Information and Software Technology 56 (2014) 873–889

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Individual empowerment of agile and non-agile software developers in small teams

Bjørnar Tessem [*]

Dept. of Information Science and Media Studies, University of Bergen, PO Box 7802, 5020 Bergen, Norway

ARTICLE INFO

Article history:
Received 14 February 2013
Received in revised form 2 January 2014
Accepted 22 February 2014
Available online 3 March 2014

Keywords:
Software development
Agile methods
Empowerment

ABSTRACT

Context: Empowerment of employees at work has been known to have a positive impact on job motivation and satisfaction. Software development is a field of knowledge work wherein one should also expect to see these effects, and the idea of empowerment has become particularly visible in agile methodologies, in which proponents emphasise team empowerment and individual control of the work activities as a central concern.
Objective: This research aims to get a better understanding of how empowerment is enabled in software development teams, both agile and non-agile, to identify differences in empowering practices and levels of individual empowerment.
Method: Twenty-five interviews with agile and non-agile developers from Norway and Canada on decision making and empowerment are analysed. The analysis is conducted using a conceptual model with categories for involvement, structural empowerment and psychological empowerment.
Results: Both kinds of development organisations are highly empowered and they are similar in most aspects relating to empowerment. However, there is a distinction in the sense that agile developers have more possibilities to select work tasks and influence the priorities in a development project due to team empowerment. Agile developers seem to put a higher emphasis on the value of information in decision making, and have more prescribed activities to enable low-cost information flow. More power is obtained through the achievement of managing roles for the non-agile developers who show interest and are rich in initiatives.
Conclusion: Agile developers have a higher sense of being able to impact the organisation than non-agile developers and have information channels that is significantly differently from non-agile developers. For non-agile teams, higher empowerment can be obtained by systematically applying low-cost participative decision making practices in the manager–developer relation and among peer developers. For agile teams, it is essential to more rigorously follow the empowering practices already established.

1. Introduction

It is now more than 10 years since the agile movement started to have a significant impact on the software development industry. Since then, agile development approaches have grown from a somewhat obscure status to mainstream, and we have been introduced to the meanings of Extreme Programming [1], Scrum [2] and Lean Development [3]. The proponents of agility suggest a collection of technical and social practices that, when combined, constitute development processes that are distinct from traditional software development, and, supposedly, should ensure higher quality in software at less cost.

The agile manifesto[1] gives us an idea of what agile software development is, by emphasising “Individuals and interactions over processes and tools”, “Working software over comprehensive documentation”, “Customer collaboration over contract negotiation” and “Responding to change over following a plan”. But in research we also need to understand the theoretical underpinnings of the concept in order to be able to analyse it rigidly, and complement the formulations found in the manifesto. Theoretical elaborations include Conboy [4] who defines agility in software development as emerging from a flexible and lean development team. Vidgen and Wang [5] define agility as built on three principles adopted from complex adaptive systems theory: (1) match co-evolutionary change rate, (2) optimise self-organising, and (3) synchronise exploitation and exploration of

* Tel.: +47 55584103.
E-mail address: bjornar.tessem@uib.no

[1] http://www.agilemanifesto.org.

http://dx.doi.org/10.1016/j.infsof.2014.02.005
0950-5849/© 2014 Elsevier B.V. All rights reserved.

knowledge. For Lee and Xia [6], central foundations for agility include the concepts of autonomy and team diversity.

Empowerment in the workplace has a long tradition in practice and research, and there is a substantial literature describing the effects of empowerment in different types of organisations [7–13]. Positive effects on productivity, product quality and innovation, as well as affective reactions like workers' job satisfaction and motivation are found both in blue-collar and knowledge-based lines of business [14], and thus we also expect to see some of these effects in the software development organisation as a result of introducing empowering processes and structures. The empowerment concept is often discussed in terms of structural empowerment, that is, the organisation's approach to empower its workers [15], and psychological empowerment, that is, the individual person's feeling of having control of one's own work [16]. Mathieu et al. [17] assert that psychological empowerment is highly dependent on structural empowerment. Further, psychological empowerment is again an antecedent for performance, job satisfaction and motivation [18], which are some of the effects aimed for and claimed for in agile software development. Hence, empowerment is a concept worth analysing in relation to agile software development. Still, research that studies the effects of empowerment in software development is scarce, even though there is some evidence that agile processes increase job satisfaction and motivation, and that this is caused by agile developers being more empowered than non-agile developers [19]. On the side of this argument, it is worthwhile mentioning here that research has not shown a correlation between job satisfaction and productivity [20], but job satisfaction has the positive effect of less turn-over in the business, and with that less cost on recruitment and training [21]. This alone makes empowerment a goal worthwhile pursuing for any software development organisation.

Much of the professional literature on agile software development focuses on practices that aim to enhance productive social interaction and participation, like stand-up meetings, iteration planning, iteration retrospective, and even pair programming, which all contribute to shaping decision processes and empowering the development teams and the individual developers [1,22–25]. McHugh et al. [26] discuss how such practices have higher motivation as a consequence when enacted in meaningful ways. Even if empowerment is not mentioned explicitly in theoretical studies of agility in software engineering, empowerment as a concept is deeply intertwined with the theoretical concepts that define agility: Team empowerment has been considered a foundation for leanness [27], which is central with Conboy [4]; it is overlapping strongly with the concept of team autonomy and self-organisation, which is considered central by Lee and Xia [6] and Vidgen and Wang [5]; and it is also considered a result of the team's ability to, on its own, effectively explore and exploit new knowledge in the work process [15], which is claimed by Vidgen and Wang [5] to be central for agility. Maruping et al. [28] use a control theory perspective and shows that the successful agile teams have control modes that provide team autonomy in development combined with a strong focus on the project objectives. We see that concepts that contribute to defining agile software development both practically and theoretically, like participation, interaction, autonomy, self-organisation, and knowledge exploration, all have a strong "team empowerment" association to them. Hence, empowering the developer team is deeply melded into the effective realisation of agile software development.

In the professional literature, practices and perspectives in agile development have the aim of increasing the performance of the *team*, often by enabling empowerment of the *individual* developer within the team. Some examples of agile practices that potentially empower the individual are

- *Stand-up meetings:* Developers present their problems, suggest solutions to other developers and to the team, and choose tasks for their own work day.
- *Pair programming:* Developers work together with another developer, influence the partner's work, and learn about solutions in the current project as well as available technologies.
- *Shared code ownership:* Developers have the freedom to change other team mates' solutions, whenever the developer finds it necessary.

Individual, affective effects like higher job satisfaction and motivation are known to be a consequence of individual, psychological empowerment [29]. At the same time we know that empowered teams have a higher level of job satisfaction [30]. This higher level applies at both the individual level and at the team level [31]. Since team empowerment is so central in an agile software development organisation's overall approach, it could with its implementation through the agile practices found in the literature, potentially lead to the effects of individual psychological empowerment, and further to job satisfaction and motivation for the individual. But, whether the agile practices in fact leads to a higher level of individual empowerment than found elsewhere, and if so, how strong it is, is not obvious. Only through empirical studies can we find out what these effects are.

At the same time, even though non-agile software development organisations work according to processes whereby empowering practices are not explicitly emphasised, this does not mean that the non-agile developers are not empowered. For example, Shrednick et al., from before the agile development era, discuss how empowerment improve on information systems development [32]. In many cases, non-agile development organisations have adopted supplementary practices that ensure high levels of participation in decisions regarding their work, as exemplified by many software process improvement initiatives [33]. These are forms of empowerment that may have the same effects as the practices found in agile software development. There is as of yet no reason to claim that non-agile software developers are less empowered than agile.

This study examines how empowerment for the developer is enabled in both agile and non-agile software development organisations, and the aim is to determine whether there is a difference between agile and non-agile development processes in this regard and further, to what extent the individual developer is empowered. As mentioned, agile processes implicitly suggest high team empowerment levels, but how this contributes to how the single developer feels empowered is not clear. Since empowerment is an underlying, maybe even defining facet of agility, we should expect to see some differences.

The starting point for this research is to try to identify the differences in how developers in software development organisations are empowered, then further to discuss why these differences occur, and finally ask ourselves whether we can conclude that there are differences in levels of empowerment, and what consequences these have. To answer these questions, data from 25 semi-structured interviews with agile and non-agile developers from Canada and Norway are used. The interviews are with businesses developing software for the oil industry, finance industry, graphical printing, public information systems and several other areas. The data came mainly from programmers and managers, but also the occasional tester and business representative. The interviews are analysed qualitatively according to a conceptual model that includes factors that influence empowerment, including involvement, power, information flow and sense of impact.

The article first presents some background material on empowerment and how it is studied in general and then continues with an overview of some of the research that has been conducted on this within the software development literature. In Section 3, a more elaborate version of the problem statement is formulated and the research model and the analysis categories used in the qualitative analysis are presented. Sections 4–6 present the collected data and an elaborate analysis of the interviews splitting the developers into agile and non-agile groups, and analysing the data according to categories for forms of involvement (Section 4), structural empowerment (Section 5) and psychological empowerment (Section 6). The findings, showing significant differences as well as similarities, are discussed in Section 7. In Section 8 some limitations are discussed and opportunities for further research are presented, before the article concludes in Section 9.

2. Background literature

2.1. Empowerment at the workplace

Empowering the workforce has been documented to have positive consequences both for the employee and the business [7,15,34]. Empowered employees enjoy their jobs, become more motivated, stay in their jobs and feel ownership of the business's goals. The business gains economically due to less turnover and lower training costs, and may lead to better performance, although this is disputed [20,35].

A definition of *structural empowerment* can be found in Mills and Ungson [11], where they define it as entailing "the delegation of decision-making prerogatives to employees, along with the discretion to act on one's own". Mills and Ungson distinguish empowerment from participation – "a communication process or technique to solicit and use employee feedback in the decision-making process", and delegation of work tasks – "discretionary control for specific activities that fall within officially prescribed limits". Participation and task delegation are viewed as tools that support empowerment.

Lawler [10] emphasises that among the structural factors that enable empowerment is first of all power, but also structures for giving employees access to relevant information for decision making, and enabling them to do their job by providing means to increase the worker's knowledge. Finally, in the successful empowering organisation, a system for rewarding the employees is necessary. These four structural factors (power, information, knowledge and reward) are repeated by for instance Riordan et al. [15], and Bowen and Lawler [7].

Several authors argue that not only structural or organisational empowerment is necessary in order to gain the benefits searched for [12,13]. In particular, they claim that psychological attributes like an individual's perceptions of personal control, proactiveness and understanding of the social context or organisation are essential. They base their views on Thomas and Velthouse [16], who defined the concept of *psychological empowerment* as consisting of the following four cognitions: meaningfulness, competence, self-determination and impact. Meaningfulness is the employee's perception of the value of the work done, competence is the employee's belief in his or her own ability to do a job well, self-determination is the perception of autonomy at work and impact is the employee's perception of being able to influence the workplace. In much of the literature, psychological empowerment is considered an effect of structural empowerment combined with other factors like individual characteristics, work design, leadership and organisational support [17]. Further, it has been shown that psychological empowerment leads to higher performance, job satisfaction, motivation and other outcomes [36]. Fig. 1 shows how a chain of factors from structural empowerment through psychological empowerment enable valuable outcomes for the organisation.

Empowerment in organisations is first of all enabled by structural means. Managers use different approaches to empower their workers, such as quality circles [37], total quality management (TQM) [38] or employee suggestion systems [39]. Hackman [40] focuses on how to enable teams to organise their own work, enabling them to construct their own structures and practices for monitoring, improving and managing the work. Such team self-management models have had a great influence on agile methods [41], and are central in both XP and Scrum. The concept of team empowerment is about how teams are enabled to control their own work [31], and is often analysed from the same perspectives as individual empowerment [18], with structural and psychological empowerment as a chain of factors that influence performance and affective outcomes. In general one expects individuals to be empowered when their team is empowered [42], as team empowerment leads to individual empowerment. Still, all team members may not necessarily be empowered to the same extent. For instance, the more knowledgeable and experienced team member will most often have more to say in decisions regarding how the team shall proceed on a task.

2.2. Empowerment and decision making in software development

Empowerment in software process improvement (SPI) is central in Moe's PhD thesis [43], which is based on a series of articles on SPI in plan-driven development [44], transitions from plan-driven to agile development [45] and team building in agile software processes [46]. A critical view on how empowerment is realised in agile teams is found in work by McAvoy and Butler [47], who among others point to the *group think* problem, exemplified by the Abilene paradox (a group agrees on something nobody wants because everyone thinks the others want it), as a cause of inefficient decision making. The previous authors have an analytical focus on team empowerment and less on the individual developer. In addition, a few articles considering the individual's participation and empowerment are worth mentioning. Melnik and Maurer [19] document that agile developers are more satisfied and motivated at work, and in their study, the developers suggest empowerment as one of the reasons for this. A small case study by this article's author (Tessem et al. [48]) describes how a growing Scrum development organisation with a high degree of empowerment is able to maintain the workers' job satisfaction and motivation. The author [49] has also used Wilkinson et al.'s [50] escalator of organisational participation and analyses interview data from different software organisations to describe how levels of participation vary in different organisation types.

Decision-making processes are related to empowerment, and as for empowerment in software development work, research work on decision making per se in software development organisations is scarce. But many types of activities in software development involve decisions that have significant consequences on the process and the final product. Examples of rather technical practices that have strong elements of decision making are code reviews [51] or pair programming [52], and these techniques have been extensively studied with a focus on productivity and quality [53,54]. White Baker [55] discusses how to construct a situational development process, which is an example of participative decision making at an organisational level.

More focused studies on how software development decisions are made include work by Aurum and Wohlin [56] and Alenljung and Persson [57], who have studied decision making in requirements engineering with different research perspectives, and Zannier et al. [58], who have studied how design decisions are made in software teams. Drury et al. [59] show how good decisions in agile teams are hindered by the way the software development organisation implements the development process. They list six obstacles to decision making that are strongly related to features of how decisions are made and implemented:

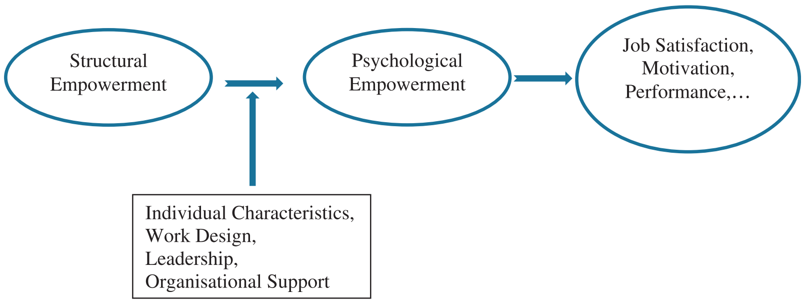
Structural Empowerment
Psychological Empowerment
Job Satisfaction, Motivation, Performance,…
Individual Characteristics, Work Design, Leadership, Organisational Support

Fig. 1. The chain of effects from structural empowerment to organisational outcomes.

1. Agile team members are unwilling to commit to a decision and rely on the Scrum master for decisions.
2. Agile teams face conflicting priorities for decisions.
3. Decisions are based on unstable staff availability during an agile iteration.
4. Agile team members are not implementing decisions and are relying on others for decisions.
5. Agile team members are not taking ownership of decisions despite ASD team autonomy.
6. Collaborative group decision making on ASD teams prevents experts from making decisions.

The obstacles are exemplified in six case studies. All these obstacles relate strongly to empowerment, and add to the motivation for studying how empowerment of software team members is enacted.

3. Problem statement and analytical framework

As described in the professional literature, agile software development is very focused on simplicity in process and communication, allowing teams and individuals to make central decisions, and allowing the team and the individual to quickly change directions in work [1–3,22]. These are all features that contribute to empowerment, even though empowerment as such is not a much-mentioned concept in this literature. An exception is the literature on lean programming [3], which puts an emphasis on *team empowerment*. The connection to individual empowerment is that the practices that enable team empowerment in agile software development are components of the structural empowerment within agile software organisations, further contributing to individual empowerment, both structurally and psychologically.

As for plan-driven (or non-agile) software development methods, they do to a slight extent focus on social practices. The research mentioned in the Section 2 suggests that there are some differences in how agile and non-agile developers are empowered, but there is still not a deep understanding of which structural and psychological factors create a difference in how software developers influence their own work in agile and non-agile teams.

On this background it should be worthwhile to investigate what agile and non-agile processes imply in terms of individual empowerment. This means that we first of all have to gain insights in how empowerment is reached in different software development organisations through structural means, then investigate whether there are any differences at a deeper level, analysing the structural and psychological factors of empowerment, and finally whether these differences lead to a difference on how individual empowerment is realised and perceived.

In this article, answers to these questions can be found by analysis of interview data with regard to the factors of the two types of empowerment. Structural empowerment is supported by the following factors from the literature:

- *Power:* The ability of employees to have an influence on decisions regarding issues at the workplace.
- *Information:* The ability of an employee to get access to information needed for the best decision making in work tasks.
- *Knowledge:* The ability of an employee to obtain the necessary skills to do the work and make good decisions.
- *Reward:* The way an employee is credited for participating in decision making, information sharing and training.

Psychological empowerment is supported by the following factors from the literature:

- *Meaningfulness:* The employee's personal assessment of the value of a work goal or purpose.
- *Competence:* The employee's personal assessment of one's own ability to perform the job skilfully.
- *Self-determination:* The employee's perception of one's own autonomy in initiation and continuation of work activities.
- *Impact:* The employee's assessment of one's own ability to influence strategic, administrative and operational outcomes at work.

The power factor is of particular importance, as it defines roles, responsibilities and possibilities for involvement for the individual software developer. To get a richer understanding of the power factors, categories of (forms of) involvement are added to the conceptual model. The power factor of Lawler [10] is closely related to Mills and Ungson's [11] participation and delegation concepts, which are two forms of involvement in the organisation's decision processes. We should consider the decision-making activity also to include taking the steps to initiate a decision-making process, and the workforce's ability to initiate processes thus constitutes a third form of involvement in decision processes. *Initiation* as a form of involvement can be viewed as distinct from the empowerment concept, on a level with participation in decision making and delegation of decision making power in limited areas. Initiation was used as a concept by the author [49], who expanded Wilkinson et al.'s [50] participation escalator with the organisation's acceptance of initiatives for decision making by the employees, as the data showed that this occurs frequently in software teams. In contrast to that work, in this article initiation is not found to be a special form of participation, but a particular form of involvement. Involvement as used in this work thus takes three different forms:

- *Participation:* An employee is asked for information or opinions on a matter of decision, and/or is explicitly involved in making the decision.
- *Delegation:* An employee is assigned to a particular task and has responsibility making the decisions needed to fulfil that task.
- *Initiation:* An employee initiates activities with the aim of having the organisation make a decision on a matter of relevance to the employee's work.

The data are analysed by finding quotations in the interviews that relate to the 11 bulleted factors above. Fig. 2 shows the factors/categories used in the data analysis and how they are assumed to contribute to empowerment. The arrow from psychological empowerment to affective and economic reactions is not studied here, as this study is limited to emphasise on how empowerment is enabled and not its effects.

The choice of a qualitative approach in this study was motivated in the search for answers to questions on how empowerment is enabled and its varying realisations, and to a little extent to quantitative comparisons of empowerment and its consequences. A grounded theory study [60,61] could also be considered here to get a theoretical understanding of the relationship between software development practice and empowerment, but as the empowerment concept already has been well studied, the categories identified in earlier empowerment studies were considered to be the relevant ones for this study.

The data used for doing this analysis are only the 25 semi-structured interviews with software developers. As the author was visiting the companies for interviews at their localities, this was and opportunity to get an impression of their work site, even though this has only been used as informal background knowledge in the analysis. It was also asked for documentation of their development processes and practices. Only two of them had this and/or were willing to share this, so these documents did not influence the data analysis much. One could have added observations at the sites in the data collection. This was omitted, as it was prioritised to get data from many diverse organisations, and such observation studies would take much time.

The topics of the interviews were not on the 11 categories themselves, but on decision making, ability to participate in decision making and ability to act on those decisions, all within the software development processes. The empowering practices touched upon as topics in the interviews were chosen on the basis of what normally goes on in a software team, identified on basis of the author's own practice, research, and teaching in software engineering. The topics are given in the left column of Table 1. The idea was that involvement or control in the practices covered in the interview would, through the data analysis, reveal how the developers were empowered in what is standard software development work. In the right column of Table 1 we find the categories (out of the 11 previously identified) that the discussion of a particular topic potentially could give data for.

Table 1
Empowering practices discussed in interviews.

Empowering practice discussed	Potential categories
Control of low-level design/ coding	Power, self-determination
Ability to choose own work tasks	Power, self-determination, meaningfulness, participation, delegation
Initiate trials of new technology	Knowledge, competence, initiation
Participate in estimation	Power, competence, impact, participation
Participate in high-level design	Power, competence, impact, participation
Participate in process improvement	Power, competence, impact, self-determination, participation
Get information and domain knowledge from customer	Information, knowledge, competence, initiation
Get technical knowledge in order to solve task	Knowledge, competence, delegation, initiation
Collaborate with peers	Information, knowledge, meaningfulness, initiation, participation
Team and individual rewards	Reward, impact, meaningfulness

Interviewees were from two different countries, Canada (14) and Norway (11), where a total of 12 worked in or with agile teams, and 13 in non-agile teams. All team could be characterized as small, as they at most consisted of 8–10 persons. Most of the developers were programmers, but there were also testers (2), customer representatives (1), project managers (2), software architects (1) and requirements specialists (1). For those working in non-agile teams, many of the programmers also had technical project management responsibilities to some extent. All the agile developers worked with variations of Scrum, whereas in the non-agile teams there were one team (5 developers) using Rational Unified Process (RUP), another team (3 developers) using an in-house-developed, plan-driven, iterative method and also developers (5) who worked in teams with no specified development methods.

The interviewees in Norway were found by contacting varying software development companies in Norway, asking for access to interviewees. To some extent, the author's contact network was used to identify the companies. In Canada, respondents were self-recruited. A request for participants was emailed to members of a professional organisation within software development. All the

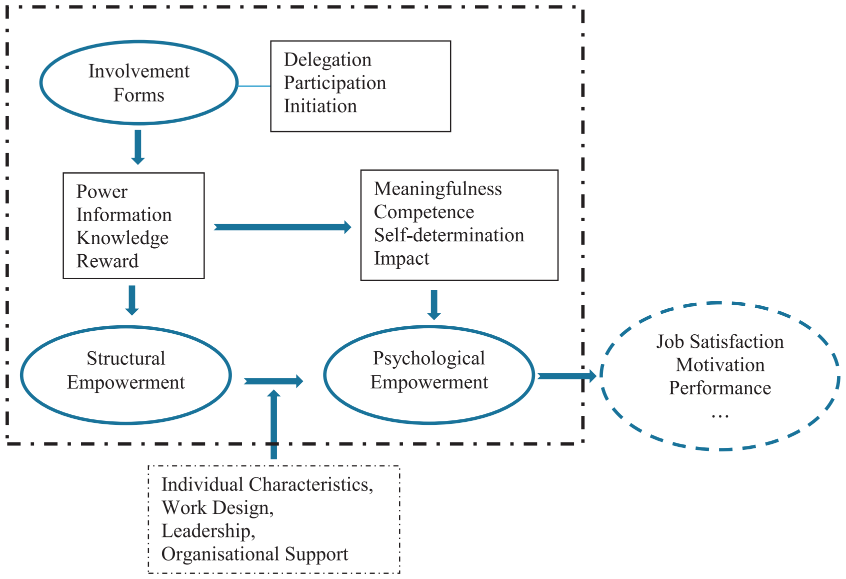

Fig. 2. Empowerment and its factors and outcomes in software development organisations. The boxes and ellipses included in the larger, dotted rectangle are concepts and factors included in the analysis.

Table 2
Overview of interviewees.

	Norway	Canada
Agile	Scrum team – finance industry (2 interviewees)	Scrum team – oil industry (5) Scrum team – graphical/marketing services (3) Scrum consultant – large consultant company, works as project manager (1) Scrum consultant – own company, works as project manager (1)
Non-agile	RUP team – telecommunications (5) Team w/no methodology – public administration (2) Team w/no methodology – public legal systems (1) Team w/no methodology – climate control for buildings (1)	Team w/structured iterative method – accounting services (3) Team w/no methodology – airport administration system (1)

Canadian interviewees were developers who responded to this invitation. Recruitment of respondents in qualitative research is a challenge as it in the use of contact networks and channels like professional organisations may lead to a bias in the responses. However, by recruiting from two different countries, although with similar cultures, a better balance could be achieved. The respondents also had varying age, experience, and industrial background, contributing to a representative respondent group.

It must be mentioned that the interviewees came from teams that were not necessarily working with software development according to ideal standards, and, accordingly, they do not represent examples of prescribed practice. Still, the research questions are about how empowerment is enabled in practice, and, in an imperfect world, the near-ideal organisations are not necessarily the best to get data from. Of course, they also do not cover the full spectrum of software development organisations, but give us a width that covers much of what is business practice in agile and non-agile teams. Table 2 summarises the distribution of the 25 interviewees according to origin, being agile/non-agile and their domains.

The analysis was done in a qualitative manner, coding text elements (quotes) in all interviews with the 11 categories/factors. Commonalities and variations in how the quotes relate to the particular category were noted, and further used to identify differences between agile and non-agile organisations. To improve quality in the coding, two individual coding processes were done, with a couple of months pause between them. The results were integrated into a final coding of the interview texts.

The next three sections will discuss how the data fit into each of the eleven categories. The analysis starts with involvement forms, because these can be said to be a part of or tools for structural empowerment and feeds into the understanding of structural empowerment. Then analysis of the structural and psychological empowerment categories follows.

4. Analysis – forms of involvement

In the conceptual model, the three categories for forms of involvement are closely related to power, and will contribute to the analysis of that particular factor. However, forms of involvement are important in their own right, as each is also an underlying contributor to the other structural empowerment factors. This section is divided into three subsections, which cover each of the involvement categories and their realisation in agile and non-agile teams, before the analysis is summarised in a table. It must be emphasised that the focus in the analysis is on how programmers and testers are involved, not managers and business representatives.

4.1. Participation

In this category, there is data that refer to situations where developers participate in activities together with peers and other stakeholder groups including managers, quality assurance people, customers, etc.

The agile development methodologies have built-in practices that open up for high degrees of participation in the development work. The interview data include references to planning meetings (scrums), estimating procedures like planning poker, retrospectives, pair programming and several other practices that are part of decision making in the daily operational work. A Scrum developer from Canada, working in-house with continuous development of the software of a company selling graphical/marketing services said:

> "My involvement is to be involved in the Scrum planning meetings, evaluating the back log and then doing the poker planning, and be part of the actual Scrum development team in implementing the tasks and ultimately satisfying the stories". (Master in software engineering, 10 years of industrial practice, 3 years with agile development)

An experienced agile developer, having been part of the agile movement from early on, explained how participation from the engaged developer ensures good decisions. Notice that the result depends on a good team structure, including respect among team members:

> "If you are engaged and respected, you know, you will participate and you will argue for what you believe in and then if you have a good team structure you know you are going to make a team decision and go on it". (Scrum project manager from Canada, provide services as project manager to organisations that use agile methods, masters in computer science and business administration, 20 years in industry, 10+ years with agile development)

The non-agile developer also participates in operational decisions about *other* developers' work as well as bringing in others in their decision making, but mostly these do not happen without an invitation to do a design or code review, or when helping with some difficult coding. In most cases, participation at this level is considered a knowledge practice done when a developer is stuck or needs to understand a topic better before starting a larger programming effort.

Developers do participate in the managers' decisions by invitation, for instance when the manager needs input to an estimation job, a technological choice or a process improvement effort. Many of the interviewees in fact expressed scepticism towards too much participation, as they felt they could lose focus and become less productive. To be delegated a task is a preferred model. A Norwegian developer working in two projects, one larger and one small that he himself had responsibility for, said this:

> "My nearest manager, usually he gets a load of things to do, and pass them out, and include us a little in what we should do, what each developer should do, and what time it might fit. But the planning and the execution of the project, we are not much involved in that. I think that is enough for my part, since

what I enjoy most is to develop". (Non-agile developer from Norway, company that provides a mix of off-the-shelf and tailored software for the public domain, bachelor in software engineering, 7 years in industry, 3 years in current company)

Although agile teams engage in explicit, participative practices, many of these seem to die out as projects approach their end, and team leads take over more of the external communication, and suggest work distribution in the team. A Canadian developer who had been on a Scrum project in the petroleum production domain for a year experienced this about retrospectives six months before the final delivery of the project:

"...we usually have the whole group, like all or everybody get together and talk about what works, what doesn't work. Now we do it in a little group, or we don't do it anymore, I think". (Agile developer from Canada, in Scrum-of-scrums project making tailored software for oil industry, master in computer science, 8 years in industry, 1 year with agile, 1 year in current company)

For both type of teams, developers mentioned the importance of trustable teams, ability to agree on solutions and personal matches for successful participative practices. Several of the interviewees told stories about people they had problems cooperating with, which had strongly influenced their willingness to work on a task.

4.2. Delegation

In this category, there is data that refer to situations where managers or presumably more powerful people delegate work to one or several team members. The really main difference found in the data here is that the agile team is the unit that is delegated work in agile methods, whereas in non-agile methods, managers always direct work to individuals. This also implies that coordination of work tasks is handled by the manager in non-agile teams. In a plan-driven team in Canada, one of the developers said:

"And they will basically hand out a feature to me or to one of the other developers, but very vaguely, so they will for example say, we need you to write a feature for login or something like that". (Non-agile programmer from Canada, tailored software for accounting systems using own in-house development method, master in software engineering, 3 years in industry, mainly in the current company)

This is in contrast to the agile team who instead of a single task at the time gets a bunch of things to do, and have to organise themselves in order to finish those:

"So then we basically try to collect a number of stories to complete in that month that matches the amount of time we have, in that month, based on our estimates. And we do everything we can to finish them". (Agile programmer from Canada, in Scrum-of-scrums project making tailored software for oil industry, master in computer science, 7 years in industry, 2 years in current company)

Notice that the agile programmer in the quote above talks about we, i.e. his team, implying that he mainly sees the team as the unit that has autonomy. The team delegation in agile development in the data nearly always applies to the development of new functionality, and in one case when a team was delegated the task of developing its own way to do agile development. The agile developer would normally choose a new functionality development task from those that are remaining on a backlog. However, they may also be assigned to more trivial tasks like small bug fixes or even needed substantial redesigns (in agile, called refactoring). In many such circumstances, the agile project manager or others with power will choose to assign work to the persons found most suited, thus breaking the team empowerment philosophy. In one agile team a test responsible did the assignment of bug fixes directly, not the team itself, but to the individual developers that was seen fit. Actually this was not part of the tester's responsibilities, but was a practice that the tester had adopted:

"I tend to do that, and I tend to go through them, look at it and I say "Well, these guys were just working on this. Makes more sense for them to fix this than somebody that hasn't been in that code to look at it". And that's the way I try to keep track of who is looking at what and who is doing what. So if something comes up that looks related I can take it to them". (Agile team member from Canada, mainly doing testing, bachelor in accounting, 5 years in software industry, 2–3 years in current company)

In both types of organisations, it is found that particular higher-level tasks, like gathering and organising knowledge about a new technology that is of potential use, are delegated to individuals.

4.3. Initiation

In this category there is data that refers to situations where programmers are initiating decision processes or work activities that include other people. For example, a Norwegian Scrum developer started to estimate bug fixes, and added these estimates into the Scrum backlogs. This had not been done before, and everybody was happy because it gave them better project control:

"I started to do it. I had already started to write it down, but then thought I had to ask. So I contacted the project manager and asked. But then it was just positive answers right away". (Scrum developer from Norway, provide tailored software for banking, master in computer science, 2 years in industry, half year with agile methods in current company)

Another example where the developer had to convince a manager was a Norwegian developer in a small development organisation without a defined development method:

"The project manager did after all realise that it was right to use versioning [...] But to make a strict system for it, I had to insist, like other times when it was necessary. But they actually did realise that this was something that was definitely sensible" (Norwegian developer in in-hose software team for a company that does building climate control systems, bachelor in computer science, 20 years in industry, 15 in current company)

As expected, employees who come up with creative and useful ideas are welcome in both types of development organisations. Initiatives are welcome, accepted and handled well in both agile and non-agile teams. There are some distinctions regarding what kind of initiatives developers do, for instance in a non-agile organisation, a developer will often be in a situation where a need exists to initiate a code review including other developers (a low-level decision-making activity), whereas developers in agile teams have other means of verification of code and designs. Another example is that the agile team members may sometimes have better access to customer representatives, and the data show that they take initiatives directly towards those. In a non-agile organisation, this could only happen through a mediating manager. A distinction is that agile team members often will test and improve their initiatives among their team members before they (or the team) bring it forth to managers, whereas in non-agile teams, the initiative is almost always directed towards managers. Some non-agile developers in fact said it was felt problematic to bring colleagues into a discussion, as they felt it was a waste of time:

Table 3
Summary of involvement patterns in software development organisations.

	Agile	Non-agile
Participation	• Many team practices support participation • Practices tend to die out towards project ending	• To provide information and knowledge to peers • To support manager's decision making • Avoid unnecessary participation
Delegation	• Development of new functionality is assigned to the team • Other tasks are often delegated to individuals	• Delegation always to individuals
Initiation	• Is appreciated • Handled in teams before going to managers	• Is appreciated • Directly towards managers

"If you try to bring others into it, if you are trying to resolve it with collaboration, then it usually ends up with, you know, big waste of time" (Canadian non-agile developer, small team working with airport administration system, master in computer science, one year in industry, in current company only)

Table 3 summarises the findings in the analysis of involvement practices. There is a clear distinction between agile and non-agile here. It is evident that the team as an empowered unit stands strong among the agile developers, as delegations and much of the individuals' initiatives and participation is directed to and through the team. For the individual developer, this may be felt as inhibiting in some cases, when the team disagrees, but also strengthening when the team supports. The team does not have such a role in non-agile development, which has the consequences that developers will take individual initiatives towards managers, and the individual's participation in decision making is on the courtesy of the manager. It is common in agile development methods that developers choose their tasks from a backlog or other collection of remaining tasks. On the other hand, agile developers also do not choose their own tasks in large portions of their work. The distinction here is not large, as much of software development, perhaps most of it, is about bug fixes, testing and redesigning the solutions.

5. Analysis – structural factors of empowerment

The structural factors that influence the ability to achieve empowerment are power, information, knowledge and reward. These factors will be addressed in turn.

5.1. Power

In non-agile teams, project managers, team leads or people in similar types of managing positions have the power to instruct each of the team members regarding work tasks and priorities. In agile organisations, this is essentially done not to the single developer, but to the team as seen in the previous section. The team is responsible for estimates and distribution of work within the team. Developers may participate in such activities (estimates, task assignment) also in non-agile teams, but managers have formal responsibility for making the decisions. They often request input for their decisions from developers, but not necessarily, and often take those decisions themselves:

"Pretty much, our planning team, which is like three guys, which I'm not a part of would plan the release at a private meeting, and they would basically, have people like me assigned to the task when they are planning the release. And they would assign during the first developer meeting during the month. Yeah, basically top down". (Non-agile programmer from Canada, tailored software for accounting systems using own in-house development method, master in software engineering, 3 years in industry, mainly in the current company)

Coding, testing, bug fixing and lower-level designs are normally considered work that the developers themselves can take care of in both kinds of organisations. It also happens in agile teams that managers suggest people to do particular tasks or even suggest design solutions, but it is still considered to be the team's formal responsibility to make the final decisions. Managers may also try to steer the activities in general by introducing issues in the discussion, like this experienced agile project manager, who uses his experience to make the team focus on the right things:

"I'll ask for how do we want to tackle this, incorporate people's ideas. If I feel that we are missing something I may then say 'Well how about this', or 'Do you want to do that', so I don't think I'm dictating the approach, but I'm just trying to get resolution on, you know, let's get this thing done and what are people's ideas on how we should do it". (Agile project manager from Canada, contracting as project manager for agile projects, bachelor in computer science, 20+ years in industry, 12 years with agile methods)

There are indications in the data that developers in some agile teams may have more possibilities to influence the customer or product owner (a Scrum role). In a non-agile team, contact between developers and business would go through project managers, and managers would be able to control the communication between those stakeholders. It is the manager's decision to assess the need for direct contact. In most of the agile companies represented in the data, we see the same, as for instance Scrum masters work as mediators. But agile developers were in one of the cases able to use direct channels to business representatives to provide opinions or get information about some issue related to the software under construction.

In non-agile teams management is enacted somewhat differently from agile teams where developers may have direct channels to the business representatives, and also have institutionalised participation in decision making. Non-agile managers, with their position as mediators and responsibility as decision makers, have significant power in deciding what the software should become and how it should be developed. One developer working on a small non-agile team making an airport administration system said:

"Usually customers, if they want something, they won't ask me, they would talk to the customer satisfaction manager in our company, and he's a person who has a lot of knowledge about, because he used to work in the customer environment, so he has knowledge about what customer actually is looking for. So usually requirements come through him..." (Canadian non-agile developer, small team working with airport administration system, master in computer science, one year in industry, in current company only)

In non-agile organisations, we see that developers with experience and initiative gradually will drift into roles where they take manager responsibilities according to the particular organisation's model. For instance, a developer will get responsibility for customer contact, estimation and work distribution in a smaller project:

"I probably am the person in line to take on a little bit of extra responsibility when we have people who are away or in participating in planning activities as well. Part of it is also probably showing interest to matters and I probably show interest by just discussing it informally, asking questions about what's coming in the next release, things like that. Our management, our VP, our president even, they probably know who is interested and who is not". (Non-agile programmer from Canada, tailored software for accounting systems using own in-house development method, master in software engineering, 3 years in industry, mainly in the current company)

The data do not show the same role evolution for the individuals in agile teams, but the experienced developers seem to be more active and have more credibility in the decision-making processes inside teams. An example where role change happened in an agile team is found in one case with many developers and large, complex software. They were using a Scrum-of-scrums approach (an organisation where teams of Scrum masters have meetings to coordinate their respective teams' work), and here experienced agile developers were put into a separate architecture team that had responsibility for overall software architecture and large refactorings, contributing to giving these developers more power in the development process.

5.2. Information

In both agile and non-agile teams, one finds a variety of information channels from personal communication via meetings to documentation. Taking a look at those practices that are present in our data and part of a prescribed methodology, it is found that they differ significantly. Agile developers get information about technical issues through pair programming, get daily updates on project status and have meetings at the beginning and end of iterations. In non-agile projects, developers use different forms of written documentation, have fewer meetings on project status and get updates on technical solutions in informal discussions and ad hoc code and design review meetings. Both kinds of projects have access to business representatives, but in non-agile projects almost only through project managers/team leads.

There is a clear distinction in how people perceive the value of information. In non-agile teams, developers often find that they get insufficient information, but on the other hand they are concerned that they will get too much information (information overload). The developers think that there is a danger that irrelevant information will take too much time. We also see that people tend to be careful with sharing information, perhaps because of the implied cost of distributing what one knows to one's colleagues:

"We talk less with the customer now than we did before. Now we are kind of more screened, we are supposed to focus more on development". (Non-agile developer from Norway, company that provide a mix of off-the-shelf and tailored software for the public domain, bachelor in software engineering, 7 years in industry, 3 years in current company)

Non-agile developers also mention that they are often sloppy with updating technical documentation, an important source for future information needs. Some agile developers also mentioned the amount of information as a problem, but on the other hand the low-cost information channels realised in such teams seem to allow information to flow more freely. One developer was satisfied with the possibility to get information from his colleagues in an informal manner:

"It is in a way a little bit like you ask very much, and then you think, wow, it's perhaps a bit too much, but it has never been a problem, and it really must be so. If you start making assumptions that things surely are like that, then you risk spending even more time on it". (Agile developer from Norway, software for bank industry, master in software engineering, four years in current company)

The agile developers are supposed to have contact with business through business representatives who have a role as information and knowledge providers to the developers. The close customer contact is partly confirmed in the data, as seen in a story about how an algorithmic problem was solved in a cooperative process involving both customer representatives and programmers. In some agile teams, information can flow rather directly between business and developers and vice versa. One of the results is that business has higher trust in the development team. One interviewee reported that the business representative, when initially starting out with agile did not trust estimates and put little emphasis on viewpoints coming from the developers, but after they had worked agile for a while, this changed, and the developers' views got to be considered important in discussions on project progress:

"As soon as they started noting that ours were pretty accurate then it became, they started believing it again" (Canadian agile project manager, contract in different industries, master in computer science, 20+ years in industry, 10 years with agile)

On the other hand, there are examples in the data about how other stakeholders like QA people do not feel comfortable with the agile way of distributing information, and also about developers who do not always get the useful contact they seek with the customer representatives.

5.3. Knowledge

Under this category is discussed the need for both technological and domain knowledge, i.e. knowledge about the business area. It is interesting to see how similarly the teams handle knowledge update on technologies. For both agile and non-agile teams, the most effective means of knowledge dissemination in the development organisation is collaboration on a task like design, coding or assessment of new technology. Non-agile developers do not work together much on coding directly, but use code and/or design reviews to gain knowledge on design techniques or new technologies. Code reviews are also used as a tool to learn newcomers about the local development practice:

"For our newer developers, we basically require them to do code reviews every week" (Non-agile programmer from Canada, tailored software for accounting systems using own in-house development method, master in software engineering, 3 years in industry, mainly in the current company)

This is in contrast to agile developers who use pair programming, i.e. collaborative coding, as a knowledge dissemination approach.

Employees are given much freedom to initiate activities where they find relevant knowledge for their project, and they may often be asked to review one or a few technologies for potential use, and report on that. This developer found out about a new xml parser and informed the colleagues about it:

"Without being asked, you go about and suggest using this one. The result is that this xml parser has caught on around here. Because everyone thinks it was much better than this Java xml parser". (Non-agile developer from Norway, tailored software for telecommunication, master in computer science, 5 years in industry, mainly in current company)

Table 4
Summary of structural factors of empowerment in software development organisations.

	Agile	Non-agile
Power	• Operational tasks like coding, bug fixes, testing, and low level design • Team estimates, distributes work, and designs • Managers facilitate activities • Managers have some control of information flow to/from business	• Operational tasks • Managers instruct on priorities, estimates, tasks, and even designs • Managers control information flow to/from business • Managers occasionally involve developers
Information	• Information channels are low cost: short meetings, pair programming, and personal communication • Some direct access to customer representatives	• Information channels include larger meetings, documentation, and personal communication • Efforts regarding information exchange have low priority
Knowledge	• Collaboration as a means of transferring knowledge • Freedom and resources to get knowledge • Some direct access to domain experts	• Collaboration as a means of transferring knowledge • Freedom and resources to get knowledge
Reward	• Perks at successful deliveries • Recognition and credibility in team	• Perks • Freedom to decide own work hours • More power to individuals with initiative

Access to business representatives to get domain knowledge is found for some of the organisations in the agile data, but it is not always easy to get access to the domain experts for agile developers. Customer representatives have other priorities and are continuously busy with other activities than software development.

5.4. Reward

Data are not as rich on this category, particularly for the agile developers. They mention perks like free lunches at the end of iterations as an example of reward. They also seem to enjoy working in agile teams and consider that a reward in itself, and they are also satisfied whenever they get recognition from doing some good work. As one of the Canadian agile developers said:

> "I think the work, the environment that we have here is quite interesting. I think I'm very happy to be here. I'm honoured to be in this project. I think it is a big challenge for me. I don't think there is a lot of companies out there to do something like this". (Agile developer from Canada, in Scrum-of-scrums project making tailored software for oil industry, master in computer science, 8 years in industry, 1 year with agile, 1 year in current company)

In non-agile teams, you see that trusted developers have similar access to perks, flexible hours and freedom at work within the limits of the delegated tasks. The distinction found in the data is that in non-agile organisations, initiative and willingness to take on challenging work will later on lead to more power and responsibility. Developers who do good jobs get more customer contact, and start taking leadership responsibilities in smaller projects. However, some of the interviewees mentioned that not all developers were willing to take on new responsibilities; they remained in pure developer positions, as they enjoyed that kind of work much more.

Table 4 summarises the data on structural factors of empowerment. One clear distinction is found on how power is distributed, where the agile developer through the team practices is able to participate in estimates, task assignment and overall designs. Information is less valued among non-agile developers, as illustrated by the lack of interest in maintaining the documentation. The data suggest that there is a difference in the reward structures in the sense that non-agile developers are more personally rewarded, whereas knowledge management is handled in a surprisingly similar manner in both kinds of organisations.

6. Analysis – psychological factors of empowerment

From empowerment research it is known that psychological empowerment is a consequence of both structural empowerment and other factors like individual characteristics, work design, leadership and organisational support. When looking for psychological empowerment and relations to structural empowerment, it is hard to control for all these factors. So, the approach is to look for trends regarding psychological empowerment in the data that are typical for the agile or non-agile organisation, and relations to practices that are parts of structural empowerment.

6.1. Meaningfulness

Developers in all teams like to work with challenging new tasks, and enjoy variation. If they experience that, they are also willing to accept tasks that are tedious or uninteresting. Still, there is a tendency that developers in non-agile teams get tasks that they feel are irrelevant for their job. Some of the developers expressed frustration with that. One developer felt that he was burdened with filling in information in the database, which was not really what a developer should do:

> "We maintain aircraft information. If a company has one additional aircraft to look after then we have to enter that information into the system. To me that is not programming, you just enter information about stuff" (Canadian non-agile developer, small team working with airport administration system, master in computer science, one year in industry, in current company only)

The communication practices among developers and from developers towards customers found in agile are considered very useful for work; developers get useful knowledge of technology and domain and feel ownership of the product. A Norwegian agile developer said:

> "There is very good collaboration in the group. And every time I ask I get help, and I think that has been a positive experience. And since I'm rather new, people are not negative when you ask about things that maybe is a bit obvious for them". (Scrum developer from Norway, provide tailored software for banking, master in computer science, 2 years in industry, half year with agile methods in current company)

Developers in non-agile teams also consider communication practices to be valuable, like solving a coding problem with a colleague, having some customer contact or understanding the domain:

> "It's always a lot to learn from people you are working with, and that's probably the best thing about it. If you're working with somebody who is more experienced that you, you can get a lot from them, and if you're working with somebody who is less experienced than you, you can still learn a lot of things". (Non-agile programmer from Canada, tailored software for accounting systems using own in-house development method, master

in software engineering, 3 years in industry, mainly in the current company)

Some non-agile developers complain about a lack of domain knowledge and possibility for being updated on technological knowledge. But they also accept the argument that too much information and communication is expensive, and they support the model that a developer should not necessarily have a complete overview of the whole process. It is interesting to note that non-agile developers find some practices to be meaningful; practices that are typically criticised by agile advocates, like documentation work, customer contact through managers, task distribution by team leads and specialisation. One non-agile programmer in a Norwegian RUP-team said:

"We have not been so good at ensuring that all design is documented. It is a very big system, and things change a lot. We have been clever to document, decisions, meetings, such things". (Norwegian non-agile programmer, company selling telecommunications software, master in computer science, 7 years in industry, 7 years in current company)

Overall, agile developers find decision-making practices in their teams to be meaningful, but here the data show that they also complain about long, meaningless meetings and unnecessary, inefficient communication with customer representatives, so the difference with non-agile development is not that large. Some newcomers in agile teams may feel redundant, having a rather passive role looking at other developers' programming or continuously having to query busy and experienced people, and for some it is a challenge to adapt to agile methods. On the other hand, once they have worked agile for a while, agile converts do not want to return to non-agile practices. According to one agile project manager who has been in the industry promoting agile for many years:

"Of the thirty companies I have worked with, none of them have gone back to the way they used to work". (Agile project manager from Canada, contracting as project manager for agile projects, bachelor in computer science, 20+ years in industry, 12 years with agile methods)

One thing they have in common is the continuous search for improvements of their current software frameworks and tools, and in both types of organisations, there seems to be a continuous process of evaluating, discussing and introducing new technology.

6.2. Competence

All the developers seem to be satisfied with their own competence, and do feel that they are able to do their job and participate in decision-making processes. Possible exceptions in our data are found for one agile and one non-agile developer. The agile developer did not seem to adapt well to the agile work practices, and did not feel well included in the team. The non-agile developer had shifted technological specialisation several times, and felt that he was not supported in his desire to specialise in a particular area. He had discussed this with management and said:

"I do not want to continue reading new and different books all the time. It is very stressful, I feel. So I would like to specialise more within one area" (Norwegian non-agile developer, in team with undocumented method, bachelor in computer science, 20 years in industry, all the time with current company)

The general observation is that competence increases with experience. Collaboration with your peers is a central means to become better at your job, and with the increased competence you also improve people's ability to participate in decision making, irrespective of development methodology.

6.3. Self-determination

Agile developers all report satisfaction with their ability to work in their own manner; they select their tasks from a list, and accept that they have to select the more boring tasks once in a while. Still, we see that agile developers have to do bug fixes, refactorings or handle immediate maintenance tasks under, if not direct orders, considerable pressure to do the job:

"One minute I might be looking at this, and then also there is a problem in production, and when there is a problem in the production those get high priority. And some of them we need to look at right away. So I may be pulled off in a day, or there may be a problem with the billing systems, within another project". (Scrum developer from Canada, continuous in-house development of software for graphical/marketing services, master in software engineering, 10 years of industrial practice, 3 years with agile development)

Most often, the non-agile developers do not choose their own tasks, but are given a mix of interesting and not-so-interesting tasks. Even though they do not have the responsibility to select themselves, they are often asked for their opinions, and often heard. There is a distinction here between agile and non-agile, but not as large as you would expect if you only read the professional literature on agile methods. A non-agile developer said about his possibility to choose:

"The project manager may have a suggestion about what we should do, but we do the discussion in the project meetings. We may say that this fits better with that person, and this person could do that because he knows the code, and we move stuff around. Sometimes you end up with tasks you don't think is very exciting, but in general I think most of us are quite open regarding tasks. If you show some interest for an area, it is natural that you do it. So really, with the project managers suggestions as a starting point, we decide ourselves what to do". (Norwegian non-agile developer, RUP project, five years in industry in same company all the time)

In both kinds of teams, people with specialities seem to be more able to steer their own work. This again has a cost, as other developers often will have to wait for specialists to finish work with a higher priority, which again leads to a sense of lack of control over one's own work.

Solutions are most often at the developers' discretion, but they have to adapt them to team practices. In both kinds of organisations, there are conflicts between the design preferences of different persons, and that one sometimes has to accept other developers' solutions. This may not and should not be a problem for most, but a couple of developers expressed frustration with the process that led up to the design conflict they had experienced.

6.4. Impact

Agile developers seem to have a clear sense of the impact they can have on their job. They are given influence through practices with daily meetings, estimation and planning meetings and often access to customers. Data show that on the other hand, access to business representatives is not as common as prescribed in the professional literature. Even one inexperienced Scrum master expressed frustration with access to business people:

"I find it hard to get them to participate, because of everything else they have to deal with it is hard to get them all in the room at the same time. And I have really enforced getting them in on the sprint planning days and most of them are committed to that, but on other meetings and stuff I find this difficult" (Scrum master in Canadian team, master in computer science, 5–10 years in industry as programmer, 1 year as Scrum master)

Experience is central for impact. For instance, experience with informal communication channels seems to be one way of getting through to business people. In general, willingness to participate and the sense of contributing increase with experience.

A couple of agile developers in the data material chose to go for a role as specialising in a particular job, i.e. database specialist or bug fix coordinator, perhaps because it fits their competence and personality. These two developers seem to prefer this situation, and obviously give them more impact at the job. One of them mentioned an example:

"A column was getting populated with value zero. And when I talked to the developer, he said it should be either null or other than zero. Then I say, 'ok, there is a zero'. And I noticed that, when I did a deeper search after a discussion with the person, he had been declaring a variable initialised by default to zero value. And as I spotted that code to him, he said: 'I though I'd done it. Let me fix it right away'". (Agile Canadian developer, database specialist in Scrum-of-scrums team, software for petroleum industry, master computer science, 10 years in industry, one year in current company)

These examples show agile teams that break out of the prescribed practice of not having specialised roles among developers. It enables some developers to have more impact, but may take away power from other.

Developers in non-agile teams seem to feel that their impact is very much constrained to decisions on solutions and designs, and as for agile developers, they are in addition allowed to influence technological choices. They may offer suggestions, but decisions are made elsewhere:

"Basically, our VP or our design architect would be making the decision of what to do when something is late. I could offer like a suggestion, like I could say I can do everything except for this feature by the end of the week. And it is still not my decision to do it that way". (Non-agile programmer from Canada, tailored software for accounting systems using own in-house development method, master in software engineering, 3 years in industry, mainly in the current company)

The non-agile developer's impact grows with experience, but only as a result of having gotten a leading position, for instance as a project manager. In contrast to the situation in agile organisations, much of the attempts of the developer to have an influence are directed towards managers and team leads. An experienced non-agile developer said that he would normally go to the relevant team lead, involve them and agree on a solution:

"... involve varying other persons dependent on the problem. If it is technical issues, frameworks, architecture, I would involve the architecture team lead, or I could involve the gui architect. If it is about requirements, it is natural to involve those who are working with requirements". (Norwegian non-agile developer, RUP team, telecommunications industry, master in computer science, 7 years in industry in current company)

The lack of prescribed formal structures for communication among non-agile developers seems to lead to fewer *collaborative* efforts among developers to initiate and influence decisions made by managers.

In Table 5 is found a summary of how the psychological factors of empowerment are found in the interview data. There is no difference in how the two groups view their own competence.

Both groups find their way of working meaningful. Even though the communication practices and the valuation of information are different in the two groups, both find their own way of handling communication with peers and others meaningful. As for self-determination, both groups seem to have a great deal of control over how they do their work. There is a marked difference in how they experience their own impact on the work. We see that the agile team and the way it is structurally empowered constructs an environment for work that gives the individual developer significant impact on the process. The non-agile developer is more alone in the efforts, but often gets significant impact in the form of new responsibilities when doing good work.

7. Discussion

The first main conclusion to draw from the analysis is that both agile and non-agile software developers are highly empowered, as one should except in this kind of knowledge-based work. They have extensive control over their daily work by doing design and programming solutions much as they prefer within the constraints defined by the system architecture and project resources. They are structurally empowered by being given the ability to initiate and participate in decision-making processes, and have rewarding practices that contribute to their sense of meaningfulness. On the psychological empowerment side, both groups find their work as a whole and most work activities meaningful, feel that their competence at work and options to improve their competence are good

Table 5
Summary of psychological factors of empowerment in software development organisations.

	Agile	Non-agile
Meaningfulness	• Wants variation, accepts tedious tasks • Communication practices are valuable • Agile decision making is meaningful • Enjoys search for technological knowledge	• Wants variation, accepts tedious tasks • Occasional meaningless tasks • Communication is valued, but too much communication brings cos • Enjoys search for technological knowledge
Competence	• Satisfaction with own competence	• Satisfaction with own competence
Self-determination	• Developers are satisfied with possibility to choose • Are sometimes assigned tasks • Control over solutions • Specialists are more in control	• Normally assigned tasks • Occasionally able to influence task assignment • Control over solutions Specialisation gives control
Impact	• Satisfied with impact, includes also estimates, process, task assignment and technological choices • Some contact with business representatives • Influence is directed through team	• Impact on solutions and designs, and technological choices • Experience and willingness creates leadership job opportunities and power • Little contact with business representatives • Initiatives directed towards managers

and at last, they both have a sense of having an impact on the development team as well as on the products delivered.

7.1. The empowering structures

Even though all the developers have high degrees of empowerment, there is a significant difference in how the individual developer's empowerment is realised among the two groups. The team has a central place in the agile developer's identity in the workplace and further how other stakeholder relate to the developer's work. The individual empowerment of the agile developer is realised through work practices in the empowered team that enables daily, weekly and monthly participation in not only programming, but also estimation, architecture and process improvement. Tasks are to some extent chosen by the developer, although not always. The relationship between managers and developers is to a large extent canalised through the developer team as a unit. Within the team, developers stand in a symmetrical relationship to each other. The developers are individually rewarded within their teams by getting recognition and more responsibility and trust after having shown good work. Contact and collaboration with the customer are normally enabled by managers who mediate contact, but it is also realised by direct access to customer representatives. This facilitates some of the information exchange and building domain knowledge. This is found in example stories where the developers participate to change business priorities and the business representatives modelling of their own domain. Still, in many agile methods, the relationship between developers and customers is considered to be very important, but the data here indicate that it is not as strong as recommended. Only one of the teams had appointed business representatives (Scrum product owners) continuously available at the development site. These difficulties with involving customers more directly are well known and have also been discussed in other research, for instance on how to establish a better relationship with the customer in agile software development [62].

Individual empowerment for the non-agile developer is like in other standard hierarchical organisation structures. There are direct lines from the manager to each individual developer, and the manager mediates communication with the business. If projects are with a team, the manager still takes responsibility for task assignment and estimates. The individual developers are often consulted, but the responsibility remains with the manager. The communication among developers may be informal or in formal procedures like code reviews, but is not regular. When the developer needs information, new knowledge and a critical look at code or designs, the peer developers are contacted. The direct link between the manager and the individual developer is also visible in how initiatives from the developer are directed to the manager. Rewards are also individual, in contrast to the situation in agile, where team rewards like free lunches at a project's termination are more common. There is little communication and influence between developers and customers/business as most of this is normally channelled through a project manager.

To get a better overview of the distinctions at the structural level, Table 6 summarises how agile and non-agile teams handle the empowering practices in software development, referring to the topics covered in interviews as presented in Fig. 1.

7.2. From structural to psychological empowerment

According to the empowerment literature, the expected gains of empowerment are to be seen when the individuals are psychologically empowered. Among the factors of psychological empowerment, impact is the one with most visible difference in how it is realised for the individual developer. The agile developers build their sense of impact on an ability to participate in a lot of team decisions regarding the software project, whereas the non-agile developers gain greater impact when as a result of good work and initiatives, they move from a pure developer role. It is rather typical for the non-agile developers to move towards more and more responsibilities and management work, and thus increased power in the development organisation is a part of the rewarding structures. Most initiatives from the agile developers are directed through the team, and if they get support in the team they are heard by management. For these developers, the team works as a quality assurance system on initiatives, and enables a potential influence in higher-level decisions with which most seem to be satisfied. As a whole, non-agile developers are less able to participate in higher-level decisions about the development team, project and organisation. If they are, they are consulted by managers by virtue of experience. So, team empowerment is a power structure that contributes, in addition to more frequent direct customer contact, to the impact agile developers have. Fig. 3 indicates the

Table 6
A summary of empowering practices.

Empowering practice	Agile	Non-agile
Control of low-level design/coding	Full control for developers, in some teams as pair programming	Full control for developers, ownership to task and product
Ability to choose own work tasks	When work is primary development of new functionality, less common in bug fixes or refactorings	Manager delegates tasks after consulting developers
Initiate trials of new technology	Approved of by team and managers, gives credibility in team	Approved of by team and managers, gives more career opportunities
Participate in estimation	Agile team practices support this, is part of iteration planning	Manager decides after consulting developers
Participate in high-level design	Agile team practices support this, is part of iteration planning and daily meetings	Developers and manager collaborate
Participate in process improvement	Agile team practices support this, is part of post-iteration evaluation meetings, but is often skipped	Developers and managers collaborate on this, but it is rare
Get information and domain knowledge from customer	Directed through manager, some teams have direct access to customer but not all	Directed through manager, developers rarely meet customers unless given special responsibilities
Get technical knowledge in order to solve task	Approved of and considered part of the team's responsibility	Approved of in organisation, normally first by asking colleagues for tips. Expected that own time is used for time-consuming learning
Collaborate with peers	Agile practices and guidelines like planning meetings, pair programming	Ad hoc when in need for information or knowledge
Team rewards	Perks like team lunches or other common activities	Rare
Individual rewards	Recognition in team as result of initiatives and good ideas	Career opportunities for those who show initiative in solving problems

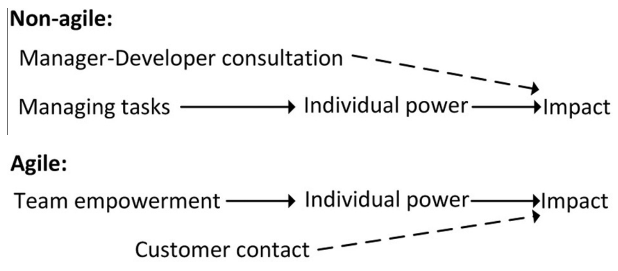

Fig. 3. From structural empowerment to individual power and sense of impact in agile and non-agile software development organisations.

differences in how impact is achieved, showing practices and factors that contribute to structural empowerment on the left-hand side of the arrows, indicating a causal relationship.

For the agile developer, meaningfulness is obtained when they get a variety of tasks, through which they are able to find the knowledge they need, and they see the low-cost information channels and ability to participate in the team activities as contributing to a meaningful work situation. In addition, variation and ability to get the needed knowledge contribute to non-agile developers' sense of meaning in their work. And this is supported in their organisations, although through other power structures. As for information channels, they are often in the form of textual and graphical documentation, and non-agile developers feel that there is a significant cost with information exchange through documentation work, and, further, that work with creating and getting information should be limited. An interesting observation is that agile developers seem to want to stay agile when given the option to go back to non-agile ways, whereas the non-agile teams had an ongoing discussion about turning to agile development methods. Both parties find their jobs meaningful, and interviewees from both sides expressed that they felt lucky being able to work with software development in their particular company. Fig. 4 summarises how meaningfulness is achieved.

The competence factor is the least different in the two different groups. Developers in both types of organisations feel that they have access to the needed knowledge, both regarding the technology needed for the development project and domain knowledge. They also seem to get the information they want from the information channels they have. In agile teams, this is to some extent realised through direct contact with customer representatives, but for both kinds of teams, the contact with customers is often routed through project managers. The relation between structural empowerment practices and the competence factor is summarised in Fig. 5.

The sense of self-determination for the two groups is similar, the main difference being that agile developers are able to choose among tasks in many situations. In non-agile teams, task assignment is a manager's responsibility. Both types of teams have control of how to solve the problems they are working with, both doing the design choices, and gathering the knowledge and information needed. Theories for agile development suggest diversity in the team as a factor that enables agility [6]. And in our data, it is evident that people that have some kind of specialisation, like expertise in databases or user interfaces, perceive specialisation as a feature that gives them a higher degree of self-determination. This is even more pronounced in non-agile teams, where team

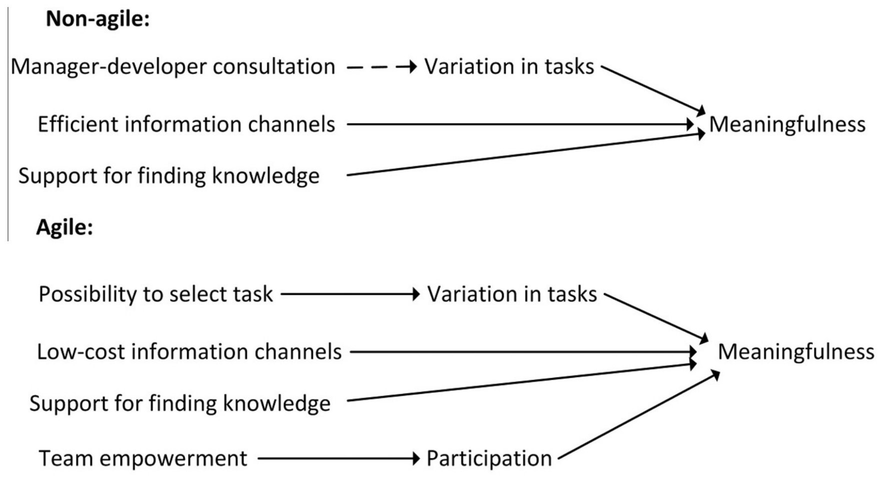
Non-agile:
Manager-developer consultation
Variation in tasks
Efficient information channels
Meaningfulness
Support for finding knowledge
Agile:
Possibility to select task
Variation in tasks
Low-cost information channels
Meaningfulness
Support for finding knowledge
Team empowerment
Participation

Fig. 4. How sense of meaningfulness is achieved in non-agile and agile software development organisations.

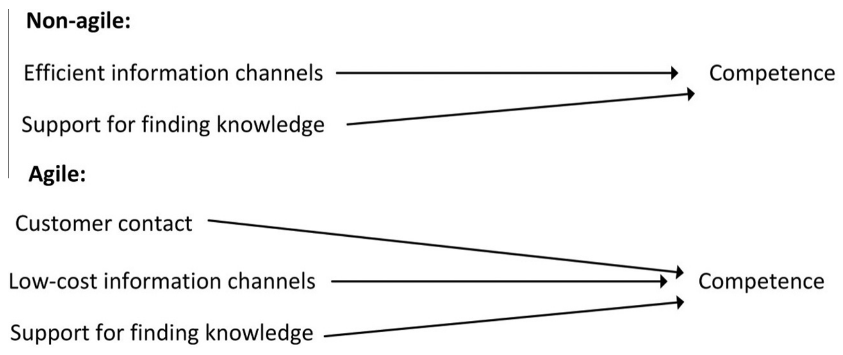
Non-agile:
Efficient information channels
Competence
Support for finding knowledge
Agile:
Customer contact
Low-cost information channels
Competence
Support for finding knowledge

Fig. 5. How sense of competence is achieved in non-agile and agile development organisations.

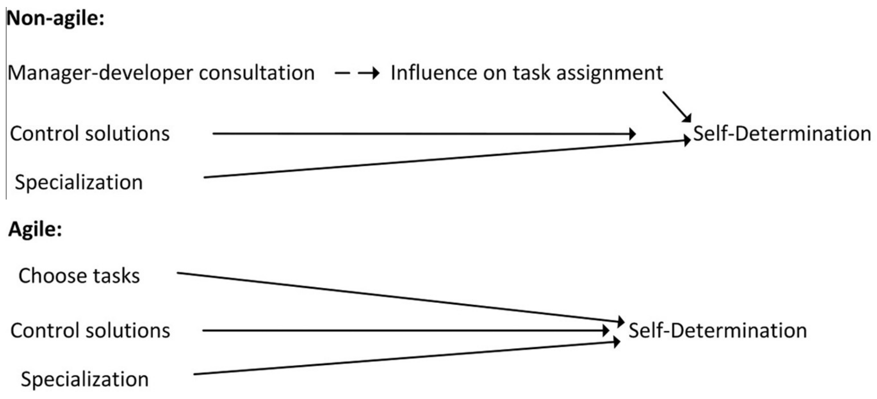

Fig. 6. How sense of self-determination is achieved in non-agile and agile development organisations.

members often have roles indicating their specialisation. Fig. 6 shows the features of structural empowerment that contribute to self-determination.

Results on psychological empowerment suggest that the four factors are additive [12]. With that premise, a conclusion of the analysis is that the individual agile developer is empowered at a higher level than the non-agile developer, mostly with the difference in the impact factor as a contributing factor. Still, both groups are highly empowered, and the difference is not very large. Agile developers do not always select their own tasks, and non-agile developers do also meet business people on occasion. A couple of stories from one of the agile teams show how agile team members experience how difficult it is to abide by all the prescribed and empowering practices, with some unwanted consequences.

The first story was told by a developer who had seen that very experienced people working in a refactoring team suggested a redesign that should improve the software. Their experience and position gave them a significant but not formal power, which again resulted in a lack of protests being directed towards the suggested solutions, which later was seen to be suboptimal. Even though the developers were free to express criticisms against the design, the experts' credibility precluded this, and this cost a lot of rework. The story is a reminder of the group think problem mentioned in the work by McAvoy and Butler [47].

The second story is about a developer who did not adapt well into the agile team, probably due to personality. He found a solution by volunteering for a job as database specialist, and gradually grew into this specialist role where he controlled the development of the databases for the software. As a consequence, the team has grown vulnerable due to the risk of losing this person's competence in the future, and he has gained influence on how the other developers should develop the software. Fig. 1 shows how several factors influence the relation between structural and psychological empowerment (Individual Characteristics, Work Design, Leadership, Organisational Support). The story above exemplifies how individual characteristics influence how a person is empowered, illustrating the complexity of the empowerment construct.

8. Limitations and future research opportunities

The findings here are based on a qualitative analysis of textual data, using a structural–psychological model for empowerment, and hence are dependent on the researcher's theoretical sensitivity and ability to interpret data in a way that compensates for biases. To avoid this, the data has been read and categorised twice, with some time delay in between, to identify variations in the researcher's interpretation over time and ensure a more critical view of the interpretations. On the other side, the number of interviews and the variation in organisational practices give the final results credibility and generalisability. Twelve agile and thirteen non-agile developers from a variety of industries and practices ensure breadth in the data. A possible weakness is that the teams are small, and that the interview data do not have representatives from larger plan-driven development projects. As the data, coming from semi-structured interviews, are qualitative, it may be that several issues are not handled to their full depth for all interviewees, which further may imply that some potential relations between empowering practices, structural factors and psychological factors may be missing, but the data are believed to be in confirmation of the relations presented in the analysis.

For the non-agile development organisation, the findings suggest that teams and their managers must establish practices that bring developers into the decision processes, relieving the traditional power structures. We have to accept that there are situations where plan-driven development (as well as short-term ad hoc development activities) is necessary, but that should not necessarily imply lower empowerment than in agile teams. Using low-cost techniques for making team decisions and organising more frequent information exchange activities with customers could enable more power to the individual developer and improved information flow, considering the structural empowerment areas where most could be gained. Also, non-agile teams seem not to be coordinated in their efforts to improve knowledge on technology and solutions. Initiatives are mediated through managers, without being assessed by peers before coming to the manager for a decision. Can we as researchers identify ways of enhancing communication among team members other than long-lasting meetings involving many stakeholders and many topics? After all, there is a pronounced scepticism in non-agile teams towards the practices used for information exchange today, due to the fact that they are felt to cost too much.

For the agile teams, developers seem to be satisfied, but there is a tendency to skip the prescribed empowering practices and drift into unrecommended practices in certain situations, for instance when priorities are changed when problems with systems operation occur, in the distribution of bug fixes, or when teams skip communication practices when deadlines are getting close. Short-term gains may be won from skipping team empowerment practices, but may have unwanted effects in the long term by lessening the developers' sense of responsibility, motivation and focus. A constant awareness among both managers and within teams of how these practices enable empowerment is necessary. Critical assessments of consequences of introducing new practices and abandoning old ones is a constant part of process improvement activities in agile teams, but how practices affect empowerment is a dimension not very often considered, and could be researched in more depth. Another question that should be investigated regarding empowerment of agile team members, is whether it is harder for the agile developer to get to a leadership position because personal initiative is not visible for outside observers like managers and their likes? Willingness to take initiatives is a main cause of increased power in non-agile teams.

9. Conclusions

Empowering practices is in research and professional literature considered to be among the features of agile software development that distinguishes it from non-agile or plan-driven methodologies. During the analysis conducted here, this was found to be true with some reservations. Agile developers have, through their participation in the empowered team, more individual power in higher-level decision making, and this is confirmed in their feeling of having an impact on the organisation. They do also have information flows in their companies that are significantly different from what is found in more documentation-oriented development teams with fewer meeting places for exchanging information. This contributes to a variation among agile and non-agile teams in how meaningfulness is constructed for the individual developer. Even though social and technological practices that enable empowerment are not always prescribed to the same extent in non-agile as in agile development it is not right to say that non-agile teams are not empowered, as they normally have a lot of control of their work situation, and they are also satisfied with their work place. Structural empowerment takes a different form in non-agile organisations, but still leads to psychological empowerment. It is also observed that agile teams are sloppy in their practices, influencing the level of agility, and further leading to effects on how empowerment is realised, with unknown consequences for the team and organisation.

Further research on how to assess and improve empowerment in a software team, measure the effects of empowerment on the organisation, and identify the influence of existing and new practices on empowerment is needed.

References

[1] K. Beck, C. Andres, Extreme Programming Explained: Embrace Change, second ed., Addison-Wesley, 2004.
[2] K. Schwaber, Agile project management with Scrum, Microsoft Press, Redmond, Wash, 2004.
[3] M. Poppendieck, T. Poppendieck, Lean Software Development, Addison-Wesley Professional, An Agile Toolkit, 2003.
[4] K. Conboy, Agility from first principles: reconstructing the concept of agility in information systems development, Inform. Syst. Res. 20 (2009) 329–354.
[5] R. Vidgen, X. Wang, Coevolving systems and the organization of agile software development, Inform. Syst. Res. 20 (2009) 355–376.
[6] G. Lee, W. Xia, Toward agile: an integrated analysis of quantitative and qualitative field data, MIS Quart. 34 (2010) 87–114.
[7] D.E. Bowen, E.E. Lawler, Empowering service employees, Sloan Manage. Rev. 36 (1995) 73–84.
[8] K. Dewettinck, M. van Ameijde, Linking leadership empowerment behaviour to employee attitudes and behavioural intentions testing the mediating role of psychological empowerment, Pers. Rev. 40 (2011) 284–305.
[9] S. Fernandez, T. Moldogaziev, Empowering public sector employees to improve performance: does it work?, Amer Rev. Public Adm. 41 (2011) 23–47.
[10] E.E. Lawler III, High-Involvement Management, Jossey-Bass, San Francisco, CA, 1986.
[11] P.K. Mills, G.R. Ungson, Reassessing the limits of structural empowerment: organizational constitution and trust as controls, Acad. Manage. Rev. 28 (2003) 143–153.
[12] G.M. Spreitzer, Psychological empowerment in the workplace – dimensions, measurement, and validation, Acad. Manage. J. 38 (1995) 1442–1465.
[13] M. Zimmerman, Psychological empowerment: issues and illustrations, Am. J. Community Psychol. 23 (1995) 581–599.
[14] D. Marsden, A. Cañibano, An Economic Perspective on Employee Participation, in: A. Wilkinson, P.J. Gollan, M. Marchington, D. Lewin (Eds.), The Oxford Handbook of Participation in Organizations, Oxford University Press, 2010, pp. 131–163.
[15] C.M. Riordan, R.J. Vandenberg, H.A. Richardson, Employee involvement climate and organizational effectiveness, Hum. Resour. Manage. 44 (2005) 471–488.
[16] K.W. Thomas, B.A. Velthouse, Cognitive elements of empowerment – an interpretive model of intrinsic task motivation, Acad. Manage. Rev. 15 (1990) 666–681.
[17] J.E. Mathieu, L.L. Gilson, T.M. Ruddy, Empowerment and team effectiveness: an empirical test of an integrated model, J. Appl. Psychol. 91 (2006) 97–108.
[18] M.T. Maynard, L.L. Gilson, J.E. Mathieu, Empowerment—fad or fab? a multilevel review of the past two decades of research, J. Manage. 38 (2012) 1231–1281.
[19] G. Melnik, F. Maurer, Comparative analysis of job satisfaction in agile and non-agile software development teams, Springer Lect. Notes Comput. Sci. 4044 (2006) 32–42.
[20] H.D. Stolovich, Human performance technology: research and theory to practice, Perform. Improv. 39 (2000) 7–16.
[21] W. Mobley, Employee Turnover: Causes, Consequences, and Control, Addison-Wesley, NY, 1982.
[22] A. Cockburn, Agile Software Development, Addison-Wesley, Reading, MA, 2002.
[23] J. Highsmith, Agile Project Management: Creating Innovative Products, Addison Wesley Longman Publishing Co., Inc., 2004.
[24] A. Kelly, Changing Software Development: Learning to Become Agile, Wiley, 2008.
[25] C. Larman, Agile and Iterative Development: A Manager's Guide, Pearson Education, 2003.
[26] O. McHugh, K. Conboy, M. Lang, Using agile practices to influence motivation within IT project teams, Scand. J. Inform. Syst. 23 (2011) 59–84.
[27] J.P. Macduffie, Human resource bundles and manufacturing performance: organizational logic and flexible production systems in the world auto industry, Ind. Labor Relat. Rev. 48 (1995) 197–221.
[28] L.M. Maruping, V. Venkatesh, R. Agarwal, A control theory perspective on agile methodology use and changing user requirements, Inform. Syst. Res. 20 (2009) 377–399.
[29] G.M. Spreitzer, M.A. Kizilos, S.W. Nason, A dimensional analysis of the relationship between psychological empowerment and effectiveness satisfaction, and strain, J. Manage. 23 (1997) 679–704.
[30] J.L. Cordery, W.S. Mueller, L.M. Smith, Attitudinal and behavioral-effects of autonomous group working – a longitudinal-field study, Acad. Manage. J. 34 (1991) 464–476.
[31] B.L. Kirkman, R. Benson, Beyond self-management: antecedents and consequences of team empowerment, Acad. Manage. J. 42 (1999) 58–74.
[32] H. Shrednick, R. Shutt, M. Weiss, Empowerment: key to IS world-class quality, MIS Quarterly 16 (1992) 491–505.
[33] R. Conradi, T. Dybå, D.I.K. Sjøberg, T. Ulsund, Software process improvement: results and experience from the field, Springer-Verlag, Berlin, Germany, 2006.
[34] A.A. Wilkinson, The Oxford Handbook of Participation in Organizations: Oxford Handbooks in Business and Management, 2010.
[35] K. Dewettinck, J. Singh, D. Buyens, Psychological empowerment in the workplace: reviewing the empowerment effects on critical work outcomes, in, Vlerick Leuven Gent Management School, 2003, p. 26.
[36] S.E. Seibert, G. Wang, S.H. Courtright, Antecedents and consequences of psychological and team empowerment in organizations: a meta-analytic review, J. Appl. Psychol. 96 (2011) 981–1003.
[37] D.C. Hutchins, The Quality Circles Handbook, Pitman Press, New York, 1985.
[38] K.O. Cua, K.E. McKone, R.G. Schroeder, Relationships between implementation of TQM, JIT, and TPM and manufacturing performance, J. Oper. Manage. 19 (2001) 675–694.
[39] M. Frese, E. Teng, C.J.D. Wijnen, Helping to improve suggestion systems: predictors of making suggestions in companies, J. Organ. Behav. 20 (1999) 1139–1155.
[40] J.R. Hackman, The Psychology of Self-Management in Organizations, in: M.S. Pallack, R.O. Perloff (Eds.), Productivity, Change, and Employment, American Psychological Association, Washington, DC, 1986.
[41] N.B. Moe, T. Dingsøyr, T. Dybå, Understanding self-organizing teams in agile software development, in: 19th Australian Conference on Software Engineering, 2008, pp. 76–85.
[42] R.E. Quinn, G.M. Spreitzer, The road to empowerment: seven questions every leader should consider, Organ. Dyn. 26 (1997) 37–49.
[43] N.B. Moe, From improving processes to improving practice: software process improvement in transition from plandriven to change-driven development, in: Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway, 2011.
[44] N.B. Moe, T. Dybå, The use of an electronic process guide in a medium-sized software development company, Softw. Process Improv. Pract. 11 (2006) 21–3434.
[45] N.B. Moe, A. Aarum, Understanding decision-making in agile software development: a case-study, in: Software Engineering and Advanced Applications SEAA '08 34th Euromicro Conference, Parma, Italy, 2008, pp. 1089–6503.
[46] N.B. Moe, T. Dingsoyr, T. Dyba, A teamwork model for understanding an agile team: a case study of a Scrum project, Inf. Softw. Technol. 52 (2010) 480–491.
[47] J. McAvoy, T. Butler, The role of project management in ineffective decision making within Agile software development projects, Eur. J. Inform. Syst. 18 (2009) 372–383.
[48] B. Tessem, F. Maurer, Job Satisfaction and motivation in a large agile project, lecture notes in computer science=, Lect. Notes Artif. Int. 4536 (2007) 54–61.

[49] B. Tessem, An empirical study of decision making, participation, and empowerment in Norwegian software development organisations, Lect. Notes Bus. Inform. Process. 77 (2011) 253–265.
[50] A. Wilkinson, P.J. Gollan, M. Marchington, D. Lewin, Conceptualizing Employee Participation in Organizations, in: A. Wilkinson, P.J. Gollan, M. Marchington, D. Lewin (Eds.), The Oxford Handbook of Participation in Organizations, Oxford University Press, 2010, pp. 3–25.
[51] K. Wiegers, Peer Reviews in Software: A Practical Guide, Addison-Wesley, Boston, MA, 2001.
[52] L. Williams, R. Kessler, Pair Programming Illuminated, Addison-Wesley, Reading, MA, 2003.
[53] M.E. Fagan, Advances in software inspections, IEEE Trans. Softw. Eng. 12 (1986) 744–751.
[54] K.M. Lui, K.A. Barnes, K.C.C. Chan, Pair Programming: Issues and Challenges, in: T. Dingsøyr, T. Dybå, N.B. Moe (Eds.), Agile Software Development: Current Research and Future Directions, Springer-Verlag, Berlin, Germany, 2010, pp. 143–163.
[55] E. White Baker, Why situational method engineering is useful to information systems development, Inform. Syst. J. 21 (2011) 155–174.
[56] A. Aurum, C. Wohlin, The fundamental nature of requirement engineering activities as a decision-making process, Inf. Softw. Technol. 45 (2003) 945–954.
[57] B. Alenljung, A. Persson, Portraying the practice of decision-making in requirements engineering: a case of large scale bespoke development, Requirements Eng. 13 (2008) 257–279.
[58] C. Zannier, M. Chiasson, F. Maurer, A model of design decision making based on empirical results of interviews with software designers, Inf. Softw. Technol. 49 (2007) 637–653.
[59] M. Drury, K. Conboy, K. Power, Obstacles to decision making in Agile software development teams, J. Syst. Softw. (2012).
[60] B. Glaser, A.L. Strauss, The Discovery of Grounded Theory: Strategies for Qualitative Research, Aldine Publishing Company, Chicago, 1967.
[61] M. Birks, J. Mills, Grounded Theory: A Practical Guide, Sage Publication, London, 2011.
[62] A. Martin, R. Biddle, J. Noble, An ideal customer: a grounded theory of requirements elicitation, communication and acceptance on agile projects, in: T. Dingsøyr, T. Dybå, N.B. Moe (Eds.), Agile Software Development: Current Research and future Directions, Springer-Verlag, Berlin, 2010, pp. 111–141.