Assembling Multiple-Case Studies: Potential, Principles and Practical Considerations

Aiko Yamashita
Mesan AS & Simula Research Laboratory
Oslo, Norway
aiko@simula.no

Leon Moonen
Certus Centre, Simula Research Laboratory
Oslo, Norway
leon.moonen@computer.org

ABSTRACT

Case studies are a research method aimed at holistically analyzing a phenomenon in its context. Despite the fact that they cannot be used to answer the same precise research questions as, e.g., can be addressed by controlled experiments, case studies can cope much better with situations having several variables of interest, multiple sources of evidence, or rich contexts that cannot be controlled or isolated. As such, case studies are a promising instrument to study the complex phenomena at play in Software Engineering.

However, the use of case studies as research methodology entails certain challenges. We argue that one of the biggest challenges is the case *selection bias* when conducting multiple-case studies. In practice, cases are frequently selected based on their availability, without appropriate control over moderator factors. This hinders the level of comparability across cases, leading to internal validity issues.

In this paper, we discuss the notion of *assembling cases* as a plausible alternative to *selecting cases* to overcome the selection bias problem when conducting multiple-case studies. In addition, we present and discuss our experiences from applying this approach in a study designed to investigate the impact of software design on maintainability.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

General Terms

theory, design, experimentation.

Keywords

empirical studies; methodology; case study; internal validity

1. INTRODUCTION

A case study is an empirical inquiry or research strategy that "investigates a contemporary instance or phenomenon within its real-life context, particularly when boundaries between instance or phenomenon and context are not clear" [24]. In Software Engineering (SE) research, the popularity of case studies is growing, although its practice is not as mature as other disciplines (e.g., Social research and Information Systems research). Runeson et al. argue for the adequacy of case study methodology for many types of SE research because the objects of study are indeed contemporary phenomena that are hard to study in isolation [16].

Case study research can be based on both quantitative and qualitative evidence and it encompasses a wide set of systematic techniques (i.e., data collection, analysis, and reporting of the results) [24]. Case studies can provide a deeper insight into key aspects that can be investigated to develop or confirm theories which *explain* an observed phenomenon [9].

However, despite the versatility and advantages of this methodology, its usage also entails some challenges. We believe that one important challenge is due to *selection bias* [11] during the selection of cases. In multiple-case studies, the units of analysis normally need to be selected to have a variation in the properties that the study intends to compare. In practice, cases are often selected based on their availability, without exercising appropriate control over moderator factors (often of contextual nature). This, in turn, can affect the internal validity of the findings, e.g., because the presence of dissimilar contextual factors make it more difficult to validate results across the multiple cases.

In this paper, we discuss the notion of *assembling cases* as a plausible alternative to *selecting cases*, to achieve better control over contextual factors across cases and tackle the selection bias challenge. We believe that constructing real-life situations (cases), with some level of control over certain variables and/or contextual factors is a feasible and purposeful approach to conduct empirical studies in SE. This approach enables the study of phenomena that is closer to real-life than what is normally attainable in experimental settings (thus, richer in details), and at the same time it adds elements of control that can facilitate comparison across cases, tackling the problem of case selection bias and reducing threats to internal validity. Furthermore, we illustrate how this notion was applied in a multiple-case study, where we attempted to control for moderator factors and conducted *case replication* [24]. We report lessons and practical considerations stemming from our experiences with this case study, and discuss the implications, limitations, and adequacy of this approach for SE research.

The remainder of this paper is structured as follows: Section 2 briefly discusses the advantages and limitations of case study research inspired on the discussion by George and Bennett [11]. Section 3 and introduces the notion of *assembling* case studies. Section 4 illustrates this approach by de-

EASE '14, May 13 - 14 2014, London, England, BC, United Kingdom
Copyright 2014 ACM 978-1-4503-2476-2/14/05 ...$15.00.
http://dx.doi.org/10.1145/2601248.2601286

scribing the design of a multiple-case study conducted in order to investigate the effects of software design on maintainability. Section 5 describes our experiences from conducting this study and discusses feasibility, practical considerations and adequacy of this approach. Section 6 summarizes the work and presents concluding remarks.

2. OPPORTUNITIES AND CHALLENGES

2.1 Opportunities

George and Bennett [11] argue that, for certain questions, case studies have advantages over controlled experiments: (a) potential for high construct validity, (b) strong procedures for fostering new hypotheses, (c) reasoning about causal mechanisms in individual cases, and (d) capacity to address causal complexity.

Case studies involve detailed consideration of contextual factors leading to higher construct validity. They can also deal with complex causal relations and complex interaction effects [15]. This is beneficial in software engineering research where constructs (e.g., maintainability or comprehensibility) are often difficult to operationalize, and the usefulness of a given operationalization can considerably depend on the context of the study. E.g., the operationalization of "readability," may be appropriate in one context but not in another. The richer context also enables a more detailed investigation of causal mechanisms. For example, inclusion of a large(r) number of variables may lead to observing unanticipated aspects of a *causal mechanism*, or the identification of conditions that activate a given causal mechanism.

Finally, case studies also help to *identify new variables* and *develop new hypotheses* by studying deviant or outlier cases via techniques such as interviews, archival research, and observational protocols. This allows for the identification of alternative theories based on findings from the field, when outcomes are not explainable by initial theories.

2.2 Challenges

Known challenges of case study research include: (a) selection of cases and bias due to case availability, (b) determining relative causal weights for variables [11].

To analyze the relation between two or more variables, one should minimize (ideally freeze) variability in other variables that may affect the investigated relation. In controlled experiments, this is approximated by dividing subjects in experimental and control groups through some form of random assignment. Traditionally, *control* in multiple-case study design is achieved through *case selection*, where the case and units of analysis are selected to be as similar as possible, but vary in the properties that the study intends to compare.

Selecting similar cases that differ only in the variables of interest is very important, otherwise it would not be possible to know if the variation seen in the cases is due to the particular variable(s) under consideration, or due to other differences between the cases [14]. In practice, however, cases are often selected because of their availability to the researchers [4]. Many SE studies examine multiple cases involving systems with different functionality and contexts, which hinders the comparability of cases [7]. One of the main underlying reasons is that it is difficult to identify, and get access to, multiple cases with the necessary characteristics.

A similar challenge applies to determining the relative causal weighs of variables. Case studies are often better at assessing *whether* and *how* a variable mattered to the outcome rather than *how much* it mattered. For example, the equivalent of beta coefficients in statistical studies is only possible in case studies when there is enough control so that extremely similar cases differ only in one independent variable. As mentioned, the level of control that is needed for this can seldom be achieved in practice.

3. CONTROL VIA ORCHESTRATION

An approach to address the above-mentioned challenges is to *orchestrate* a real-life situation where *moderator* (contextual) factors across multiple cases are made as similar as possible and the variable of interest is made dissimilar. This approach differs from the typical multiple-case studies, because the cases are *assembled* rather than *selected.*

The use of 'control' in case studies within software engineering was first reported by Salo and Abrahamson [17]. It also applied by Anda et al., who investigated the degree of replicability of software projects by contracting four Norwegian software companies to each develop a system based on the same requirements specification [2].

Note that *hiring* or *contracting* software engineering companies is one way to assemble cases, but it does not constitute the only means. Other approaches include the use of *crowd-sourcing* [1], and designing cases under the schema of *Action Research* [3]. Regardless of the means to assemble cases, the two most important aspects to observe are: (1) the degree of control that can be achieved throughout the study design, and (2) the presence of in-depth inquiry methods via the data collection and analysis techniques offered by the case study repertoire.

Whenever cases are constructed or assembled, it is natural to expect that there is a trade-off between the degree of *realism* and the degree of *control* in such studies. We argue that an appropriate study design can ensure the degree of realism and representativeness of the cases in the study while adding the required control.

The running example that is used in this paper consists of an *assembled* multiple case study in software maintenance in which the same maintenance tasks were performed on functionally equivalent systems by six professional software developers with similar skills [21]. Because we were in charge of defining the maintenance project, it was easier to control for different moderator factors such as tasks, technology, etc. (described in more detail in the next section).

The example study was done as a follow-up on the study by Anda et al., [2], mentioned above. After Anda's study, Simula possessed four functionally equivalent systems with significant differences in their design and implementations. These systems were designed and implemented to interact with Simula's website, which was being revised at the time the example study was set up. Roughly at the same time, another colleague finished a comprehensive study on developer skills for which he contacted a large group of professional developers [5]. As a result, several conditions were satisfied that enabled us to systematically set the parameters for the context of a multiple case study:

- a number of systems with identical functionality and different implementations,
- a concrete need (and budget) for a realistic software maintenance effort, with well-defined tasks,
- a pool of developers with similar skills selected from our colleague's skill study participants.

Table 1: Size and cost for all four systems.

System	A	B	C	D
Java	8,205	26,679	4,983	9,960
Jsp	2,527	2,018	4,591	1,572
Others	371	1,183	1,241	1,018
Total	11,103	29,880	10,815	12,550
Costs	€25,370	€51,860	€18,020	€61,070

This starting point allowed for the definition of a study that assembled a number of *very similar cases* (same tasks, same technology, similar developers), which displayed a *variation* in one factor (software design and implementation). Because the systems had different design and implementation, the effect of software characteristics (metrics, presence of code smells, anti-patterns) could be investigated as the variable(s) of interest or independent variable(s). Moreover, the fact that the systems displayed (nearly) the same functionality allowed control for the *maintenance task*: identical tasks could be performed on all four systems, which constitutes a typical moderator factor of studies on software maintainability. Access to the pool of developers that participated in the study on programming skills allowed us to select and invite developers that displayed most similar skill levels, controlling for yet another moderator factor. Designing, conducting and observing a multiple-case study on these premises enabled us to collect a level of detail and richness of data on maintenance processes that could not have been achieved in a controlled experiment.

4. THE MAINTENANCE PROJECT

4.1 Systems Under Study

Simula's Software Engineering department sent out a tender in 2003 for the development of a new web-based information system to keep track of their empirical studies. Based on the bids, four Norwegian consultancy companies were hired to independently develop a version of the system, all using the same requirements specification (more details on the initial project are described in Anda et al. [2]). In the remainder, these four systems are referred to as Systems A, B, C, and D. The systems were primarily developed in Java and had similar three-layered architectures but had considerable differences in their design and implementation. This is reflected in Table 1, which displays the physical lines of code (LOC) for all the different types of files in the system (Java, JSP, and other files, such as XML and HTML).

The cost of contracting development of these systems also differed notably (Table 1). The price may have been affected by business factors within the companies (e.g., different business strategies to profit from a project; some companies were willing to bid low to enter a new market) but may also reflect differences in skill and coding quality.

The main functionality of the systems consisted in keeping a record of the empirical studies and related information at Simula (e.g., the researcher responsible for the study, participants, data collected, and publications resulting from the study). Another key element of the functionality was to generate a graphical report on the number of different types of studies conducted per year. The systems were all deployed over Simula's Content Management System (CMS), which at that time was based on PHP and a relational database system. The systems had to connect to a database in the CMS to access data related to researchers at Simula as well as information on the publications therein. During the operational stage of all four systems, the defects and change requests were recorded.

4.2 Maintenance Goals

In 2008, Simula introduced a new CMS called *Plone*,[1] and due to the differences with the old CMS, it was no longer possible for the systems to remain operational. This provided the motive to conduct and investigate a maintenance project. To conduct this maintenance project, developers from two software companies were contracted at a total cost of 50,000 Euros. Table 2 shows the tasks implemented during the project. The first two tasks consisted of adapting the system to the new platform, and the third task consisted of adding a new functionality. It was assumed that the adaptive and perfective tasks would introduce new defects and hence would include work on corrective tasks (i.e., the study also indirectly covers corrective tasks).

4.3 Developers and Location of the Study

The tasks from Table 2 were conducted individually by six developers from two software companies, one located in the Czech Republic and another located in Poland (three developers from each company). The developers were recruited from a pool of 65 participants in a previous study on programming skills [5], which also included maintenance tasks. All the selected developers had been evaluated to have a good and, perhaps more important for the study design, a similar level of programming skill. The skill scores of the instrument were derived from the principles in [5], where the performance on each task was scored as a structured aggregate of the quality (or correctness) and time for a correct solution for each task. Developers were also selected based on their availability, English proficiency, and motivation for participating in the study. The project was conducted between September 2008 and December 2008 at the companies' sites (four weeks in the Czech Republic and three weeks in Poland).

4.4 Activities, Environments, and Tools

Initially, the developers were given an overview of the project (e.g., the maintenance project goals and project activities). They also completed a questionnaire and a set of programming exercises to familiarize themselves with the domain of the systems. A specification was given to the developers for each maintenance task, and when needed, they discussed it with the researcher that was present at the site. An acceptance test was conducted once all the tasks were completed for one system. The defects identified during acceptance testing were recorded in an issue tracking system and in the acceptance test reports. In addition to the acceptance test, an open interview was conducted (individually) where the developer was asked about his/her opinion on the system(s) on which he/she had worked so far. Eclipse was used as a development tool along with MySQL[2] and Apache Tomcat.[3] Defects were registered in Trac,[4] and Subversion[5] was used as the versioning system.

[1] http://plone.org
[2] http://www.genuitec.com
[3] http://tomcat.apache.org
[4] http://trac.edgewall.org
[5] http://subversion.apache.org

Table 2: Maintenance tasks carried out during the study.

No.	Task	Description
1	Adapting the system to the new Simula CMS	The systems in the past had to retrieve information through a direct connection to a relational database within Simula's domain (information on employees at Simula and publications). Now Simula uses a CMS based on the Plone platform, which uses an OO database. In addition, the Simula CMS database previously had unique identifiers based on Integer type, for employees and publications; a String type is used now. Task 1 consisted of modifying the data retrieval procedure by consuming a set of web services provided by the new Simula CMS in order to access data associated with employees and publications.
2	Authentication through web-services	Under the previous CMS, authentication was done through a connection to a remote database using authentication mechanisms available at that time for the Simula web site. Task 2 consisted of replacing the existing authentication by calling a web service provided for this purpose.
3	Add new reporting functionality	Provide options for configuring personalized reports, where the user can select the type of information related to a study to be included in the report, define inclusion criteria according to researchers who were in charge of the study, sort resulting studies according to the date that they were finalized, and group results according to the type of study. The configuration should be stored in the system's database and only be editable by the owner of the report configuration.

5. DESIGN OF THE STUDY

5.1 Control over Moderator Factors

The functional similarity of the systems allowed the investigation of cases with very similar contexts (e.g., identical tasks and programming language and similar development environments), and the differences in the systems' code design allowed us to investigate the effect of *code smells* (indicators of potentially problematic code [10]) on *software maintenance*. The study design enabled an in-depth investigation of a realistic industrial software maintenance context. An overview of the current literature in code smells indicates that most empirical studies in this area consist predominantly of correlation studies conducted on post-development projects on OSS or controlled experiments [23]. As discussed previously in Section 2.1, correlation studies typically do not allow for an in-depth understanding of causal effects due to limited contextual information on the projects, as opposed to case studies. In the context of this study, having four functionally equivalent systems with different code enabled the design of a multiple-case study, with software maintenance tasks embedded within almost identical *maintenance contexts* and differing in the variable of interest: *code smells*. The design of the study therefore, enables better control over the moderator variables, such as system functionality, tasks, programming skills, and development technology (see Figure 1), to better observe the relations between *code smells* and different maintenance aspects.

5.2 Case Replication

In multiple-case designs, it is possible to perform two types of replication: theoretical replication and literal replication. According to Yin [24], in *literal replication*, cases that are similar in relation to certain variable(s) are expected to support the analysis of each and give similar results. When *theoretical replication* is used, the cases that vary on the key variable(s) are expected to have different results.

Figures 2 and 3 illustrate the use of case replication for instances with similar and dissimilar presence of code smells. The developers in the study were required to perform the maintenance tasks in more than one system, so that there would be enough cases to perform both types of replications. Note that if only theoretical replication was required, it would have been sufficient to have four different maintenance projects, each involving one system, and then to compare the results across the four systems.

As the use of both theoretical and literal replications can support each other's analyses, each of the six developers was asked to first conduct all tasks in one system (in the order that they were presented in Table 2) and then to repeat the same maintenance tasks on a second system, resulting in 12 observations (six developers $\times$ two systems). Thus, we make a distinction between first-round cases and second-round cases. "First round" denotes a case in which a developer has not maintained any of the systems previously, and "second round" denotes a case in which developers repeat the tasks on a second system.

Figure 4 describes the order in which the systems were assigned to each developer. This assignment was done randomly. The order (and thus combination) of systems maintained by each developer was decided so that all the four systems should be equally represented within the cases and so that all the four systems should be maintained at least once in the first round and once in the second round (i.e., to support the adjustment for potential learning effects from the first to the second round).

5.3 Variables of Interest

The variables of interest within this study are as follows:

1. *Code smells*: *Number of code smells* and *code smell density* (code smells/kLOC) are used as measures for this variable.
2. *Developers' perception of the maintainability of the systems*: This includes subjective and qualitative aspects of maintainability to be reported by the developer once the three maintenance tasks of one system had been

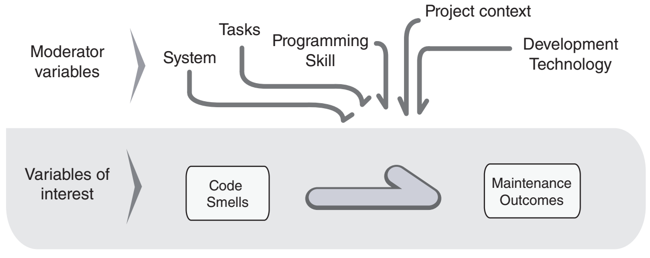

Figure 1: High-level design of the case study.

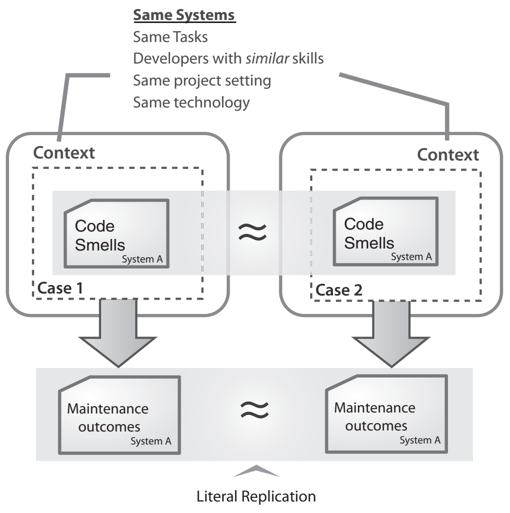

Figure 2: Literal replication in example study

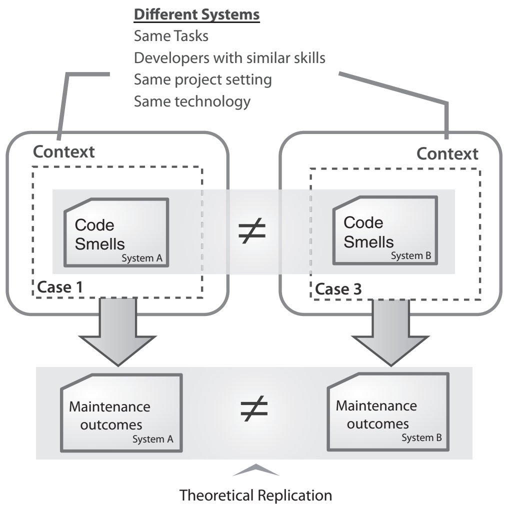

Figure 3: Theoretical replication in example study

completed.

3. *Maintenance problems encountered by the developers during maintenance*: These include a qualitative aspect of the maintenance process based on problems reported through interviews or think-aloud sessions or observed by the researcher during the maintenance work.
4. *Change size (measured by churn)*: This variable constitutes an outcome variable from the maintenance project, reflecting the sum of LOC added, changed, and deleted.
5. *Effort (measured in time spent on the tasks)*: This variable is an aspect of the maintenance outcome.
6. *Defects introduced during maintenance*: This variable is an aspect of the quality of the system after the tasks were completed. As such, it is also an aspect of the maintenance outcome.

Figure 5 describes the moderator variables (those we control in the analysis), the variables of interest (those whose relationships we analyze), and the data sources for the variables. The figure discriminates between outcomes and aspects that were observed at the system level (marked with one asterisk) and at both system and file levels (marked with two asterisks). The figure also distinguishes *qualitative aspects* such as maintenance problems and maintainability perception (which are drawn as circles) from *quantitative outcomes* such as change size, effort, and defects (which are drawn as squares). Finally, this figure shows from which of the data sources each of the maintenance outcomes/aspects was derived. We conducted a comprehensive collection of both qualitative and quantitative data. Table 3 displays the complete list of data sources and their corresponding data collection activities.

5.4 Types of Analysis Conducted

The design of the study allowed to investigate the effects of code smells on system-level maintainability, by comparing the total amount of code smells across each of the systems, and comparing the total effort, defects and change size in each of the cases involving the different systems. This approach is also known as *Pattern Matching* [20], where theoretical assumptions (e.g., systems with more code smells are more difficult to maintain) are verified through measurable outcomes (e.g., maintenance outcomes such as effort, defects).

The design of the study and the diversity of the data collected allowed to perform different types of analysis, whose results can be verified by each other. Thus, for investigating one particular research question, one could use different data sources *and* different data analysis techniques to cross-validate the findings. This is part of what is known as *mixed approach* [6], where quantitative analysis can be followed by qualitative analysis or vice versa. One example of data triangulation is reported in [18], where a multiple regression analysis was conducted at file level, using number and type of code smells contained in a file, the file size and the file change size, to explain the effort (time) spent maintaining (reading and modifying) a Java file. In this analysis, we use different data sources such as the source code, the SVN, and the Mimec Logs (sources 1, 4 and 7 in Table 3).

Another example of mixed methods approach is reported in [22], where *quantitative* (i.e., principal component analysis) is followed by *qualitative* analysis (i.e., explanation building) to investigate if the presence of code smells could explain maintenance challenges for Java files. The fact that we can conduct *case replication* (as described in subsection 5.2) is very important when using qualitative techniques such as *explanation building* because plausible explanations can be discarded or verified via comparing cases with specific similarities or dissimilarities in the variables involved in the explanation [24].

Given that we counted with a rich set of qualitative data (e.g., sources 2 and 3 in Table 3) across multiple cases that

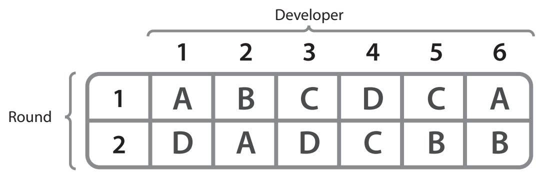
Developer
Round
1 2 3 4 5 6
1 A B C D C A
2 D A D C B B

Figure 4: Assignment of systems to developers

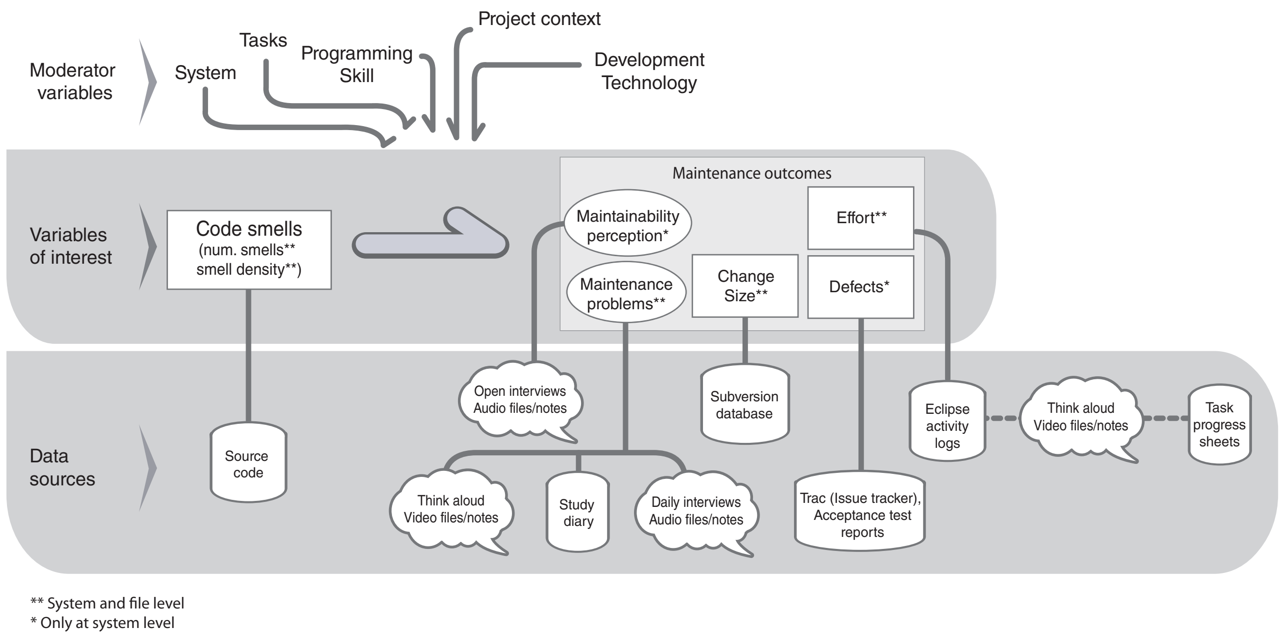

Figure 5: Illustration of variables involved in the study and the corresponding data sources.

Table 3: List of data sources in the study.

No.	Data sources	Data collection activity
1	Source code	The system's source code. (Note: The scope of the code smells is limited only to Java files, and consequently, jsp, sql files and other artefacts were not considered for measurement).
2	Open-ended interviews Audio files/notes	Individual *open-ended interviews* (40-60 minutes): were held after all three maintenance tasks were completed for each system. The developers were asked about their opinion of the system(s) on which they had worked so far (e.g., *how difficult was it to understand the systems?*).
3	Daily interviews Audio files/notes	Individual *progress meetings* (20-30 minutes): were conducted daily between all developers and the researcher present at the study to keep track of the progress, and register difficulties encountered during the project (ex. Dev 1: "*It took me 3 hours to understand this method...*")
4	Subversion database	The repository database where the source code was kept (Subversion was used).
5	*Trac* (Issue tracker)	The developers registered defect reports in an issue tracking system called *Trac*.
6	Acceptance test reports	Results from each of the acceptance test scenarios
7	Eclipse activity logs	Developer's activities were logged by a plug-in called Mimec [13]. This plug-in logged all the actions performed on Eclipse at the GUI level.
8	Think-aloud video files/notes	Video-recorded *think-aloud sessions* (ca. 30 minutes) were conducted every second day to observe the developers in their daily activities. If the developer felt uncomfortable "speaking-out", the session would be limited to record the screen and the responsible of the think-aloud session will take observational notes.
9	Task progress sheets	The developers filled in these sheets, where they compared estimations vs. actual time for each of the sub-tasks required for the maintenance.
10	Study diary	A logbook was kept by the researcher present at the study where the most important aspects of the study were annotated in a daily basis

share certain characteristics (e.g., different developers worked with the same system), results from *grounded theory* techniques [19] can be verified by comparing the most similar cases (e.g., those involving the same systems) through *cross-case comparison* [24].

We have used such a cross-case comparison to extract maintainability factors that were important from a developers' perspective from the open-ended interviews and investigated their relation to code smells was investigated [21]. The chain of evidence that was used to summarize and analyze the data in this study is shown in Figure 6. When conducting *cross-case comparison*, evidence strength is determined by how many cases support a statement. As such, we consider as indicator of relevance whether a maintainability factor was mentioned by many/all developers who worked with the same system, as opposed to factors only mentioned by one developer.

6. DISCUSSION AND LESSONS LEARNED

6.1 Achieving Realism

One of the concerns we had at the beginning of using this type of study design was the level of realism, as mentioned in section 3. Once the maintenance project was finalized in each of the companies, an open discussion was held where developers, researchers and the company representative had the opportunity to comment on the level of realism of the project.[6] The outcomes from these sessions showed rather different insights across the companies. While one of the companies described the project as slightly "unrealistic", the other company described the maintenance project as "normal practice". Upon further investigation, it became clear that this discrepancy could be explained mostly by differences in the normal *modus-operandi* of the companies.

Our study used a "solo-project" modality. This was seen as very unusual by the developers from one company, as they were used to a highly collaborative environment. Some developers at this company commented that this even affected their motivation during the project. This was not the case for the second company, where developers were used to working individually. We opted for "solo-projects" to reduce the complexity of the analysis, since collaborative work requires far more complex analysis than studies involving individuals. This relates to George's position, that there is a need to balance *theoretical parsimony* and *broad generalizations* when conducting case studies [11]. Although solo projects may not be the standard case, the study design still provides an in-depth, clear-cut view on how developers conduct maintenance in an *individual basis*, and we have seen that this working schema occurs in some companies (such as the second company involved).

Similarly, the fact that developers repeated the same maintenance tasks on different systems was perceived as unusual for one company, in contrast to the other company, who consider it acceptable. The developers who found it acceptable stated that the tasks they had to perform in this project were "standard tasks" and that they often faced similar challenges and tasks in other projects, so they did not consider the repetition to be *that* different from their normal practice. From our side, we found that the choice of having developers repeat tasks on several systems in the end represented some advantages and some disadvantages. On the one hand, it allowed us to compare one developer's opinion while they worked on the same tasks on different systems. On the other hand, it introduced an element of artificiality in the project, which potentially affected the perceived degree of realism for the participants.

[6] Note that the developers and their companies were aware from the start that this project was part of a scientific study.

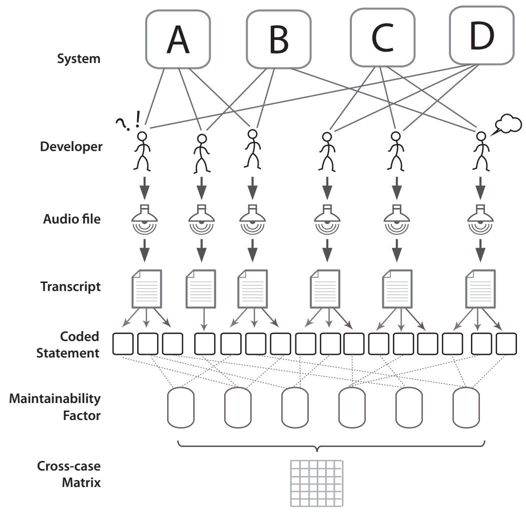

Figure 6: Chain of evidence for data summarization and analysis, from [21]

One way to address this issue would be to hire more developers and increase the number of cases involving different systems, given that there is enough budget (in terms of time and economical resources) to do so. However, this would not have allowed us to compare an individual's impression on the maintainability of several systems with same functionality and different design. Again, this is a matter of finding the right balance between the degree of control and realism, often the choice depends on the specific research questions and the topics to be investigated.

Other challenges for achieving high levels of realism were the level of granularity and techniques used for the data collection. From the open session, we found that for example the *Task progress sheets* (source 9 in Table 3) where found to be tedious by more than one developer, and *Think-aloud sessions* (source 8 in Table 3) where found to be rather intrusive. Half of the developers agreed that for the think-aloud sessions, they did not mind to be observed and have their screens recorded, as long as they were not required to verbalize their thoughts, contrary to the other half, who expressed having no discomfort on doing so. However, these are normal challenges for any type of case study, regardless of if they are multiple- or single-case designs, and whether the cases are *assembled* or *selected*.

Finally, the developers stated that they were not used to get into deep technical details and challenges during the *daily interviews* (source 3 in Table 3), as their usual customers seldom are interested in those details and show more interest on the actual functionality of the systems. However, we believe that daily interviews in these type of projects are necessary in order to achieve an adequate granularity level for investigating the delays and problems that occur in a maintenance project.

From all the data collection techniques used, we found that the least intrusive one was the use of an instrumented IDE (i.e., the Mimec plug-in for Eclipse) to record the developer's activities (source 7 in Table 3) and the inspection of the versioning and issue tracking systems (sources 4 and 5 in Table 3). The Mimec plug-in provided an unique opportunity to observe the maintenance at a high level of detail, which would have not been possible with the the progress sheets written by the developers. Based on our experiences from this study, we see great potential for the usage of a combination of: *screen recording* (e.g., via a screen-recording program installed in the developers' computer), *IDE instrumentation* (e.g., Mimec plug-in) and *direct observation*, for investigating *in-vivo* maintenance, since the first two methods are rich in details and do not constitute intrusive approaches, and the third method can complement/explain/index the data from the first two sources.

6.2 Outsourcing

We consider this aspect of special relevance when conducting these type of studies, given that one way to assemble a multiple-case study is by "hiring" developers or outsourcing software companies to conduct specific tasks. Although there are alternatives to outsourcing (e.g., crowd-sourcing, action research), this option could be seen as the most optimal one to achieve the required level of 'control' in the design of a study. In this project, we opted for hourly-rate contracts instead of a fixed-rate contracts, which led to delays in some of the projects. These delays constituted a major concern for the researchers, both due to time constrains and annual budget constrains on the research funds. Delays and dispersed working times were also caused by a lack of coordination between the researchers and the contractors, in relation to local public holidays.

Based on our experience, we conclude that fixed-rate contracts are desirable as the outsourced companies need to comply with the objectives and goals of the project, and also need to account internally for aspects such as public holidays. Another pitfall within our contracting process was an insufficient overview of the project goals for the developers and the contact persons at the company. Although this may sound as counter-intuitive (considering that it may increase the *hawthorne effect*), we conclude from this study that developers are more open to collaborate and more communicative if they are more involved with the goals of the project and the scientific work rather than being kept away from it. Of course, this implies a delicate balance between information that can be conveyed to the developers and information that should be kept concealed, not only from a research validity perspective, but also from an ethical standpoint. Finally, we learned that having a fixed-rate contract and clearly defined objectives make the evaluation much easier during the project closure, and facilitates the settling process.

6.3 Use of Resources

This study was conducted with a considerably large budget (50,000 euros), supported by resources from the Norwegian Research Council and resources from Simula Research Laboratory. The effective use of the resources to ensure high quality data and maximize research value was an important issue for the study design. Different features of the design (e.g., the number of developers/companies hired, the choice of asking developers to repeat tasks on a second system) were in part the result of a trade-off between the achieving a high number of cases within the available time/budget. The fact that several maintenance projects (i.e., three at each company) where observed in parallel was due to time constrains placed on the availability of the funds. To maximize research value and enable data triangulation, a comprehensive study protocol was devised, so that different aspects of the maintenance project could be analyzed in detail, and from different perspectives.

The comprehensiveness of the study protocol, together with the fact that several projects were studied in parallel, resulted on a high and intensive work-load for the researchers present on-site, as they were responsible for the management of the project, the testing of the systems, and following the different data collection procedures. Despite the fact that is important for the researchers present at the study to keep an overview on the progress of the project, we conclude that for studies of this magnitude, counting with enough research/technical staff with a well-defined division of responsibilities (e.g., project management, testing, different data collection techniques) is very important for achieving efficient data collection and analysis. Eisenhardt et al., [8] suggests that between four and ten cases are desirable for theory building using case study design. Our study design allowed us to observe twelve maintenance projects. An alternative to this design would have been to hire less developers and use more resources for hiring research/technical staff supporting the study. Yet again, this relates back to the difficult matter of accommodating the intended research scope into the available resources.

Not only human resources constituted a challenge in this study. Conducting research in industrial settings poses a degree of uncertainty on the availability of other types of resources. This becomes of paramount importance when we want to control for different moderator factors, as in our design. Examples of some unforeseen resource problems we experienced, were the low speed of internet connection at the companies' site, and the stability/availability of external resources (such as our web-services). In the beginning of the project, we had an extensive period where adjustments were needed to the web-services required for the completion of tasks 1 and 2 (see Table 2). This could have been avoided with more comprehensive testing of the web services prior to the study. Another issue were the delays in configuring the development environment, due to the location of the installers and resources needed, and the speed of the internet connection at the working site. This also affected the storage of the data collected during the study into our data repository. In order to avoid delays in subsequent projects, installation files were provided in usb-keys to the developers, and to reduce the usage of internet for heavy data uploading (e.g., video files from think-aloud sessions), we switched to a local repository and leave the uploading to the remote repository for low-traffic periods of the day (e.g., evenings).

Based on these experiences, we suggest using some form of risk-analysis to examine the study protocol, and to create contingency plans to address potential issues during the study and avoid loss of data or resources.

6.4 Study Protocol and Data Analysis

Another lesson learned is the importance of a study protocol that can be tested by means of a pilot study. This allows for the identification of potential threats to validity, and points out practical issues not contemplated in the protocol. A pilot study helps also to adjust the time that is allocated to each of the different activities (project-related and research related), which can support better management of the study, and the prioritization of data collection activities. In practice it is often difficult to conduct a pilot study if the study design is highly complex and expensive. An alternative (followed in our study) is to use the first case as a kick-off stage, to adjust the timing for the activities and to perform troubleshooting. The risk of this approach is that the data on that case cannot be used in comparison to other cases when serious issues were encountered.

The study also showed us the importance of being "receptive" to the developers' input when it comes to introducing new working schemas into an organization. Remember that most case studies try to interfere as less as possible with the organization, and attempt to observe/investigate the phenomena within. Action research, on the other hand, includes an element of *intervention* as means to assess and improve a software process, or product. Whenever we want to add elements of control in a case study, we are interfering with the normal working mode of the individual or the organization. Although certain level of control is required for ensuring the validity of some results, we also need to be flexible to not impose a very strict "working schema" on the developers. To ensure that the design maintains a proper level of realism, it is important that the researcher first observes the interaction between the developers, and takes note of the culture and modus-operandi of the company before setting the final rules of the project, or instructing the developers with the different tasks/requirements.

Despite this need for flexibility in defining the protocol, it is also important to make sure the developers understand the importance of following certain protocols during the study once they are set. One problem we faced was the differences in the companies' habits concerned to management of issues and defect reporting. In order to investigate correlations between defects and code metrics at file level, we needed the developers to associate each bug fix to a single commit of a single file. Many developers were used to this procedure, but some developers had worked mostly with GUI development in the past and were accustomed to other types of testing/defect reporting. As result, some developers did not follow the protocol for reporting defects and bug fixes, which resulted on an incomplete data-set of defects at file level. These kind of situations can be avoided by close communication with the developers, and monitoring of the issue tracking system, or via IDE instrumentation. This last option can support diverse development activities while avoiding interruptions in the developers' work-flow.

Assumptions on potential threats to validity should be carefully weighted during the planning stage of a study, and while building the study protocol. In our study, we assumed that the *learning curve* implied by the choice of asking the developers to repeat the tasks on a second system was not going to affect the maintenance effort. Unfortunately, this assumption did not hold, despite the fact that we tried to mitigate its effects by asking the developers to perform several programming exercises to get used to the systems domain. The maintenance effort between systems that were maintained in the second round constituted roughly the half of the effort in systems maintained during the first round. This difference was not only due to increased knowledge of the developers on the system domain, but also because the third task allowed developers to reuse work that was done in the first round to perform the task in the second round. As tasks 1 and 2 were adaptive, they were highly dependent of the internal structure of the systems (e.g., data transference) and learning did not play a role. However, since task 3 consisted of adding a functionality, the developers found it natural to simply use the already developed solution and "plug" it into the second system. Although we could have ensured a lower learning curve by requesting the developers to conduct more comprehensive exercises prior to the maintenance project, it would have been hard to control for the code-reuse activity and this was an oversight on our side. To this we have to add the fact that developers many times base their solutions on code snippets or examples found on the internet, and it would have been very counter-intuitive and even tedious for the developers to write exactly the same functionality from scratch. Due to the learning curve, we had to discard the cases involving second round systems for the analysis at system level, as it was not possible to compare system level efforts. Fortunately, this threat to validity did not apply for other types of analysis, in particular those performed at file level, and also for those analysis which did not involve the effort measure. The bottom line of this discussion is that, as the complexity and expense of a given study increases, the importance of foreseeing difficulties/validity threats during the study execution becomes critical. Thus, this type of design resembles an experimental setting, in the sense that changes are more difficult to perform in later stages or when the study has already started.

Finally, a considerable challenge of this study after the data collection, was the summarization, integration and analysis of the data. In particular, processing and indexing large amounts of log-data (e.g., the Mimec logs amounted for 1400 hours in approx. 87MB of .csv files) and qualitative data (e.g., 3000 minutes for daily interviews and 480 minutes of recorded audio from the open interviews) was found extremely time consuming. Also, the integration of the different data sources for the purposes of data triangulation had to be automated via a Java program written for that purpose. Although we had a structured and functional repository for storing the data from the study, more "on the fly" analysis and summarization during the experiment could have eased the navigation and exploration of the data in latter stages. In our case, the lack of human resources during the study execution limited the amount of "pre-analysis" that could be done to enrich the data at the time of collection. A very useful tool to facilitate the indexing of data in this study was the *logbook* (source 10 in Table 2) in which the researcher logged all observations during the study. The logbook turned out to be an invaluable resource to pinpoint and identify observations of interest that could be further examined in through other data sources, and to synchronize time-wise the different events during the project across the different data sources. Moreover, as the logbook contained the dates of the observations, it was intuitive to navigate the progress of the project and validate the events via multiple sources sharing the same date, such as the SVN or the progress report written by the developers.

7. CONCLUDING REMARKS

In this paper, we discuss the notion of *assembling cases* as a plausible alternative to *selecting cases* when conducting multiple-case studies, in order to overcome some of the limitations of case study research. To illustrate its effectivity, we described in detail the design of a study that implemented this approach to investigate the impact of software design on maintainability, and discussed our experiences from conducting this study.

Although the increased element of control introduced a certain degree of *artificiality* in the study, we found that *assembled multiple-case studies* still provide the richness in detail of an in-vivo context that cannot be achieved by experimental settings. Yet, it is up to the skills and responsiveness of the researchers conducting the study to reduce this element of artificiality, by understanding and anticipating the working habits and modus-operandi of the organization involved in the study, without compromising the validity of the study. Morover, we found that the degree of *intrusiveness* that comes with this approach does not differ from traditional case studies (where cases are selected rather than assembled): whenever qualitative data is collected, a certain level of intrusion in the workflow of individuals and/or organizations can not be avoided.

We observed certain similarities in the challenges entailed by controlled experiments and this type of "controlled" case studies. One particular challenge is that, although it is already important to adhere to protocols in normal case studies, in the context of *assembled* case studies the protocol becomes of paramount importance for the validity of results. Because the environment for studies of this nature often constitutes an industrial setting, researchers will be faced with situations where certain factors cannot be controlled for. Consequently, it is important to adequately prepare, and create contingency plans that allow one to overcome potential issues from a practical and research perspective. We strongly recommend running a pilot study, even if it constitutes a down-scaled version of the final study.

Another challenge in (potentially complex and expensive) assembled case studies is dealing with the practical issues alongside the research issues, and allocating resources among them. For example, an adequate balance must be found between how much data should be collected, versus how much resources can be used in terms of time, staff and funds. Our experiences also warn against targeting an overly wide research scope, especially when lacking the staff to fully carry out, summarize and analyze such a large study.

Despite the intricacies and challenges of experimenting with this new approach, it allowed us to conduct both *theoretical* and *literal* replication in a case study, which is often very difficult to attain. This enabled cross-validation of observations across cases, and strengthened the internal validity of findings derived from qualitative sources. By combining qualitative and quantitative data collection, we could apply a mixed method approach that not only can identify trends or connections between variables, but also help to derive theories or explanations for those relationships.

Hannay et al. assert that theory-driven research does not yet play a major role in empirical software engineering and they collect and discuss evidence from SE literature that comments explicitly on the lack of relevant theory [12]. Case study research could significantly contribute to the development of theories from observations in relevant fields and contexts (i.e., *inductive research* [24]). Runeson et al. argue for the adequacy of case study research in SE because the complexity of the context plays an intrinsic role on the different phenomena being investigated [16]. The goal of this paper is to provide an example and discuss experiences to help other SE researchers with the design and conduction of multiple-case studies with an increased element of control. We aim to support the proliferation of *inductive research* in SE, leading to a wider set of theories in our field.

Acknowledgments. The authors thank Gunnar Bergersen for his support in selecting the developers and Hans Christian Benestad for providing technical support in the planning stage. They thank Bente Anda and Dag Sjøberg for insightful discussions, and for finding the resources to conduct the study that formed the basis of this paper.

8. REFERENCES

[1] O. Alonso, D. E. Rose, and B. Stewart. Crowdsourcing for relevance evaluation. *ACM SIGIR Forum*, 42(2):9, 2008.

[2] B. C. D. Anda, D. I. K. Sjøberg, and A. Mockus. Variability and Reproducibility in Software Engineering : A Study of Four Companies that Developed the Same System. *IEEE Trans. Softw. Eng.*, 35(3):407–429, 2009.

[3] D. E. Avison, F. Lau, M. D. Myers, and P. A. Nielsen. Action research. *Comm. ACM*, 42(1):94–97, 1999.

[4] I. Benbasat, D. K. Goldstein, and M. Mead. The Case Research Strategy in Studies of Information Systems. *MIS Quarterly*, 11(3):369, 1987.

[5] G. R. Bergersen and J.-E. Gustafsson. Programming Skill, Knowledge, and Working Memory Among Professional Software Developers from an Investment Theory Perspective. *J. Individual Differences*, 32(4):201–209, 2011.

[6] J. W. Creswell and V. L. P. Clark. *Designing and conducting mixed methods research.* SAGE, 2007.

[7] D. S. Cruzes, T. Dyba, P. Runeson, and M. Host. Case Studies Synthesis: Brief Experience and Challenges for the Future. In *2011 Int'l Symp. Empirical Softw. Eng. and Measurement*, pages 343–346. IEEE, 2011.

[8] K. M. Eisenhardt. Building Theories from Case Study Research. *Academy of Management Review*, 14(4), 1989.

[9] B. Flyvberg. *Five misunderstandings about case-study research.* In Qualitative Research Practice. Sage, 2007.

[10] M. Fowler. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.

[11] A. L. George and A. Bennett. *Case studies and theory development in the social sciences.* MIT Press, 2005.

[12] J. E. Hannay, D. I. K. Sjøberg, and T. Dybå. A Systematic Review of Theory Use in Software Engineering Experiments. *IEEE Trans. Softw. Eng.*, 33(2):87–107, 2007.

[13] L. M. Layman, L. A. Williams, and R. St. Amant. MimEc. In *Int'l Ws. Cooperative and Human Aspects of Softw. Eng. (CHASE)*, pages 73–76. ACM Press, 2008.

[14] A. Lijphart. Comparative Politics and the Comparative Method. *The American Political Science Review*, 65(3):pp. 682–693, 1971.

[15] C. Ragin. *The comparative method: Moving beyond qualitative and quantitative strategies.* University of California Press, 1987.

[16] P. Runeson, M. Höst, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples.* Wiley, 2012.

[17] O. Salo and P. Abrahamsson. Empirical Evaluation of Agile Software Development: The Controlled Case Study Approach. In *Product-Focused Softw. Process Improvement*, pages 408–423, 2004.

[18] D. I. Sjoberg, A. Yamashita, B. C. Anda, A. Mockus, and T. Dyba. Quantifying the Effect of Code Smells on Maintenance Effort. *IEEE Trans. Softw. Eng.*, 39(8):1144–1156, 2013.

[19] A. Strauss and J. Corbin. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory.* SAGE, 1998.

[20] W. M. K. Trochim. Outcome pattern matching and program theory. *Evaluation and Program Planning*, 12(4):355–366, 1989.

[21] A. Yamashita and L. Moonen. Do code smells reflect important maintainability aspects? In *Int'l Conf. Softw. Maintenance*, pages 306–315. IEEE, 2012.

[22] A. Yamashita and L. Moonen. Exploring the impact of inter-smell relations on software maintainability: An empirical study. In *2013 35th Int'l Conf. Softw. Eng. (ICSE)*, pages 682–691. IEEE, 2013.

[23] A. Yamashita and L. Moonen. To what extent can maintenance problems be predicted by code smell detection? - An empirical study. *Information and Softw. Technology*, 55(12):2223–2242, 2013.

[24] R. Yin. *Case Study Research : Design and Methods (Applied Social Research Methods).* SAGE, 2002.