Perceived Productivity Threats in Large Agile Development Projects

Jo E. Hannay
PREPARE Group, Simula Research Laboratory
Pb. 134, NO-1325 Lysaker, Norway
johannay@simula.no

Hans Christian Benestad
PREPARE Group, Simula Research Laboratory
Pb. 134, NO-1325 Lysaker, Norway
benestad@simula.no

ABSTRACT

Applying agile methodology in large software development projects introduces many challenges. For example, one may expect that the combination of autonomous teams and the necessity for an overall organizational control structure may lead to conflicts, and one may expect that Agile's informal means of knowledge sharing breaks down as the number of project participants increases. Such issues may in turn compromise the project's productivity. In order to better understand potential threats to productivity in large agile development projects, we conducted repertory grid interviews with 13 project members on their perceptions of threats to productivity. The project was a large software development project consisting of 11 Scrum teams from three different subcontractors. The repertory grid sessions produced 100 issues, which were content analyzed into 10 main problem areas: (1) Restraints on collaboration due to contracts, ownership, and culture, (2) Architectural and technical qualities are given low priority, (3) Conflicts between organizational control and flexibility, (4) Volatile and late requirements from external parties, (5) Lack of a shared vision for the end product, (6) Limited dissemination of functional knowledge, (7) Excessive dependencies within the system, (8) Overloading of key personnel, (9) Difficulties in maintaining well-functioning technical environments, (10) Difficulties in coordinating test and deployment with external parties. Using critical-case reasoning, we claim that projects deploying agile practices in projects with less favorable conditions than those enjoyed in the current project, and that are larger and more complex, are likely to face similar challenges.

Categories and Subject Descriptors

D.2.9.d [**Software Engineering**]: Management—*Productivity*; D.2.9.g [**Software Engineering**]: Management—*Software process models*

Keywords

Agile, Large Development Projects, Practitioners' Mental Models, Repertory Grid, Case study

1. INTRODUCTION

Agile software development methodologies are intended as a set of practices for the timely delivery of software of high value to its users [12]. The agile approach is a reaction against traditional relay-style development with exhaustive up-front planning and design. Instead, it is argued that uncertainty and change is inevitable and should therefore be dealt with rather than avoided. Customer involvement and collaboration is a key tenet.

In many peoples' minds, agile approaches are tailored to small projects (say, less than 20 developers): Large projects should either not use agile [39, 9] or one should not run large projects but rather break large projects into smaller sub-projects [25, 4]. However, with the current global investment and reliance in IT, for example in the public sector, large and partially sub-contracted projects seem inevitable and are likely to increase in number. Further, agile development has gained a firm foothold among developers and also increasingly with management. Thus, recent discussions have focused on how to reconcile "large" and "complex" with "agile" [38, 26, 5]. These discussions regard both management issues (e.g., command and control versus autonomous teams) [41, 18, 27, 32, 15, 30, 17] and technical issues (e.g, architecture in the face of code centeredness) [10, 24].

Several authors have given guidelines on how to scale up agile projects, e.g, [10, 24]. It is important that such accounts be evaluated and complemented by further empirical evidence and experience. An immediate source of evidence is practitioners' perceptions of what the most challenging issues are while working in large agile projects, which may then be systematized and analyzed in a manner that lends itself to knowledge building [1, 2, 14, 21].

As a case, we followed a large, agile project (Project X) in the construction phase. A total of 176 people are directly involved in the project, including 88 developers in 11 Scrum teams from three sub-contractors. The project is planned to last for four years and has a budget of approx. EUR 100 million. When the project had been in its construction phase for about one year, the project managers felt that there might be glitches in productivity at various locations in the project. In order to get a clearer picture of possible threats to the project's productivity, we conducted concept elicitation sessions with project members. Threats to productivity is here informally understood as circumstances that may cause a less than optimal amount of features completed per unit time. This paper describes the method and main results of this concept elicitation.

We used the repertory grid technique for concept elicitation [13, 20, 40] on 13 project members: two in-house developers, two product owners, two architects, two test managers, one project manager for the Development sub-project, one Scrum team leader, two project owners, and

ESEM'10, September 16–17, 2010 Bolzano-Bozen, Italy

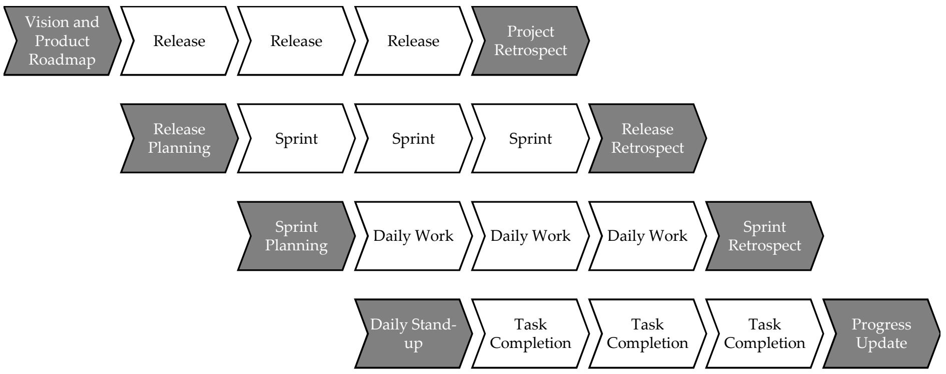


Figure 1: Project lifecycle (adapted from [38])

the Project Manager. The interviews were analyzed both individually and on an aggregated basis. This gave a picture of the main challenges, their relative importance, and their tentative causal relationships.

Section 2 describes the case, Section 3 describes our research method for the study, Section 4 summarizes the findings of the study, and Section 5 discusses the applicability of results and threats to validity. Section 6 concludes.

2. THE CASE

Project X was initiated to (re)develop a large system commissioned by a governmental body for administering pensions and loans. New political reforms and legislation necessitated major reorganizations of case procedures. Three software suppliers are involved: The customer has five in-house Scrum teams (40 developers), and two external consultancy houses have six teams (supplying 48 developers between them). In addition, there is one cross-cutting architectural team, one cross-cutting test team, and one development environment team supporting the 11 Scrum teams. The choice of following agile development practices was taken on grounds that requirements were guaranteed to change during the project's lifetime due to political reforms that were not fully decided upon yet, and whose indirect effects through other governmental bodies were not yet clear.

The project's lifecycle can be modeled by the "agile fractal" [38] in Figure 1. Iterations (white boxes) permeate the project at multiple levels of abstraction. At each level, iterations are preceded by a planning phase and proceeded by a retrospective phase (gray boxes).

At the top level, the project progresses in terms of releases (mainly three a year). The project's initial planning stage (to establish the vision and product road map) was conducted over a six-month period. However, the initial road map was perceived as difficult to follow by the product owners, and also, the chief architect left the project along with key knowledge. Therefore, the road map was revised about a year into the project. The outcome of this revision was master product backlog with a total of about 300 high level items for the entire project. Each item comes with a priority (value for customer) and a relative estimate (story points). The construction phase is planned to last for three years.

At the next level, a release progresses in terms of five to six sprints. At the end of a release, the evolved system is put into production. The input to a release is a release backlog which is a designated portion of the master product backlog to be realized in the release. A release is planned as follows: At about six weeks prior to the first sprint, key stakeholders meet to refine and develop a design specification of the items in the release backlog. Then, the release backlog is split into three subcontractor-specific backlogs. The subcontractors estimate their backlog items using their preferred estimation method and establish an hours per story-point ratio (120 man-hours per story point, say). This ratio is then used in negotiations with project management to determine a target cost for the release with a 50-50 shared responsibility agreement for over-/underruns. The parties use the target cost-based [33] *PS2000* contract [34].

At the lowest level, the input to a sprint is a sprint backlog that consists of detailed tasks derived from the subcontractor-specific backlog items that are to be realized in the sprint. Sprints are set to last for three weeks.

3. RESEARCH METHOD

The study followed principles for applying scientific methods to important industrial challenges in a more flexible manner than in traditional large-scale research programs:

1. *Relevant, concrete and general interest*: The research objective is initiated by a desire to solve an important concrete challenge in a software development company. The challenge is perceived to be of interest to the software industry in general.

2. *Practitioners' theories*: The research questions are defined and refined by consulting (possibly by eliciting the mental models of) practitioners working on the problem.

3. *Scientific-strength research methodology*: The research questions are answered by applying sound scientific research methods, if necessary, with the help of researchers' expertise on such methods.

Principle 1 is inspired by [19, 28]. In our setting, the research was initiated by a desire from Project X to discover productivity threats in their project, and secondly, by the large general interest in the benefits and challenges of large agile development projects. Our research objective in the specific context of the project thus became:

> *Research objective*: To lay out and systematize knowledge and perceptions on the most salient threats to productivity in Project X.

By applying critical case reasoning [45], we claim that the results have applicability beyond the immediate case context: Project X has several critical characteristics that are expected to work in favor of effective software development in general and agile development in particular [42, 7]. First, efforts have been made to attract the best people on the market in all roles of the project. These efforts were facilitated by the fact that the project is a prestige project within the national IT-development industry. Second, the project's development strategy has been formally committed to at all organizational levels. Third, the project receives considerable attention and resources from business experts and end users. Fourth, developer teams and business experts are co-located with the required support functions and infrastructure. Hence, threats to productivity that we find in the context of this project are also likely to surface in comparable projects where the above four conditions are equally or less favorable. We elaborate on this reasoning in Section 5.

The impetus of Principle 2 comes from work by, among others, Argyris, Gigerenzer, Jarvis, and Schön [1, 2, 14, 21], and is based on the view that practitioners hold valuable implicit and explicit knowledge of their work domain that should give input to scientific knowledge. Since practitioner knowledge is often tacit, various methods for eliciting such tacit knowledge may be employed.

Principle 3 was met by using the repertory grid technique (see Section 3.1 below). This is a semi-structured interview technique which has had numerous applications in other domains of research [40], such as marketing, quality control, work training, and software engineering [8, 3, 35]. For software engineering practice, the technique has also been proposed for e.g., requirements elicitation [31]. The idea of practitioner's theories (Principle 2) permeates the repertory grid technique. The technique is based on Kelly's personal construct theory [22, 23] which implies that people try to understand the world and their place in it by constantly testing hypotheses about the world. The resulting construct system reflects a person's current understanding or beliefs and will change over time. In our context, this translates to eliciting a software professional's perceptions regarding threats to productivity in Project X. The technique allows one to gather salient information in a short time compared to other relevant expert elicitation techniques. This was essential, since Project X was in a particularly hectic phase.

3.1 The Repertory Grid Technique

The repertory grid technique's main methodological objective is to elicit the interviewee's current perception of a topic. The interviewee determines the salient concepts of the interview as well as a value system on which to rate the salient concepts relatively to each other. We followed the guidelines for the technique described in [13, 20]. The key parts of a repertory grid session are the *topic*, the *elements*, the *constructs*, and the *grid*, as explained in the sections below.

The interviews took place during November and December 2009, and were conducted by the two authors at Project X's premises. The interviews lasted from 1 to 2 hours. All interviews were audio recorded for the purpose of subsequent clarifications. Anonymity was guaranteed. The interview session began by the interviewer briefly introducing himself, the field of empirical software engineering, and the ideas behind evidence-based practice. The interviewees were then asked to talk informally for 3–5 minutes about their role in Project X and to present any initial thoughts on productivity threats in the project. The interviews then moved on to describing the topic in a more concise manner, eliciting the elements, the constructs, and the grid.

3.1.1 Topic

The *topic* in a repertory grid study forms the domain of interest in which the interviewee's personal constructs are to be elicited. The overall topic for the study was presented as "obstacles to productivity in large, agile software projects", where "productivity" was to be understood informally as "circumstances that may cause a less than optimal amount of completed features per unit time in the project".

3.1.2 Elements

The salient concepts are the first things to be elicited in a repertory grid session, and are called *elements*. The elements were elicited by asking the interviewees to contemplate

> "your personal opinion on concrete events, problems, circumstances that may have compromised, or might compromise productivity in Project X".

Each interviewee was provided with a set of blank cue cards and was asked to write keywords for each subject he or she came up with when thinking about the above. Each cue card constitutes an element. The interviewee was free to write down as many elements as he or she liked.

In element elicitation, it often turns out that elements overlap or subsume other elements. Such elements are problematic, since the subsequent rating becomes more difficult for the interviewee [40]. The interviewee was therefore given the opportunity to merge, split, or rethink elements. Subsequently, the interviewee was asked to pick out the eight most salient elements that he/she had written down. This was intended as a quick quality check, while also limiting the number of elements. The recommended number of elements is suggested to be between five and 12 in most cases [20]. It would be possible to limit the number of elements at the outset by simply asking for, say, the eight most salient elements at the start. However, this would require the interviewee to hold a large number of elements in working memory simultaneously, which is cognitively taxing, e.g., [29]. In our sessions, it happened that more than eight elements were kept if they were perceived as important, and in some cases the number of elements elicited was less than eight. The resulting elements (with explanations) of one of the interviewees are presented in the first half of Table 1.

3.1.3 Constructs

After the elements have been elicited, the interviewer pools three elements at a time and asks the interviewee to name a characteristic of two of them that distinguishes them from a named characteristic of the third one. This pair of bipolar characteristics goes under the name of a *construct*. For example, given the elements *Clarification of dependencies*, *Multiple roles*, and *Upfront plans* (Table 1), the interviewee grouped the first two and contrasted them to the third, by characterizing the first two as *Unclear responsibilities*, and the third as *Traditional distribution of responsibility*. This gives the bipolar construct *Unclear responsibilities – Traditional distribution of responsibility*. Direct opposites, such as *Unclear responsibilities – Clear responsibilities* are discouraged, since such constructs typically are self-contained

Table 1: Elements and constructs of one of the interviewees

Elements

1. *Process disagreement*: The project feels non-agile at times. Vendors want to work agile, and now we're suddenly supposed to work waterfall. Flexibility is compromised.
2. *Non-shippable code from sprints*: Three annual main releases with designated test runs and test teams. Testing is inadvertently postponed from sprints to these main releases, and is therefore not performed satisfactorily en route, resulting in not potentially shippable code. QA sets in too late and since error fixing has high priority, this then diverts resources from ensuing activities.
3. *Clarification of dependencies*: Don't know what the other teams are doing, and don't know early enough what dependencies there are. Ensuing clarification regarding these things steals time from ongoing efforts.
4. *Multiple roles*: Multiple and concurrent roles (e.g., architect, responsible for builds, responsible for DB) leads to context switching that in itself drains resources. Also, people from the Scrum teams get picked out to contribute in the release teams which have their own separate runs in parallel.
5. *Long planning horizon*: Often, details are planned for things further into the future, instead of focusing on details in, say, the next sprint. This comes from the old school were things should be speced out beforehand, but this leads to badly specified user stories.
6. *Bottleneck personnel*: Key people are sparse and often not available when needed. They might be involved in other projects.
7. *Upfront plans*: Twist toward waterfall model and measurement. New project management introduced "masterplan". People were a bit put off by this new model from above. Mismatches between peoples' perceptions of models is demotivating and gives bad priorities.
8. *Development environment*: The development environment is often down and is not efficient enough. Demotivating. VDMs lock up. It's important to coordinate development efforts, but the templates aren't good enough.

Constructs

1. *Framework – Personal*: Organizational aspects of the project versus things experienced by an individual.
2. *Method – Resources*: The process method (or breaches in method, i.e. in Scrum) versus all types of resource in the project.
3. *Project size – Risk aversion*: Many teams, obscurity in what we're doing, and things that are difficult in this large project versus a fear toward going all agile; one dares not abstain from detailed planning of future events.
4. *Procrastination – Mental models*: Issues that explicitly or implicitly are put off to later because some other process will deal with it somehow versus the various process models people in the project have in their minds.
5. *Srum/agile – Specification*: Method and framework for method versus weakly defined tasks.
6. *Unclear responsibilities – Traditional distribution of responsibility*: Many dependencies, loose ends, and multiple roles that make it easy to think that someone else has the responsibility (ill-defined Scrum autonomies) versus old-fashioned command and control ideas.
7. *Feeling productive – Distribution of competence*: To feel efficient and motivated versus how knowledge and expertise is distributed.

and do not relate to the rest of the construct system. This process of construct elicitation is repeated with new sets of three elements until it is apparent that no essentially new constructs emerge. The focus is not on eliciting all possible constructs, but on eliciting sufficiently salient constructs. In our case, the triads were pooled in a random manner, but it was ensured that all elements were used approximately the same number of times, see [13] for further suggestions.

The repertory grid technique is developed to minimize interviewer bias. Nevertheless, the interviewer has a responsibility to guide the interviewee to present constructs that relate to the research objective. A process known as *laddering* [13] was performed if constructs needed clarification. Laddering is the process of going from general constructs to specific constructs or vice versa according to what is relevant for the research objective. Another way to guide the interviewee is to introduce the phrase "in terms of" when comparing elements. We asked "what contrasts events/problems A and B from event/problem C, in terms of how they compromised, or might compromise productivity'. This is meant to guide the interviewee to present relevant constructs without the interviewer actually suggesting constructs [13, 40]. All constructs of one of the interviewees (with explanations) are presented in the second half of Table 1.

It may be pertinent to provide predefined elements and constructs [13, 20, 40]. For our study, three constructs were provided for all interviewees:

1: *Not serious – Serious*: How serious you perceive a particular element to be in terms of negative impact on productivity in Project X.

2: *Cause – Consequence*: Whether you perceive an element to be a cause of further threats to productivity in Project X (not necessarily captured by the elicited elements) or whether you perceive it as a consequence of other threats.

3: *Easy to handle – Difficult to handle*: How easily you think an element may be handled or resolved in Project X.

These constructs were provided to translate the interviewees' perceptions into immediate improvement initiatives: It makes sense to start with elements that are classified as *Serious*, *Cause*, and *Easy to handle.* Supplied constructs that consist of direct opposites are not considered a problem since they are not part of the elicited construct systems.

3.1.4 Grid

The last phase in the repertory grid session, is to rate each element on each of the constructs. The two poles of a construct, which were written on two separate cards, were placed at opposite ends of a rating mat laid out on the table, with five places (Figure 2). The interviewee was then asked to place all elements relatively to each other between the two construct poles, according to how close the interviewee felt an element associated to one or the other pole. In modern applications of the technique, one normally uses a 5- or 7-point scale [20]. The interviewee was encouraged not to use the mid-point of the scale as a way out when he/she was unsure where to put the element or if the element was not possible to rate on the given construct. Instead, such elements were left out. During the rating, each interviewee was encouraged to think aloud. This gives the interviewee the opportunity to verbalize his or her decisions. Reasoning or explaining action (Type 3 verbalization, in Ericsson and Simon's terminology [11]), has been shown to increase performance in decision making, e.g., [6]. Also, the verbalizations helped the interviewer understand the often tacit choices made by the interviewee.

4. FINDINGS

The data from the repertory grid sessions were analyzed per interviewee, and then the individual results were content analyzed to gain an overall understanding of the perceptions regarding productivity in the project.

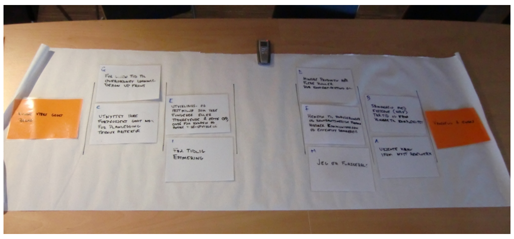


Figure 2: Rating mat with elements (white) and bipolar construct (colored).

Table 2: Problem areas.

1: Restraints on collaboration due to contracts, ownership and culture.
2: Architectural and technical qualities are given low priority.
3: Conflicts between organizational control and flexibility.
4: Volatile and late requirements from external parties.
5: Lack of a shared vision for the end product.
6: Limited dissemination of functional knowledge.
7: Excessive dependencies within the system.
8: Overloading of key personnel.
9: Difficulties in maintaining well-functioning technical environments.
10: Difficulties in coordinating test and deployment with external parties.

4.1 Analyses per Interviewee

The responses of a single interviewee may be analyzed in different ways [13, 20, 40]. In our case, we conducted a range of analyses of the elements and the constructs (cluster analyses, principle component analyses, map analyses) using *WebGrid* (Centre for Person-Computer Studies `repgrid.com`) which were fed back to each interviewee. We omit these analyses, since the main purpose of this paper is to present the aggregated results. The full set of elements and constructs can be accessed by contacting the authors.

4.2 Aggregated Analyses

In total, 100 elements and 66 constructs were elicited. In order to summarize and find trends in this information, we content analyzed the elements into categories of similar meaning. The constructs helped in understanding the individual elements, but were not content analyzed.

The content analysis of elements was carried out in an exploratory inductive manner following guidelines in [20]. All 100 elements were spread out on a table. If two elements seemed to be related, they were grouped, thus forming an initial inductively formed category. Other elements might be added to such a category, or they might form a new category. This aggregation process continued until all element cards were allocated to a category. Both authors performed this process independently. Inter-rater reliability in this case concerns agreement on categories that arise and then on how elements are allocated to categories. Prior to computing agreement scores, we consulted each other's categories and decided which ones corresponded semantically to the other's categories. As expected in such inductive approaches [20], inter-rater agreement was low on the first pass: 50% when considering both corresponding and non-corresponding categories and 59% when considering only corresponding categories. Then, the non-corresponding categories were negotiated upon until a full category set, with agreed-upon names, was obtained. The remaining discrepancies were discussed and resolved. To check the reliability of the category set, all elements were recategorized independently giving an agreement score of 77%, which is acceptable. The ten resulting categories give rise to the ten problem areas in Table 2, and are sorted roughly w.r.t. Project X's lifecycle.

In the following, we provide for each of the problem areas:

- An elaboration of the problem area and examples of elements, as well as the associated perceived causes and consequences.
- All elements belonging to the problem area, plotted according to their ratings on the three supplied constructs. Each element is labeled with an interviewee identifier (A–M). It is thus possible to see how widely a problem area is represented among the interviewees.
- Proposed initial actions toward the problem area based on the elements that are located in the *Serious*, *Cause*, *Easy to handle* segment of the plots. These actions are thus derived from the project's perceptions of itself.

1: Restraints on collaboration due to contracts, ownership and culture. The stakeholders in the project need to uphold their interests. Such interests are rooted in the contracts between subcontractors and customer, but pertain also to the perceived ownership of tasks and competence, as well as to individuals' and subcontractors' views on methodology. Often, such interests are not explicitly stated or communicated adequately.

These interests constrain how tasks and expertise are distributed in the development teams, leading to a certain rigidity. Consequences are teams with incomplete competencies, teams that might run out of tasks, and inter-team communication that is not as open as it should be. In summary, these restraints limit process learning and the development toward common goals.

Figure 3 maps the elements in this problem area relative to the three supplied constructs. Based on the position of the elements with regards to this mapping, one may imply the following starting points for improvement.

1: Maintain a designated backlog of overflow tasks along with incentives to solve these tasks.
2: Develop routines prior to each sprint for evaluating needs for particular expertise.

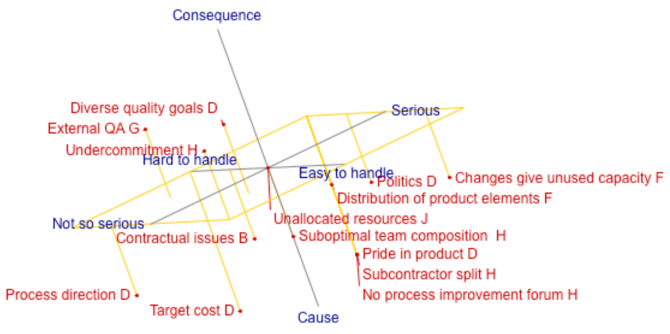
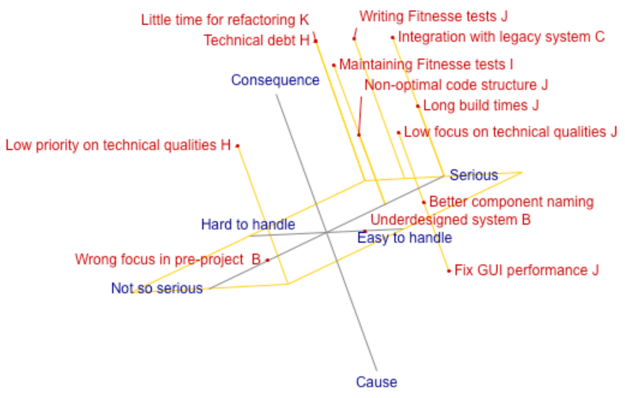


Figure 3: Map of elements in Problem Area 1.

2: Architectural and technical qualities are given low priority. The teams encounter problems that might have been avoided, had there been a stronger focus on architectural and technical qualities, both at the start of the project and throughout the construction phase. The problems encountered pertain to qualities such as software performance, stability, testability and maintainability.

However, a stronger focus on architectural and technical qualities is recognized as diametrical to concerns such as the time pressure of the project, the product owner's field of expertise, and the agile focus on value for customer. The perception is nevertheless, that technical qualities should receive more attention.



Little time for refactoring K
Writing Fitnesse tests J
Technical debt H
Integration with legacy system C
Maintaining Fitnesse tests I
Consequence
Non-optimal code structure J
Long build times J
Low focus on technical qualities J
Low priority on technical qualities H
Serious
Better component naming
Underdesigned system B
Hard to handle
Easy to handle
Wrong focus in pre-project B
Fix GUI performance J
Not so serious
Cause

Figure 4: Map of elements in Problem Area 2.

Implied starting points for improvement:

1: Maintain a designated backlog of technical improvement tasks along with incentives to solve these tasks (one item in the backlog might be to designate effort to optimize the use of the test execution tool).

2: Conduct more systematic tests of technical qualities at the sprint checkpoints.

3: Conflicts between organizational control and flexibility. There are three main conflicts associated with this problem area: First, the master product backlog (Section 2) is subjected to early and binding estimates. This conflicts with the agile practice and perceived need to maintain a continuing reevaluation of requirements and estimates. Second, features, or groups of features, are allocated early to certain resources in the project. This induces ownership issues and may hinder opportunities to redistribute subtasks more evenly across teams. Third, at the end of each release, there is an acceptance test. This diverts attention away from the sprint tests, resulting in non-shippable code from the sprints, and in the accumulation of architectural and technical issues. In short, bounds on flexibility compromises motivation and optimal allocation of resources to tasks.

Implied starting points for improvement:

1: Make sure to reevaluate whether the master product backlog elements are necessary and sufficient.

2: Ensure flexibility in prioritizing and allocating tasks.

3: Consider estimates for larger portions of subtasks.

4: Move resources from release testing to sprint testing.

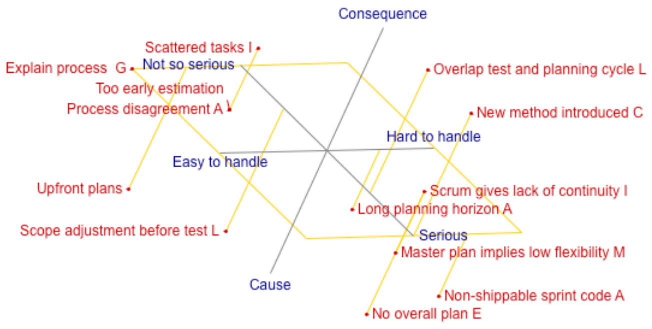
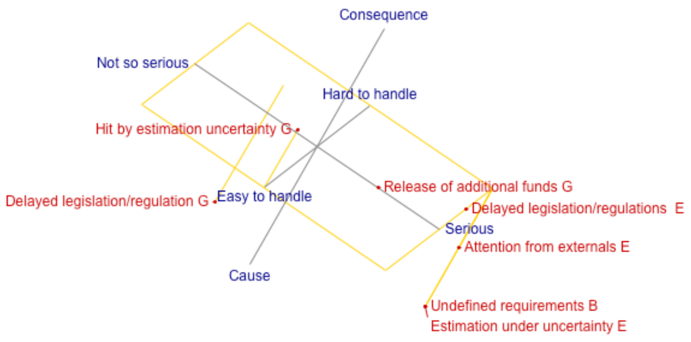


Figure 5: Map of elements in Problem Area 3.

4: Volatile and late requirements from external parties. Many of the project's software requirements depend on new political reforms. Due to time pressure, the project needed to start the construction phase before all ramifications of new legislation were clarified. In addition, the system must integrate with other systems across the public sector, which themselves are under redevelopment. Overall, volatile and late requirements introduce uncertainty in effort estimates, and are perceived to be a direct cause of delays. For example, personnel capable of translating legislation into software requirements are often overloaded. Although the funding bodies are prepared to cover additional project costs due to late requirements, project management finds it challenging and time consuming to produce the documentation required to release these funds. Project management also finds it time consuming and unproductive to defend the original estimates and uncertainty analyses when queried by external QA auditors hired by funding bodies.



Consequence
Not so serious
Hard to handle
Hit by estimation uncertainty G
Release of additional funds G
Delayed legislation/regulation G
Easy to handle
Delayed legislation/regulations E
Serious
Cause
Attention from externals E
Undefined requirements B
Estimation under uncertainty E

Figure 6: Map of elements in Problem Area 4.

Implied starting points for improvement:

1: Ensure documentation that links added costs and delays to pre-agreed risk factors

2: Document and convey the quality of the process that led to estimates and uncertainty analyses.

3: Reevaluate time scope and level of detail for estimates.

4: Ensure that people with competence on the software solution participate in estimation.

5: Lack of shared vision for the end product. There is uncertainty as to the presence of a shared vision of how the software product will support the business needs. Without such a shared vision, there are concerns that redundant work

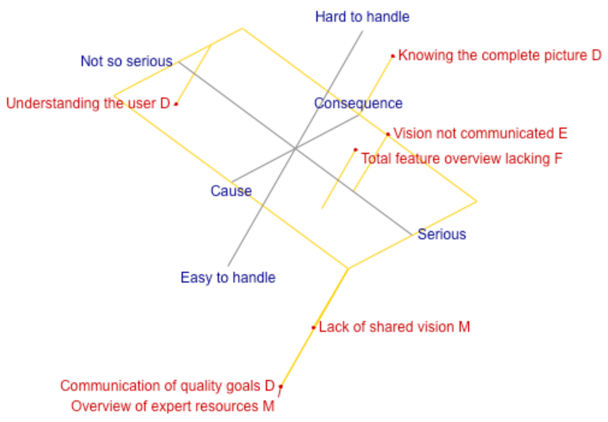

Hard to handle
Not so serious
Knowing the complete picture D
Understanding the user D
Consequence
Vision not communicated E
Total feature overview lacking F
Cause
Serious
Easy to handle
Lack of shared vision M
Communication of quality goals D
Overview of expert resources M

Figure 7: Map of elements in Problem Area 5.

is being done, that code will have to be withdrawn, and that the system will eventually not fully satisfy the user's needs.

Implied starting points for improvement:

1: Clarify how and by whom the overall solution is envisioned.

2: Allocate time to describe and communicate vision.

6: Limited dissemination of functional knowledge. Knowledge about the legacy system, its users, the dependencies (technical and non-technical) to external systems, and the overall domain knowledge is perceived as critical for the developers to receive apt specifications. Concerns were expressed that the lack of such knowledge impedes well-designed solutions, and that this had already resulted in erroneous code. There were concerns that the external subcontractors possessed insufficient functional knowledge.

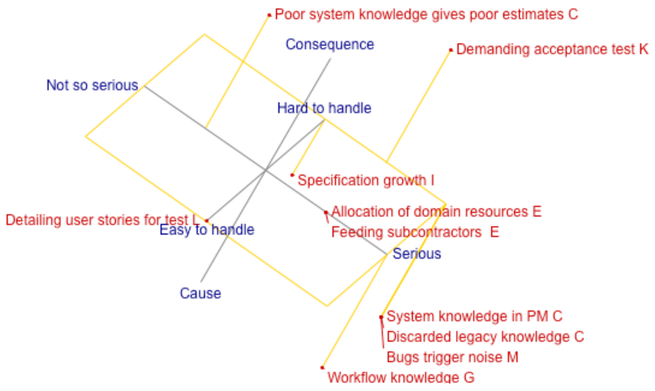


Figure 8: Map of elements in Problem Area 6.

Implied starting points for improvement:

1: Ensure that the Scrum teams have access to people with adequate functional knowledge for the tasks at hand.

2: Improve the timing for transferring knowledge from business to development.

3: To a greater extent, involve people with rich functional understanding in the specification of tests.

7: Excessive dependencies within the system. Subcontractors and their Scrum teams are responsible for designated system parts, where partitioning follows both vertical architectural layers and horizontal sub-domains. Problem Area 7 refers to dependencies beyond those inherent to the business domain and those made explicit in the target architecture. It is perceived as particularly challenging to handle interdependencies between code that is developed in parallel. Due to the compressed lead time/effort ratio, a large number of dependencies must be resolved by developers as part of the coding activity. Communicating about this steals substantial development time. Due to the priority on lead time and deployable functionality, focus is not always on minimizing the technical dependencies being introduced, as discussed under Problem Area 2. In some cases, expensive corrections to incompatible code parts had to be done, and in a few cases, teams were unknowingly developing code for the same features. The problem of interdependencies in newly developed code has also led to challenges with configuring integration test environments and test data, as discussed under Problem Area 9. Some interviewees expressed concerns that project management underestimates the negative effects of compressing the lead time/effort ratio.

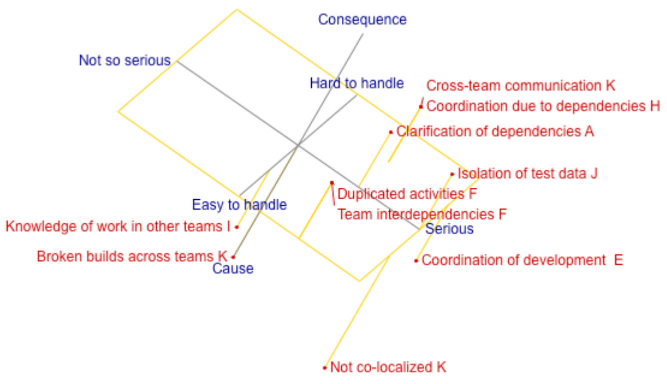

Consequence
Not so serious
Hard to handle
Cross-team communication K
Coordination due to dependencies H
Clarification of dependencies A
Isolation of test data J
Duplicated activities F
Team interdependencies F
Easy to handle
Serious
Knowledge of work in other teams I
Broken builds across teams K
Cause
Coordination of development E
Not co-localized K

Figure 9: Map of elements in Problem Area 7.

Implied starting points for improvement:

1: Identify dependencies between product queue elements at an early stage of release planning.

2: Clarify the responsibilities for configuring test data.

3: Assess the need for broader knowledge of the business domain and the solution in different parts of the project.

8: Overloading of key personnel. It has been claimed that some elements of agile methodology can lead to employee burnout [43], and that initial productivity gains is challenging to sustain over time [36]. Examples of such elements are constant pressure on delivering working code, the expectancy to act socially and share shortcomings and problems in daily meetings, and the reliance on the project's transactive memory, rather than on externalized knowledge in the form of software system documentation. Problem Area 8 is related to such claims. In particular, only a very small group of people in the project deeply understand business rules and system requirements, and thoroughly understand the IT solution. These people experience great pressure and are perceived by themselves and other project members as bottlenecks, with a constant backlog in serving developers with specifications. A large degree of context switching, necessary to serve the 11 teams working on different sub-domains, is perceived to further decrease the productivity of key personnel.

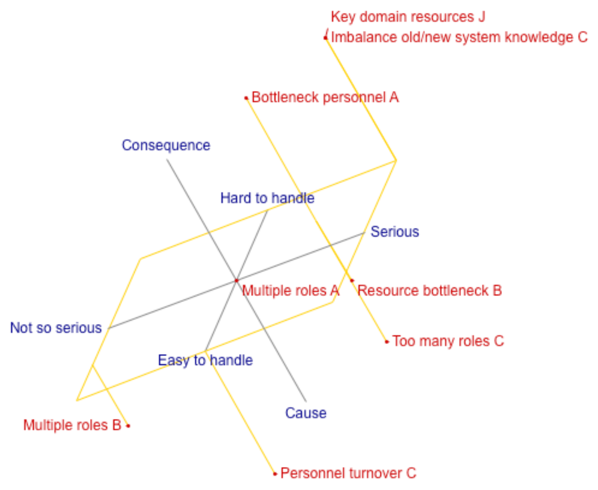


Figure 10: Map of elements in Problem Area 8.

Implied starting points for improvement:

1: During release planning, clarify the need for and availability of key personnel.

2: Allocate key personnel to specific teams while not sacrificing flexibility.

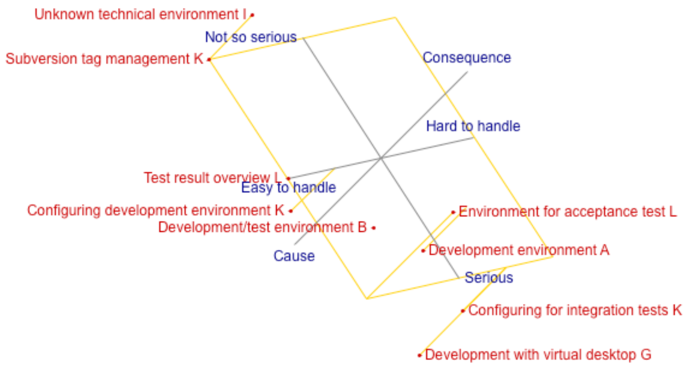
9: Difficulties in maintaining well-functioning technical environments. For small projects, configuring the developers' technical environment, such as development tools, change management systems, build support and automatic test execution is a relatively well understood task. In Project X, the system evolves at a high pace, with a high degree of parallel development. This puts demands on the technical environment to evolve constantly. Achieving acceptable response times and maintaining consistent test data are mentioned as particularly challenging. Substantial resources are used to maintain the technical environment, and developers' time is sometimes wasted when problems do occur.

Implied starting points for improvement:

1: Focus resources for preparing the test environment for the release tests.

2: Ensure better prediction of test needs in different parts of system, using historical data.

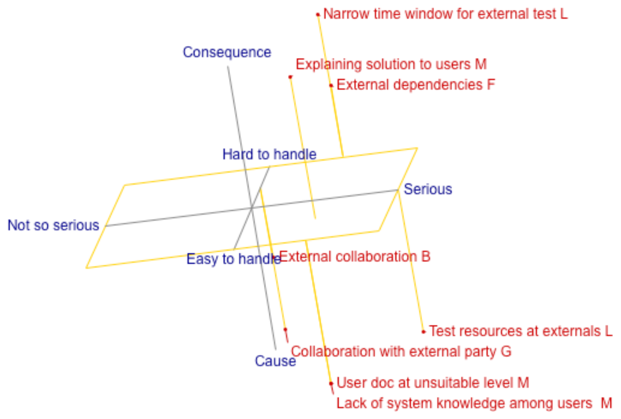
3: Reassess the technical platform used for development and test.



Unknown technical environment I
Not so serious
Subversion tag management K
Consequence
Hard to handle
Test result overview L
Easy to handle
Configuring development environment K
Development/test environment B
Environment for acceptance test L
Cause
Development environment A
Serious
Configuring for integration tests K
Development with virtual desktop G

Figure 11: Map of elements in Problem Area 9.

10: Difficulties in coordinating test and deployment with external parties. The system under development integrates with external systems, such as web services delivering data from other public bodies. Also, as a consequence of incremental deliveries, the system integrates with the legacy system being replaced. Substantial effort is needed to coordinate the integration of new versions, for example to transfer knowledge of new features and to set up appropriate test data. Unfortunately, short time windows are available for performing such integration, and such windows might not become available at the appropriate time. This escalates problems in the affected organizations, leading to unplanned effort expenditure at several levels of the project.



Narrow time window for external test L
Consequence
Explaining solution to users M
External dependencies F
Hard to handle
Serious
Not so serious
Easy to handle
External collaboration B
Test resources at externals L
Cause
Collaboration with external party G
User doc at unsuitable level M
Lack of system knowledge among users M

Figure 12: Map of elements in Problem Area 10.

Implied starting points for improvement:

1: Identify and commit the right people in the affected external organizations.

2: Produce documentation about new features at a level found to be appropriate by the user organization.

5. DISCUSSION

5.1 Applicability to other agile projects

In Section 3, we claimed that Project X has certain characteristics that are favorable for running an efficient (agile) development project. We argued, by critical case reasoning, that any threats to productivity uncovered in Project X would likely surface in similar-type large and agile projects where these favorable conditions are less present.

It is possible to augment the critical case reasoning done hitherto, by extracting project characteristics that are plausible root causes of the ten problem areas. Based on a informal assessment of the project as a whole and the findings of this study, we propose that the following project characteristics are at the root of the 10 problem areas:

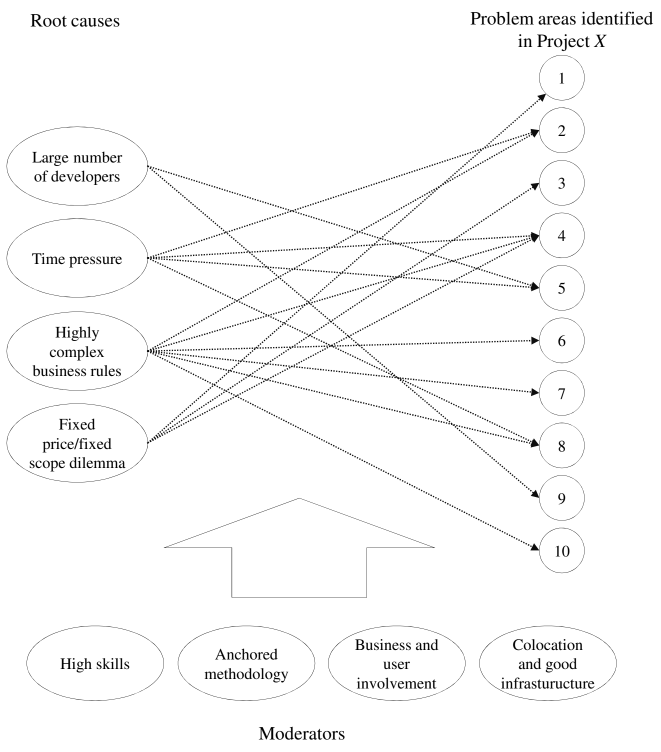
1: Large number of developers.

2: Time pressure.

3: Highly complex and externally connected business processes and business rules.

4: Fixed-price/fixed-scope dilemma in subcontracted development services.

These four root causes are postulated to underly the 10 problem areas, whereas the favorable conditions mentioned



Root causes
Problem areas identified in Project X
Large number of developers
Time pressure
Highly complex business rules
Fixed price/fixed scope dilemma
1
2
3
4
5
6
7
8
9
10
High skills
Anchored methodology
Business and user involvement
Colocation and good infrasturucture
Moderators

Figure 13: Conceptual model for the effect of project characteristics on large agile development projects.

earlier are postulated to moderate the effect of these root causes, see Figure 13.

The four root causes and the four favorable conditions are all project characteristics that are recognized as important factors in project productivity [44, 42, 7]. For example, the literature review in [42] suggests that the three top cross-contextual factors influencing productivity are, (1) Team capabilities and experience, hereunder, programming language experience, application experience and familiarity, and project manager experience and skills, (2) Software complexity, hereunder architecture complexity, complexity of interface to other systems, and database size and complexity, (3) Project constraints, hereunder schedule pressure and decentralized/mulitisite development, and (4) Tool usage and quality/effectiveness; all of which relate to the project characteristics under discussion here. Moreover, the summary of industrial experience from the information systems domain in [42] suggests that the top factors influencing productivity are, (1) Requirements quality, in particular requirements volatility, (2) Team capabilities and experience, in particular project manager experience and skills, (3) Customer/user involvement, and (4) Method usage.

One may thus argue that in addition to a project's failure to meet the four favorable conditions enjoyed by Project X, projects that have even more developers, more time pressure, even more complex business rules and less control over the fixed-price/fixed-scope dilemma, will also encounter problems akin to the ten problem areas elicited in this study.

5.2 Threats to Validity

The main threats to validity of the study concern the reliability of the elicitation and interpretive processes involved in the repertory grid technique and the subsequent content analyses. To reduce these threats we recorded the interviews for context understanding, we fed analyses back to interviewees, and we conducted dual content analyses.

External validity of case studies is problematic, because the variables under study are case specific and, thus, less likely to be the basis of generalization, see [16] for a discussion on this. Preferably, generalization from case studies should be done through arguments of construct validity. This remains a challenge due to the low number of validated constructs in software engineering. In particular, several of the concepts in Figure 13 are not agreed-upon validated constructs. Thus, our critical case argumentation is, in fact, on the level of variables specific to our study, and is therefore an external validity argument: We posit that our results are valid for variations (to the worse) on these study-specific variables [37]. Hence, our proposals for root causes and the conceptual model in Figure 13 are suggestive only.

6. CONCLUSION

In this study, we uncovered productivity threats in Project X, as perceived by project members. These productivity issues were summarized in ten problem areas, and these problem areas together with the elicited recommendations for action, are now taken into consideration by project management in efforts to improve the planning and running of the remaining releases. For example, during the current release planning, cross cutting communication and coordination is facilitated by the introduction of daily business analysis and release planning stand-ups, as well as weekly summary meetings involving all relevant stakeholders. Further, developer leaders are now instructed to prioritize architectural and technical qualities, the subcontractors are urged to cooperate better, and the developers are instructed to use documentation routines to check and uphold common goals.

The reflective practitioner reflects and learns from practice and builds personal practitioner's theories that, in turn, influences daily practices [21], see Figure 14. Our hope is that our *concept elicitation* has aided in the process of reflection by making projects member's personal theories about large agile development explicit. Our *application* of relevant scientific knowledge on productivity is a further input to building practitioner's theories. Scientific knowledge and theories concern mechanisms that underlie phenomena encountered in daily practice. Understanding such mechanisms enables practitioners to control these phenomena to a larger degree. The findings from studies such as the one presented here

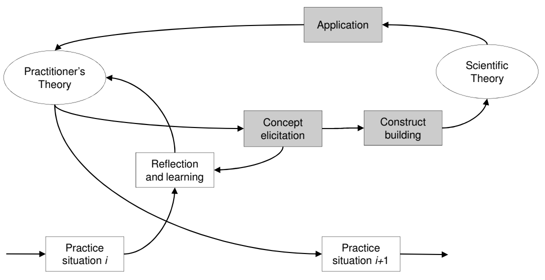


Figure 14: Building practical and scientific knowledge (adapted from [21]). Researchers' contributions in grey.

may contribute to scientific knowledge by *building constructs* through further systematization and evaluation.

This study focused on the elicitation of potential problems and not on their solutions. Preliminary suggestions for improvement were nonetheless given based on the repertory grid analyses. We note that the resulting problem areas are broad in scope. To gain a deeper and more focused understanding of root causes and solutions, future research might therefore target specific activities in which the problem areas arise. We are conducting further investigative studies in Project X which focus on release planning in a large and agile context. Release planning activities seem to intensify aspects of all ten problem areas elicited in this study.

Acknowledgments

The authors are grateful to the interviewees for their insights and cooperation, to the project management of Project X, and to Mika Mäntylä for valuable input. The authors wish to thank the anonymous referees for insightful feedback.

References

[1] C. Argyris. *Knowledge for Action.* Jossey-Bass Publishers, 1993.

[2] C. Argyris and D. A. Schön. *Organizational Learning II. Theory, Method, and Practice.* Addison-Wesley Publishing Company, Inc., 1996.

[3] N. Baddoo and T. Hall. Practitioners roles in software process improvement: An analysis using grid technique. *Software Process Improvement and Practice*, 7:17–31, 2002.

[4] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, second edition, 2003.

[5] B. Boehm and R. Turner. *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison Wesley, 2003.

[6] M. T. H. Chi, N. de Leeuw, M.-H. Chiu, and C. LaVancher. Eliciting self-explanations improves understanding. *Cognitive Science*, 18:439–477, 1994.

[7] T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50:833–859, 2008.

[8] H. Edwards, S. McDonald, and S. M. Young. The repertory grid technique: Its place in empirical software engineering research. *Information and Software Technology*, 51:785–798, 2009.

[9] H. Erdogmus. Let's scale agile up. *Agile Times*, 2(1):6–7, Apr. 2003.

[10] H. Erdogmus. Architecture meets agility. *IEEE Software*, pages 2–4, Sept./Oct. 2009.

[11] K. A. Ericsson and H. A. Simon. *Protocol Analysis.* The MIT Press, revised edition, 1993.

[12] M. Fowler and J. Highsmith. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[13] F. Fransella, R. Bell, and D. Bannister. *A Manual for Repertory Grid Technique.* John Wiley & Sons, Ltd., 2004.

[14] G. Gigerenzer. *Gut Feelings. The Intelligence of the Unconscious.* Viking, Penguin, Ltd., 2007.

[15] R. Guzzo and M. Dickson. Teams in organizations: Recent research on performance and effectiveness. *Ann. Rev. of Psychology*, 47:307–338, 1996.

[16] J. E. Hannay and M. Jørgensen. The role of deliberate artificial design elements in software engineering experiments. *IEEE Trans. Software Eng.*, 34:242–259, Mar/Apr 2008.

[17] J. D. Herbsleb and A. Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *Proc. European Software Engineering Conf. and ACM SIGSOFT Symp. Foundations of Software Engineering*, pages 112–121. ACM Press, 2003.

[18] M. Hoegl and K. P. Parboteeah. Autonomy and teamwork in innovative projects. *Human Resource Management*, 45(1):67–79, 2006.

[19] F. Houdek. External experiments—a workable paradigm for collaboration between industry and academia. In N. Juristo and A. M. Moreno, editors, *Lecture Notes on Empirical Software Engineering*, volume 12, chapter 4, pages 133–166. World Scientific, 2003.

[20] D. Jankowicz. *The Easy Guide to Repertory Grids.* John Wiley & Sons, Ltd., 2004.

[21] P. Jarvis. *The Practitioner-Researcher.* Jossey-Bass Publishers, 1999.

[22] G. A. Kelly. *The Psychology of Personal Constructs.* Norton, 1955.

[23] G. A. Kelly. *A Theory of Personality.* Norton, 1963.

[24] P. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 6:45–50, 1995.

[25] P. Kruchten. Scaling down large projects to meet the agile "sweet spot". *The Rational Edge*, Aug. 2004.

[26] C. Larman and B. Vodde. *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum.* Addison Wesley, 2008.

[27] C. C. Manz and H. P. Sims Jr. *The New Superleadership: Leading Others to Lead Themselves.* Berrett-Koehler, 2001.

[28] L. Mathiassen. Reflective systems development. *Scandinavian J. Information Systems*, 10(1&2):67–118, 1998.

[29] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1956.

[30] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proc. 30th Int'l Conf. Software Engineering (ICSE)*, pages 521–530. ACM Press, 2008.

[31] N. Niu and S. Easterbrook. So, you think you know others' goals? a repertory grid study. *IEEE Software*, pages 53–61, March/April 2007.

[32] C. L. Pearce and H. P. Sims Jr. Vertical versus shared leadership as predictors of the effectiveness of change management teams: An examination of aversive, directive, transactional, transformational, and empowering leader behaviors. *Group Dynamics*, 6:172–197, 2002.

[33] M. Poppendieck and T. Poppendieck. *Implementing Lean Software Development: From Concept to Cash.* Addison-Wesley, 2006.

[34] PS2000 standard contract for iterative development. `www.dataforeningen.no/index.php?cat=134112`, accessed 2010.

[35] H. Rognerud and J. E. Hannay. Challenges in enterprise software integration: An industrial study using repertory grids. In *Proc. 3rd Int'l Symp.Empirical Software Engineering and Measurement (ESEM)*, pages 11–22. IEEE Computer Society, 2009.

[36] R. Schatz and I. Abdelshafi. The agile marathon. In *Proc. AGILE 2006*, pages 139–146. IEEE Computer Society, 2006.

[37] W. R. Shadish, T. D. Cook, and D. T. Campbell. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference.* Houghton Mifflin, 2002.

[38] M. Sliger and S. Broderick. *The Software Project Manager's Bridge to Agility.* Addison Wesley, 2008.

[39] M. Stephens and D. Rosenberg. *Extreme Programming Refactored: The Case Against XP.* APress, 2003.

[40] V. Stewart and A. Stewart. *Business Applications of Repertory Grid.* McGraw-Hill, 1981.

[41] H. Takeuchi and I. Nonaka. The new new product development game. *Harvard Business Review*, pages 137–146, Jan./Feb. 1986.

[42] A. Trendowicz and J. Münch. Factors influencing software development productivity—state-of-the-art and industrial experience. *Advances in Computers*, 17:185–241, 2009.

[43] E. Whitworth and R. Biddle. The social nature of agile teams. In *Proc. AGILE 2007*, pages 26–36. IEEE Computer Society, 2007.

[44] C. Wohlin and A. Amschler Andrews. Prioritizing and assessing software project success factors and project characteristics using subjective data. *Empirical Software Engineering*, 8:285–308, 2003.

[45] R. K. Yin. *Case Study Research: Design and Methods*, volume 5 of *Applied Social Research Methods Series.* Sage Publications, third edition, 2003.