What Are Problem Causes of Software Projects?

Data of Root Cause Analysis at Four Software Companies

Timo O.A. Lehtinen
Department of computer science and engineering
Aalto University School of Science
Espoo, Finland
timo.o.lehtinen@aalto.fi

Mika V. Mäntylä
Department of computer science and engineering
Aalto University School of Science
Espoo, Finland
mika.mantyla@aalto.fi

***Abstract*—Root cause analysis (RCA) is a structured investigation of a problem to detect the causes that need to be prevented. We applied ARCA, an RCA method, to target problems of four medium-sized software companies and collected 648 causes of software engineering problems. Thereafter, we applied grounded theory to the causes to study their types and related process areas. We detected 14 types of causes in 6 process areas. Our results indicate that development work and software testing are the most common process areas, whereas lack of instructions and experiences, insufficient work practices, low quality task output, task difficulty, and challenging existing product are the most common types of the causes. As the types of causes are evenly distributed between the cases, we hypothesize that the distributions could be generalizable. Finally, we found that only 2.5% of the causes are related to software development tools that are widely investigated in software engineering research.**

Key words: Root Cause Analysis, Problem Prevention, Software Process Improvement, Grounded Theory

I. INTRODUCTION

The discipline of software engineering was born in 1968 due to problems in software projects [1]. The key for effective problem prevention is to know why the problem occurs [2]. Problems and challenges of software engineering have been introduced, e.g., Demir [3] indicates that scope management, requirements management, estimation, and communication are usual areas of challenges. Unfortunately, the causes of these challenges have not been comprehensively presented. There is only little value to know the problems in contrast to the value of understanding what causes them.

Root cause analysis (RCA) is a structured investigation of a problem to detect the causes that need to be prevented [4]. RCA takes the problem as an input and provides a set of problem causes as an output. It states what the problem causes are, in addition, where they occur. This helps with software process improvement in various contexts and across all software organizations, including product development, hardware design, product engineering, and manufacturing [5].

In some prior RCA studies, the causes of defects have been presented. Card [6] indicates that the defect causes are related to the methods, people, input, and tools, but his classification is quite coarse-grained and is lacking the software process dimension completely. Grady [7] states that the top eight causes of defects are specifications, user interface, error checking, hardware interface, software interface, logic, data handling, and standards. Grady's classification on the other hand sees causes from a technical perspective but does not go beyond that, e.g., "this line of code has the error" vs. "why does this line of code have error whose symptoms are visible in the released product". In this work, we want to understand problem causes from wider than just the technical perspective that Grady provides, furthermore, we want to provide more details than Card provides and we want to map the problem causes to the process dimension. A final difference to prior work is that the high number of particular types of software defects is not the only target problem that should be analyzed, e.g., negative project experiences, delayed product releases, and challenging product installations are all industrially relevant and severe problems but have only been exiguously explored using RCA [4].

In our previous paper [4], we presented the development and evaluation of ARCA, an RCA method, in terms of effort, usefulness and ease of use. The ARCA method consists of four steps, i.e., target problem detection, root cause detection, corrective action innovation, and documentation of the results [4]. In this paper, we introduce a detailed classification system for the detected causes of the ARCA method, that we developed by using the grounded theory approach introduced in [8]. Our classification system is based on a literature review and causes of four industrial RCA investigations focusing on complex software engineering problems. The classification system is thereafter used to show what types of causes were detected and where in the development processes they occurred. We discuss the similarities and dissimilarities of the causes and show what types of causes were common between the cases and what were not. This paper makes two important contributions as it introduces the output of the ARCA method and

simultaneously creates hypotheses on the challenges of software engineering for future research.

The research goal is as follows: *study the causes of complex software engineering problems by applying grounded theory to the target problem causes detected in four medium-sized software companies*. The research aimed to answer the following questions: RQ1: *What types of causes are related to the target problems of cases?* In the context of this study, it describes what the causes are, e.g., wrong working methods, lack of instructions, and challenging existing product. RQ2: *In which process areas the causes of the cases can be mapped?* Every cause occurs somewhere. In the context of this study, this means the development processes wherein the causes occurred. The causes are not isolated, instead, they are divided between the software processes, e.g., the causes of a late product release are not occurring in the development work only but also in the requirements engineering and software testing.

II. METHODOLOGY

The cause data was collected in industrial field studies [9] by using the ARCA method [4] in four medium-sized software product companies (100 to 450 employees) located in Finland. The target problem of the ARCA method was defined in a focus group meeting by the key representatives of the case company, who also selected the case participants. The common high-level goal of the companies was to understand why their software projects are delayed and how to avoid that. As recommended in the ARCA method, the target problem causes were detected through an anonymous e-mail inquiry followed by a causal analysis workshop, which is a meeting where the case participants write down the target problem causes and present them to others. The causes were organized to a cause-effect diagram [4].

The cause data was analyzed by creating two classification schemes to classify the causes. The development of the schemes was done in iterations. We started by a literature review on software engineering root cause analysis to conclude what kind of cause classification schemes have been previously introduced [6, 7, 10]. Thereafter, we created a preliminary classification scheme for both the types and related process areas of causes. Third, we combined the preliminary classification schemes to the grounded theory approach [8] and classified samples of the causes of the cases. During this step, we modified the preliminary classification schemes to create finalized classification schemes that actually correspond the causes of the cases. Finally, we applied the finalized classification schemes to all causes of the cases.

III. RESULTS

A. Classification Schemes

This section introduces the classification schemes that were developed in this study. The first scheme describes *what types of causes were detected* and the second scheme describes *what software engineering process areas were affected*.

There are three important terms used in this study. The *process area* describes where the cause occurs, e.g. "requirements engineering" or "software testing". The *type* describes what the cause actually is, e.g., "lack of instructions and experiences" or "lack of monitoring". The *class* means a set of similar types of causes, i.e. "people", "tasks", "methods", and "environment". It describes on general level what types of causes were detected in the cases and makes it possible for us to compare our results to the results of the prior studies of RCA [6, 10].

TABLE I. introduces the classification scheme used to describe the process areas of the causes. These process areas are similar to the ones found in software engineering process literature. The list of process areas was created based on our initial understanding of common software process steps, and it was refined by the data analysis. If we compare the process areas to commonly recognized software processes such as RUP [11] or the waterfall model [12] we can see several similar steps such as requirements engineering, testing, change management, product release and deployment. However, there are also some differences. First, we have merged software implementation and software design under a process area called development work. It would not have been feasible to separate whether technical problems of the product were due to poor design or implementation, because our cause data did not support such a division. Another difference is that we have a process area called management that gathers causes such as insufficient resource allocation, bad estimates, poor prioritization decisions, and bad organizational culture. Such issues undoubtedly are causes for problems in software projects, but they cannot be placed under other process areas. Thus, the management process area is needed to enable descriptive and honest presentation of the causes. The final difference to commonly recognized process areas is the Unknown process area, which includes causes that cannot be classified into any other process area, e.g., "laziness".

TABLE I. THE PROCESS AREAS OF CAUSES

Process area	Description
Requirements engineering, Re	Causes are focused on the requirements engineering and input from customers.
Management, Ma	Causes are focused on the company support and the way the project stakeholders are managed and allocated to tasks.
Development Work, Dw	Causes are focused on implementation of features and its output.
Software Testing, St	Causes are focused on software testing and its output.
Change Management, Cm	Causes are focused on implementation of change requests.
Product Release and Deployment, Pd	Causes are focused on installing and releasing the product.
Unknown, Un	Causes that cannot be focused on any specific process area.

TABLE II. presents the classification scheme used to describe the types of causes, which are additionally organized under four classes: people, tasks, methods, and environment. In prior works, Jalote [10] and Card [6] present a similar coarse-grained classification of cause classes. Unfortunately, these classification schemes are too general as they do not go under the classes. We wanted to extend these classification schemes to provide more details of the problem causes. Thus, we added the type level. For the type level there was no prior work, thus, it is completely based on our analysis of the problem causes.

TABLE II. THE CAUSE CLASSES AND RELATED TYPES OF CAUSES

Class / Type	**Description**
People, P	This class includes the people related causes
Instructions and Experiences	This type includes causes of missing documentation and lack of experience. The needed documentation is missing or inaccurate, and the lack of experience complicates the work.
Values and responsibilities	This type includes causes of bad attitude and lack of taking individual responsibility. The people do not care about important things and they look out for number one.
Co-operation	This type includes causes of inactive, inaccurate, and missing communication between the stakeholders. The people do not communicate actively or share knowledge on their own will.
Policies	This type includes causes of not following the company policies.
Tasks, T	This class includes the task related causes
Task Priority	This type includes causes of task priority. The priority is missing, wrong, or too low.
Task Output	This type includes causes of low quality task output. In our terminology the task is a general term which corresponds the tasks of all stakeholders, e.g. the managers may do inadequate resource allocation whereas the developers may do bad code, etc.
Task Difficulty	This type includes causes of challenging tasks. The task requires too much effort, time, or it is too difficult.
Methods, M	This class includes the methodological causes
Work Practices	This type includes causes of lack of current working methods. The method is missing or inadequate.
Process	This type includes causes that are focused on the current operations model. The model is unclear, vague, too heavy, or inadequate.
Monitoring	This type includes causes of lack of monitoring. The management does not know the project status caused by the lack of monitoring the progress.
Environment, E	This class includes the environment related causes
Existing Product	This type includes causes of the existing product, which is too complex and the old low-quality code creates challenges.
Resources and Schedules	This type includes causes of wrong resources and schedules.
Tools	This type includes causes of missing or insufficient tools.
Customers	This type includes causes of customer requests and users expectations and needs.

The people class includes types of causes that correspond to human aspects. The tasks class includes types of causes that correspond to the causes that were closely related to implemented tasks. The methods class includes types of causes that correspond to the causes of wrong working methods. The environment class describes the type of causes that are related to external settings of the work. The detailed types of causes including their descriptions are placed under each class as can be seen in TABLE II. .

B. Cause Distributions

TABLE III. summarizes the types of target problem causes and shows how they divide into the software processes. We also report the totals of causes for each process area and type. Next to the totals is the standard deviation *between cases*. This is reported to help in analyzing the external validity of the cause distribution. High standard deviation indicates that the cause distribution is highly affected by the case context. Low standard deviation between cases suggest that the distribution could be generalizable, but with only four cases it is only possible to draw initial hypotheses. It should be noted that when looking at the standard deviation one should always contrast it with the total average.

The lack of instructions and experiences included the highest number of causes (18.1 %), which was mainly divided into the requirements engineering, development work, software testing, and product release and deployment. The wrong work practices included the second highest number of causes (15.7 %), which was mainly divided into the software testing, development work, management, and product release and deployment. Looking at the deviations we can see that shares of Instructions and Experiences could be generalizable (deviation 3.4 units from total share of 18.1%), but that shares of existing product do not seem generalizable (deviation 7.8 from the total share of 8.5%)

From the process perspective, the software testing (23.1 %) and development work (22.6 %) included the highest number of causes. The causes of software testing divided mainly into the wrong work practices, lack of instructions and experiences, insufficient task output, task difficulty, and wrong resources and schedules. The causes of development work were mainly similar to those in the software testing, but the existing product was more often referred (2.5 %) whereas the insufficient task output was less often referred (0.9 %).

The deviations between cases are higher in the process areas than it is in the types of causes. From the recognized process areas, only the development work process area has a low standard deviation (7.9) in comparison to the total share of causes (22.6%). Thus, we can hypothesize that shared causes per process area are more dependent on the case context than the type of causes, which seem more general.

C. Limitations

As the total number of cases was only four, the results need to be validated by further studies. However, in contrast to

TABLE III. PERCENTAGES OF THE TYPE OF CAUSES IN SOFTWARE PROCESS AREAS (A TOTAL OF 648 CAUSES)

Cause type, Class	Process Area								
	Re	***Ma***	***Dw***	***St***	***Cm***	***Pd***	***Un***	***Total***	***Std****
Inst. and Exp., P	3.4	2.3	**5.1**	2.9	0.3	2.6	1.4	**18.1**	3.4
Work Pr., M	0.8	2.3	4.3	**5.6**	0.9	1.5	0.3	15.7	5.2
Task Output, T	3.2	**3.5**	0.9	2.9	0.2	0.6	0.3	11.7	4.8
Task Difficulty, T	0.6	0.3	**3.1**	2.9	0.5	2.0	0.2	9.6	4.5
Existing Pr., E	0.3	0.3	**2.5**	0.9	0.3	2.2	2.0	8.5	7.8
Res. and Sch., P	0.6	1.1	1.9	**2.5**	0.2	0.5	0.8	7.4	3.5
Val. and Resp., P	0.9	**1.5**	**1.5**	1.4	0.5	0.6	0.5	6.9	5.2
Process, M	0.0	0.2	0.6	1.2	1.1	**1.4**	1.1	5.6	0.7
Policies, P	0.5	0.2	0.5	**0.8**	0.3	**0.8**	0.5	3.4	1.6
Co-operation, P	**1.4**	0.9	0.8	0.0	0.2	0.0	0.0	3.2	2.9
Customers, E	**1.5**	0.0	0.3	0.2	0.0	0.5	0.8	3.2	1.6
Tools, E	0.2	0.3	0.2	**1.5**	0.2	0.2	0.0	2.5	0.6
Task Priority, T	0.0	0.2	0.8	0.2	**0.9**	0.0	0.3	2.3	2.6
Monitoring, M	0.0	**0.8**	0.2	0.2	0.3	0.2	0.2	1.7	2.5
Total	13.4	13.9	22.6	**23.1**	5.7	13.0	8.2	100	
Std*	12.3	10.9	7.9	15.2	6.1	17.5	1.8		

Std* = deviation of % units between the cases

prior studies [6, 7, 10] our results are based on more than one case and thus are more externally valid than they are. Effect of the case context, both the company context and the chosen RCA focus is likely to be high. The deviation between the cases varied between process areas and types.
The classification scheme was jointly developed and partly based on the existing literature. The classification of the causes was done only by the first author, which increases the possibility of the researcher bias. We plan to address this in our future work on this topic.

IV. CONCLUSIONS

In this paper we have created a two-dimensional classification of software problem causes based on four industrial RCA field studies resulting in 648 causes. The first dimension of the classification is based on common software engineering process areas. The second dimension describes the type of causes and it extends prior works of software engineering root cause analysis [6, 7, 10] by giving more detailed types under the general classes of people, tasks, methods, and environment. Our classification is useful for understanding problem causes as it highlights both the process areas where improvements should be made and also the types of improvements that need to be made, e.g. do we have a problem with tools or work practices.

We have also presented a distribution of causes with our two-dimensional classification system. In it, we found that instructions and experiences was the most common cause type followed by insufficient work practices. It is interesting to note that tools were mentioned in only 2.5% of the causes, although a great deal of software engineering research is focused on building new tools. In the software process dimension the process areas with most causes were development work and software testing. However, the deviation between the cases was higher in the process area dimension. Therefore, we believe that case context and focus has a larger effect on the process area of the causes compared to the types of causes.

V. REFERENCES

[1] P. Naur and B. Randel, Software engineering: A report on a conference sponsored by the NATO science committee, Nato, 1969.
[2] J. J. Rooney and L. N. Vanden Heuvel, Root cause analysis for beginners, Quality Progress 37 (7) (2004) 45 - 53.
[3] 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.
[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] R. G. Mays, Applications of Defect Prevention in Software Development, IEEE Journal on Selected Areas in Communications 8 (1990) 164-168.
[6] D. N. Card, Learning from our mistakes with defect causal analysis, IEEE Software 15 (1) (1998) 56-63.
[7] R. B. Grady, Software failure analysis for high-return process improvement decisions, Hewlett-Packard Journal 47 (4) (1996) 15 - 25.
[8] 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.
[9] T. C. Lethbridge, S. Elliott Sim and J. Singer, Studying software engineers: Data collection techniques for software field studies, Empirical Software Engineering 10 (3) (2005) 311-341.
[10] 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.
[11] I. Jacobson, G. Booch and J. Rumbaugh, The Unified Software Development Process. Addison-Wesley, 1998.
[12] W. W. Royce, Managing the development of large software systems: Concepts and techniques, Proceedings of Wescon, 1970, pp. 1-9.