4. Results

* Alle figurer tegnes om for å forenkles og ha samme stil

* Foreløpig er programmet (prosjektet) og firma anonymisert, men vi endrer til riktige navn om det ok for SPK og hovedleverandørene.

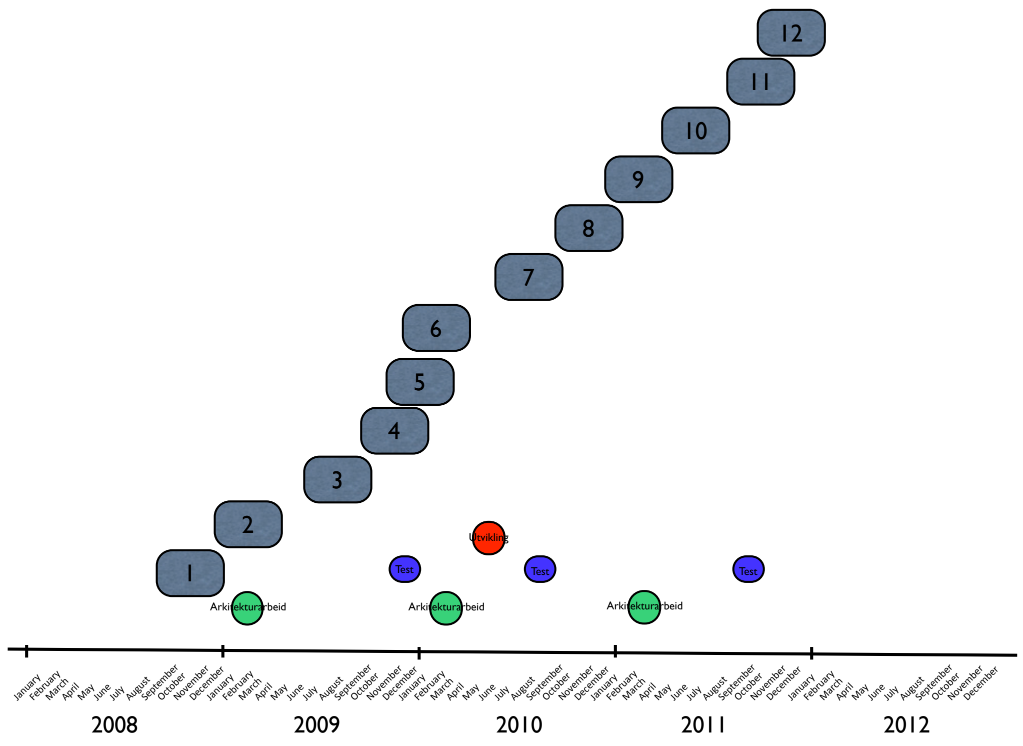
4.1 The Omega programme

The Omega programme was conducted by the public department Gamma, which needed a new office automation system. The main arguments for initiating the programme was a public reform, and that the existing office automation system was on a platform, which was to be abandoned.[2] When the programme started, the content of the public reform was 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 provide accurate and timely services to customers and to ensure a cost-effective implementation of the reform. Because of the reform, the programme had a strict deadline. It is one of the largest IT programmes in Norway, with a final budget of about EUR 140 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.

[2] Source: official report from the project. This description is mainly based on this report and an internal experience report.



(Erstattes av figur som viser innsats i hver leveranse og tar bort "peaks" av arbeid i prosjektene).

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

As an example, release 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. Most user stories were identified prior to the first release, but were supplemented and reprioritised for every release.

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 provides a richer interface using Flex.[3] The database from the old system was kept, but the data model was changed. The regulations and legislations are implemented in

[3] Flex was developed by Adobe as a framework for "expressive web application", and is now an open source product.

the new system as rules, using 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.

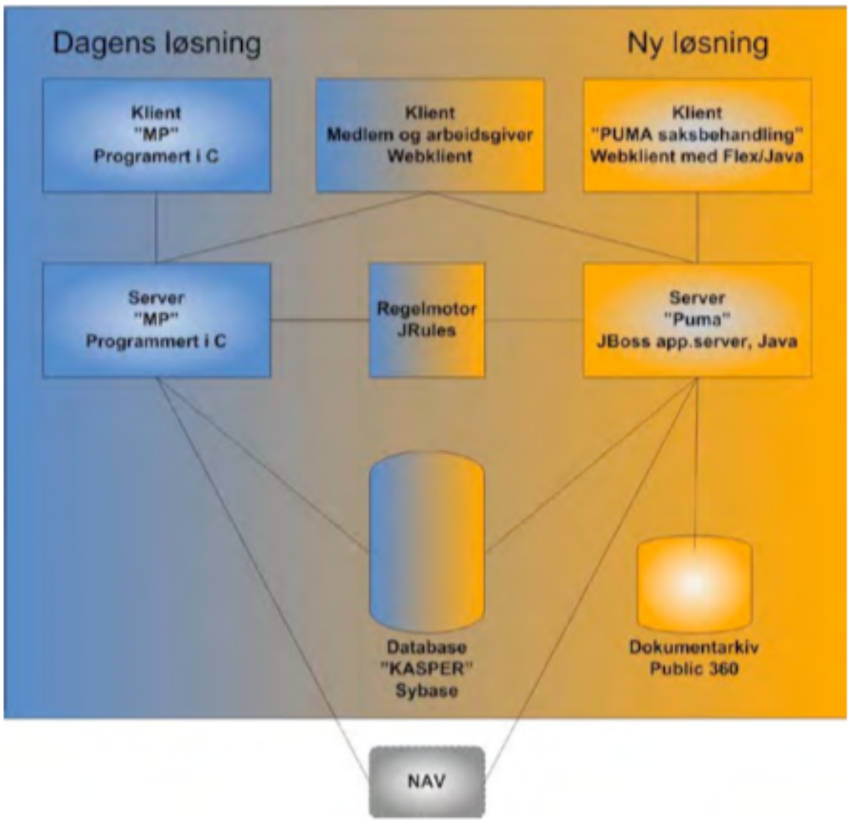

Dagens løsning
Ny løsning
Klient "MP" Programert i C
Klient Medlem og arbeidsgiver Webklient
Klient "PUMA saksbehandling" Webklient med Flex/Java
Server "MP" Programmert i C
Regelmotor JRules
Server "Puma" JBoss app.server, Java
Database "KASPER" Sybase
Dokumentarkiv Public 360
NAV

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

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 analysis of needs through defining and prioritizing epics and user stories in a product backlog. This project was manned with product owners and in total 30 employees[4] from the line

[4] The number of people involved in the project varied, we use numbers in the peak period in the project, from 2009 to 2011.

organization in the department. In addition, functional and technical architects from development teams contributed to this project.

- *Architecture* - responsible for defining the overall architecture in the programme and also for detailing user stories in the solution description phase. Consisted of a lead architect and technical architects from the feature teams. Suppliers Alpha and Beta participated on a time & material basis.
- *Development* - 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). The feature teams worked according to scrum with three-week iterations, delivering on a common demonstration day. The feature teams had additional roles than the scrum master, as listed in Table 2. In addition to the 12 feature teams, the project had an environment team responsible for development and test environments.
- *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.

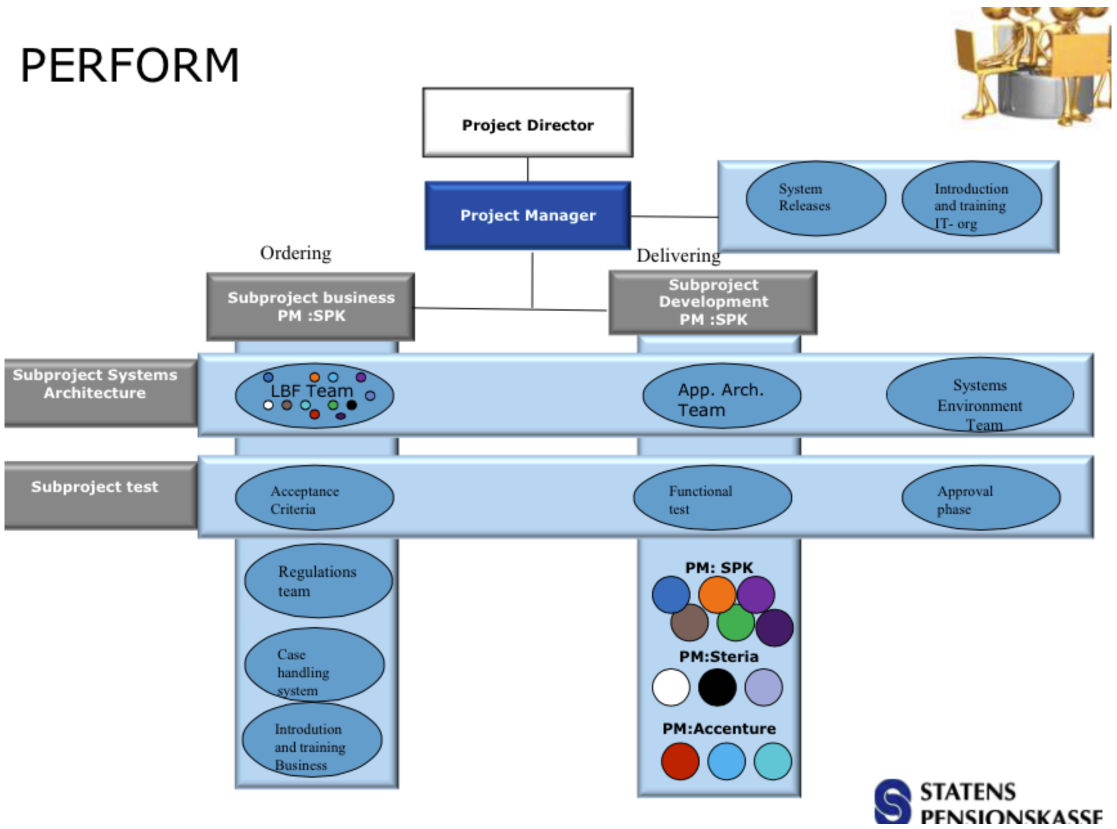


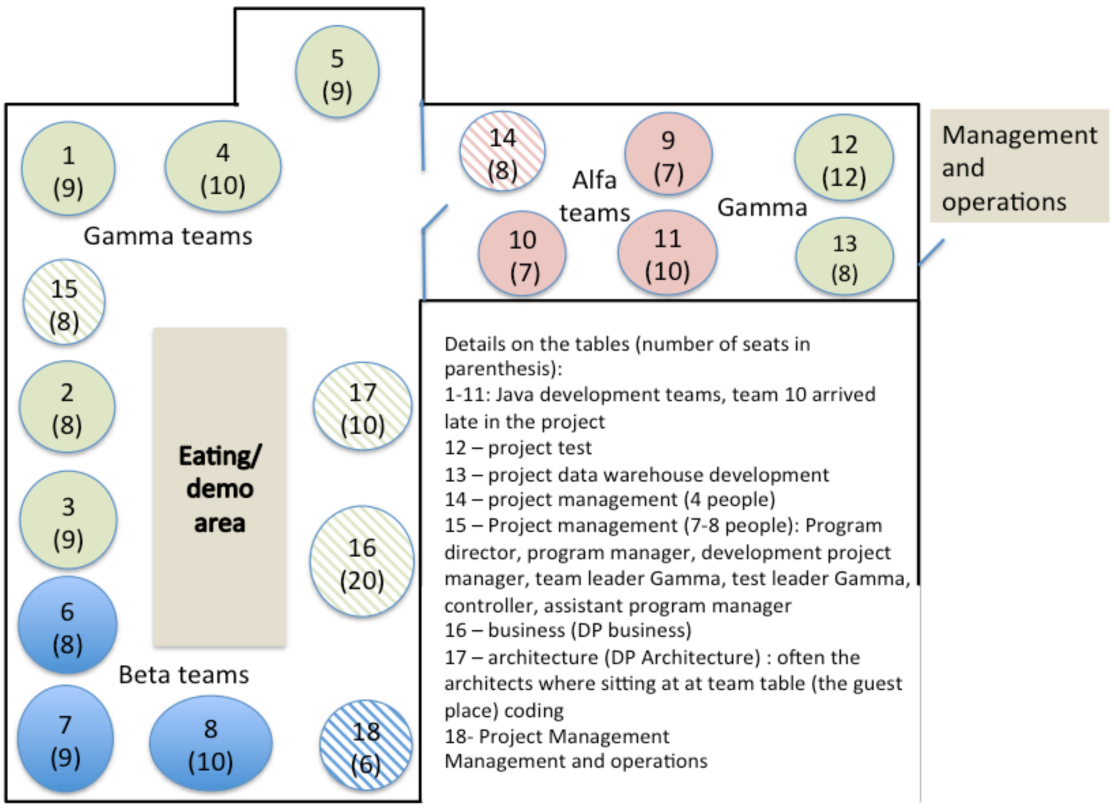
Figure 3: Project organization (Source: Presentation by Mette Gjertsen)

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.

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.

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 4.



Details on the tables (number of seats in parenthesis):
1-11: Java development teams, team 10 arrived late in the project
12 – project test
13 – project data warehouse development
14 – project management (4 people)
15 – Project management (7-8 people): Program director, program manager, development project manager, team leader Gamma, test leader Gamma, controller, assistant program manager
16 – business (DP business)
17 – architecture (DP Architecture) : often the architects where sitting at at team table (the guest place) coding
18- Project Management
Management and operations

Figure 4: Floor plan for the area where the development project was situated from 2009 until project end in 2012. (TODO: reduce complexity of figure)

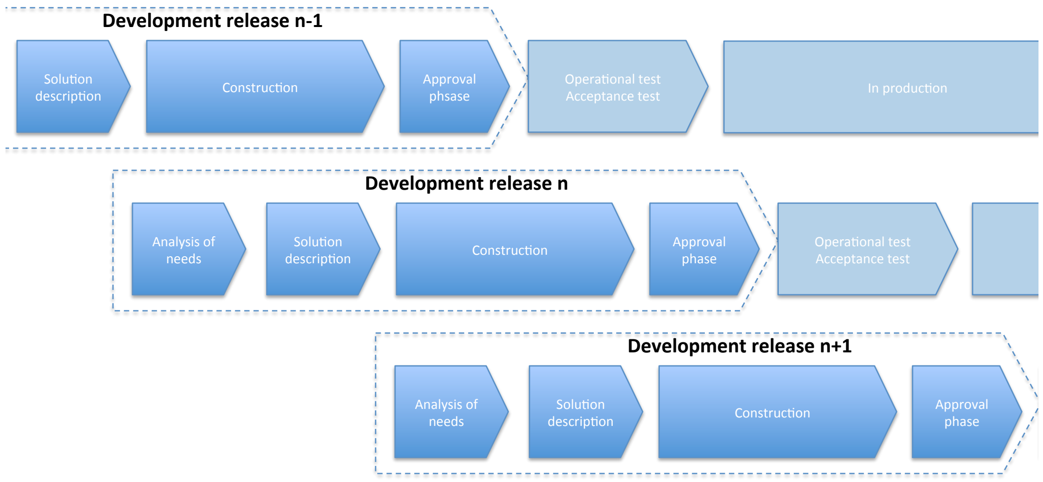


Figure 5: Initial development process.

Initially, the development process included the four phases described in Figure 5:

- *Analysis of needs* - this phase starts with a walkthrough of target functionality of a release, and identification of user stories. The product backlog is prioritized by product owners.
- *Solution description* - the user stories are assigned to epics, and the user stories are described more in detail, including design and architectural choices. User stories are estimated and assigned to a feature team.
- *Construction* - development and delivery of functionally tested solutions from the product backlog. Five to seven iterations per release.
- *Approval* - a formal functional and non-functional test to verify that the whole release works according to expectations. This includes internal and external interfaces as well at interplay between systems.

In order to keep the schedule of the project, there was a pressure to have solution descriptions ready for the feature teams. This meant that releases were constantly under planning, being constructed and under test. This work structure was then reflected down to work in the feature teams, given the roles listed in Table 2.

After approval from the programme, new releases were acceptance tested, set in production and underwent an approval phase before being accepted by the operational IT section of the department.

4.2 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 analysis of needs and solution description and construction were blurred. We describe each of these characteristics in the following:

Solution description teams

The solution description team included people that participated in the 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 5).

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. 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.

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: Analysis of needs, 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, and 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 feature 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]

4.3 Software architecture

Two topics emerged as primary characteristics 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 simpler 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 tensions stemming from too little up-front work. 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. Functionality was given priority over refactoring. We learned that team architects took steps to avoid integrating refactoring into normal work tasks: “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 Construction. 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.

4.4 Arenas for inter-team coordination and knowledge sharing

When discussing arenas for coordination and knowledge-sharing between feature teams in the development project, we identify three main findings: First, that there was a number of arenas 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 arenas

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*
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.
Demo	Demonstration of developed user stories after every iteration, from morning until lunch. Everyone could participate. 10 minutes devoted to each team. To get rapid feedback, feature teams started organizing "mini demos" within iterations.
Experience forum	A forum at subcontractor *Alpha* for scrum masters, development manager and agile coach focusing on development method.
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.
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.
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.
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.
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.
Radical co-location	From 2009, the project with all teams and project management was situated in an open-plan office space on one floor.
Retrospectives	Mainly used on team level, but a few times on sub-project level. All retrospectives were documented in the project wiki.
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.
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.
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.

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 programme as well for the projects 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.