causes that were proposed and selected. Furthermore, it seems that in Case Defects, the role of the release & deployment was very low as it included only 1.6% of the detected causes, and no proposed or selected causes.

Figure 4 depicts the identified causes, and the relationships between the process areas. In the figure, the selected causes are in **bold.** Normal texts and lines depict the sub-causes of the selected causes. A dashed line or grey text indicates that the cause was neither a selected cause nor a sub-cause. Looking at the figure, we can see that the project failure was caused by management ignoring the importance of software quality. This is in line with the early assumptions of the key representatives, being closely related to the attitudes of the company employees, a cause emphasized in the interviews. Our quantitative results support this conclusion, as a high number of causes related to

the task output and values & responsibility taking in management were detected, proposed, and selected, as shown in Table 4. These causes were interconnected to software testing and implementation, as shown in Figure 4. Our qualitative results indicate that lack of respect, lack of resources, and incorrect scheduling of software testing caused difficulties in verifying the fixed defects quickly. Additionally, software testing was influenced by lack of cooperation and low quality of implementation. Considering the causal relationships of implementation, software quality was systematically prioritized lower than new feature development. While the development work continued, the technical debt increased as new functionality was built upon a low quality implementation. This was caused by that the state of product quality was unclear during the project as the development work and software testing did not report the work properly. Simultaneously, the feature requests made by sales & requirements and the defect reports given by software testers were unclear. Thus, the workload of the implementation was difficult to estimate and the tasks were allocated to the wrong developers with incorrect priorities.

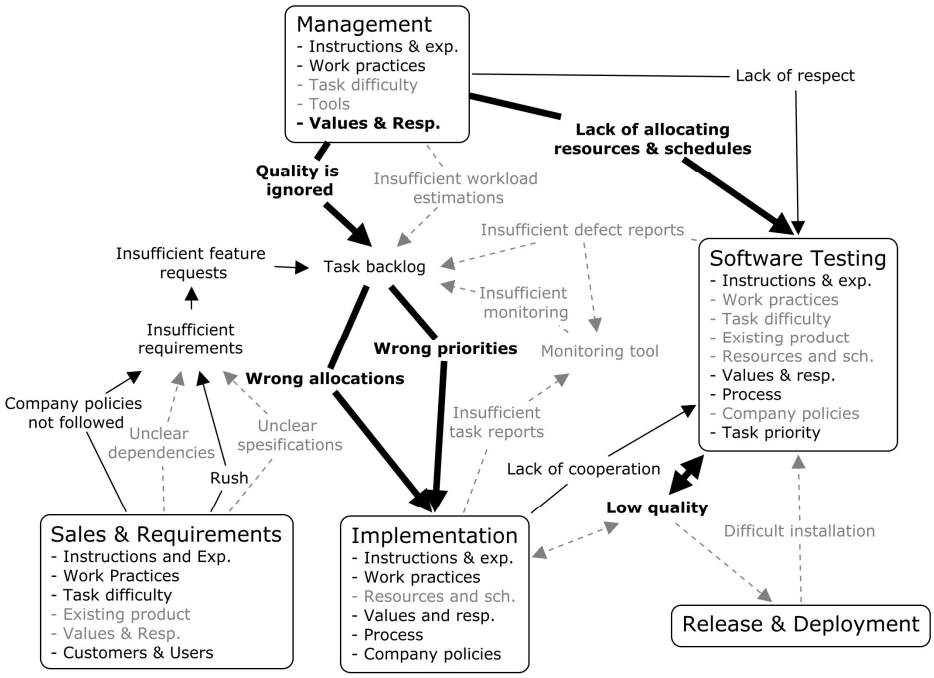

Figure 4. The causes and causal relationships in Case Defects

(Bolded text/line indicates the selected causes; Normal text/line indicates the sub-causes of the selected causes; Dashed line/grey text indicates that the cause was neither a selected cause nor a sub-cause; Lines with arrows and text between the process areas indicate the direction of causal relationships interconnecting the process areas)

4.2 Case Quality

4.2.1 The Failure and the Background

The second case was a medium-sized international software product company with approximately 450 employees. The company releases new software versions regularly and its products can be characterized as complex model-based software.

The failure of the case was that blocker-type defects are often detected after the product releases, which increases the costs through re-work and customer dissatisfaction. The main goal of the company was to understand why blocker-type defects were made and not detected during the project. The company has recently reacted to this failure by setting a clear goal to decrease the number of defects that leak to the customers. The key representatives characterized the failure as very complex and related to many different causes. The main causes for the failure were believed to be that new code is built on the old, low-quality code; too many different methods are used in the development work; and the lack of different hardware set-ups decreases the coverage of the software testing. They said that the failure could be best prevented by refactoring the old code. They also believed that the failure is not very severe because the customers are currently highly satisfied.

4.2.2 The Causes and Causal Relationships

In Case Quality (see Table 4), the process area of software testing included the highest number of causes (38.5%). However, the implementation (34.0%) and sales & requirements (19.5%) also included a relatively high number of causes. Similarly, the causes related to sales & requirements (5.3%), implementation (9.6%), and software testing (8.0%) were often proposed. These results indicate that Case Quality was highly focused on sales & requirements, implementation, and software testing. However, the selected causes were related to implementation and software testing. These included the task priority and cooperation related to the implementation, and work practices, instruction & experiences and task output related to software testing. Perhaps this was because the key representatives perceived that the case participants—mostly developers—were incapable of developing good corrective actions for the process area of sales & requirements. Furthermore, our results indicate that management and release & deployment played a minor role in Case Quality.

Figure 5 shows that the failure of Case Quality arose from the uncontrollable side effects of the existing product, which were difficult to take into account during implementation and detect during software testing. Our quantitative results supplement this conclusion by indicating a high number of causes related to the existing product in implementation and software testing, see Table 4. The assumption of the key representatives was that the failure was caused by the existing product seems to be in line with this conclusion.

However, the uncontrollable side effects of the existing product were mostly symptoms of other causes. This may explain why the causes related to the existing product were not selected. Lack of cooperation caused insufficient requirements, which lowered the efficiency of the development work and software testing through missing information. This may explain why the causes related to cooperation were proposed and selected. Due to lacking instructions & experience, the people of sales & requirements were incapable of taking the information needed for implementation and software testing into account. This was caused by a lack of collaboration in reviewing the

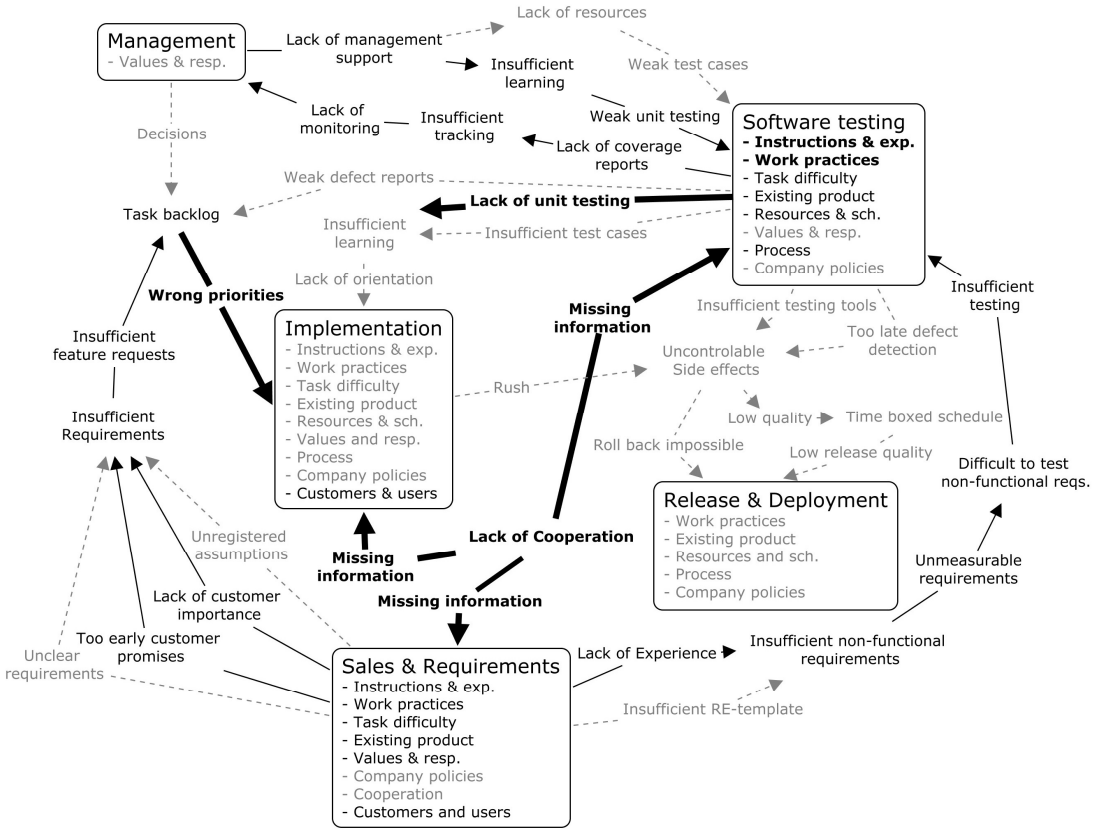

Figure 5. The causes and causal relationships in Case Quality

(Bolded text/line indicates the selected causes; Normal text/line indicates the sub-causes of the selected causes; Dashed line/grey text indicates that the cause was neither a selected cause nor a sub-cause; Lines with arrows and text between the process areas indicate the direction of causal relationships interconnecting the process areas)

requirements. It was also claimed that the people of sales & requirements did not take enough responsibility for producing good requirements, as they assumed that the developers understand ambiguous specifications.

New feature development was prioritized higher than the quality of the existing features. This was caused by too early customer promises and a time boxed project schedule. Thus, while development continued, the technical debt increased as new functionality was built on top of the low quality existing implementation. Thus, there was not enough time left to improve the software quality at the end of the project, which may explain why the causes of failure related to task priority of implementation were proposed and selected. Furthermore, management did not sufficiently support learning of unit testing, nor allocated sufficient testing resources to make good test cases, which was experienced as an important problem. These causes had following effects: defect detection was insufficient through weak unit testing, and junior developers did not have enough examples on how to use the existing functionality. The causes of failure related to weak unit testing were perceived as important, as such causes were often proposed and selected. The insufficient defect detection was also caused by implementation and sales & requirements. The development tools did not help detecting the uncontrollable side effects of the existing product, and in particular non-functional requirements were specified insufficiently and in an immeasurable way. This caused difficultly in detection and reporting defects, since the expected level of quality was unclear. Interestingly, such causes were not selected or proposed. Perhaps the company people experienced that controlling such causes is highly difficult.

4.3 Case Complicated

4.3.1 The Failure and the Background

The third case was a medium-sized international software product company with approximately 100 employees. The main product can be characterized as a highly configurable software service. The product is delivered for the customers through installation projects that occasionally include the development of new features. New software versions are released regularly.

The failure selected for analysis in the case was that the installation projects are too challenging to be performed efficiently. Re-engineering due to unexpected defects caused by the complex software configurations and development of new features during the installation projects was common. The main goal of the company was to understand why new product installation and updating are highly challenging tasks. The company hadn't expended much effort to manage the failure earlier. However, the key representatives stressed that the failure has a significant impact on their customer relationships and that it is very complex to prevent. They said that the main cause for the failure was that the employees have too many different ways in which to perform a product installation. Additionally, the number of different stakeholders was considered too high with respect to the quality of communication between them. They also indicated that creating checklists and simplifying the installation process could minimize the failure.

4.3.2 The Causes of failure and Causal Relationships

In Case Complicated (see Table 4), the causes of failure were related to all process areas except sales & requirements. The process area of release & deployment included the highest number of causes (52.5%), but software testing (18.2%), implementation (21.0%), and management (8.4%) also had their share. Interestingly, the number of causes related to management implodes while considering the proposed causes, while the number of causes related to other process areas remains high. Thus, our results indicate that Case Complicated was mostly focused on the release & deployment, software testing, and implementation. These process areas included a high number of the proposed and selected causes. The selected causes included the impact of the existing product in implementation, lack of instructions & experience, task difficulty of product deployment, and task difficulty of software testing.

Figure 6 shows that the failure of Case Complicated arose from complex version dependencies, insufficient software testing, and difficult manual configurations during release & deployment. Our quantitative results supplement these conclusions as we can see a high number of causes related to the impact of the existing product and task difficulty in the release & deployment, and insufficient task output of the implementation and software testing, see Table 4. Interestingly, the assumption of the key representatives indicating the failure being caused by the employees having too many different ways in which to perform a product installation supplements only the conclusion related to the difficult manual configurations. Additionally, the causes related to the quality of communication between different stakeholders were not detected in the causal analysis workshop.

The failure was driven by causal relationships between management, implementation, software testing, and release & deployment. The work practices of release & deployment were not planned, resulting in unsystematic

work practices. Additionally, there was very little documentation to support the work of release & deployment. This caused a lack of information during the release & deployment, and explains why the installation work was experienced as difficult. Most of the selected and proposed causes were related to these problems. The existing product was also relatively complex, unreliable, and required a high number of manual configurations during release & deployment. Considering the causes related to the existing product, there was a problem that newly developed features occasionally crashed the existing functionality, which may explain why such causes were selected and proposed. The problem was influenced by uncontrollable version dependencies and low priority of software quality in contrast to the amount of new features. The version control system was used inadequately, as occasionally new software versions were not added to the version control system, and different customer versions were overlapped under the same version branch. Furthermore, the company employees did not get enough relevant information related to the version history. Thus, it was difficult to conclude what configurations were relevant and for what version. The misuse of the version control system was influenced by management giving little value to the usage of the version control system. This was caused by a lack of knowledge.

Furthermore, software testing before release & deployment was insufficient. This problem was influenced by lack of time and resources. There was no feature freeze in the development process before releasing new versions. Thus, there was very little time to test the software and fix the defects before the release & deployment, which may explain why the causes related to task difficulty of software testing were selected and proposed. The developers also utilized the test environment as an active development environment, which meant that the detected defects were difficult to map to any active product version. The company product was also occasionally integrated into the server environments of the customers. Unfortunately, it was practically impossible to test these installations in a safe test environment in advance, as the customer environments varied a lot, and were difficult to clone to the test environment.

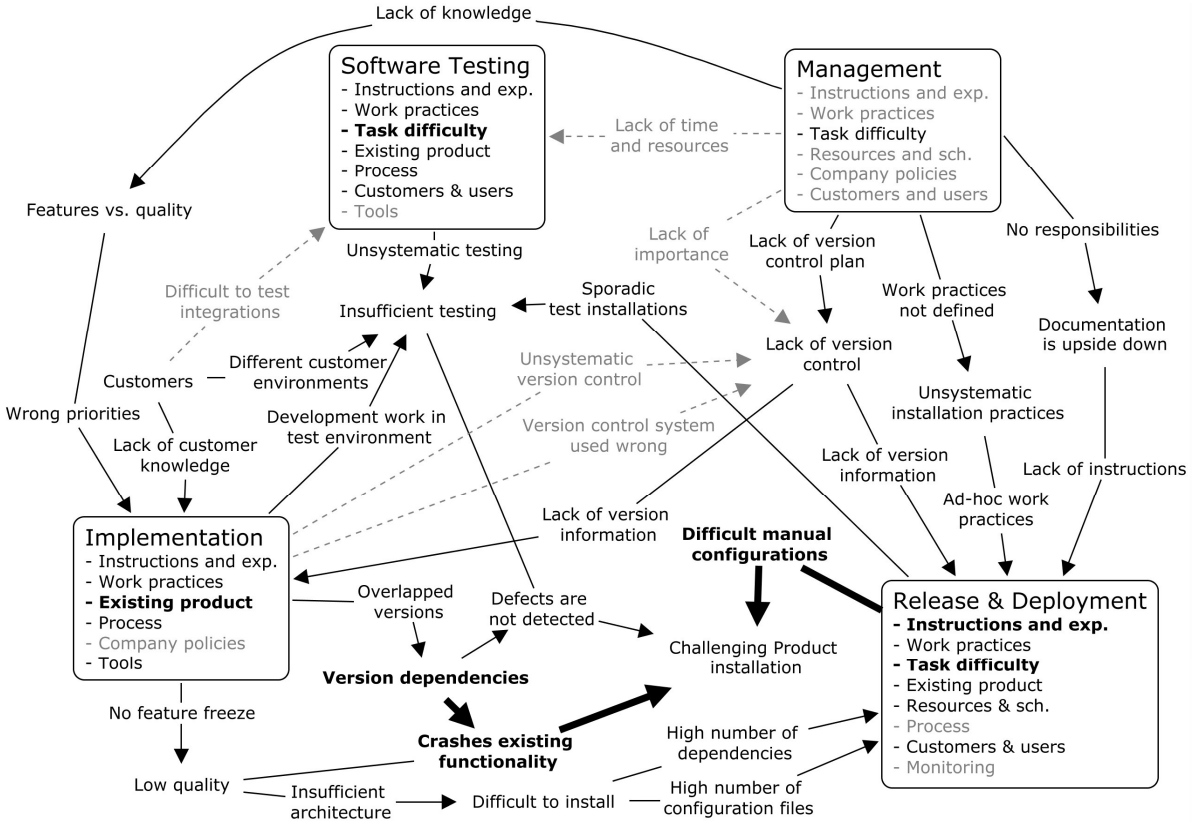

Figure 6. The causes and causal relationships in Case Complicated

(Bolded text/line indicates the selected causes; Normal text/line indicates the sub-causes of the selected causes; Dashed line/grey text indicates that the cause was neither a selected cause nor a sub-cause; Lines with arrows and text between the process areas indicate the direction of causal relationships interconnecting the process areas)

4.4 Case Isolated

4.4.1 The Failure and the Background

The fourth case was a medium-sized international software product company with approximately 110 employees. The main product can be characterized as a highly complex software system. The product is delivered to customers through complex integration projects where the product is configured into the software systems of the customers.

The failure selected for analysis in the case was that the lead time of an issue is occasionally intolerably long, resulting in delays in projects. The main goal of the company was to understand why some project issues were not implemented and verified quickly. The company hadn't expended much effort to manage the failure earlier. However, they have tried to improve communication between the stakeholders of the company. The key representatives valued the failure as high because it had a severe financial impact. It follows that the projects are not finalized on time. They said that the main causes of the failure include lack of communication between the stakeholders and the way the company divides resources between the issues. Usually, an issue with fairly low priority does not get enough resources. They concluded that preventing the failure is not an easy task. This would require increasing face-to-face meetings, increasing the number of inspections, and allocating skilled project managers to be responsible for the issues.

4.4.2 The Causes of failure and Causal Relationships

In Case Isolated, see Table 4, the causes were related to all process areas. However, implementation (36.2%), sales & requirements (36.0%), and management (20.9%) had significantly higher numbers of causes than software testing (3.6%) and release & deployment (2.4%). The distribution of proposed causes follows this trend. However, almost every cause from software testing (2.4%) and release & deployment (2.4%) was proposed for further processing. This indicates that the causes related to software testing and release & deployment were considered important even though they were not often underlined in the causal analysis workshop. Furthermore, most of the selected causes were related to sales & requirements, which indicate that the key representatives found these causes to be feasible targets for process improvement.

Figure 7 shows that the failure was partially caused by lack of cooperation, task priorities, and the inflexible development process, which is in line with the conclusion of the key representatives indicating the failure was caused by lack of communication between the stakeholders and the way the company divides resources between the issues. However, our results also indicate that the failure was caused by company policies and lack of individual responsibility. Our quantitative results supplement the conclusions on the inflexible development process, company policies, and lack of taking individual responsibility by showing a high number of detected, proposed, and selected causes related to these cause types. Instead, the central roles of the causes related to the cooperation and task priorities dissipate while focusing only on the numbers of detected causes.

Lack of cooperation was caused by distributed team members who ignored the team. Management did not arrange project meetings or reviews. Lack of cooperation had causal relationships to all process areas. Management was strongly relying on the project management tool. Resource allocation and task planning including task prioritization was done solely with the tool. Unfortunately, the tool did not support the project members' communication needs, nor did it include relevant information related to task history. This resulted in the project issues being allocated to the wrong developers and testers. Another consequence was that the project members did not receive clarifying information related to their tasks. The specifications were inadequate, including duplicate project issues. Similarly, the developers and testers did not understand what the task required. Interestingly, the selected causes did not include causes related to cooperation. Perhaps the key representatives knew that such causes are highly difficult to control as they already had tried to improve cooperation. It is also possible that such causes were already under the process improvement.

Incorrect task priority was caused by insufficient requirements and customer coordination. Additionally, the age of project issues was ignored by management. The insufficient requirements were influenced by difficult customer requests. The customer requests were driven by external consultants who had little knowledge about the required level of details needed in development. The requests were unclear and promised to be implemented immediately. The customer requests were not sufficiently managed, resulting in a problem that project issues were unclear with ambiguous specifications.

The company policies were not followed in sales & requirements, which was a cause both selected and proposed. It was an effect of the inflexible development process, which may explain why the causes of failure related to the process of the implementation were also selected and proposed. It was decided that all project issues have to be registered into the project management tool so that all project issues are systematically handled through the

development process including specification, task prioritization, resource allocation, implementation, and software testing. Unfortunately, some customer requests were experienced as too severe to be processed through the dilatory development process. Instead, the customer requests were directly allocated to the developers without specifications. This caused a problem of missing information in implementation and software testing. Additionally, the people in sales & requirements modified their prior issues to decrease the lead time of their new feature requests. They claimed that adding a new feature request as part of some prior issue would decrease the lead time of the feature request being disguised as a bug of the prior issue.

Lack of individual responsibility had causal relationships to sales & requirements and implementation. It was claimed that there is a problem of lack of orientation in the implementation and sales & requirements, which was caused by systematic interruptions and resource changes during the project. It was difficult for new developers and requirement engineers to get started, because they had to continue the work of other people without proper instructions. The systematic interruptions of the implementation and resource changes were caused by changing task priorities and lack of taking individual responsibility, which was driven by difficult tasks. The project members did not want to continue to work with tasks that some other members may execute easier and faster. This also caused a change in priorities.

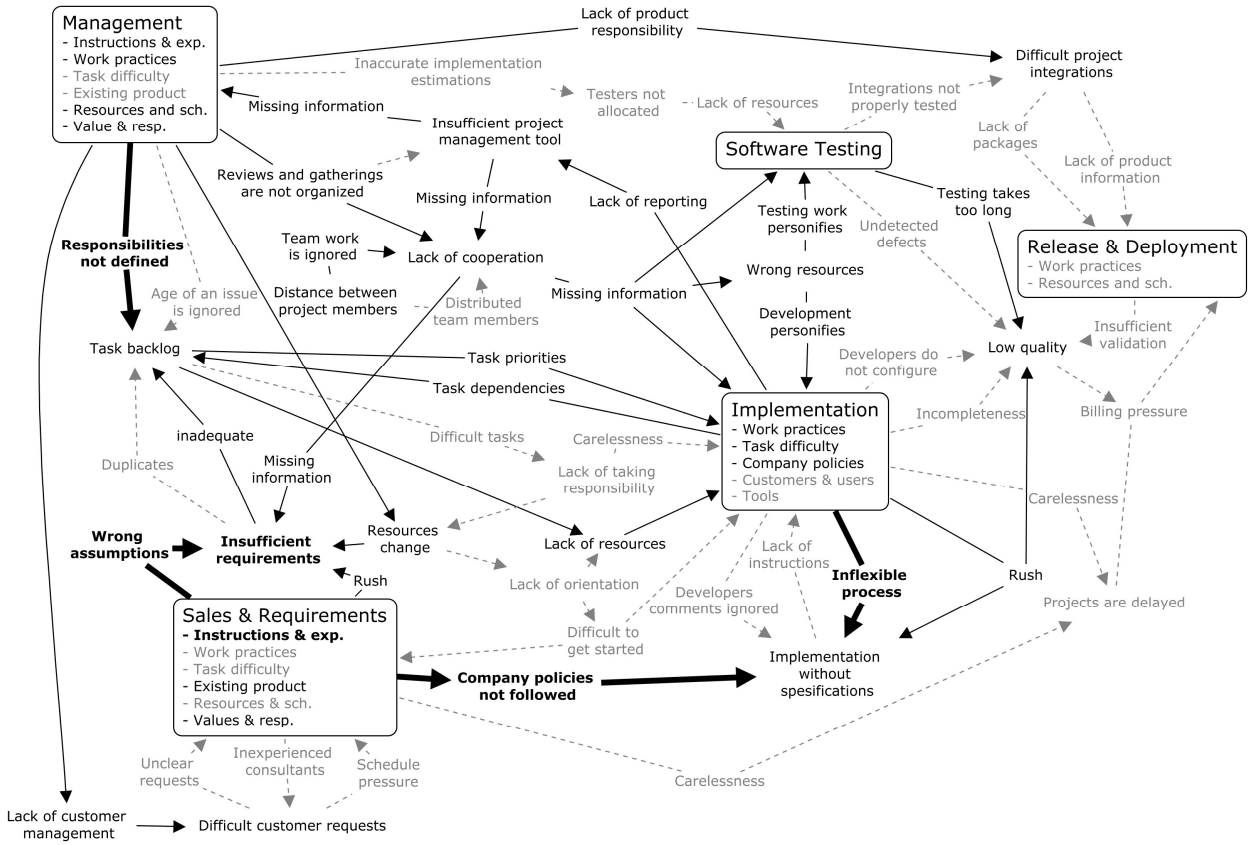

Figure 7. The causes and causal relationships in Case Isolated

(Bolded text/line indicates the selected causes; Normal text/line indicates the sub-causes of the selected causes; Dashed line/grey text indicates that the cause was neither a selected cause nor a sub-cause; Lines with arrows and text between the process areas indicate the direction of causal relationships interconnecting the process areas)

4.5 Cross Case Analysis

In this section, we compare our cases by focusing on the detected causes and their causal relationships bridging the process areas. The comparison is based on two questions: 1) Are the cases similar in terms of the detected, proposed, and selected causes, and 2) Are the common causes of failures related to similar causal relationships interconnecting the process areas?

4.5.1 Similarities of the Causes of failures

Table 4 summarizes the detected, proposed, and selected causes in the cases. It can be seen that the distributions of causes in process areas vary heavily between the cases. This means that the causes of failures were different and dependent on the case context, e.g., in Case Quality, the problem was software testing being a highly difficult task, whereas in Case Defects, the problem was software testing being a low priority task.

Despite the differences in the distributions of the causes in process areas, all of the cause types were frequent in all cases and process areas. These included causes related to People (avg. 29%, std. 6%), Tasks (avg. 26%, std. 4%), Methods (avg. 22%, std. 3%), and Environment (avg. 22%, std. 5%), as can be concluded from Table 4. Considering the sub-types of causes, seven sub-types are common between the cases and over most of the process areas. These include the instructions & experience (avg. 16%, std. 4%), values & responsibilities (avg. 8%, std. 6%), work

Table 4

Detected, proposed, and selected causes of the cases

			Case Defects			**Case Quality**			**Case Complicated**			**Case Isolated**		
Type & Sub-type		**Examples from the cases**	Detected %	Proposed %	Selected %	Detected %	Proposed %	Selected %	Detected %	Proposed %	Selected %	Detected %	Proposed %	Selected %
Management			*24.0*	*6.1*	*3.0*	*2.7*	*0.5*		*8.4*	*0.7*		*20.9*	*12.2*	*0.6*
Inst. & Exp.	P	*New managers are not familiarized enough.*	5.4	1.5					0.7			2.9	1.2	
Value & Resp.	P	*Managers behavioral give bad examples.*	6.2	2.3	1.5	1.1						1.7	1.2	
Comp. Policies	P	*Resource allocation neglected for some tasks.*							0.7					
Work Practices	M	*Workload estimations were not done.*	3.1						0.7			6.4	5.2	
Task Output	T	*Managerial duties are conducted insufficiently.*	7.7	2.3	1.5	1.6	0.5		4.2	0.7		5.2	2.3	0.6
Task Difficulty	T	*Prioritizing hundreds of issues is difficult.*	0.8						0.7			0.6		
Existing Product	E	*Only few people know the product technically.*										1.2		
Res. & Sch.	E	*Project manager is busy.*							0.7			1.7	1.7	
Cust. & Users	E	*SLA forces us to react to the complaints.*							0.7			0.6	0.6	
Tools	E	*PM tool does not support knowledge sharing..*	0.8									0.6		
Sales & Requirements			*11.6*	*0.8*		*19.5*	*5.3*					*36.0*	*16.9*	*2.4*
Inst. & Exp.	P	*RE people does not understand the problem.*	3.1			2.7	0.5					7.6	4.7	0.6
Value & Resp.	P	*Assumption that insufficient requirements are ok.*	0.8			2.2	0.5					5.2	1.7	0.6
Comp. Policies	P	*Many requests are documented in one request.*	1.5			0.5						2.9	1.7	0.6
Cooperation	P	*Lack of communication with developers.*				1.1	0.5					1.2	0.6	
Work Practices	M	*Requirements are not verified with customers.*	1.5			1.1						5.2	3.5	
Task Output	T	*Requirements are documented insufficiently.*	1.5	0.8		4.3	2.7					3.5	1.2	0.6
Task Difficulty	T	*Very difficult to make a perfect specification.*	0.8			1.6						1.2		
Existing Product	E	*The product requires new issues constantly.*	0.8			1.1						1.7	0.6	
Res. & Sch.	E	*Lack of resources to document the issues.*	0.8									5.2	1.2	
Cust. & Users	E	*Customers cannot describe their needs.*	0.8			3.8	1.1					2.3	1.7	
Tools	E	*Missing fields in the requirement template.*				1.1								
Implementation			*23.0*	*10.8*	*0.8*	*34.0*	*9.6*	*1.1*	*21.0*	*9.1*	*0.7*	*36.2*	*11.6*	*0.6*
Inst. & Exp.	P	*Lack of orientation to implement features.*	2.3			4.9	1.6		2.8	1.4		3.5	0.6	
Value & Resp.	P	*Features quality is ignored in the development.*	2.3	1.5		3.2	0.5					3.5		
Comp. Policies	P	*Definition of done is not obeyed.*	0.8	0.8		1.6	1.1		0.7			0.6		
Cooperation	P	*Inactive communication with the task requester.*	1.5	0.8		2.2	0.5	0.5				1.7	1.7	
Work Practices	M	*Tasks are not finalized once started.*	2.3	0.8		3.2			4.9	2.1		7.6	4.1	
Process	M	*No stabilization phases in the process.*	1.5	0.8		2.7	1.6		1.4	0.7		4.1	1.7	0.6
Monitoring	M	*The progress of development tasks is unclear.*	1.5	1.5								0.6		
Task Output	T	*The quality of code is low.*	2.3	0.8		4.9	1.1		5.6	2.1		3.5	1.7	
Task Difficulty	T	*Difficult to design high quality interfaces.*				1.6	0.5					1.2		
Task Priority	T	*Features are more important than the quality.*	5.4	2.3	0.8	1.1	0.5	0.5	0.7	0.7		2.9		
Existing Product	E	*Architectural dependencies hamper dev.*				4.3	2.2		2.1	0.7	0.7			
Res. & Sch.	E	*Lack of time to implement.*	3.1	1.5		3.2						5.2	1.2	
Cust. & Users	E	*Customers' requests often break the product.*				1.1			2.1	0.7		1.2		
Tools	E	*Insufficient version control system.*							0.7	0.7		0.6	0.6	
Software Testing			*37.6*	*11.5*	*0.8*	*38.5*	*8.0*	*1.6*	*18.2*	*9.8*	*0.7*	*3.6*	*2.4*	
Inst. & Exp.	P	*Lack of instructions on what to test.*	4.6	1.5		3.8	0.5	0.5	3.5	0.7		0.6		
Value & Resp.	P	*Manual testing work is frustrating.*	5.4	0.8		2.2								
Comp. Policies	P	*Verification guidelines are not obeyed.*	1.5			0.5								
Work Practices	M	*Lack of test automation.*	4.6	0.8		8.1	3.2	0.5	2.1	2.1				
Process	M	*Process for software testing is missing.*	3.1	0.8		2.2	0.5		1.4	0.7				
Monitoring	M	*Test coverage not reported.*	1.5	1.5		1.1								
Task Output	T	*Low quality of testing work.*	3.1	1.5	0.8	6.5	1.1	0.5	3.5	2.1		1.2	1.2	
Task Difficulty	T	*Difficult to define inoperative test cases.*	3.8	0.8		6.5	1.1		2.8	2.8	0.7	0.6	0.6	
Task Priority	T	*Other tasks are prioritized higher than testing.*	1.5											
Existing Product	E	*Units of the old code cannot be separated.*	2.3			4.3	1.6		0.7					
Res. & Sch.	E	*No schedules for software testing.*	6.2	3.8		2.2			1.4			1.2	0.6	
Cust. & Users	E	*No testing in the production environment.*							2.1	1.4				
Tools	E	*Lack of tools to detect the side effects.*				1.1			0.7					
Release & Deployment			*1.6*			*4.7*	*1.0*		*52.5*	*18.9*	*2.1*	*2.4*	*2.4*	
Inst. & Exp.	P	*Lack of instructions on how to install the system.*	0.8						14.7	4.9	1.4			
Comp. Policies	P	*Blockers were detected, but release was done.*				0.5	0.5							
Work Practices	M	*Most of the tasks have to be conducted manually.*				0.5	0.5		10.5	4.9		1.2	1.2	
Process	M	*The release process is time boxed.*				1.1			0.7					
Monitoring	M	*It is not monitored which installations are in use.*							0.7	0.7				
Task Output	T	*Wrong files are overwritten during installation.*				0.5			4.2	2.8		0.6	0.6	
Task Difficulty	T	*The product installation is difficult.*	0.8						6.3	3.5	0.7			
Existing Product	E	*System configuration files are scattered.*				0.5			9.1	2.1				
Res. & Sch.	E	*Too busy to do installations carefully.*				1.6			2.1			0.6	0.6	
Cust. & Users	E	*Some installations are done to customers' servers*							4.2					
Unknown Process Area			2.2	0.8	-	1.1	0.5	-	-	-	-	1.2	0.6	-
TOTAL %			100	30.0	4.6	100	24.9	2.7	100	38.5	3.5	100	46.1	3.6
Total $\sum$			130	39	6	185	47	5	143	55	5	172	79	6

practices (avg. 16%, std. 4%), task output (avg. 16%, std. 2%), task difficulty (avg. 7%, std. 3%), existing product (avg. 7%, std. 5%), and resources & schedules (avg. 9%, std. 4%).

Another similarity between the cases is related to the selected and proposed causes. While the detected causes (a total 630) distributed equally to the cause types, the selected (a total 22) and proposed (a total 216) causes distributed mostly into the cause types of People and Tasks. At each case, the selected causes were most often related to the cause types of People (avg. 41%, std. 7%) and Tasks (avg. 45%, std. 15%). Furthermore, the proposed causes were most often related to the cause types of People (avg. 26%, std. 6%), Tasks (avg. 29%, std. 9%), and Methods (avg. 27%, std. 6%).

Considering the bridge causes, it seems that the company people perceived them as feasible targets for process improvement. Figure 8 shows that the proportion of the bridge causes increases in the proposed (average 56%) and selected causes (average 68%) when compared with the detected causes (average 50%). This indicates that the company people perceived being feasible to control the causes related to the causal relationships interconnecting the process areas.

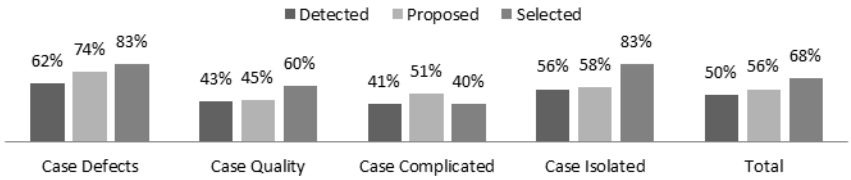

Figure 8. Proportion of the bridge causes in the detected, proposed, and selected causes

4.5.2 Common Causal Relationships Bridging the Process Areas

Figure 9 summarizes the common causes and related causal relationships bridging the process areas. In prior studies, the cause of failure has been concluded as "common" if it occurs in 60% to 80% of the software project failures, e.g., in [8, 16]. Our inclusion criterion for combining the cases was that the cause and its related causal relationship occurred at least in three of our four cases, i.e., 75%.

Similar causes were often related to similar causal relationships. Despite the fact that the failures selected for analysis in the cases were highly different, three common causal relationships bridging the process areas, shown in Figure 9, stand out in our cases: *Weak Task Backlog*, *Lack of Cooperation, and Lack of Software Testing Resources.*

The causal relationship *Weak Task Backlog* bridges the sales & requirements, management, implementation, and software testing. Management is a cause for Weak Task Backlog through incorrect decisions on task priorities, i.e., features vs. quality. Caused by lack of instructions & experience, and the high priority of new feature development, it was claimed to be difficult for managers to prioritize hundreds of opaque tasks. Sales & requirements is also a cause for Weak Task Backlog through vague requirement specifications. Caused by lack of instructions & experience, it was claimed to be difficult to write good requirement specifications. Implementation is an effect of Weak Task Backlog through vague task descriptions and incorrect task priorities. Caused by opaque specifications, it was difficult for the developers to know what their tasks required. The high priority of new feature development caused difficulties in implementation through increasing technical debt. Software testing is a cause and effect for Weak Task Backlog through missing verification criteria and vague defect reports.[1] As an effect, the testers did not know what to verify. As a cause, the defect reports registered to the task backlog were vague complicating the work of managers when prioritizing the tasks.

The causal relationship *Lack of Cooperation* interconnects sales & requirements, implementation, and software testing. Missing information in sales & requirements is an effect of Lack of Cooperation through the missing assistance of developers while making the specifications. Caused by the difficult existing product and the lack of instructions, the people of sales & requirements were incapable of documenting the specifications detailed enough to support the implementation and software testing. Similarly, missing information in implementation is an effect of Lack of Cooperation through the inactive assistance from the people of sales & requirements while the developers are trying to understand the task descriptions. Furthermore, missing information in software testing is an effect of Lack of Cooperation through the inactive assistance from the people of sales & requirements and implementation while the testers try to understand what to verify. The common cause for Lack of Cooperation seems to be missing.

[1] this cause was detected only at cases Quality and Defects

Perhaps the managers did not force the people to work together or maybe the people ignored cooperation, as it was in Case Isolated.

The causal relationship *Lack of Software Testing Resources* bridges management and software testing. Management is the cause for Lack of Software Testing Resources through the causes related to values & responsibility. In Case Defects, the testers were forced to do other tasks than testing, and thus they simply did not have enough time to do testing. In Case Quality, the managers allowed new feature development at the very end of the project schedule, and thus there were not enough testing resources available to verify that the developed features actually worked. In Case Complicated, the resource allocation of software testing was considered a failure without more detailed explanations. In Case Isolated, the tasks related to software testing were allocated to the wrong testers having only a little knowledge on the issue to be verified.

Many of the causes were a case specific. Thus, explaining the failures solely with the common causal relationships between the process areas is infeasible, but improves the knowledge related to possible common causes of software project failures. Analyzing the causal relationships between the causes helps us understand why the failures occur. However, the common causal relationships alone or even at together do not explain any of our cases solely. Comparison of the case-specific results (Table 4, and Figures 4 to 7) shows that each failure was also caused by different, case-specific causes having different causal relationships. Additionally, none of the common causes were either proposed or selected at every case, and causes other than the common ones were proposed at each case.

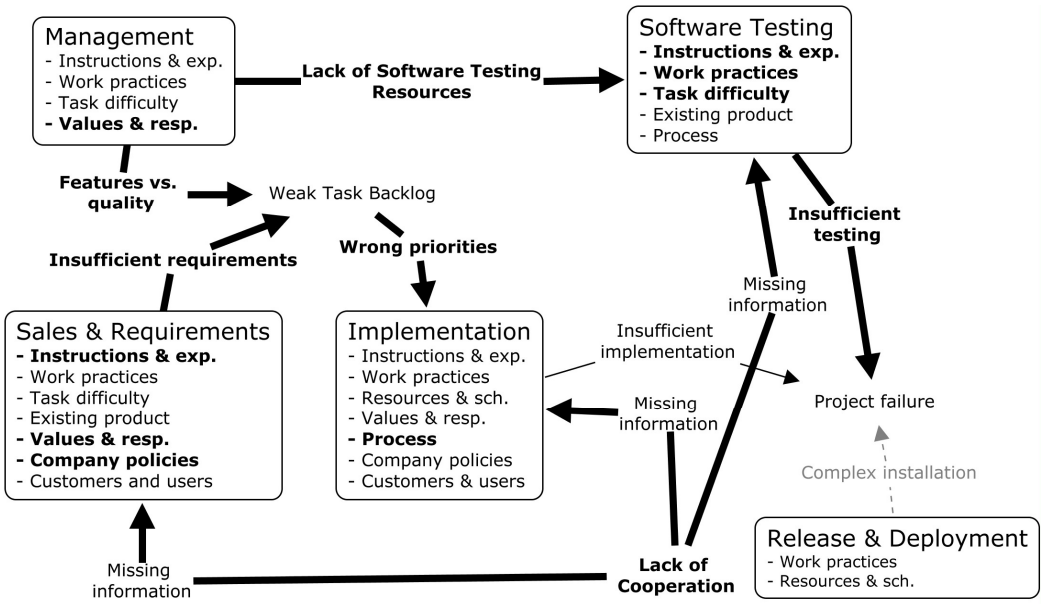

Figure 9. Common causes and bridging causal relationships found in at least three out of the four cases
(Bolded text/line indicates the selected causes; Normal text/line indicates the sub-causes of the selected causes; Dashed line/grey text indicates that the cause was neither a selected cause nor a sub-cause; Lines with arrows and text between the process areas indicate the direction of causal relationships interconnecting the process areas)

5. Discussion

In this section, we answer the research questions and compare our findings with the prior studies of common causes for software project failures. Additionally, we discuss the validity threats related to our conclusions.

5.1 Answering the Research Questions

RQ1: *Which process areas and cause types were frequently used to explain the software project failures?*

The causes of software project failures analyzed in this study were equally distributed into the types of People, Tasks, Methods, and Environment, see Section 4.5. We believe that this finding can be generalized as similar conclusions are also presented in prior studies. McLeod at al. [27] summarized in an extensive survey of literature that the outcome of software system development is affected by People & Action, Development Processes and Project Content.

The causes of failures were commonly related to the instructions & experience, values & responsibilities, work practices, task output, task difficulty, existing product, cooperation, and resources & schedules. On average, 81% of the detected causes in the cases (std. 2%) were related to these sub-types of causes. We believe that this finding is generalizable as it can be logically compiled and prior studies [27] have resulted in similar findings. In software product companies, the projects are related to maintaining and improving the existing product. Over the years, the existing product grows, increasing the complexity of both the product and the project tasks. This leads to an increasing need for instructions & experience, values & responsibilities, cooperation, resources & schedules, and improved work practices. McLeod at al. [27] present that development practices and the project outcome are influenced by domain knowledge, experience, values & beliefs, communication & social skills, and motivation.

Considering the process areas, the causes were related to management, sales & requirements, implementation, software testing, and release & deployment. The cause distributions over process areas varied between the cases, see Table 4, which indicates that the process areas where the causes occur are dependent on the case context. However, despite the differences between the cases, their commonality was that each of them was influenced by insufficient management, as indicated by Verner et al. [16], and by the problems of software testing and implementation.

In the prior studies of the common causes of failures, management and sales & requirements are mentioned most frequently. For example, inaccurate estimating and schedule planning [8, 10, 11, 16, 45, 46], overoptimistic status reporting [10], insufficient quality control [10, 16, 46], unrealistic schedule pressures [8, 10, 16], lack of top management support [11, 13, 16, 45], weak project manager [13, 46], task priorities [13], and insufficient development process [14, 16] are all part of management, which influences software project failures. Furthermore, unrealistic expectations of customers [8, 11, 16, 18], lack of customer support when gathering the requirements [8, 11, 14, 16], changing scope [8, 11, 13, 16, 45], scope creep [14], failure to specify appropriate measures [18], and inadequate requirements [11, 13, 14, 18, 46] indicate that there are many improvement opportunities at the sales & requirements too.

Interestingly, prior studies have recognized, but not emphasized, the central role of implementation and software testing in causing software project failures. In our cases, these two process areas included the highest number of causes and they had a central role at each failure. It was claimed that the quality of the implementation was low. The work was not well reported and the defect density was high. These problems were influenced by the values & responsibility and prioritizing implementation of features higher than quality. Other problems of the implementation were related to the technical debt and low maintainability of the existing (legacy) product. Furthermore, it was also claimed that the quality of software testing was low. The defect reports were vague and a high number of defects were not detected. Software testing was claimed to be a difficult task. It was challenged by the existing product and lack of resources & schedules. Software testing and implementation are essential parts of software engineering. However, despite this, the prior studies have not introduced these two as highly common reasons for project failures. We hypothesize that this is due to the data collection methods. Using the RCA workshops allowed many individuals to participate and encouraged deeper thinking instead of using factor-based large-scale surveys [27] resulting in the answers of premade questions that are filled out by a single company employee. Furthermore, we studied product companies rather than bespoke software projects, which might have caused these differences.

RQ2: *What causal relationships bridge the process areas?*

A high number of the causes of failures in implementation, software testing, and release & deployment were interconnected to the output of management and sales & requirements. This finding contributes and consolidates the prior studies as it indicates that software project failures are caused by management work and sales & requirements, see Figure 9. For example, the lack of resources in software testing was claimed to be caused by management, and similarly, the lack of instructions in implementation was claimed to be caused by sales & requirements. Thus, in order to prevent the failures, should we focus on management and sales & requirements only?

Management was influenced by sales & requirements and software testing, which means that solving the problems related to management requires improvements in these process areas. Task difficulty, lack of instructions & experience, work practices and values & responsibility were the types of causes related to management. These types were used to explain the insufficient task output including the wrong prioritization of implementation work and lack of resources & schedules in software testing. However, solving these two problems requires improvements in sales & requirements and software testing. In three cases, the participants claimed that the requirements were vague. In two cases, the participants claimed that the test reports were insufficient. The managers were doing resource allocation and prioritization with inadequate information. Furthermore, in one case, the participants noted that the service level agreement with the customers forced the managers to prioritize the customer requests higher than fixing defects. As an effect, severe defects and low priority customer wishes were transposed. The low priority of software quality was perceived as a problem in each case.

Sales & requirements was influenced by implementation. This means that solving the problems of sales & requirements requires improvements in implementation. The insufficient task output of sales & requirements was caused by instructions & experience, work practices, task difficulty, existing product, values & responsibilities, company policies, and customers & users. Making good requirements was likely very difficult in the case companies. The existing products were complex and required extensive technical knowledge. Additionally, the managers, developers, and testers had somewhat different needs, which meant that extensive practical knowledge was needed while documenting the requirements. In three cases, the participants noticed that the existing product was so complex that it was likely that nobody knew the system fully. It was also claimed that the customers and users cannot express the features they really need and that the customer requests are not well analyzed by the company people. Furthermore, in one case, the participants claimed that the process for handling the customer requests was ignored. Considering these challenges, it was claimed that the people at the sales & requirements were not well instructed, especially because of the lack of cooperation with the implementation.

Lack of Cooperation, Weak Task Backlog, and Lack of Software Testing Resources indicated causal relationships common to the software project failures of our cases. Obviously, these three are also related to one another. The people in sales & requirements, implementation, and software testing did not work enough together. In two cases, the participants used the problem of lack of cooperation to explain why the output of sales & requirements was insufficient for the developers and testers. In two cases, the problem was relevant between the developers and testers. Respectively, in two cases, the defect reports were claimed as vague, which made it difficult for the managers to monitor the status of ongoing project and for the developers to fix the defects. Furthermore, in each case, the managers prioritized new features higher than defect fixing, which was partially caused by the insufficient output of sales & requirements and software testing, but also because of the values of the managers. The values of the managers also explained the lack of software testing resources, a problem in each case. Furthermore, in each case, the participants used the low priority of defect fixing to explain why the implementation resulted in low product quality. While considering what the managers could have done better, enabling the cooperation and critically monitoring the task backlog would have been important. This is because many of the causes of failures were interconnected to the cooperation and task backlog. Considering the values of the managers related to the defect fixing, we should understand that technical debt is not always a negative thing for the software product company [74].

Dependencies over various process areas have been introduced [58] and causal relationships between the common causes of software project failures have been considered [8]. Our results indicate that many of the common causes of failures are not isolated. Instead, they explain one another through the bridge causes. For example, in Case Isolated, vague requirements were explained by the lack of collaboration which was explained by stakeholder conflicts. Prior work has presented the same causes as separated [17]. In order to correctly react to the failures, we need to understand, not only the causes of failures, but the actual mechanism of how the multiple causes together manifest as a failure.

More attention should be given to analyzing the causal relationships between the causes. Considering the failures of our cases, it would be too general to state that the failures were caused by the lateness or inaccurate scope estimations only, as presented in [75]. Instead, our case companies needed to understand why the lateness and inaccuracy occurred. The quality of implementation and software testing had a central role in the failures. Additionally, management and sales & requirements explained the problems occurring in implementation, software testing, and release & deployment. Furthermore, management and sales & requirements were influenced by the complex existing product and organizational issues including motivation and collaboration. In order to prevent the failures, there was a need to manage the existing product including its requirement debt, test debt, architectural debt, and documentation debt. Additionally, there was a need to improve the collaboration and motivation among the company people.

RQ3: *Do the causes perceived as feasible targets for process improvement differ from the other detected causes, and if so, how?*

Considering the distributions of the causes in the process areas, it seems that the proposed and selected causes do not differ from the other detected causes. The process areas that include the highest number of detected causes also include the highest number of proposed and selected causes. Similarly, it seems that the process area that includes the highest number of causes in one case does not include the highest number of causes in the other cases. This indicates that the importance of a specific process area for process improvement, e.g., management, is dependent on the case context.

Our results show that the causes of failures perceived as feasible targets for process improvement are related to the cause types of Tasks, People, and Methods. Most of the selected causes are related to Tasks (45%) and People

(41%). Furthermore, when looking at causes proposed for process improvement, the types of Tasks (29%), People (26%), and Methods (27%) were more frequent than Environment (18%). A comparison of the selected and proposed causes with the detected causes shows that the share of the causes related to Tasks, People and Methods increases. Furthermore, the comparison of the selected causes with the proposed causes shows that the selected causes are related to People highly more often than Methods. This indicates that the key representatives perceived that the improvements need to be focused on the company employees whereas the case participants perceived that also the work practices should be improved.

Xiangnan et al. [25] used the concept of internal and external causes to indicate the causes of failures that are under the control of the project team. They claimed that the internal causes are under the control of the project team and they include the project manager, project team, process & technology, and completion of project delivery results. Instead, the external causes are not under the control of the project team and they include the customers and causes related to other stakeholders. This may explain why only a few of the proposed and none of the selected causes included causes related to customers. Furthermore, the causes related to customers belong to the sub-type of Environment. The other sub-types of Environment were rarely proposed or selected. Perhaps such causes are more difficult, or even impossible, for the company to control compared with those related to Tasks, Methods and People.

Considering the causal relationships interconnecting the process areas, we hypothesize that the causes of failures perceived as feasible targets for process improvement point out and explain weaknesses between the process areas. The causes of failures were distributed into various process areas at each case whereas the failures surfaced at implementation, software testing, and release & deployment. This means that you must be willing to make process changes outside of the process areas where the failure surfaces. Comparison of proposed and selected causes with the other detected causes shows that the proportion of the bridge causes increases in the proposed and selected causes. This means that the company people found such causes as feasible targets for the process improvement. Perhaps this is because these causes cannot be controlled solely by the people in the process area where the failure surfaces, as indicated by Keil et al. [11]. For example, the failure in Case Defects, i.e., “Fixing and verifying defects delay the project schedules”, surfaced at implementation and software testing. However, the failure of Case Defects cannot be prevented by making improvements only to the implementation and software testing, as the failure was also caused by sales & requirements and management. This was also the case in the other companies.

5.2 Implications

In this section, we present the implications of our results and provide recommendations for future works. We start with a discussion on the implications for the causal analysis of software project failures. Thereafter, in Section 5.2.2, we discuss the applicability of the RCA method for causal analysis. Finally, we discuss the implications of our results for software outcome prediction models.

5.2.1 Causes of software project failures

We propose that studying the relationships between the factors affecting the outcome of software projects requires modeling the software development as a system where process areas are interconnected multi-directionally. McLeod et al. [27] claimed that people and technology are interconnected through multidimensional processes. Our results indicate that a software project failure is an effect of various important causes bridging the process areas together. Considering software product development as a set of linked activities [59], the direction of cause and effect relationships over the bridge causes follows the logical workflow from sales & requirements to verification and software release. Thus, the detection of bridge causes could start by considering problems in the workflow items (e.g. requirements, resource allocations, estimations, developed features, test reports, etc.). Studying the problems in the workflow items, however, requires explaining why the problems occur. In the cases of this study, the process areas were bridged together multi-directionally. The insufficient task output of process areas created one direction, but insufficient social and informative interaction between people in different process areas created another direction being more exploratory and unforeseeable. The bridge causes related to the interaction between people explained the bridge causes related to the task output. Thus, explaining the problems in workflow items required that possible bridge causes and local causes in all process areas were considered.

We also noted that software project failures are different and dependent on the case context. Therefore, the data collection should not be limited to the common causes of failures, because such causes alone do not cover the case specific problems comprehensively enough. For example, the common causes of software project failures introduced in Section 2.1 would not have covered the project failures in Case Complicated (see Section 4.3) where the process areas of release & deployment, implementation work and software testing had a central role. Despite that many of

the causes of failures were common in the cases of this study, a high number of cause and effect relationships were case specific. Thus, a case specific data collection and analysis was needed.

5.2.2 Root cause analysis

We used RCA in the data collection. Therefore, considering the outcome of RCA in the context of software project failures, our results contribute to the research on RCA. It seems that in the case of software project failures, RCA helps to explain the perceptions of domain experts on what happened, where it happened, and why it happened. Prior studies have used questionnaires to conclude the causes of project failures [27]. The difference between these two approaches is that RCA provides information about the perceived cause and effect relationships. On the other hand, a questionnaire helps to generalize the findings as it can be used to collect information from a high number of subjects. We hypothesize that future works aiming to understand the causes of software project failures should utilize both heavy data collection for statistical analysis as well as RCA depicted in this paper. RCA could be used to model the problem domain whereas questionnaires could be used to generalize it.

Considering the data analysis, the advantage and disadvantage of RCA is a high number of detected causes. The high number of causes helps to reveal process improvement targets [4]. However, this also makes it somewhat challenging to conclude the causal mechanisms related to the failure under analysis. Thus, the data analysis techniques for RCA should be a part of future works. In order to reveal the causal mechanisms, we found it useful to classify the detected causes into four dimensions: process area, type, interconnectedness, and feasibility for process improvement (see Section 3.3). This made it possible to model the bridge causes and local causes. Additionally, we were able to characterize the causes perceived as the most important for process improvement. We hypothesize that our analysis approach is useful when RCA is applied to software project failures. However, this approach needs further validation.

The efficiency and ease-of-use of RCA were evaluated by the case participants of the companies. The participants considered RCA as more efficient than the prior practices used in the companies. Additionally, they perceived RCA as easy-to-use. The analysis resulted in tens of high quality corrective actions, which resulted in process changes in the companies. Thus, we hypothesize that RCA is a useful practice to detect process improvement opportunities in software product companies. Considering the limitations, we used the ARCA method to conduct RCA. Therefore, the evaluation results of RCA are also limited to the work practices of ARCA. The evaluations of the participants in detail can be found in our prior paper [4], which also includes the development of ARCA and literature review about prior RCA methods.

5.2.3 Prediction models

Prediction models have been used to predict the likelihood of success and failure in ongoing software projects [76, 77]. The models are used to consider corrective actions for a failure before the failure occurs [76]. They take the state of risk factors as an input and provide the likelihood of a failure as an output [76, 77]. The knowledge about the current state of risk factors is based on questionnaires [76, 77]. Thus, the accuracy of prediction models is dependent on the accuracy of the used questionnaire. The accuracy of prediction models is also dependent on the causal model used in the likelihood calculation. The causal model is based on prior statistical evidence about the correlations between the risk factors [77, 78].

We believe that our results provide useful domain knowledge for prediction models. Alaeddini and Dogan [79] introduced an approach where the output of RCA is used with the Bayesian network in order to improve the real-time identification of the causes of failures. Cheng and Greiner [80] claimed that one way to improve the prediction accuracy is to use “domain knowledge” when creating the causal model. Such knowledge includes the order of causes and effects, forbidden links, and cause and effect relationships [80]. Steck and Tresp [78] presented a similar idea by stating that learning the structure of data provides better information about the domain than basing the reasoning on correlations and distance measures only. Our results provide rich knowledge about the situation in a particular case, including cause and effect relationships between the risk factors and the process areas where a specific risk factor causes a problem. Thus, we assume that our results could improve the prediction models. First, the bridge causes and local causes could be used to extend the questionnaires of risk factors used in the prediction models. Second, they could be used to consolidate the potential cause and effect relationships found by using statistical correlations. Third, their types and process areas (see Table 4) could help to consider various important aspects while making the questionnaires. Considering the risk factors used in prior works [48, 77], it seems that our results include more factors particularly in the process areas of implementation, software testing, and release & deployment. By extending the prior works with these missing process areas, it could be possible to detect bridge causes from the requirements and management by using statistical methods.

We conclude that the outcome of prediction models is important for RCA and vice versa. We also believe that RCA could be used among the questionnaire in order to improve the knowledge about the current state of the project under analysis, as presented in [63]. RCA is not limited to specific risk factors as it is with premade questionnaires, e.g., [77]. Instead, the outcome of RCA reflects the "true" domain knowledge being limited to the experience of the group of participants.

5.3 Threats to Validity

This section discusses the validity of our empirical results using a validation scheme presented by [67]. We will present the construct validity, the reliability, and the external validity of the study.

5.3.1 Construct Validity

Construct validity indicates whether the studied operational measures really represent what is investigated according to the research questions [67]. The validity of the RCA outcome has been criticized as relying strongly on assumptions and critical thinking of practitioners [81]. Using the ARCA method creates a threat to the construct validity as the research data was relying on human input. This means that the causes of failures and their causal relationships were perceived, and dependent on the experience, awareness, memory, expertise and analytical capabilities of the case participants. In our prior work [4], we evaluated the accuracy, correctness and usefulness of the detected causes of this study by utilizing interviewing, questionnaires, and observations. Even though our results indicate that the outcome of the ARCA method was correct, we were not able to prove that the detected causes and their causal relationships were 100% correct and completely covered the failures. It should be noted that this problem is also relevant in other studies utilizing RCA, interviews, or surveys.

In order to analyze the causes of failures and their causal relationships systematically, we developed a detailed cause classification system. Regardless of our effort trying to make the classification system as comprehensive as possible, classifying the causes of failures always dissipates the dissimilarities and simultaneously highlights the similarities. This means that there is a risk that using the classification system creates systematic errors. This validity threat was controlled by developing the classification system based on the detected causes. Thus, our cause classifications likely correspond the detected causes accurately.

5.3.2 Reliability

Reliability indicates the extent to which the data and analysis are dependent on a specific researcher [67]. Considering the reliability of the classification system, inter-rater agreement of the classifications was analyzed by using Kappa values for randomly selected 10% of the cause statements of each case resulting to the classification of 62 causes by the second author of this paper. Kappa value for the process area dimension was 0.65 that can be concluded as good agreement between the raters. Kappa value for the type dimension was 0.55 that can be concluded as moderate agreement between the raters. As far as we know the reliability of cause classification systems has not previously been reported. In comparison with the reliability of the classification systems used in code reviews, higher Kappa values 0.80 [82], 0.76 [82], and 0.79 [83] have been achieved. We think that the classification systems of code review defects are more mature and code review defects are simpler to classify as the possible defects are limited to the expressiveness of the programming languages while the causes of failures were expressed in natural language which is richer but also more opaque reducing the agreement level between raters. Additionally, our classification system includes a higher number of categories and dimensions.

The qualitative analysis on the causal relationships interconnecting the process areas creates a threat for the reliability, because the qualitative analysis was based on the interpretations of the first author. As already presented, the inter-rater agreement for the process area dimension was good. This means that the causes analyzed qualitatively likely covered most of the causal relationships bridging the process areas. Furthermore, the causes bridging two process areas were mostly similar and their amount was low (around one to five). Thus, it was not difficult to summarize the relationships when possible.

5.3.3 External Validity

External validity indicates whether it is possible to generalize the findings of the study [67]. All of our cases varied and thus considered the causes of failures and their causal relationships from different perspectives, e.g., the case companies, the case attendees, and the case failures varied. This increases the external validity. There were also similarities between the cases, which made them more comparable while consolidating the case study results, e.g., each failure was highly complex, similar stakeholders were present at each case, and the same amount of effort (120 minutes) was used at each causal analysis workshop.

Considering the results on the types of the causes of failures and the importance of bridge causes for process improvement, we believe that the external validity is high (see Section 5.1). However, as discussed earlier, the specific causes detected in the cases varied. Therefore, the validity of the results on the common causes for failures needs further validation. The software projects studied in this paper were medium-sized (less than 100 people involved), which means that we cannot conclude that our results are valid in small-sized (a few people) or large-sized projects (hundreds people). Additionally, the projects that we studied were geographically distributed and most of them were conducted in European countries only. Therefore, we cannot generalize our findings to collocated projects or other than western cultures. Furthermore, the software development processes used in the projects varied from "more traditional" to "less traditional" processes. Thus, we cannot limit our results to traditional waterfall based software development processes or modern agile methods either.

Our study is based only on four cases conducted in product companies, and as far as we know, there are no prior studies on the causal relationships between the causes of software project failures. Thus, it was difficult to compare our findings with the findings of the prior studies. Therefore, in order to increase the external validity of our results, replicative studies are needed in different case contexts including projects with different size, cultures, geographical distribution, software development methods, and failures.

6. Conclusions

This paper makes four contributions. First, our results indicate that there is no single root cause for software project failures, as also claimed in [16]. Instead, a software project failure is the result of several causes — in our cases, we had 130 to 185 causes per analyzed failure. Furthermore, these causes form a network where the causes are connected to each other. Additionally, the causes come from many process areas. This matches prior works (see Section 2.1) arguing that software project failures are influenced by social and technical causes, which are spread over various process areas. However, in the prior works, analyses on the causal relationships between the causes of failures are missing, an oversight that we have tried to correct in this paper.

Second, our results consolidate the prior works by showing that a software project failure is a result of a multidimensional process where people, tasks, methods and project environment are interconnected. Lack of cooperation, weak task backlog, and lack of software testing resources were common bridge causes for the failures of our cases. These causes interconnected process areas of management, sales & requirements, implementation, software testing, and release & deployment. Furthermore, the prior studies have commonly recognized the process areas of management, requirements engineering, and implementation causing failures. We found that also software testing has a central role in the software project failures. Based on our industrial experience it seems unlikely that the case companies in this paper would have had more problems in software testing than companies in average. We hypothesize that the discrepancies between this work and prior studies are caused by the data collection methods employed.

Third, the bridge causes, and causes related to tasks, people, and methods were particularly common among the causes perceived as the most feasible targets for process improvement. The causes of software project failures were equally distributed into the types of People, Tasks, Methods, and Environment. However, the causes related to Environment were seldom perceived as feasible targets for process improvement. We also found a high number of bridge causes interconnecting the process areas, 50% on average. The bridge causes had even a higher share of the causes perceived as feasible for process improvement, 68% on average. For software practitioners, this means that to fix weaknesses in a specific process area, one must be willing to make changes outside of the process areas where the failure surfaces, e.g., the failure of ineffective testing might be most effectively fixed with improvements to collaboration between developers and sales people rather than hiring more testers.

Fourth, our results indicate that the causes of failures and their causal relationships are diverse and depend on the case context. Despite the notification that some of the causes of failures were common between the cases, the detailed analysis showed that a high number of the causes were a case specific. Thus, a case specific analysis is likely needed every time a failure occurs.

Considering future work, we believe that the research methods used in this study allow constructing *the story behind the data,* which is a key component also in more formal works on causal reasoning [54, 84]. We see that future works aiming to understand the causes of software project failures should utilize both heavy data collection for statistical analysis as well as RCA depicted in this paper. Replicating studies are needed to increase the external validity of our findings and test our hypotheses. More empirical research is needed to better understand the complicated mechanisms and relationships of causes leading to software project failures. Furthermore, for industry it is recommended that process improvement is done with teams that represent different process areas as causes interconnecting process areas were often perceived the most important for fixing.

References

[1] P. Naur and B. Randel, Software engineering: A report on a conference sponsored by the NATO science committee, Nato, 1969.

[2] A. Burr and M. Owen, Eds., Statistical Methods for Software Quality: Using Metrics for Process Improvement. ITP A division of International Thomson Publishing Inc., 1996.

[3] J. J. Rooney and L. N. Vanden Heuvel, Root cause analysis for beginners, Quality Progress 37 (7) (2004) 45 - 53.

[4] T. O. A. Lehtinen, M. V. Mäntylä and J. Vanhanen, Development and evaluation of a lightweight root cause analysis method (ARCA method) – field studies at four software companies, Information and Software Technology 53 (10) (2011) 1045-1061.

[5] D. N. Card, Learning from our mistakes with defect causal analysis, IEEE Software 15 (1) (1998) 56-63.

[6] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray and M. Wong, Orthogonal defect classification - A concept for in-process measurements, IEEE Transactions on Software Engineering 18 (11) (1992) 943 - 956.

[7] K. Ishikawa, Ed., Introduction to Quality Control. 3A Corporation, Shoei., 6-3, Sarugaku-cho 2-chome, Chiyoda-ku, Tokyo 101, Japan: JUSE Press Ltd., 1990.

[8] N. Cerpa and J. M. Verner, Why Did Your Project Fail? Communications of the ACM 52 (2009) 130-134.

[9] J. Dye and T. van der Schaaf, PRISMA as a quality tool for promoting customer satisfaction in the telecommunications industry, Reliability Engineering & System Safety 75 (3) (2002) 303-311.

[10] J. Capers, Social and technical reasons for software project failures, Crosstalk the Journal of Defense Software Engineering (6) (2006) 4-9.

[11] M. Keil, P. E. Cule, K. Lyytinen and R. C. Schmidt, A framework for identifying software project risks, Communications of the ACM 41 (11) (1998) 76-83.

[12] K. A. Demir, A survey on challenges of software project management, Proceedings of the 2009 International Conference on Software Engineering Research Practice, 2009, pp. 13-16.

[13] L. A. Kappelman, R. McKeeman and L. Zhang, Early Warning Signs of IT Project Failure: The Dominant Dozen, Information System Management 23 (2006) 31-36.

[14] K. Moløkken-Østvold and M. Jørgensen, A comparison of software project overruns - flexible versus sequential development models, IEEE Transactions on Software Engineering 31 (9) (2005) 754-766.

[15] K. Moløkken-Østvold and M. Jørgensen, A review of survey on software effort estimation, Proc. 2003 ACM-IEEE Int'l Symp. Empirical Software Eng. (ISESE 2003), 2003, pp. 220-230.

[16] J. Verner, J. Sampson and N. Cerpa, What factors lead to software project failure, Proceedings of Research Challenges in Information Science (RCIS 2008), 2008, pp. 71-80.

[17] L. J. May, Major Causes of Software Project Failures, Crosstalk the Journal of Defense Software Engineering 11 (1998) 9-12.

[18] J. M. Verner and L. M. Abdullah, Exploratory case study research: Outsourced project failure, Information and Software Technology 54 (2012) 866-886.

[19] R. B. Grady, Software failure analysis for high-return process improvement decisions, Hewlett-Packard Journal 47 (4) (1996) 15 - 25.

[20] J. Jacobs, J. Van Moll, P. Krause, R. Kusters, J. Trienekens and A. Brombacher, Exploring defect causes in products developed by virtual teams, Information and Software Technology (47) (2005) 399-410.

[21] M. Leszak, D. E. Perry and D. Stoll, A case study in root cause defect analysis, Proceedings of the 2000 International Conference on Software Engineering, 2000, pp. 428 - 437.

[22] T. Nakashima, M. Oyama, H. Hisada and N. Ishii, Analysis of software bug causes and its prevention, Information and Software Technology (41) (1999) 1059-1068.

[23] T. Stålhane, Root cause analysis and gap analysis - A tale of two methods, EuroSPI 2004, Trondheim, Norway, 2004, pp. 150 - 160.

[24] M. Kalinowski, G. H. Travassos and D. N. Card, Towards a defect prevention based process improvement approach, Proceedings of the 34th EUROMICRO Conference on Software Engineering and Advanced Applications, Parma, Italy, 2008, pp. 199 - 206.

[25] L. Xiangnan, L. Hong and Y. Weijie, Analysis failure factors for small & medium software projects based on PLS method, The 2nd IEEE International Conference on Information Management and Engineering (ICIME) 2010, pp. 676-680.

[26] X. Lu, H. Liu and W. Ye, Analysis failure factors for small & medium software projects based on PLS, Proceedings of Information Management and Engineering (ICIME 2010), 2010, pp. 676-680.

[27] L. McLeod and S. G. MacDonell, Factors that affect Software Systems Development Project Outcomes: A Survey of Research, ACM Computing Surveys 43 (2011) 24-55.

[28] R. J. Latino and K. C. Latino, Eds., Root Cause Analysis: Improving Performance for Bottom-Line Results. 6000 Broken Sound Parkway NW, Suite 300 Boca Raton, FL 33487-2742: CRC Press, 2006.

[29] F. O. Bjørnson, A. I. Wang and E. Arisholm, Improving the effectiveness of root cause analysis in post mortem analysis: A controlled experiment, Information and Software Technology 51 (1) (2009) 150 - 161.

[30] P. Jalote and N. Agrawal, Using defect analysis feedback for improving quality and productivity in iterative software development, Proceedings of the Information Science and Communications Technology (ICICT 2005), 2005, pp. 701 - 714.

[31] R. G. Mays, Applications of Defect Prevention in Software Development, IEEE Journal on Selected Areas in Communications 8 (1990) 164-168.

[32] S. O. Al-Mamory and H. Zhang, Intrusion detection alarms reduction using root cause analysis and clustering, Computer Communications 32 (2) (2009) 419 - 430.

[33] M. Siekkinen, G. Urvoy-Keller, E. W. Biersack and D. Collange, A root cause analysis toolkit for TCP, Computer Networks (2008) 1846-1858.

[34] A. Traeger, I. Deras and E. Zadok, DARC: Dynamic analysis of root causes of latency distributions, SIGMETRICS '08, Annapolis, Maryland, USA, 2008, pp. 277-288.

[35] I. Bhandari, M. Halliday, E. Tarver, D. Brown, J. Chaar and R. Chillarege, A case study of software process improvement during development, IEEE Transactions on Software Engineering 19 (12) (1993) 1157-1170.

[36] Z. X. Jin, J. Hajdukiewicz, G. Ho, D. Chan and Y. Kow, Using root cause data analysis for requirements and knowledge elicitation, International Conference on Engineering Psychology and Cognitive Ergonomics (HCII 2007), Berlin, Germany, 2007, pp. 79 - 88.

[37] W. Al-Ahmad, K. Al-Fagih, K. Khanfar, K. Alsamara, S. Abuleil and H. Abu-Salem, A taxonomy of an IT project failure: Root Causes, International Management Review 5 (2009) 93-104.

[38] R. L. Glass, Evolving a new theory of project success, Communications of ACM 42 (1999) 17-19.

[39] N. Agarwal and U. Rathod, Defining 'success' for software projects: An exploratory revelation, Int. J. Project Manage. 24 (2006) 358-370.

[40] J. D. Procaccino, J. M. Verner, K. M. Shelfer and D. Gefen, What do software practitioners really think about project success: an exploratory study, J. Syst. Software 78 (2005) 194-203.

[41] T. O. A. Lehtinen and M. V. Mäntylä, What are problem causes of software projects? Data of root cause analysis at four software companies, ESEM '11 Proc. of the 2011 International Symposium on Empirical Software Engineering and Measurement, 2011, pp. 388-391.

[42] M. Nasir and S. Sahibuddin, Critical success factors for software projects: A comparative study, Scientific Research and Essays 6 (2011) 2174-2186.

[43] N. H. Arshad, A. Mohamed and Z. Matnor, Risk factors in software development projects, Proceedings of the 6th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, 2007, pp. 51-56.

[44] M. Tarawneh, H. AL-Tarawneh and A. Elsheikh, Software development projects: An investigation into the factors that affect software project success/failure in Jordanian firms, First International Conference on the Applications of Digital Information and Web Technologies, ICADIWT, 2008, pp. 246-251.

[45] K. El Emam and A. G. Koru, A replicated survey of IT software project failures, Software, IEEE 25 (2008) 84-90.

[46] E. Egorova, M. Torchiano and M. Morisio, Actual vs. perceived effect of software engineering practices in the Italian industry, JSS 83 (2010) 1907-1916.

[47] R. C. Mahaney and A. L. Lederer, The effect of intrinsic and extrinsic rewards for developers on information systems project success, Proj. Manage. J. 37 (2006) 42.

[48] J. Drew Procaccino, J. M. Verner, S. P. Overmyer and M. E. Darter, Case study: factors for early prediction of software development success, Information and Software Technology 44 (2002) 53-62.

[49] N. Cerpa, M. Bardeen, B. Kitchenham and J. Verner, Evaluating logistic regression models to estimate software project outcomes, Information and Software Technology 52 (2010) 934-944.

[50] C. Jones, Software Tracking: The Last Defense against Failure, Crosstalk–Journal of Defense Software Engineering 21 (2008).

[51] R. Kaur and J. Sengupta, Software Process Models and Analysis on Failure of Software Development Projects, International Journal of Scientific & Engineering Research 2 (2011) 2-3.

[52] M. P. Álvarez, The four causes of behavior: Aristotle and Skinner, International Journal of Psychology and Psychological Therapy 9 (2009) 45-57.

[53] D. Hume, A Treatise of Human Nature [1739]. Oxford: Clarendon Press, 1896.

[54] J. Pearl, Ed., Causality: Models Reasoning, and Inference. United States of America: Cambridge University Press, 2000.

[55] C. W. Granger, Some recent development in a concept of causality, J. Econ. 39 (1988) 199-211.

[56] D. Galles and J. Pearl, Axioms of causal relevance, Artificial Intelligence 97 (1997) 9-43.

[57] C. Eden, Analyzing cognitive maps to help structure issues or problems, European Journal of Operational Research (2004) 673-686.

[58] P. Monteiro, R. J. Machado, R. Kazman and C. Henriques, Dependency analysis between CMMI process areas, PROFES, LNCS 6156, 2010, pp. 263-275.

[59] Y. Wang and G. King, Software Engineering Processes: Principles and Applications. CRC Press LLC, 2000.

[60] K. Lyytinen and D. Robey, Learning failure in information systems development. Info Systems (1999) pp. 85-101.

[61] D. L. Cooke, Learning from incidents, Proceedings of the 21st International Conference of the System Dynamics Society, New York, NY, USA, 2003, pp. 1-30.

[62] T. Dingsøyr, Postmortem reviews: purpose and approaches in software engineering, Information and Software Technology 47 (2005) 293-303.

[63] B. Collier, T. DeMarco and P. Fearey, A defined process for project post mortem review, Software, IEEE 13 (1996) 65-72.

[64] D. N. Card, Defect-causal analysis drives down error rates, Quality Time 10 (4) (1993) 98 - 99.

[65] A. Gupta, J. Li, R. Conradi, H. Rönneberg and E. Landre, A case study comparing defect profiles of a reused framework and of applications reusing it, Empirical Software Engineering 14 (2) (2008) 227 - 255.

[66] R. K. Yin, Ed., Case Study Research: Design and Methods. United States of America: SAGE Publications, 1994.

[67] P. Runeson and M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering (14) (2008) 131-164.

[68] T. D. Jick, Mixing qualitative and quantitative methods: Triangulation in action, Adm. Sci. Q. 24 (1979) 602-611.

[69] W. Foddy, Ed., Constructing Questions for Interviews and Questionnaires. Hong Kong by Colorcraft: Cambridge University Press, 1994.

[70] S. Salinger, L. Plonka and L. Prechelt, A coding scheme development methodology using grounded theory for qualitative analysis of pair programming, 19th Annual Psychology of Programming Workshop, Joensuu, 2007, pp. 144-157.

[71] I. Jacobson, G. Booch and J. Rumbaugh, The Unified Software Development Process. Addison-Wesley, 1998.

[72] W. Royce, Managing the development of large software systems, Proceedings of IEEE WESCON 26 (August), 1970, pp. 328-338.

[73] CMMI Product Team, "Capability Maturity Model Integration, version 1.2, CMMI for Development" CMU/SEI-2006-TR-008, ESC-TR-2006-008 (2006).

[74] P. Conroy, Technical Debt: Where Are the Shareholders' Interests? IEEE Software 29 (2012) 87-88.

[75] T. DeMarco, All late projects are the same, IEEE Software (2011) 103-104.

[76] F. Reyes, N. Cerpa, A. Candia-Véjar and M. Bardeen, The optimization of success probability for software projects using genetic algorithms, J. Syst. Software 84 (2011) 775-785.

[77] Y. Takagi, O. Mizuno and T. Kikuno, An empirical approach to characterizing risky software projects based on logistic regression analysis, Empirical Software Engineering 10 (2005) 495-515.

[78] H. Steck and V. Tresp, Bayesian belief networks for data mining, Proceedings of the 2nd Workshop on Data Mining Und Data Warehousing Als Grundlage Moderner Entschidungsunterstuezender Systeme, DWDW99, Sammelband, Universität Magdeburg, 1999.

[79] A. Alaeddini and I. Dogan, Using Bayesian networks for root cause analysis in statistical process control, Expert Systems with Applications (2011) 11230-11243.

[80] J. Cheng and R. Greiner, Comparing bayesian network classifiers, Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, 1999, pp. 101-108.

[81] A. Ayad, Critical Thinking and Business Process Improvement, Journal of Management Development 29 (2010) 556-564.

[82] K. El Emam and I. Wieczorek, The repeatability of code defect classifications, ISSRE '98 Proceedings of the the Ninth International Symposium on Software Reliability Engineering, 1998.

[83] M. V. Mäntylä and C. Lassenius, What Types of Defects Are Really Discovered in Code Reviews? IEEE Transactions on Software Engineering 35 (2009) 430-448.

[84] R. M. Daniel, M. G. Kenward, S. N. Cousens and B. L. De Stavola, Using causal diagrams to guide analysis in missing data problems, Pubmed 21 (2012) 243-256.