Improving the Interface Between Business and Product Development Using Agile Practices and the Cycles of Control Framework

Jari Vanhanen
Helsinki University of Technology
jari.vanhanen@hut.fi

Juha Itkonen
Helsinki University of Technology
juha.itkonen@hut.fi

Petteri Sulonen
Avain Technologies Oy
petteri.sulonen@avaintec.com

Abstract

The paper describes how we created and adopted an agile product development process in a small software company based on the Cycles of Control framework by combining selected agile practices and principles from the Scrum and XP methodologies. Describing the development process using the framework helped in identifying the crucial control points between business and development and enabled defining practical and well-functioning connections between them. The control points enable visibility and flexible management of product development status and direction. Currently Business understands development status better, which has led to fewer interruptions between the control points, and thus improved working conditions for Development. Positive experiences are reported of newly adopted practices such as scrum meetings, pair programming, and unit testing. However, finding and adopting technical tools to facilitate the process proved to be challenging.

1. Introduction

Sooner or later, most growing software companies face the challenge of bringing more structure to their product development process. The reasons are many, e.g., getting more predictability to the release schedules, providing visibility to the development progress, improving software quality, and facilitating initiation of new workers. Recently, we created and adopted an agile product development process for a small but growing software company facing these and many other challenges. The new process was created based on the Cycles of Control framework [3] and by combining selected agile practices and principles from the Scrum and XP methodologies into this framework.

1.1. The company

The company, Avain Technologies Oy, specializes in building secure digital transaction solutions. While delivering several customer-specific solutions, the company has simultaneously created a first product version out of their solution. The product is a system for secure digital signatures of XML forms over the Internet.

Previously the company had only a few developers, who mostly worked for customer-specific projects. They followed ad-hoc development practices but managed to perform well as a small team. At present, the strategic focus of the company is to move to the product business and grow the company to become a global player in the market. This means increases in the number of developers and other personnel and strict requirements for software quality. The need for a more rigorously defined development process is evident. However, the company does not want to lose flexibility, efficiency, and innovative work culture.

1.2. The Cycles of Control framework

Cycles of Control [3] is a general framework for managing software product development in small organizations created in the SEMS research project at Helsinki University of Technology. It provides a common language in which the practices, pacing and phasing of the incremental software development process can be communicated to the whole organization.

The basic idea of an iterative and incremental development process is to deliver working software early to get user feedback on it. At the same time technical feedback on system performance or other non-functional aspects can be made available. The feedback is used in planning the subsequent development cycle(s).

The framework (Figure 1) combines business and product development processes through four cycles of control: (1) portfolio management provides the interface between business and product development and manages the set of products and services offered by the company, for example, by product roadmapping, (2) release project management handles the development of individual product versions, (3) increment management deals with the incremental development of product functionality within release projects, and, (4) heartbeats are used for daily or weekly task scheduling and monitoring and synchronizing the effort of individuals or teams to get an indication of system status during development. Portfolio management should also provide guidelines for how product development efforts should be organized in terms of release cycle length and development rhythm [6].

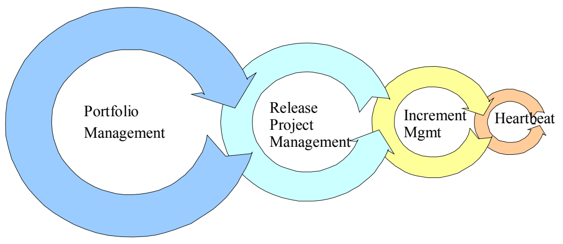

Figure 1. The Cycles of Control framework

1.3. Creating the new development process

The process definition and improvement work at Avain Technologies began in October 2002. The co-operation between the company and three researchers of the SEMS research project began around the same time. The product development manager led the process improvement work, spending a considerable amount of his work time for this from October to December. The researchers participated actively by observing, commenting on the plans and providing new ideas and references. The process improvement began with a kick-off day, where the whole development team and the researchers participated. Later, senior developers and the managing director participated in some of the meetings where the new process was discussed. Other developers read through the process descriptions and were encouraged to comment on them.

Several weaknesses in the current ad-hoc development process had already been identified. The unpredictability of release dates and vague ideas of their content made planning hard for the sales and marketing departments. The process could not be efficiently scaled up for a larger team, or taught to another team. Transfer of knowledge within the team did not work well. The valuable time of senior developers was wasted in customer support and maintenance tasks. The challenge was to make it possible to plan, predict, and steer the development and yet to maintain the responsiveness. The answer was to devise a minimum-overhead process that provides the necessary degree of control, and to create a separate service team for supporting the existing customer projects and installations.

First, the new process and practices were brainstormed in a rather ad-hoc manner. The ideas were documented as a narrative text document. However, the communication of the process was problematic, because it was hard to form a common vocabulary and understanding of the concepts that were used to describe the process. Soon after that, the Cycles of Control framework was introduced in the company and used to give clear cyclic structure for the process. The framework also clarified the terms and concepts for all stakeholders and thus improved communication and made process construction significantly easier. Some agile practices were introduced as a part of the implemented process to define the work practices in the heartbeats.

2. The new development process

In this section we describe the proposal for a new development process as it was initially adopted at Avain Technologies. It was clear that the process did not yet cover all activities and was not perfect, but the best way to improve it was through piloting. We present the experiences of the adoption, and the changes and improvements to the process during the first release cycle in Section 3.

2.1. Organizational structure

The company has four departments: Business, Development, Service, and Administration. The managing director, two sales managers and the international functions manager, who are responsible for the sales and marketing, represent Business. Development contains the product development manager, five developers and a tester. The development team is split into three teams, two product teams and one team producing special components for the products. One of the developers works in a different country, but others are co-located in two big rooms next to each other. Service takes care of delivering solutions to the customers.

Improving the communication between Business and Development was a high priority challenge for the new process, and is in a key role in the development process description. This can be seen in the strong emphasis on requirements management. An overview of the rhythm of the development and the control points available for Business to change development plans can be seen in Figure 2.

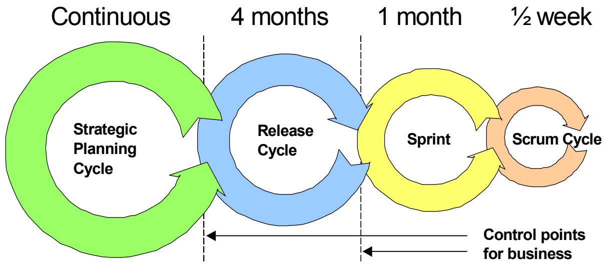

Figure 2. The implementation of the Cycles of Control framework at Avaintec

2.2. Cycles of Control

The control cycles were adopted straight from the Cycles of Control framework. The practices for managing requirements in each of these cycles are largely affected by the backlog management practices in Scrum [4].

In the **strategic planning cycle**, a coherent product vision is continuously maintained. The product vision is expressed as a Product Backlog (PBL) and an idea pool. The PBL contains a prioritized list of requirements to be implemented sometime in the future. All changes to the content and priorities of the PBL go through the managing director. The idea pool contains all kinds of ideas for the product's future not yet in the PBL. Anyone can commit ideas to it.

A feasibility study must be made for any item before proposing it to the PBL. The depth and scope of feasibility studies vary depending on the scope and immediacy of the requirement being studied. At its simplest it can mean a rough estimate by Business about the commercial feasibility of the item followed by an equally rough estimate about the effort required to develop and maintain it and other possible technical implications from Development. At its most complex it can mean two sprints worth of requirements and architectural exploration for a large set of new functionality.

The **release cycle** results in a release that can be either commercial or internal. There may be different types of releases of different products in the end of the same release cycle. Commercial releases are made based on strategic decisions, e.g., once a year. Other releases are internal and their main purpose is to converge the work regularly showing the real progress status.

The release cycle contains three phases: planning & exploration, development and stabilization. The lengths of the phases are planned at the beginning of the release cycle and embedded into the four one-month sprints. The lengths may vary a lot, e.g., the planning & exploration phase may take from a minimum of one day to even a whole sprint. In the planning & exploration phase Business and Development play a kind of planning game, where the highest priority items in the PBL are discussed in more detail, and estimated by Development. The results of the game are the definition of the release goal, preliminary sprint goals and sprint themes, and a prioritized list of items for the release, i.e. the Release Backlog (RBL). An item may be a feature, high-level architectural task, or exploration of, e.g., domain-specific standards or legislation. The priorities are based on both architectural and business criticality, the former having more weight in a case of a conflict. After the development phase the system is acceptance tested and documented in the stabilization phase.

At the beginning of the **sprint cycle**, Business and Development hold a sprint-planning meeting, where the sprint goal and theme(s) are revised. Here Business may propose changes to the RBL as long as achieving the release goal is not compromised. Then development picks a reasonable amount of items from the top of the RBL and expands them to tasks. These tasks form the Sprint Backlog (SBL), i.e. the plan for the sprint. In the end of the sprint Development holds an internal sprint debrief session and gives a demo for Business in order to communicate what has been accomplished.

At the beginning of the **scrum cycle**, Development has a status meeting. Based on these meetings the scrum master, who is the leader of scrum meetings and responsible for managing the development work at the scrum level, maintains the SBL and tracks the progress of the sprint.

Feedback from each of the cycles is propagated to the previous cycle in a pre-defined rhythm via scrum meetings, sprint demos, and releases.

The role of general goals for the cycles is to push decisions of details later and allow being more adaptive to changes. Upper management need not be involved in accepting changes as long as the cycle's goal is not compromised. The control points are clearly defined both in schedule (timing of cycles) and in the significance of the accepted changes (goals of the cycles). Only highly critical changes in the environment may override these control points.

2.3. Development practices

All developers agreed on using some development practices: automated unit testing, pair programming [7] in a slightly modified form (real pair programming for hard tasks, otherwise close co-operation between two developers), CVS for version control, and coding standards. XP practices [1], simple design, collective ownership, continuous refactoring, 40-hour week, were listed in the process definition, but they were not emphasized to the developers as much as the other practices.

"Red-Flag"-practice is used to manage unexpected work not related to achieving the sprint goal, such as preparing demonstrations for potential customers or fixing problems in previous customer installations. Red flags are included in the SBL as high-priority tasks. A constant amount of effort (40%) is reserved for non-development work, including red flags.

2.4. Quality Assurance

Before the first release, the focus of the process definition and improvement work had been on the overall structure, pacing, cycles and the core development practices. Because the communication between Business and Development was one of the main issues, the process had been defined from the requirements management viewpoint and the testing and quality assurance issues were intentionally left with less attention. Also, there were not many testable features expected from the first release due to its technical nature.

By the first release the defined quality assurance practices were automated unit tests written by the developers and the test-first practice for writing those unit tests. In addition, the process description stated that before a sprint backlog item could be labeled as completed, the system tests for that functionality must be created and the test results provided to the scrum master to support the decision of completeness.

The process work on QA practices continued during the first release. The primary focus areas in this work were defining the defect management process, deploying the Bugzilla open source defect tracking system and describing the hand-off and release procedures between developers, testing and the service team. These hand-off procedures between developers, testing and service are essential because cyclic, incremental development is based on frequent code hand-offs between different stakeholders. The communication that accompanies these hand-offs is the most important and accurate information on the current functionality and status of the system since no prescriptive specifications are created beyond the backlog items. This means that the system testing will be based on the hand-off documentation as well as communication and collaboration between the developers and testing.

The testing processes were not defined during the first release cycle and the tester was hired to the company only after the release cycle had started. In addition, the testing environments and hardware were not established during the first release, which made it difficult to start system testing during the first release cycle.

3. Realized process

The first release has now been carried out using the new process. The SEMS researchers have been observing and affecting the process improvement work during the first release, and the representatives from both Business and Development were interviewed after the release about how they had followed the new process and felt about it.

Quantitative data is scarce because neither the old nor the new process contains more than very simple basic metrics. Thus the evaluation of the new process is largely based on the subjective opinions of the employees.

3.1. Organizational structure

The structure of the development team was changed after the third sprint from the product and component team based division to front-end/back-end division. The problem was that the component team could not successfully synchronize their work with the product teams. In the new structure both products have a responsible lead developer, and the "external" component team has been embedded to the back-end team. The new division has improved the customer/producer relationship between the developers, and allows developers to work better within their area of expertise. For example, if the lead developer of product 1, who is in the front-end team, recognizes that a feature would best be implemented as a back-end component, he can order one from the back-end team. Formerly, the path of least resistance would have been to try to figure out a "front-end" solution to the problem, or attempt to write the back-end component himself (which would have been less efficient and would have had a negative impact on product quality). The role of the product development manager also changed from developer-manager to full-time manager. Managing the team and simultaneously being able to contribute to high-concentration development tasks proved to be too strenuous in the long run.

The amount of effort spent by a developer on development work was typically between 50-60%, which matched quite well the original estimate of 60%. However, one of the developers could use only 26% of his effort on development work due to intense participation in existing customer projects.

3.2. Cycles of Control

Previously, no common idea pool had been used, but the product ideas were scattered here and there. At the beginning of the first release all existing ideas were collected into an Excel sheet called the product backlog. There were a total of 30 items identified. Each item was documented by one row containing a column for all the attributes listed in Table 1. The item is described in the product backlog by a short textual description and each item is assigned a priority number that describes the implementation order of the items. The scope of the item is described by estimating how long it would take to implement the feature. The scale is from less than one scrum cycle (1/2 week) to more than two releases (over eight months). The schedule is estimated by assigning each feature to some future release, e.g., II/03 means the second release of year 2003. The business feasibility and architectural importance of items are estimated using the scale: high, medium, or low. Each item is also associated to a product or common component. Links are provided to connect additional documentation to backlog items.

An idea pool that was initially empty was also deployed. During the first release, service and sales proposed three new ideas to the idea pool. They described the idea by a few paragraphs of text, each in a separate file. The product development manager added his comments about the feasibility of the ideas to the same files. The files were collected to a shared directory accessible for the whole organization.

Table 1. The product backlog

Attribute	Type/scale
Description	A few words
Priority	Implementation order
Scope	<1 scrum to >2 releases
Release	I/03, II/03, III/03, undefined
Business feasibility	High, medium, low
Architectural importance	High, medium, low
Product	Prod. 1, prod. 2, components
Links	Supporting documentation

The release-planning meeting took one day. The managing director, sales managers, and the whole development team participated. In the planning meeting all items in the product backlog were briefly discussed, and most of the items were tentatively assigned to one of the three next releases. Three items were selected for the first release: porting the product to a new version of the technology platform (size: 1 release), starting the development of a new product (>2 releases), and one new, big feature (2 releases). The sizes of the selected items were large, but splitting them would not have provided useable features from the customer perspective. Most of the other items were of smaller sizes (1-2 sprints or smaller). The rough unit for the estimates was used in order not to produce too high confidence in the rather vague estimates. The release goals and sprint goals were defined for the product teams and the component team, i.e. they all had their own goals.

Table 2. The release backlog

Attribute	Type/scale
Description	A few words
Priority	Implementation order
Scope	<1 to 2 scrums (½ week)
Sprint	I, II, III, IV
Product	Prod. 1, prod. 2, components
Status	Completed, prototype, open, on hold, deferred

The three selected items were split into 18 more detailed items, and documented in another sheet called the release backlog (Table 2). The release-planning meeting for the second release has also been carried out. It was a clearer and more structured meeting than the first one, because the agenda for the meeting, the terminology used and the development process were more familiar to the participants.

At the beginning of each sprint a sprint-planning meeting was held. Only the developers participated in the three first meetings, because there were no changes to the plans that Business wanted to propose. However in the fourth sprint-planning meeting there was a need to cut down the scope. The progress was late due to lack of resources for the development work, and too small estimates for some tasks. The tuning of the scope for the last sprint was made together with Business in a preliminary meeting between the managing director, a sales director, and the development team manager.

Each item from the release backlog generated several tasks to the sprint backlog (Table 3), meaning that the sprint backlog contained 20-40 tasks in each sprint. The person who would be responsible for implementing a task estimated its effort. Estimation was not done in a very disciplined manner. The tasks were estimated roughly in man-scrums required to complete each of them. At the scrum meetings, the estimates were refined to man-hours, with an assumption of 12 man-hours available per developer per scrum

The realized effort and status fields were not typically updated in the sprint backlog, because the progress was followed mostly through the scrum meetings and the resulting scrum logs.

Table 3. The sprint backlog

Attribute	Type/scale
Description	A few words
Priority	Implementation order
Scope (estimated)	<1 to 4 scrums (½ week)
Scope (realized)	<1 to 4 scrums (½ week)
Product	Prod. 1, prod. 2, components
Status	Completed, prototype, open, on hold, deferred

Scrum meetings were held on Mondays and Wednesdays. Only a few scheduled meetings were cancelled, although scrums were occasionally rescheduled due to customer meetings, sickness, leave, or other reasons. There was no week during which no scrums were held. In the meeting developers told what they had done since the last meeting, and what they were going to do in the next scrum cycle. A developer told the hours he had spent on each task, the status of the task, and possible problems. The scrum master wrote the data into the scrum log (excel sheet per person) immediately and compared what had been done to the plan the developer had presented in the previous meeting. Then the developer told which tasks he is going to spend time on and how much time in the next scrum. Tasks were related, but not explicitly linked to those in the sprint backlog. Red flags were listed to the scrum logs, but not added to the sprint backlog.

In the end of each sprint a demo and a short presentation evaluating the success of the sprint were given by Development. The managing director participated in these, but there were problems getting the sales directors to the first three meetings due to sicknesses, other meetings, etc. This made the developers feel that Business did not consider these meetings important. In addition, some developers questioned the value of the demo, because they had been doing very technical things, and there were not really many new features to show. After the third demo, the managing director emphasized the importance of the demos to the sales directors, and they participated in the last demo.

3.3. Release and sprint goal achievement

The release goals were reached even though the lowest priority item of the three selected items from the product backlog was not finished. Implementing the big, new feature failed because a third party could not provide necessary specifications in time. This item was discarded in the second sprint-planning meeting, and it released quite a lot of resources to other tasks. However, all the released resources were needed to fulfill the other tasks, meaning that the original plans had been too optimistic, and were achieved only due to this “lucky” decrease in scope.

After the first two sprints, the sprint themes and the phases of the release faded to the background. After the exploration and design phase, the release project remained in the development phase without reaching the stabilization phase. This was largely due to the fact that the planned release was an internal, not a commercial one: the real stabilization phase is expected in the later release cycles of the year.

In the first sprint the sprint goal was reached even though some low priority tasks were discarded. The second sprint was optimal, no unexpected problems were faced, the amount of red flags was low, and the goal was easily reached. The third sprint was quite the opposite, containing big red flags, and hard technical problems. The problems were solved but it took so much time that the goal was not completely reached. Some of the unfinished tasks had to be postponed to the fourth sprint, meaning that the original goal for the fourth sprint had to be tuned to be realistic. The tuned sprint goal was reached, and luckily the tuning did not compromise the remaining release goals.

4. Evaluating the process

Generally, all stakeholders have accepted the new process very easily. There may be several reasons for this. From Business, the new process had not really required more effort or change in their way of working. However, the new process has made it possible for them to do their work better, because they can get better knowledge about the current status and future of the product they are selling, and they know when they can affect the development plans (sprint meetings, release planning), and what the consequences of these changes are. From the developers the new process has of course required more adaptation, except from the two new hires that never saw the ad-hoc process. Their positive reactions are probably due to the improvements that the process has brought without causing unnecessary bureaucracy. The product development manager was one of the developers earlier and knew the situation from their point of view. This combined with the fact that developers participated in the process definition work probably helped to avoid the not-invented-here syndrome. The chemistry within the development team has been very good, allowing talking about all issues in the scrum meetings.

The product development manager was very pleased with the use of the Cycles of Control framework as an aid in defining the process. He believes that the process improvement has certainly progressed faster with the aid of the framework than without it. Strong support from the managing director has also helped the adoption of the new process.

4.1. The linkage between Business and Development

Common understanding of the interfaces of the development and sales processes improved the communication between Business and Development. Business knows the release dates and contents well in advance, allowing them to prepare for marketing and start sales at the right time. Business can also clearly see what is left out of a release if new features are added to it during the release, or if un-manageable amounts of non-product-development work are assigned to the developers. Of course, just knowing the plans does not lead to success. Changing the plans must be possible and the plans must be realizable. The short duration of cycles on different levels allows seeing possible problems early and reacting to them. All this means that the organization's risk of committing to unrealistic development plans has decreased significantly.

There were, however, some misconceptions about the types of the releases. For Development it was clear that only an internal release was under way, but for Business it was not clear what this meant. Could they start demonstrating the system or delivering it to pilot customers? The problem was emphasized when discussing the tentative plans for the future releases. The solution was to define the type of release for each product/releasable component separately in a clear way. Alpha release means an internal release for the purpose of demonstrating the system and converging the work. Beta release can be delivered to reliable pilot customers, but it has not yet passed a thorough stabilization phase. Commercial release is adequate for being sold to customers. Especially knowing the dates of commercial releases in time is crucial for Business in order for them to prepare marketing and start sales in a timely manner.

Overall, Business considered they had enough possibilities to affect the plans, and were able to pick a reasonable amount of functionality for the releases. Business followed the development progress through discussions with the product development manager, sprint backlogs, and sprint demos. However, it seemed that Business did not really use the possibility to change the plans. They did not propose any changes after the sprint demos, and the number of new ideas sent to the idea pool was only three, out of which two came from the service team and one from Business. Actually the service team is in closer contact with the existing customers, who are able to propose detailed new improvement ideas to the product. Business sells the system to new customers who really do not know the system yet. This may mean that Business hears only very high-level ideas for the product, if any.

The clear division of roles and responsibilities simplified, streamlined, and organized intra-company communication. The company now has a coherent product vision for the next year and concrete near-term development plans. Business understands development status better, meaning fewer interruptions by Business between the control points, and thus improved working conditions for Development.

4.2. Development practices

The experiences from pair programming have been positive, although the practice of pair programming turned out to be different from what was anticipated. Pair programming in the strictest sense was used only for difficult tasks in order to implement them carefully avoiding bugs and bad design. However, most code modules had two persons responsible for them and the pair bounced the code back and forth in order to edit and audit it. Even this kind of practice made the programmers write more understandable code. The stated purposes of pair programming were achieved: code ownership became collective, knowledge about the code was spread between the programmers, and the expertise of the developers was shared among the team. Pair programming proved to be a good way to get new developers started. New developers also considered pair programming more useful than more experienced ones. Pair programming together with a more understandable development process made taking on new developers much easier than previously.

The experiences from unit testing have also been positive. Unit tests have greatly reduced the debugging time, improved the design, and lead to smaller components. The productivity gains outweigh the time spent developing unit tests.

Getting unit tests to run has been more work than expected, but once the testing framework is in place, it is expected to become easier. Of course, all the modules cannot be unit tested. Automated unit-tests were written for those modules where it was technically possible without unreasonable effort. The developers found automated unit tests useful and felt that the existence of the tests made them more confident when making changes to the code. However, the most experienced developer pointed out that the unit tests do not find any new defects.

The test-first practice was more difficult to adopt and two of the developers said that they write tests first and two others said that they write tests first only occasionally. Developers that were not using the test-first practice consistently felt it unnatural and laborious even though they also felt the test-first practice useful.

Developers considered the scrum meeting a very good practice. It makes you work a little bit harder, helps you learn to plan and estimate your work better, and is an efficient way to find out what others are doing. However, it could be even more efficient if all developers prepared themselves for the meeting in order to be able to briefly describe their situation.

Only two red flags were large enough to be added to the sprint backlog during the release. Approximately two or three smaller ones a week on the average were assigned directly to the developers in scrum meetings or during the scrum cycles.

4.3. Tools

Not much effort was put to create the first "tool" to facilitate the management of the backlogs and scrum logs. The idea was to get the process running and get better tools, as the requirements are understood better. Some effort was spent in order to find a suitable open source tool, but no good candidate was found. The initial "tool" contained a few simple Excel sheets without any automation. After the experiences from the first release a better tool is clearly required for managing the backlogs. The current Excel-based solution is awkward when moving items from one backlog to another, when updating estimates, and when making reports. Also, it cannot preserve history of the contents of the backlogs or previous work estimates of the items. The problem is not just the tool, but also a certain level of vagueness in the management of backlogs, estimates and reporting. Probably it had not even been possible to specify the real needs for the tool, before using the process for a few sprints.

5. Improvement suggestions

After the first release where the new process was piloted, it is natural that we came up with several suggestions for improving the process further.

5.1. Product roadmap

A common understanding of future releases is one of the most crucial pieces of information Business needs. This information should be presented in an easily understandable and available form. A solution could be a single picture, a product roadmap, which presents all planned releases of all products on a time axis for the next 1-2 years [5]. Different types of releases (alpha, beta, commercial) should be clearly differentiated.

The contents of the currently developed releases are specified in detail in the release backlog. For the future releases tentative content may be specified in the product backlog. In each release-planning meeting, or before them, the contents of the future releases may be refined, or even the product roadmap updated by changing the dates and types of the planned releases.

The roadmap can also help in high-level resource planning in a longer perspective. Because the products of the company require some specialized skills from some developers, the roadmap may also help arranging their optimal utilization and help avoiding unrealistic plans. High loads of non-development work by the developers should also be noted as limitations in the roadmap. The roadmap can also help evaluate the need for further recruitment.

5.2. Continuous product planning

The number of new ideas generated for the product during the first release was very low. In dynamic markets the company should continuously observe the environment and brainstorm new ideas for the product, even if the development plans for the next couple of releases are defined. In each release planning the new and old ideas should be re-prioritized based on changes in the environment.

Even small improvement ideas should be stored somewhere, so that they are not forgotten and can be seen by everyone in the company. Bugzilla could be used also for this kind of "reports" as an easy solution.

The service team could participate more actively in the sprint-planning meeting, because they may actually have the best vision of the improvement needs based on close customer collaboration.

5.3. Estimation

Estimation needs to be made with different granularity in different backlogs. In the product backlog there may be both very detailed feature proposals and very large, unclearly defined items, e.g., implement a new sub-product. It is impossible to accurately estimate the effort of large items, whose content is not specified. Instead it is possible and probably useful to roughly estimate how much effort is needed to implement some kind of alpha prototype.

When large items are considered for the release backlog they should be refined by splitting them to more concrete parts and re-estimating the parts. The new estimates should be available for Business, before they make final decisions on the content of the release backlog. Making these estimates more accurate may require doing small spikes etc. This means that the release-planning meeting must be split into two parts containing enough time for making spikes in between.

On all levels the estimates should be done using units of effort, whose purpose is clearly understood by all stakeholders. If there are special limitations to the minimum calendar time required for a task this should be an additional note to clearly separate the effort estimation from schedule estimation. Even if the effort needed for an item is hard to estimate, it should be estimated using a clear figure. Otherwise, it is hard to track release/iteration progress and remaining effort, and learn to estimate things better in the future. In addition, another figure could be used to express the uncertainty related to the estimate. Being wrong in the estimates in the beginning is natural and should be understood by Business. However, improving estimation skills and lightweight estimation techniques should be a constant learning process.

5.4. Backlog tools

The initial Excel sheets for managing backlogs were not satisfactory. Making any summaries of the effort spent on items on any level should be available automatically based on the time logs entered into the sheets in the scrum meetings. It should be possible to easily compare the original estimates and realizations on all levels in order to improve the estimation accuracy on a personal level as well as on the iteration/release level. The ratio of the development and non-development work should be available in order to know the amount of development effort available for future releases. When the data is entered in a more structured form, a burn-down graph of the remaining effort can be drawn. All these changes can be implemented easily using quite simple Excel sheets.

Currently the different backlogs are in several files, some containing several of the backlogs. The current backlogs (product, release, sprint, scrum) should be easily available, e.g., linked to an intranet page. The goals of the releases and sprints and the phases of the release should also be collected from the sheets to a more visible form, e.g., to the intranet page or placed on the walls.

For the second release, the company deployed an internal NNTP news server with a newsgroup devoted to the planning game. Its intended use is preliminary discussion of product ideas, before proceeding to a full-scale feasibility study or committal to the “formal” idea pool. It is too early to have concrete results from working with the newsgroups, but they have been favorably received and are in daily use. In addition, the communications model between the different departments of the company (Business, Service, Development) was defined, and tools for facilitating it should be researched.

5.5. Reflection meetings

As things change and the team’s experience increases, it is necessary to tune the process continuously. Reflection meetings proposed by Cockburn [2] after each iteration and release, where the process is evaluated and improvements conceived, might be a good way to do this constant improvement. Basic metrics for progress, product quality, estimation accuracy etc. should be analyzed in order to evaluate the effects of process improvements.

6. Conclusions

The Cycles of Control framework increased the understanding of the development process throughout the company and helped understanding the linkage between the product development and business processes. Describing the development process using the framework helped in identifying the crucial control points between Business and Development and enabled defining practical and well-functioning connections between them. With the framework we created and adopted an effective agile development process in a short time with reasonable effort.

Overall, the social change needed to adopt the process and get Business to understand development status better turned out to be easier to achieve than anticipated. This means fewer interruptions by Business between the control points, and thus improved working conditions for Development. However, it turned out to be difficult and time-consuming to find and adopt technical tools needed to facilitate the process.

As expected there are several details in the process that still require improvement, but also some areas that were seen very successful from the beginning, such as clearer communication between Business and Development and scrum meetings.

References

[1] Beck, K., *Extreme Programming Explained*, Addison-Wesley, 2000.

[2] Cockburn, A., *Agile Software Development,* Addison-Wesley, 2002, pp. 184-195.

[3] Rautiainen, K., C. Lassenius, and R. Sulonen, "4CC: A Framework for Managing Software Product Development", *Engineering Management Journal*, Vol. 14, No. 2, June 2002.

[4] Schwaber, K. and M. Beedle, *Agile Software Development with Scrum*, Prentice Hall, 2002.

[5] Vähäniitty, J., C. Lassenius, and K. Rautiainen, "An Approach to Product Roadmapping in Small Software Product Businesses", *7th European Conference on Software Quality* (ECSQ2002) *Conference Notes*, Helsinki, Finland, June 2002, pp. 13-14.

[6] Vähäniitty, J., "Key Decisions in Strategic New Product Development for Small Software Product Businesses", *EuroMicro Conference 2003 (forthcoming),* Antalya, Turkey, October 2003.

[7] Williams, L and R. Kessler, *Pair Programming Illuminated*, Addison-Wesley, 2002.