**Ambidextrous software development - how a large-scale development project balanced agility and discipline**

Forfatterliste (forslag!): Torgeir, Tor Erlend, Nils Brede, Tore, Eva, Svein

Introduksjon (2S)

*How does a large-scale development project balance agility and discipline?*

Bakgrunn (3s)

Metode (3,5s)

Resultat

Oversikt (4s Torgeir)

Tidslinje

Masterplan

Utvikling av prosjektet over tid

Organisasjonskart

Roller i team

3 faser i parallell

Romplan

Nøkkeldata

Arkitekturskisse

Involvering av forretning

Viktig prosjekt: levere på tid

Prosjektledelse (2s Tore, Eva)

\* Gå i takt: synkroniseringsregime

\* Målprismodellen med brukerhistorier

\* Fra tre til en backlog

\* Folk versus gjennomføringsmodell

Løsningsbeskrivelse (2s Nils, Tor Erlend) "requirements elicitation"

\* Kontinuerlig løsningsbeskrivelse: 3 faser på en gang

\* Forretningsinvolvering; ressurser fra teamene (involvering versus beskytte)

\* Up-front versus smidig løsningsbeskrivelse (kort og lang horisont)

\* Modning

Arkitektur og informasjonssikkerhet (2s Tor Erlend, Svein)

\* Endringer i arkitekturen ("produkt")

\* Arkitekturarbeidet ("prosess")

\* Betydning av arkitekturarbeid for utviklingsprosjektet

Teamkoordinering og kunnskapsdeling (2s Torgeir, Tore, Nils)

\* Mange arenaer

\* Endring i arenaer over tid

\* Åpent landskap / uformelle arenaer

Tabell: "mekanismer" for koodinering/kunnskapsdeling

Diskusjon (4s)

Forskningsspørsmål

Implikasjoner for praksis

Implikasjoner for teori

Forbehold

Konklusjon (1/2)

Referanser

Vedlegg

Intervjuguider

# 1. Introduction

<Large development projects>

Software development is challenging, due to increasing demands, the complexity of problems and the scale of work addressed by software. There are challenges with cost overruns, late completions and outright project failures (Flyvbjerg and Budzier, 2011).

Since the formulation of the agile manifesto in 2001, agile methods have transformed software development practice by strongly emphasizing change tolerance, evolutionary delivery and active end-user involvement (Dingsøyr et al., 2012). Rajlich describes agile development as a paradigm shift in software engineering that “brings a host of new topics into the forefront of software engineering research” (Rajlich, 2006).

In the first special issue on agile development, in *IEEE Computer*, Williams and Cockburn (Williams and Cockburn, 2003) stated that agile methods “best suit *collocated teams* of about *50 people or fewer* who have easy access to user and business experts and are developing projects that are not life-critical”. The success of agile methods for small, co-located teams has inspired use in new domains: Companies increasingly apply agile practices to large-scale projects.

However, there are challenges with achieving the same productivity gains in these areas, as in the “home ground” of agile methods. Agile methods are based on the idea that high-quality software can be developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change (Nerur et al., 2005). As agile development techniques are used on large-scale projects, new challenges arise. “Agile in the large” was voted “top burning research question” by practitioners at the XP2010 conference (Freudenberg and Sharp, 2010).

Fundamental assumptions in agile development are severely challenged when using these practices in large-scale projects. Self-management is a central principle in agile methods, but studies from other fields than software development indicate that self-management can reduce the ability to effectively coordinate across teams (Ingvaldsen and Rolfsen, 2012). Also while the teams need to self-manage, team members need to have an effective knowledge network and collaborate closely with experts outside the team in large-scale agile (Moe et al., 2014). To have an emerging architecture could hamper project progress when many teams are working in parallel, and some practices like the scrum of scrum has been found to be inefficient in large projects (Paasivaara et al., 2012a). An international survey on agile adoption[[1]](#footnote-1) shows that agile practice has primarily been successful in small teams.

The term ‘large-scale agile development’ has been used to describe agile development in everything from large teams to large multi-team projects to making use of principles of agile development in a whole organization.

There is an established discussion on what constitutes agile software development, with Conboy (Conboy, 2009) providing the most thorough discussion. He defines agility as the continuous readiness “to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment.”

Arguments for a definition based in the number of teams is presented in (Dingsøyr et al., 2014), where large-scale agile is defined as “*agile development efforts with more than two teams.”*

This definition excludes agile methods applied in large organizations from ‘large-scale agile’, and we propose that this is considered as a research direction on its own.

The workshop on principles of large-scale agile development defined a research agenda where topics with high priority included "organisations of large development efforts", "variability factors in scaling", " inter-team coordination" and "release planning and architecture". This article is an explorative case study of a large-scale development project with extensive use of agile methods. We raise the following research question: *How does a large-scale development project balance agility and discipline?*

\* Kieran: smidig

# 2. Challenges in Large-Scale Development

**<Related work>**

**Project Management**

**Solution description**

**Architecture**

The software architecture is the fundamental organization of a system. IEEE defines it as “embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution”. How and when to make architectural decisions has been a major debate in the software engineering field (Abrahamsson et al., 2010). In traditional development, the architecture was defined prior to implementation and testing, in agile development the architectural design is to emerge.

In large agile projects, several approaches have been taken. Some start with the architecture (“big upfront design”), and then use agile methods, others spend the first iteration focusing on architecture. Others again start directly on development and let the architecture emerge. To construct a large software system, developed by a number of teams, it is of vital importance that the architecture is agreed on and communicated without introducing the bureaucracy and overhead associated with traditional methods. There is little support in how to achieve this in the prescribed methods available.

Software architecture has been studied for a number of years in software engineering, but the challenges of large-scale agile development have not been addressed. Further, the field of software architecture shares the general challenges of software engineering of use of theory and industrial relevance.

--

Agility can be described as an ability to encourage, create and respond to change in order to create customer value (Conboy, 2009). Proponents of agility argue that it provides direct and empirically testable value to stakeholders at short intervals. Software architecture, on the other hand, is associated with up-front designs and stable structures that accommodate pre-defined, non-functional requirements. The value of good architectural decisions are likely to surface at a later point in time in the form of easier maintainable code, scalability etc. (Faber, 2010). A sound architecture is largely invisible; it provides an effective structure for subsequent functionality.

So, how did Omega reconcile both agile development and the need to satisfy architectural (non-functional) requirements? What were the tensions between agility and up-front planning in terms of software architecture in the Omega program? How were these tensions balanced? Is the often-mentioned dichotomy between agility and software architecture false (Abrahamsson et al., 2010)?

To understand these tensions it is imperative to understand that software architecture has meaning and significance for a wide variety of stakeholders. However, different stakeholders apply different meanings and matter of significance to software architectures. For example, software architecture may be seen as blueprints, literature, language or decisions (Smolander, 2002). This variety contributes to exaggerate tensions.

We will now focus on two core elements of the architecture tensions in Omega.

**Inter-team coordination**

In small agile projects the development team coordinates work through frequent informal interaction amongst themselves and with customers as in the customer-on-site practice in eXtreme Programming. Scrum has dedicated meetings for planning, review and retrospectives. Many teams use visual boards, like in Kanban, to show who is working on what and status of work tasks. For large-scale projects there is less support. Scrum prescribes regular meetings between Scrum teams (“Scrum of Scrums”) in order to manage the interfaces between teams. Eckstein shows techniques that are applicable to large projects in order to facilitate planning, status information, integration and retrospectives in her book with recommendations to practitioners (Eckstein, 2004). Some frameworks for large scale agile have been suggested by practitioners, such as Larman and Vodde (2013). Central challenges in coordination are to identify the right form or artifacts, arenas and degree of formalization in a large project with high uncertainty.

There are few studies on inter-team coordination in practice, but “Scrum of Scrums” have been found not to be sufficient (Paasivaara et al., 2012b). Other solutions such as feature-specific or site-specific forums has not led to satisfactory coordination. To understand coordination, we can draw on the literature on communities of practice (Wenger, 1998). Further, there are team coordination models from of small agile teams (Strode et al., 2012), and a large body of literature from management science and organizational theory.

A study from management science (Ingvaldsen and Rolfsen, 2012) suggests that inter-group coordination is a major challenge when groups are self-managing. If there are practices in agile methods that have overcome these challenges, such findings will be of large interest also outside of the software engineering field

# 3. Method

To investigate the research question *How does a large-scale development project balance agility and discipline,* we have chosen an exploratory case study (Runeson and Höst, 2009). This is because there is little existing literature on balancing agility and discipline in large-scale projects. We wanted to understand challenges in this area, and generate ideas for further research.

Paradigmatisk case (Flyvjerg).

We chose to focus on one case, which was described as the most successful project with extensive use of agile methods in Norway so far - what we will refer to as the Omega programme to develop a new office automation system for the public department *Gamma.* This programme ran from 2008 to 2012, and had 12 development teams in parallel at the most. The development project was divided in three parts, one carried out by an internal development unit (6 teams), and two other parts conducted by consulting companies *Alpha* and *Beta.* We provide more details of the project in our results section. After an initial meeting with the manager of the development project from *Gamma*, we had meetings with project managers from *Alpha* and *Beta* and were granted access to do data collection from all three parties.

Before starting data collection, we studied public presentations and an official report from the programme in order to increase our understanding. We quickly realised that the programme had a very complex organization with 175 people involved and a number of project and sub-projects. Our data collection would start after the project was finished, and all resources were then allocated on new projects. This fostered a challenge both to get access to people, and that for some it was over a year since they were working on the project.

## 3.1 Data collection

We chose to use two types of data for this explorative study: First, documents, and here we mainly rely on three documents: An official report after project completion (2012), an internal experience report (2012) and a report made to quality assure the project, which was written by an external consulting company in 2010. In total, these reports contained 277 pages of text.

In addition, we chose to conduct focus groups on aspects that we found particularly challenging in large-scale projects. We discussed the list of topics with the three organizations, and ended up with four topics of interest:

1. Project management
2. Solution descriptions
3. Architecture
4. Inter-team coordination and knowledge sharing

Some of the advantages of focus groups include the ability to collect large and rich amounts of research data, that the researcher can interact directly with respondents for clarification of responses or follow-up questions and that focus group participants can react to and build upon responses from other focus group members (Stewart et al., 2007). Focus groups are applicable to quickly obtain information on emerging phenomena through structured, moderated discussions with groups of practitioners. For each topic, we organized three focus groups, one for each organization. The reason for conducting three focus groups on each topic was that we knew there were slightly different approaches in the three subprojects, it was also easier to set times for the focus group meetings, and in addition having meetings in each organization might have led to more openness than in joint focus group meetings.

We asked the three organizations to invite the most relevant people to attend each focus group. At *Alpha,* the key persons working on architecture was no longer working in the company, but we instead spoke to a person in a new large-scale project building on the approach used in Omega. Table 1 gives an overview of the focus groups and lists the number of participants for each group. In total, we had 33 people participating in focus groups. At *Beta* and *Gamma,* some participated in several groups, making it a total of 26 individuals participating. These people had a number of roles in Omega, from project director, project managers, subproject managers, technical architects, functional architects, tester, development leader, GUI responsible, controller, responsible for information security, scrum master and developers. Note that there is a bias in participation towards employees in management positions, given the availability of people and the topics in the focus groups we did not include any pure developers. However, many of the people we talked to had started in the programme as developers and gotten other roles during the programme.

To conduct focus group meetings, six researchers formed four teams so that two researchers would participate in each meeting. We developed interview guides, and a rough timeline for the project, which was shown to participants in order to remember key events in the project. The researchers had roles of leading and moderating the discussion and taking notes. All focus groups would start by participants explaining their role in the project. Two focus groups included only one person and were conducted as an interview. One focus group included six persons, and there we made use of brainstorming techniques in order to ensure contribution from everyone invited, to brainstorm on key events and on what they thought was conducted well in the programme within the topic of discussion, and what they thought could be improved. All meetings were recorded and transcribed, and we took pictures of drawings on whiteboards. This produced a total of 326 pages of transcribed material. Minutes of the meetings were sent to participants for information and for comments.

*Table 1: Participants in focus groups. In total, 33 project members participated in focus groups.*

|  |  |  |
| --- | --- | --- |
| ***Theme*** | ***Organisation*** | ***Number of participants*** |
| Project management | Alpha | 2 |
|  | Beta | 3 |
|  | Gamma | 3 |
| Solution description | Alpha | 1 |
|  | Beta | 6 |
|  | Gamma | 3 |
| Architecture | Alpha | 1 |
|  | Beta | 3 |
|  | Gamma | 3 |
| Inter-team coordination and knowledge sharing | Alpha | 2 |
|  | Beta | 3 |
|  | Gamma | 3 |

## 3.2 Data analysis

Data analysis started with the research teams presenting first impressions immediately after data collection. We imported all data documents into a tool for qualitative analysis, and did a descriptive and holistic coding (Saldaña, 2009) of the topic team coordination and knowledge sharing. Three researchers read the relevant documents and did individual coding, and then we agreed on coding in joint meetings. Units of text ranged from sentences to whole pages and were coded into topics such as "Scrum of Scrums" and "Metascrum". The results of this pilot was presented and discussed with the rest of the research team, and then we proceeded to code the material for the remaining topics. Given the various topics and background of researchers, the level of detail in the coding varied between the research teams.

After coding the material, we discussed what were the most interesting findings in the material. This was then made into a slide presentation, and feedback meetings with all three organisations were conducted for member checking. These meetings were recorded and partially transcribed to add further data material.

# 4. Results

\* Alle figurer tegnes om.

\* Terminologi: solution description / detailed design

Lurer på:

\* Antall deltakere fra SPK i prosjektene konstruksjon, arkitektur, test

## 4.1 The Omega programme

The Omega programme was conducted by a public department in Norway which needed a new office automation system. The main arguments for initiating the programme were reforms in legislation and regulations, and also because the department had an existing office automation system on a platform which was to be abandoned.[[2]](#footnote-2) When the programme started, the content of the new legislation and regulations were not known. This was the main reason for choosing agile development practices for the project.

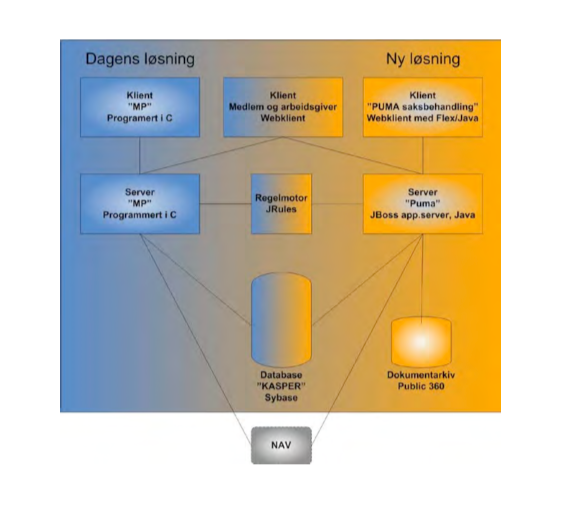
The public department has about 380 employees, and provides 950.000 customers with several types of services. The department integrates heavily with one other public department.

Omega was initiated to enable the department to fulfil their most important service. The programme had a strict deadline because of the reform in legislation. It is one of the largest IT programmes in Norway, with a final budget of about EUR 120 million. The programme started January 2008 and lasted until March 2012. 175 people were involved in the project, of which 100 were external consultants from five companies. The programme used both time and material and target price contracts for subcontractors. About 800 000 person hours was used to develop around 300 epics, with a total of about 2 500 user stories. These epics were divided into 12 releases as in Figure 1.

Macintosh HD:Users:dingsoyr:Dropbox:Smidig prosjektledelse i store utviklingsprosjekt (102003231):B Case Perform Statens Pensjonskasse:tidslinje_perform.pdf

*Figure 1: Timeline for the omega project, showing 12 releases during the four year project. (Source: Official programme report).*

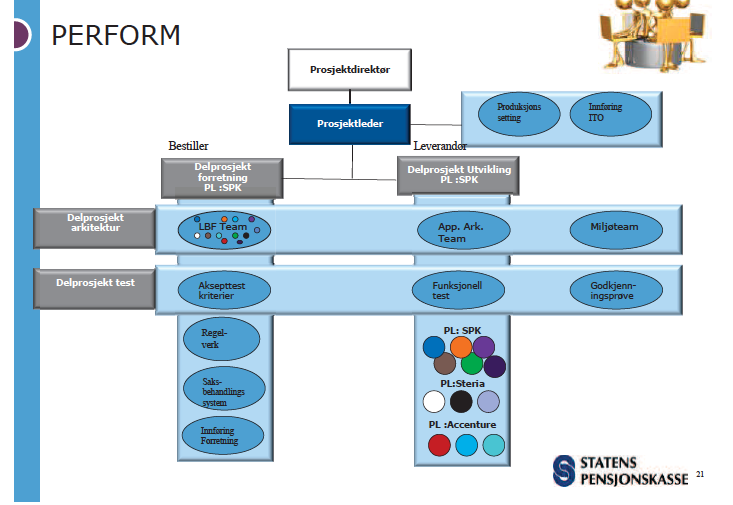
The existing office automation system was client/server-based and written in C. The new system is a service-oriented system written in Java and which provided a richer interface using Flex.[[3]](#footnote-3) The database from the old system was kept, but the data model was changed. The regulations and legislations are implemented in the system as rules, and these are represented in JRules. The system is integrated with a new document archive and with systems from another public department. Figure 2 gives an overall description of the existing and the new solution.



*Figure 2: Overall systems description of existing and new solution. (Kilde:)*

The programme was managed by a programme director who mainly focused on external relations, a programme manager focusing on the operations, a controller and four project managers responsible for the projects *business, construction, architecture* and *test:*

* *Business* - responsible for defining user stories and prioritizing between the user stories. The project developed high-level requirements and solution descriptions, which were given to development teams. The project was manned with product owners and in total 30 employees[[4]](#footnote-4) from the line organization in the department. In addition, functional and technical architects from development teams contributed to this project.
* *Construction* - development was divided into three subprojects, one led by the public department Gamma (6 teams) with own people and people from five consulting companies. The two other subprojects were led by Alpha (3 teams) and Beta (3 teams). Teams worked according to Scrum with three week iterations, delivering on a common demonstration day. The scrum teams had additional roles than the scrum master, as as listed in Table 1. In addition to the 12 feature teams, the project had an environment team responsible for development and test environments.
* *Architecture* - responsible for defining the overall architecture in the programme. Consisted of a lead architect and technical architects from the development teams.
* *Test* - responsible for testing procedures and for approving deliverables from the development teams. Consisted of a lead tester as well as test resources from development teams.



*Figure 3: Project organization.*

In addition, there was a project for communication and adoption to prepare users for the new systems. As shown in Figure 3, the programme used a matrix structure, where the business and construction projects took part in the architecture and test projects.

Initially, development of each release was planned in four phases as shown in Figure 4: Establishing high-level requirements, solution description, construction and acceptance.



*Figure 3: Development process as planned initially.*

As we will describe later, this phase model was changed during the project.

Figure 2 shows how deliverables were organized in time, with three deliverables per year. As an example, deliverable 8 contained coupling of workflow in the office automation system to the archive solution, self-service solution for new legislation, simulation of services towards external public department and first data warehouse reports on new data warehouse architecture.

Each of the 12 releases in Omega contained a phase of solution description with the intention of ‘preparing the work’ for the actual construction during that release. An epic should be implementable during a specific Release (typically 1000 hours of work), while a User Story should be suitable for delivery (meeting the ‘definition of done’) during one iteration. In Omega the goal was that user stories should be of such a size that a development team could deliver four to seven user stories during each iteration. The user stories in the product backlog could contain both functional and non-functional requirements, they should be estimated and prioritized, and they should contain acceptance criteria (the definition of done.)

As the user stories were generated they were stored in an issue tracker[[5]](#footnote-5) together with the corresponding epic. Together they constituted the product backlog.

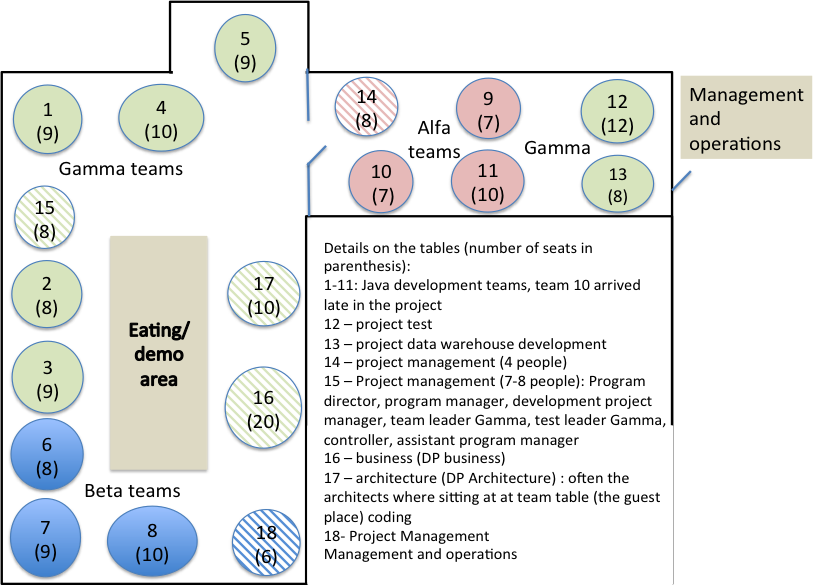
The customer managed the solution description work through the architecture project, and suppliers (Alpha, Beta and Gamma) participated on a time & material basis. Roughly 20 % of the suppliers' resources were consumed in solution description, 80 % in construction.

After approval from the programme, new deliverables were acceptance tested, set in production and underwent a trial period before being accepted by the operational IT section of the department.

The programme started working at the main office of the public department, but from 2009 it was moved to a separate office building located in the same city. Here, all teams were organized around tables as shown in Figure 5.

*Table 2: Roles in the teams.*

|  |  |
| --- | --- |
| Role | Description |
| Scrum master | Facilitated daily meetings, iteration planning, demonstration and retrospective. |
| Technical architect | Responsible for technical design, working 50% on this and 50% on development. |
| Functional architect | Responsible for this role was usually allocated 50% to analysis and design, and 50% to development. |
| Test responsible | Made sure that testing was conducted at team level: unit tests, integration tests, system tests and system integration tests. Delivered test criteria to the subproject test. |
| Developers | 4-5 developers were allocated to a team, a mixture of junior and senior developers. |



*Figure 5: Floor plan for the area where the development project was situated from 2009 until project end in 2012.*

## 4.2 Project management

## 4.3 Solution description

We found three key characteristics of solution description, first that solution description was teamwork, second that it was conducted continuously and iteratively, and third, that the boundaries between high-level requirements work and solution description and construction were blurred. We describe each of these characteristics in the following:

### Solution description teams

Solution description teams, consisting of people with various roles from project business and project construction, collaborated to produce sufficiently detailed work descriptions.

The solution description process was a part of the Architecture project (Figure 2). Being a horizontal element in the program organization structure, project architecture had responsibility for ensuring effective collaboration and knowledge sharing between the two verticals – project business and project construction.

The solution description team included people that would participate in the actual construction and people with relevant business skills from the customer. Membership could also span contractors: “If it was only Beta that was involved, then it [solution description] was a business architect, the business representative and some representatives from the Beta construction teams. If the work spanned contractors, we would involve people from Alpha, Beta or Gamma.” [Gamma, H.] In addition, other people believed to be particularly knowledgeable for the work to be planned could be invited.

At least one member from the construction team (usually the team architect) would participate in the solution description team. Furthermore, solution description work was spread among construction team members: “It wasn't one person's responsibility the whole time. It was natural that you did solution description for a while, and then we changed it to another from the same team to ensure that everybody knew what we were working on.” [BA2, p.24] This practice of involvement reduced the need for overly detailed solution descriptions. Another benefit was pointed out: “It gave good continuity in the work and helped ensure that the construction went smooth and easy.” [Gamma, GF3].

While it was seen as an effective practice regarding knowledge sharing and collaboration, and was taken into account during iteration planning it was challenging because the construction team had to give up team members for a long period – often the most skilled and often in the last phase of the iteration.

### A continuous and iterative process

Solution description was a continuous process in two dimensions. It was continuous inside one release, and it was going on in several releases at the same time. Solution description for release n would start while construction for release n-1 was still underway (Figure 3).

We learned that solution description work in Omega was characterized as “feeding the machine.” The metaphor effectively conveys the property of the construction project as a machine that requires raw material (here: requirements) to produce its products (construct the software system.) The main purpose of solution description in Omega was to ensure that the product backlog contained enough work to fill one to three iterations for the construction teams. <KILDE?> Furthermore, it was essential to give the machine sufficient raw materials to avoid idle time. Most of the construction project members were allocated full-time and therefore the cost of running the project ‘engine’ would be constant, whether the backlog was empty or filled. Hence, from a cost perspective, it was important ‘to keep the machine busy’ for as much of the time as possible.

The detailing level in solution description was subject to change during the project. As the construction teams gained experience, they understood more about the problem domain and less detail was required to convey the essential parts of the tasks. We learned that solution description work became more efficient because “they [the construction teams] understood what we wanted.” It is much easier to communicate effectively when you know the audience.

Furthermore, solution description was an iterative activity based in a pragmatic ambition to “detail as much as time permits.” [GF1, p.9] Hence, the level of detail was limited by the time available.

Working with a pipeline of completed solution descriptions as short as 1-3 iterations created tensions. There were noticeable differences between the three suppliers Alpha, Beta and Gamma in terms of preferences for up-front planning. Alpha had a preference for more detailed, up-front work descriptions. Alpha wanted reduced risk: “Regarding Gamma we had to ‘discipline’ the customer somewhat, to clarify what they wanted before we took it into our list of committed work, so that we get the least amount of rework.” [AF1, p.3].

Beta, on the other hand, was comfortable with more open specifications and continuous collaboration to resolve the details of the software in construction. Gamma had more up-front descriptions than Beta, but less than Alpha.

### Blurred boundaries between phases

Each release consisted of four consecutive phases: Identifying high-level requirements, solution description, construction and approval. “We could talk to anyone that might have the answer”. Customer representatives could approach the construction teams if they needed to discuss or clarify some issues. As a result, the boundaries between high-level requirements, solution description and construction were blurred: “In practice, high-level requirements and solution description merged more and more". This enabled a more efficient collaboration: "As the contractor starts to understand more about the context then the customer's real problem is more visible and this means we can find a solution together. Fast.” Blurring the boundaries also gave another interesting result. It became possible for people working on solution description to question output from the high-level requirement work. A contractor could challenge Gamma: “But you could do it this way instead? Why do it like this?” [Beta, BA2] Project participants found that this way of working stimulated creativity.

The iterative style of detailing the solution descriptions also contributed to blur the boundaries. For example, the acceptance criteria were supposed to be defined in the solution description phase but in practice they were often defined during construction by the teams, and our respondents were unable to agree on what phase the definition of acceptance criteria belonged to. Sometimes the solution descriptions were too vague to enable construction. In those situations the descriptions were passed back for revision.

We found pragmatism in the allocation of tasks to construction teams. We learned that often the most challenging tasks were given to Gamma construction teams because they had more domain knowledge and a more flexible contract model: “If there were many changes, if there were complicated modules and we didn't quite know how to design them, then the Gamma teams would get these tasks. I experienced that Gamma was more flexible … without the same constraints in relation to budgets and contracts.” [GF2, p.14]

## 4.4 Software architecture

Two topics emerged as primary challenges related to managing work on software architecture: The tension between up-front and emergent architecting, and the demanding role for architects in large-scale agile projects:

### Up-front versus emergent architecting

In February 2009, GA2 entered Omega as chief architect. At this point in time the architecture of the new system was described in terms of architectural strategies but with little compliance in the system under construction. The core principle of the initial architecture was ‘service orientation’ but it resulted in a complex system with many dependencies and high vulnerability in the event of changes. It was during this period decided to scale up Omega to more teams. GA2's most pressing concern was to create a system architecture that allowed the maximum parallelism of work in the different teams by “avoiding people stepping on each others feet.”

The chief architect and his team created a slightly less complex system architecture – focused around service modules and a horizontal ‘work surface’ to give users a unified environment for their daily tasks, across different service modules. The GUI layer, the “work surface” made use of all the service modules. The architecture also established rules of ownership to specific parts of the database. An important benefit of the architecture was that it helped to separate the work of different teams into different parts of the code. In Omega, we found that the software architecture for the most part allowed the teams to work autonomously and effectively.

Even if this architecture manifested important decisions for the continued work in the teams there were occurrences of tensions stemming from too little up-front work. <HADDE ALLE DEN HOLDNINGEN?> Our respondents stated wishes for more ‘proof of concept’ experiments before new technologies were adopted. A notable example was that the Flex technology used in the GUI layer had not been properly understood in order to use it effectively and establish sound ‘application architecture guidelines.’ Our respondents argued that the lack of proof of concepts was due to the contract model: “This was partly governed by the payment model and the program's execution. If they had allocated resources for concept trials that possibly would have to be thrown away, or that resources had been allocated for the development of multiple candidate solutions – but resources were not allocated for that.” [PB1, feedback session] At the end of the program, there was much work in optimizing the GUI layer code in order to obtain acceptable performance. Performance had been largely neglected for the main part of the project, but got significant attention at the final stages. This triggered some re-work for the architecture team: “When we started to use this technology [Flex] we established an architecture that was fit only for small applications and small solutions. It was not appropriate for us when we had to fill it up with more logic and GUI elements.” Also, refactoring was done in the object-persistence layer (Hibernate) on top of the relational database management system for performance reasons. These refactoring efforts were packaged into specific user stories for non-functional requirements.

The main architectural structure of the Omega system remained stable throughout the project. Mostly, the subsequent architectural work was local within the service modules. A particularly successful architectural design was task management within the work surface that was inspired from task management on Android devices.

The programme put emphasis on common architectural principles: “There needs to be openness with regards to suggesting architectural ideas from ‘below’ [in the teams], but you must also be able to convey merits of these ideas to convince the people in the appropriate forums so that it ends up as a unified architecture – so that each team does not use their own solutions.” [BA5, feedback]

### The demanding role of architects in agile projects

We learned that the architect role in agile projects is demanding because it requires continuous coordination and negotiation between many stakeholders, including representatives from the business side and the developers. The architect must be able to ‘sell’ technically sound structures and technologies that may have short-term negative impact on project progress and costs.

Another tension was the approach to decision-making in the Architecture project. For the most agile-oriented participants this top-down decision-model came to conflict with their own ambitions to satisfy the business representatives. Team members pointed to a tendency of project Architecture to step away from the discussions with business representatives when development teams pointed out that a particular architectural design had cost consequences for the business representatives. When developers argued to the business representatives that a proposed architectural approach would give penalties in terms of cost or performance, sub-project architecture responded that it was unfair of developers ‘to hide behind the business people’ instead of complying with architectural principles.

However, architecture work was increasingly more difficult for project Architecture as well. In the last half of the second year the pressure to deliver functionality increased: “At this point project management and other management was concerned about progress ... It became much more difficult to promote improvements [in the architecture].” [GF5, p.20] “Yes, more difficult. And then came these issues regarding technical debt. You know? Issues that are horrible in an agile setting.” [GA2, p.20] We see that architectural work in agile projects became more integrated, having to relate to a wider range of people.

Due to the strict schedule in the programme, the business needs were often given priority to long-term architecture. Refactoring had to be sold to the business representatives at the expense of functionality. We learned that team architects took steps to avoid this: “For me it was equally important to establish the right attitude in the teams regarding code quality as it was to make the business side allocate resources for refactoring.” [BA2, feedback session].

The two suppliers Alpha and Beta used different approaches to architecture work during the project. Alpha made many architectural decisions during the solution description phase, while Beta delayed many such decisions to the Iterations. Alpha's motive was to ensure the highest possible efficiency of the teams' work during the Iteration. The culture in Beta appeared more flexible and collaborative, seeking rather to clarify issues as they emerged.

## 4.5 Mechanisms for inter-team coordination and knowledge sharing

When discussing mechanisms for coordination and knowledge-sharing between teams in the development project, we identify three main findings: First, that there was a number of mechanisms involved in order to achieve coordination and knowledge sharing. Second, that the use of these arenas changed over time, and third, that many emphasize the importance of the informal coordination arenas which were enabled by radical co-location. We describe results from each of these three areas in the following:

### A number of mechanisms

Each feature-team would follow Scrum practices like having a planning meeting, daily stand-up meetings, a retrospective and demo for each three-week iteration. In order to coordinate work and share knowledge between the teams, we learned that there were a number of arenas in use as shown in Table 3. This list includes mainly formal arenas such as the Scrum of Scrums, and the Metascrum at a level above the Scrum of Scrum. But also a number of informal arenas such as experience forums, lunch seminars and instant messaging. Some of the arenas were only used within a subproject, such as the "experience forum" at Alpha and the "technical corner" as Beta.

*Table 3: Coordination arenas used in the Omega project.*

|  |  |
| --- | --- |
| *Mechanism* | *Description* |
| Scrum of Scrums | All three sub-projects had scrum of scrum meetings with their teams, 2-3 times a week. Scrum masters and project manager attended, and sometimes others such as product owners and test managers. |
| Metascrum | Two meetings per week with project managers from the development, architecture, test and the business projects, as well as subproject managers from the development projects. |
| Demo | Demonstration of developed user stories after every iteration, from morning until lunch. Everyone could participate. 10 minutes devoted to each team. |
| Retrospectives | Mainly used on team level, but a few times on sub-project level. All retrospectives were documented in the project wiki. |
| Experience forum | A forum at subcontractor *Alpha* for scrum masters, development manger and agile coach focusing on development method. |
| Lunch seminars | Seminar where 2-3 persons gave short presentations during lunch on topics such as new architectural components, project management or on how to follow up on a team. |
| Open space technology | A process where all participants suggested topics for discussion, which is made into an agenda and participants are free to join discussion groups of interest. Used per release during parts of the project. |
| Rotation of team members | Members sometimes rotated between teams, in particular in build-up phase when existing teams were split and new members added to all teams for a subproject. |
| Board discussions | Every team had a board with tasks and status on one side and free space for drawing on the backside. The backside was used frequently for informal discussions. |
| Technical corner | Architectural briefings in the subproject at *Beta,* where team architects briefed the teams. About 1.5 hours, used in the beginning of the project. |
| Wiki | Architectural guidelines, team routines, cross-team routines, system documentation, reports from retrospectives, solution descriptions and functional tests were documented on the project wiki. |
| Instant messaging | Instant messaging was set up after a need was identified in an open space session. Was used for open technical questions but also for social activities such as wine lottery. |
| Radical co-location | From 2009, the project with all teams and project management was situated in an open-plan office space on one floor. |
| Masterplan | The programme established a common product backlog as a master plan, with the 2500 user stories organized into 300 epics. The master plan was maintained in an issue tracker. |

### Changing arenas over time

A characteristic of the Omega programme was that arenas for coordination was changing over time.

*"There were meetings that came and disappeared, and some that remained"* (functional architect).

There were information needs that the programme saw was not covered, and new arenas were then established. Also, arenas were abandoned because the programme found that they were no longer useful. Open space is an example of an arena that existed for a while and then was abandoned. Some state that this arena had an effect:

*"during a phase, I felt it had a mission, in particular to motivate people"* (subproject manager)

while other were more critical to the practice:

*"...but in total I feel that we did not get much out of it. Maybe we got to know each other better. Maybe."* (another subproject manager)

Several commented on the change from formal mechanisms to informal mechanisms. Over time there was:

*"more usage of boards, more common coffee breaks, more lunches"* (subproject manager)

Some arenas changed function over time. Internal meetings within the Alpha subproject over time developed into an arena for sharing technical knowledge. This illustrates that there was both a change in arenas and in how arenas functioned over time.

### Radical co-location

From 2009, the project was co-located in an open-plan office floor. The teams were organized around tables, and project management for the whole project as well for the subprojects were on the same floor (see Figure 5). Many stated that being on one floor contributed to efficient coordination and knowledge sharing.

*"It was the best possible solution that the project managers were at one table, in immediate reach of the development teams"* (subproject manager)

*"I think being on the same floor is important. That is something I notice now being at "Zeta" [another large development programme], where we are not located on the same floor, it is much harder to know what is going on"* (another subproject manger)

We learned that much of the decisions that were made in the project were discussed between relevant stakeholders informally, and then officially decided in one of the formal arenas, the daily scrums, scrum of scrums or in the metascrum.

Although mainly seen as vital to coordination and knowledge sharing, loud discussions on team tables in the open landscape led to some starting to use earphones in order to avoid noise, and to indicate that they were not to be disturbed while working.

# 5. Discussion

Our research question is *how does a large-scale development project balance agility and discipline?* From the results we see that there was a number of tensions in the project in order to balance agility and discipline. In the following, we elaborate of some of these tensions that we propose are especially interesting themes for future research on large-scale development projects. We formulate research questions that should be further addressed in future research.

**Self-management versus control in large projects**

Self-managing teams offer potential advantages over traditionally managed teams because they bring decision-making authority to the level of operational problems and uncertainties and thus increase the speed and accuracy of problem solving. In large projects, there are a number of problems that span development teams, and decisions that affect a number of teams. Omega found a balance between self-management and a centralized decision structure. In the results section, we describe how

**RQ3: How can coordination in large-scale agile development projects be understood by drawing on coordination theories from management of large-scale projects?**

Points in favour of agility:

\* self-managing

\* gradual focus on more informal arenas

\* blurred boundaries between projects

\* continous solution description

\* solution descripton teams

\* architectural decisions from "below"

Points in favour of discipline:

\* Project organisatoin; architecture and solution description as separate projects

\* Roles at team level

\* Control points; end of iterations

\* Meeting structure (dependency detection, bugboards)

\* Rythm: Maskinmetafor

**Functionality versus architecture**

Points in favour of agility:

\* Release plan; "masterplan"

\* Continous and iterative solution description

\* Solution description teams

Points in favour of discipline:

\* Architecture project

\* Upfront planning of architecture (little changes, yet calls for more upfront design)

**Blurred boundaries**

Points in favour of agility:

\* Mindre fokus på faser

\* Mindre fokus på roller

\* åpent landskap

\* forskyving mellom faser

\* slack

Points in favour of discipline:

\* Initiell fokus på faser, roller

\* Organisation structure

-----

\* Inter-team coordination

\* Zipper-model?

\* CoPs?

\* The role of architecture for inter-team coordination

\* Little uncertainty / little change

\* Formal and informal controls in large-scale projects

\*Masterplan

\* What needs to be scaled?

\* Impact on practices

Agile development seems to increase the complexity (…) of architecting, and the architect role:

* Agile architecting demands a continuous, effective and integrative dialogue regarding benefits, disadvantages and trade-offs between architectural decisions.
* Finding sponsors for non-functional requirements in agile projects can be difficult.
* Managing bottom-up architecture design in the large. Many good architecture ideas come from the individual teams, but the need for standardization (across teams) creates friction with other teams. [Tradeoff between ‘ownership’ to local solutions and acceptance for standardization.]
* Ambler has suggested ways to describe and communicate the architecture in his book on agile modeling (Ambler, 2002), but there are no studies on what practices are used in large-scale projects and how successful they are in structuring the development process and in ensuring a high quality product. There are several related research areas that can improve the understanding of architectural evolution. Naturalistic decision making (Klein, 2008) focuses on how decisions are made in situations with high time pressure, stakes and uncertainty, and with team and organizational constraints. Another relevant field is studies of how shared mental models in teams has a direct impact on team productivity (Mathieu et al., 2008). Further, many have focused on managing architectural knowledge in traditional projects (Babar et al., 2009), but there has been little attention on how to do this in agile projects.

---

Architecture has a key role in defining how work is coordinated in large-scale development efforts.

The level of change and level of uncertainty will influence how the architecture work should be organized.

RQ3: How can coordination in large-scale agile development projects be understood by drawing on coordination theories from management of large-scale projects?

RQ4: What are challenges and success factors in approaches to define and maintain the software architecture in large-scale agile development projects?

Common norms and values facilitate inter-team coordination.

Effective knowledge networks are essential in large-scale development due to the knowledge-intensive nature of software development.

--

Describing the context for agility and scale is essential for understanding how to improve agility in large-scale agile.

For large-scale embedded systems development, agility should scale both with respect to the number of involved teams, and the systems engineering activities in each iteration due to the co-dependency of software and hardware development.

* SOLUTION DESCRIPTION:
* Requirements engineering takes a new form in agile development (Fitzgerald et al., 2006). In Omega we saw that the process turned into a dialogue-driven process where cooperation between the different stakeholders enabled efficient clarification of the tasks to be implemented in the final system.
* [An important enabler for continuous solution description was the collaborative, trusting working atmosphere among partners. Changes in the formal agreements – guarantees removed. Co-location and the possibility to quickly resolve issues was important.]
* We argue that it is an example of an on-going conversation between the customer and the developers with the intention to prepare the construction teams for the work to be done.

## Limitations

Time

Number of researchers

Complexity of case

Management bias

# 6. Conclusion

# Acknowledgement

# Appendix: Interview guides

# References

ABRAHAMSSON, P., BABAR, M. A. & KRUCHTEN, P. 2010. Agility and Architecture: Can They Coexist? Introduction. *IEEE Software,* 27**,** 16-22.

AMBLER, S. 2002. *Agile Modelling: Effective Practices for eXtreme Programming and the Unified Process,* New York, Wiley.

BABAR, M. A., DINGSØYR, T., LAGO, P. & VLIET, H. V. 2009. *Software Architecture Knowledge Management: Theory and Practice,* Berlin Heidelberg, Springer Verlag.

CONBOY, K. 2009. Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. *Information Systems Research,* 20**,** 329-354.

DINGSØYR, T., FÆGRI, T. E. & ITKONEN, J. What is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development. Accepted for publication at Profes 2014, 2014 Helsinki. Springer Verlag.

DINGSØYR, T., NERUR, S., BALIJEPALLY, V. & MOE, N. B. 2012. A Decade of Agile Methodologies: Towards Explaining Agile Software Development. *Journal of Systems and Software,* 85**,** 1213-1221.

ECKSTEIN, J. 2004. *Agile Software Development in the Large,* New York, Dorset House.

FABER, R. 2010. Architects as Service Providers. *IEEE Software,* 27**,** 33-40.

FITZGERALD, B., HARTNETT, G. & CONBOY, K. 2006. Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems,* 15**,** 200-213.

FLYVBJERG, B. & BUDZIER, A. 2011. Why Your IT Project May Be Riskier Than You Think. *Harvard Business Review,* 89**,** 23-25.

FREUDENBERG, S. & SHARP, H. 2010. The Top 10 Burning Research Questions from Practitioners. *IEEE Software***,** 8-9.

INGVALDSEN, J. A. & ROLFSEN, M. 2012. Autonomous work groups and the challenge of inter-group coordination. *Human Relations,* 65**,** 861-881.

KLEIN, G. 2008. Naturalistic decision making. *Human Factors,* 50**,** 456-460.

MATHIEU, J., MAYNARD, M. T., RAPP, T. & GILSON, L. 2008. Team effectiveness 1997-2007: A review of recent advancements and a glimpse into the future. *Journal of Management,* 34**,** 410-476.

MOE, N. B., SMITE, D., SABLIS, A., BØRJESSON, A.-L. & EASSON, P. A. 2014. Networking in a large-scale distributed agile project. *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.* Torino, Italy: ACM.

NERUR, S., MAHAPATRA, R. & MANGALARAJ, G. 2005. Challenges of migrating to agile methodologies. *Communications of the ACM,* 48**,** 72-78.

PAASIVAARA, M., LASSENIUS, C. & HEIKKILA, V. T. 2012a. Inter-team Coordination in Large-Scale Globally Distributed Scrum: Do Scrum-of-Scrums Really Work? *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'12)***,** 235-238.

PAASIVAARA, M., LASSENIUS, C., HEIKKILA, V. T. & IEEE 2012b. Inter-team Coordination in Large-Scale Globally Distributed Scrum: Do Scrum-of-Scrums Really Work? *Proceedings of the Acm-Ieee International Symposium on Empirical Software Engineering and Measurement.* New York: Ieee.

RAJLICH, V. 2006. Changing the paradigm of Software Engineering. *Communications of the ACM,* 49**,** 67 - 70.

RUNESON, P. & HÖST, M. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering,* 14**,** 131-164.

SALDAÑA, J. 2009. *The coding manual for qualitative researchers*, Sage Publications Ltd.

SMOLANDER, K. 2002. Four metaphors of architecture in software organizations: Finding out the meaning of architecture in practice. *Proceedings 2002 International Symposium on Empirical Software Engineering.* Nara, Japan: IEEE Computer Society.

STEWART, D. W., SHAMDASANI, P. N. & ROOK, D. 2007. *Focus Groups: Theory and Practice*, Sage Publications.

STRODE, D. E., HUFF, S. L., HOPE, B. G. & LINK, S. 2012. Coordination in co-located agile software development projects. *Journal of Systems and Software,* 85**,** 1222-1238.

WENGER, E. 1998. *Communities of practise : learning, meaning and identity,* Cambridge, UK, Cambridge University Press.

WILLIAMS, L. & COCKBURN, A. 2003. Agile Software Development: It’s about Feedback and Change. *IEEE Computer,* 36**,** 39-43.

1. Scott Ambler: Agile Adoption Rate Survey 2008, http://www.ambysoft.com/surveys/agileFebruary2008.html [↑](#footnote-ref-1)
2. Source: official report from the project. [↑](#footnote-ref-2)
3. Flex was developed by Adobe as a framework for "expressive web application", and is now an open source product. [↑](#footnote-ref-3)
4. The number of peple involved in the project varied, we use numbers in the peak period in the project, from 2009 to 2011. [↑](#footnote-ref-4)
5. The project used Jira from Atlassian. [↑](#footnote-ref-5)