A Tentative Framework for Managing Software Product Development in Small Companies

Kristian Rautiainen, Casper Lassenius, Jarno Vähäniitty, Maaret Pyhäjärvi and Jari Vanhanen
Helsinki University of Technology
Software Business and Engineering Institute
POB 9600, FIN-02015 HUT, FINLAND
firstname.lastname@hut.fi

Abstract

Deploying an appropriate software process can improve the effectiveness of software engineering. Still, small companies find it hard to allocate resources to software process improvement and tailor existing process models for their needs. In this paper we present a tentative framework for managing software product development in small companies. The framework combines business and process management through four cycles of control: (1) Strategic release management provides the interface between business management and product development. (2) Release project management handles the development of individual product versions. (3) Iteration management deals with the incremental development of product functionality within release projects, and (4) Mini-milestones are used to get an indication of system status during development.

1. Introduction

It is widely understood that deploying an appropriate software process can improve the effectiveness and efficiency of software engineering. However, small companies find it hard to allocate resources for software process improvement (SPI) and tailor existing process models for their needs. Many of the well known software process models and reference models, such as the Capability Maturity Model (CMM) developed by the Software Engineering Institute (SEI) at Carnegie Mellon University (see e.g. [5]), provide a good basis for SPI, but they also provide excessive overhead if deployed in full. Specifically, they do not take the business aspects and the fact that different process models might be needed in different situations enough into consideration. Successfully managing software product development demands more than having a suitable software process in place – a more holistic view is needed.

In this paper we present a tentative framework for managing software product development in small companies. By small companies we mean companies with less than 100 employees and less than 50 developers. The framework is partly based on our previous research on improving the controllability of product development, during which we identified the basic components of a control system for managing product development. To add to this knowledge and to focus on software product development we have studied different process models found in literature. These models provide valuable insight and alternatives to managing the software engineering activities of a company. We have also studied the practices of so-called "agile" processes in order to find alternatives that focus on small teams and projects. Also, the framework is based on interviews, discussions and observations made with the participating companies in our ongoing research project.

In this paper we focus on providing an overall view of the framework. The details of the different parts of the framework are left for subsequent work. First, we present the components of a control system for managing product development derived from our previous work. Second, we shortly present our research project. Third, we present the tentative framework. Finally, we round up with discussion and implications for further work.

2. A control system for managing product development

In our previous research project we studied the controllability of product development and one of the findings was a set of basic components for a control system for managing product development, shown in Figure 1. Many organizations face problems in managing their product development operations. The problems can be summarized into four groups: lack of direction, competence, motivation, or opportunity.

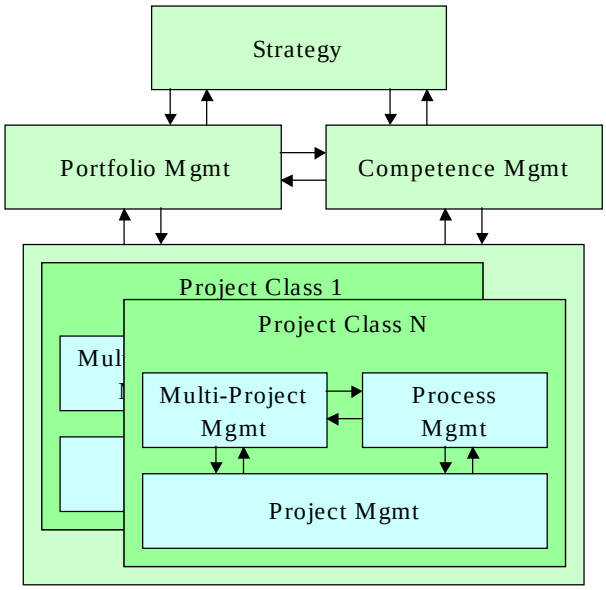

Figure 1. The basic components of a control system for managing product development

Lack of direction: Getting somewhere demands understanding where you are (current state) and where you want to go (goal or target state). Lack of direction can be caused, e.g., by an unclear business strategy. In a turbulent business environment the target may be moving so fast that it is elusive. Lack of direction may also concern, for instance, process improvement on any organizational level. Measurement and goal setting plays an important role in overcoming such a lack of direction. If you can use measures to indicate the present status of the process and give target values for the measures for a preferred status, you have better chances to control your actions towards reaching the goal state.

Lack of competence: Even if you know where you are and where you want to go, you might not know how to get there. This is a problem of lack of competence. For example, the company may be moving into previously unknown markets, or the technology used in a product may be so new that the competence has not yet been acquired. At that point it is very hard to estimate how long it takes to reach a sufficient level of competence, which influences the time to reach the target.

Lack of motivation: In processes performed by humans, direction and competence are not enough. If people are not properly motivated to reaching a given target, any effort may be futile. Even if we know where we want to go and have the necessary competence to get there, lack of motivation will slow down or even halt our progress. Lack of motivation can be caused by many things, which we will not go deeper into in this paper. E.g. Wiegers talks about culture builders and killers in his work [22].

Lack of opportunity: Even if the direction is clear, people have the necessary competence and are well motivated, lack of opportunity to achieve a given target can occur. The target may be unrealistic or there may simply be too many ongoing projects, too many targets to try to reach, which causes distraction. This may result from, e.g., an inability to prioritize work or unrealistic expectations about the existing resources and the effort needed. DeMarco and Lister have discussed the subject of productive projects and teams in their classic work [11].

By thinking in terms of the components in Figure 1, one should be able to create a control system to better manage the product development efforts of the company and improve the ability to control those efforts. A product development strategy should provide the overall direction of the organization, project portfolio management should ensure that the project load is not unreasonable, competence management should handle the skill building aspects, and good processes and project management practices are partly responsible for creating a pleasant working environment for increased motivation. The next sections briefly describe each of these components.

Strategy: By strategy we mean the product development strategy of the company, which should be derived from the overall corporate strategy. The business environment of the company must be considered, for instance the speed of change of technology or the markets. An important issue is to understand that there are different types of product development projects that need to be staffed and managed in different ways. For example, making a breakthrough product is different from making derivative products to already existing product lines. If product maintenance is considered as part of product development, it is also managed differently. The product development strategy can be summarized as one or multiple roadmaps (product, service, marketing, etc.), where for instance the product roadmap should show the different types of projects and a rough resource allocation. This is then used as an input to project portfolio management and competence management. E.g. Cooper talks about these issues in [7] and [8]. Also, these issues have been touched in [18]. For examples of project classifications, see [19] and [21].

Portfolio management: By portfolio management we mean the management of the whole set of projects of the product development organization. The input to portfolio management is the product roadmap, especially the project type classification and the rough resource allocation. To be successful in portfolio management, one must also know the existing resources and competences in the organization. Another input is, of course, the feedback from ongoing projects. The purpose of portfolio management is to specify in more detail the projects needed to fulfil the strategic goals of the organization, thus linking projects to strategy and operationalizing the

product roadmap. An important task is to prioritize projects and select the order and mix of projects to be executed. The output of portfolio management is an aggregate project plan or a project roadmap. The plan has to be updated at regular intervals to reflect the current situation. For example, many projects can be interrelated and if one project is lagging in its schedule, other projects can be influenced. This may lead to replanning and reprioritizing the order and mix of projects. For more reading, see for example [8], [12] or [18].

Competence management: The purpose of competence management is to keep track of the competences - existing and needed - in the organization and plan for training and recruiting to fulfil those needs. Competence management is tightly connected to the product development strategy and portfolio management. Also, competence needs can arise in ongoing projects. One approach for obtaining a list of professional competences can be found in [20].

Multi-project management: The purpose of multi-project management is to balance and allocate resources between projects at a regular and short-term basis. Having people work on multiple projects and moving them around between projects is not easy, though, and can cause more harm than gain [4]. Multi-project management is, naturally, closely linked to project portfolio management, and could even be considered part thereof.

Process management: With process management we mean here managing the product development process. The process model works as a map for the development projects providing the stages, milestones, roles, etc. It provides a common vocabulary and the "rules of the game", i.e., how things are supposed to be done in the organization. The process model should also be a tool, providing, e.g., templates and checklists for the projects. It is important to realize that one process model cannot accommodate the needs of all different project types. Therefore some thought has to be put into choosing appropriate process models. Some examples of issues affecting the choice of process model are the speed of change in technology or the markets, the size and length of the projects, the size and complexity of the product being developed, and the initial uncertainty of the project, i.e. how well we know the requirements up front. One part of process management is collecting data and feedback from projects for process improvement purposes.

Project management: Project management is about executing the individual projects in a systematic way, using the guidelines provided from the process models.

One of the main lessons from our previous research project, where we worked with organizations to create a control system for product development is that in order to improve controllability you have to look at the whole. Concentrating only at one part, for instance project management, you can still end up with too many projects and the project load will then at some point bring matters out of control. Another problem we encountered was that if we did not understand the whole, we could end up with a procedure that was sound in theory, appreciated in practice, but failed because of some other practices already in place.

3. The SEMS research project

In the ongoing SEMS (Software Engineering Management System) research project we are studying software engineering in small companies in the software product business. The project started in the autumn of 2000 and is planned to go on to the end of 2003. Our main focus is on the software development process and in finding links between the business model(s) the company has chosen and the software processes and software engineering practices needed to support the business model(s). One of the goals is to find a light but high-impact way of systematically performing the software engineering practices that are required in developing high-quality software products. By light we mean that introducing process or system thinking into the company should require as little resources as possible and minimize disruption. Another goal is to determine which the most important practices are and package the lessons learned into a software engineering management system for small companies in the software product business.

McCormick's opinion summarizes the ideas brilliantly: *"What's needed is not a single software methodology, but a rich toolkit of process patterns and 'methodology components' (deliverables, techniques, process flows, and so forth) along with guidelines for how to plug them together to customize a methodology for any given project."* ([17], p. 110).

We currently cooperate with four companies in a mass-market type of business, meaning that customer tailoring is not a significant part of the business. The products are not shrink-wrapped and in three of the cases some tailoring has to be made when the product is installed. One of these companies also has an ASP solution for end users. Two of the companies are in a fiercely competed, extremely fast-pace business environment, where being first really counts. Being in the software product business, the companies make different types of product releases. The way of working is iterative and incremental. The release cycles are short, ranging from one month to a week, if counting the bug fix releases.

As a first step in the project, we have developed a tentative framework for managing software development in such companies, based upon our earlier work in new product development management. This framework is the subject of the next chapter.

4. Towards a framework for managing software product development in small companies

There are two main issues that have to be addressed in order to get from the control system presented in Chapter 2 to a framework for managing software product development in small companies. First, focusing on software product development, and second, focusing on small companies. Focusing on software product development introduces, among other things, the concepts of software engineering processes and practices. The small company perspective brings constraints, especially concerning resources.

Most of the software engineering processes or software development processes in literature concern building large and complex systems, and therefore can create excessive overhead for a small company. The CMM (see [5]), for instance, provides a way to build organizational capability for performing software engineering. It even provides you with a recommended path of improvements to follow. But the CMM was written to address the process for large, complex software efforts, something a small company with 3-10 developers probably would not undertake. In [3] Brodman and Johnson showed how small businesses and small organizations viewed the CMM. Especially some points are interesting to us: the need for the CMM to be more flexible and scalable in order to accommodate different types of projects, and that the attitude of the personnel can be a big contributing factor to not applying the CMM or any other new process approach for that matter. The CMM does not dictate which development process model one should use, it only tells you to use the one that suits you best and tailor it for different needs. So there actually is flexibility in the CMM, it only gives criteria for mature processes, specifically a process must be: defined, documented, trained, practiced, supported, maintained, controlled, verified, validated, measured, and improvable [5]. CMM also recommends that maturity and effectiveness of processes should be interpreted in the context of the business environment of the company and the specific circumstances of the projects. A closer explanation of this, however, is left to other sources. Since one of our research goals is to find a link between the business models companies use and the software processes that support them, we took the business perspective as a starting point in moving towards a framework for managing software product development. Bays has summarized software release methodologies in his work [1]. He points out some important issues to consider in release management. These, combined with some other issues picked from best practice lists, such as the Airlie Council's list (cited from [24]) form a basis for the tentative framework described in this chapter.

Bringing a software product release management point of view into the discussion divides our framework into two levels: long-term or strategic release management and release management in individual release projects. Projects are executed in an iterative and incremental manner for increased flexibility, using mini-milestones to gain more controllability. Figure 2 depicts these four *cycles of control*. In relation to Figure 1, the strategic release management cycle covers the top three boxes as well as partly multi-project management, whereas the other three cycles present one way of implementing the rest of the general control framework. The radius of a cycle symbolizes the time perspective taken. The larger the radius, the longer the time perspective.

The four control cycles, strategic release management, release project management, iteration management, and mini-milestones are described in the following sections.

4.1. Strategic release management

The outermost control cycle, *strategic release management* is the interface between business management and product development. The main purpose of strategic release management is to plan the release cycles and the content, role and timing of each individual release project. This means that the overall strategic ambitions and goals of the company have to be considered, together with the availability and competences of the people that do the actual work. Product line decisions may also be of concern here, especially when a company grows and diversifies its product offering. An important task is to elicit, specify and prioritize requirements from different stakeholders, for instance marketing, customer services and users. Requirements engineering also forms the main interface to the individual release projects.

One of the biggest problems in requirements engineering is that the customer does not really know what he wants, or at least cannot express it coherently. In mass-market products the problem can be that the end customer is not heard directly, and the requirement engineers must rely on, e.g., market research data. In fast-pace markets some requirements change during the project. Cusumano and Selby report in [9] that at Microsoft a vision statement and outline specification are used to give enough structure to the development effort, but at the same time accommodate change and flexibility during the development process. The specification will then evolve during the development project. Also, features are prioritized so that the most important features can be implemented first. The development is then done in several incremental cycles, between which the requirements can be reprioritized and new requirements can be added if necessary.

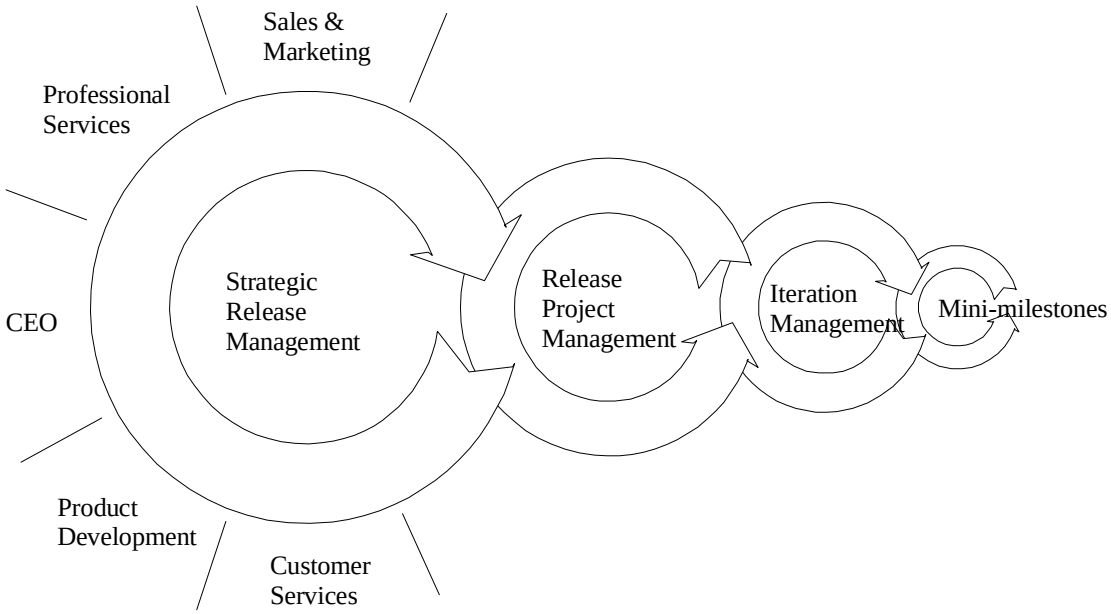

Figure 2. The control cycles of the framework

In eXtreme Programming (XP) software development is seen as "an evolving dialog between the possible and the desirable". A practice called "the Planning Game" brings together the two players: Development and Business. Requirements are collected on story cards, where Business writes the story and Development estimates how long the story will take to implement. The stories can be split into more stories if necessary. Then the stories are sorted and Business chooses the scope and date of the next release.[2] The point here is that there should always be an effort estimation attached to the requirements or features, otherwise you cannot consider the resource implications to the release projects.

The important thing to remember in requirements engineering is that the requirements should depict what the system is supposed to do for the user, not the structure of the system. The point is to fulfil a business need, which the requirements should reflect. That way design decisions and decisions to add or drop features are easier to make during development.

Figure 2 gives an example of some of the possible stakeholders or stakeholder representatives that might be involved in strategic release management. The variety of stakeholders and their different areas of expertise propose a challenge: the requirements or the features to match the requirements that are discussed should be presented in a way that everyone understands. The Unified Software Development Process (USDP) suggests use cases for capturing the requirements and communicating them to the customer and the designers [14]. This approach is probably too detailed for this level of discussion. The vision statement used at Microsoft is a better way to communicate the purpose of the product. That way the abstraction level is more suitable for discussing future releases and products. USDP also suggests business or domain models to be used, which can add width to the information on top of the vision statement. In a very small company a single person most likely acts in multiple roles and strategic release management is done by as few as 3-4 people. Even when the company grows the group should be kept fairly small for the meetings to be effective.

4.2. Release project management

The next control cycle, release project management, is concerned with individual release projects developing the actual product versions. In a small company there should not be many concurrent projects, simply because there are not enough developers. This does not mean that there would not be many different types of projects in the company. The same developers can be, e.g., working on improving the product platform, developing new features to an existing product, installing the product at the customer's site, maintaining the product (fixing defects), or developing an entirely new product. The implication of different types of projects is that they should be managed and controlled differently.

Two main project types are functionality driven and schedule driven. A new operating system is an example of a product that requires a functionality-driven project. Certain functionalities have to be in place for a system to be able to work as an operating system. This means that the schedule is allowed to slip so that the development team can build the required functionality. Microsoft Office is an example of a product (or product family) that is developed in schedule-driven projects. The release

deadline is set in advance and the product is released on schedule. It is impossible to say beforehand what the final configuration of features in the product is, because the features are decided upon as the project progresses. Features are dropped if they cannot be finished in time for the release. For this to work, the features have to be prioritized.

What this means in relation to Figure 2 is that the release project management has to consider the length, content and number of iteration cycles in a project. The task is to plan and specify the release project according to the priorities specified in strategic release management. In this paper the details of project planning are left out. In one of the companies we have studied, experience has shown that a maximum controllable iteration cycle length for a new product or larger new features to an existing product is three months. In the beginning the company tried longer cycles, but it always led to the projects going out of control. If there is only one iteration cycle, the process followed resembles the traditional waterfall model. When a product matures and there is experience from multiple product generations, the changes to the product, e.g., adding new features can probably be done with less effort and in shorter cycles, given that there are no changes in the personnel developing the product. This should be a consideration in iteration planning. As a rule of thumb, the length of a release project should lie between three and twelve months.

USDP suggest an architecture-first approach in planning and performing the iterations. The purpose is to find and develop a baseline architecture that will facilitate implementing features now and in the future. MacCormack's findings support that investments in architectural design are associated with better performing projects, with good performance indicated by product quality as perceived by the user [16]. Another consideration is perceived risk. The greater the perceived risk impact, the earlier the feature should be implemented. This way there is enough time to react to the possibly realized risk and gain better control of the project.

We have observed that when projects begin, planning quality assurance is often poorly done. Quality does not just appear into the product – we have to think about it right from the start. We have seen organizations that have left testing "to the last weekend" before the release, and the consequences have been less than impressive. A difficult decision in testing is how much and exactly what to test with limited resources. Prioritizing test cases and parts of the system is the key. Also, understanding what "good enough" quality means in each case is important. The testing process should be planned in parallel with the development project, so that testing is considered at every stage.

4.3. Iteration management

In each iteration a set of use cases or features are identified, specified in detail, designed, implemented and tested. This way there should be a working product at the end of each iteration, which can be delivered to users to get early feedback on further development. At the end of each iteration strategic release management is revisited to check the market situation and possible new or changed requirements, so that the next development cycle can focus on the relevant features. This approach has been found good for developing high-quality products in an environment with high uncertainty and rapidly changing requirements [16], which is the environment of the companies we have worked with.

An example of this approach is Microsoft, where large projects are divided into multiple incremental cycles at the end of which a shipment of the product is made to stabilize the product (Figure 3). That way Microsoft can fall back on the previous shipment if the next cycle fails. The individual engineers synchronize their work by doing daily builds, which are also tested daily. The process has been accordingly named Synchronize-and-Stabilize.[10]

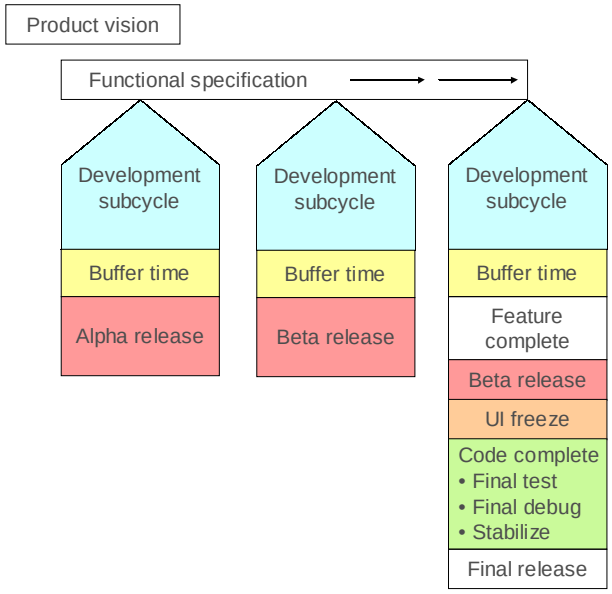

Figure 3. The Synchronize-and-Stabilize process (redrawn from [10])

XP approaches incremental development by doing the development in short iterations, lasting 1-3 weeks. Highsmith talks about time-boxing projects as a mechanism for managers to force periodic convergence of a system [13]. All this implies that a certain amount of freedom can be given to the developers during the iteration cycle, as long as the system is stabilized at the end, thus adding controllability by showing the exact status of the system at that point in time.

An incremental development process makes regression testing especially important, as does a daily build practice. Inspections or peer reviews could be seen as a part of the testing process, since one main point is trying to detect, identify and track the defects as early as possible. This can give an indication of defect levels in the final product and eliminating defects early is less costly [24]. Tracking defects is almost as important as finding them in the first place. Without tracking, the same defect might be "found" over and over again, and a simple defect count would be misleading. Tracking should also facilitate learning about the testing process for improvement purposes.

4.4. Mini-milestones

In order to have a better indication of the status of development and thus better control the development effort, mini-milestones are used, for example in the form of daily builds, as in the case of Microsoft. At Microsoft the daily build - daily test cycle makes early detection of defects possible. If something breaks the system, the defect must have been introduced the same day, which makes finding the defect easier.

In XP the idea of "test first" is introduced. The idea is to write a unit test for every production method that could possibly break. The tests should be written before the code is written, serving at the same time as a specification or explanation for the methods and features. The unit tests are then supposed to be running at 100 % all the time. If something breaks the system, a test is written that will detect the defect before it is fixed. Automated testing gives confidence to refactoring, since the tests should pick up any defect introduced to the system. Another interesting practice in XP is pair programming, where two persons sit at the same computer. One person writes the code and the other person watches, in principal doing inspection online. This should result in better quality code [23].

Source code control or configuration management becomes crucial when we use practices like Microsoft's daily build - daily test. Also, the more often we make releases, the better we have to be able to manage the source code. One would think this all seems very straightforward and clear, but we have observed that in practice source code control is not properly done, at least in many small companies. The main reason seems to be that although everyone uses tools like CVS, the practices have not been agreed upon, which leads to almost as many different practices as there are developers. We noticed one instance of this in a company that was confident about their source code control. When they started doing more rigorous defect tracking, they suddenly noticed that the already fixed defects repeatedly popped back into the system. This happened because some of the developers did not check in their code or update to someone else's revised code in time. This could have been avoided if common rules for the source code control had been established.

To avoid leaving all system testing "to the last weekend" also requires that a version of the system can be moved to a testing environment at will. When defects are found they must be identified with the exact version of the system that is under test. Especially in the case of common ownership of code, writing proper change notes to the source code control system is important. It can save time in forming an understanding of how the changes might have influenced other parts of the system.

4.5. Summary and lessons learned

The tentative framework for managing software product development in small companies combines business and process management for developing high-quality software products that fulfil market requirements. Strategic release management is the interface between business management and product development taking a long-term view to release management. This means processing the available market information and making decisions about the content, role and timing of each individual release project.

The products are developed in release projects in an iterative and incremental fashion. The basic idea of an iterative and incremental development process is to deliver early to get user feedback on the system. 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). Frequent integration of the system, or mini-milestones, such as daily or weekly builds, is used to get a better indication of system status during development. This way project management finds early warning signs and can take proper controlling actions.

Change management concerns the entire development effort, starting from requirements and going down to the test cases and source code. Depending on what details the changes concern, the practices differ. If requirements change, they may influence anything from the architecture of the system to just a part of a module. Other changes, like code changes, seem to be of less impact and importance, but they should also be documented in a commonly agreed way.

One might assume that managing change gets easier the smaller the development team is. That is partly true, since the team is very often co-located, which improves communication between the members of the team. But there will always be a need to write down the changes, because leaving the details to memory only is very risky.

The organization should establish commonly agreed upon rules and guidelines to align the efforts of individuals and teams. Each rule should have a tolerance

level attached to it, so as to show how strictly it must be followed and how much freedom each person has in applying that rule. For example, coding conventions may be very strict to facilitate easy maintenance of the software, whereas more freedom can be given on, e.g., choosing the code-writing tool.

One of the main lessons we have learned so far is that establishing a common language is one of the most valuable and tangible results of applying "process thinking" in an organization. We thought that in a small company people communicate and interact with each other more frequently thus almost automatically creating a common language, but we were proven wrong. For example, when we started developing the product roadmapping process for strategic release management, we observed that different people were using different terms for the product parts, even within the product development team. Creating a conceptual model of the product that all people could agree upon and understand, facilitated more meaningful discussions and decision-making concerning the product and its future releases. The same has applied to, e.g., quality assurance.

5. Discussion and further work

In this paper we have presented work in progress in our research project where we are developing a framework for managing software product development in small companies. The framework is still tentative, and some issues that we know are important have been left out so far. As an example, measurement is not discussed at all. On that front we have been working for a longer time on a tool set for the creation, management and use of a measurement system. We plan to continue and integrate our earlier work on measurement into this framework.

We also plan on adding more detail to the framework as we deepen our understanding of the challenges of managing software product development, as well as find workable solutions. The details will be prioritized and most likely sorted in some kind of hierarchy for different situations and needs.

Alistair Cockburn has developed the Crystal family of methodologies [6], where he uses three dimensions for methodology selection: the number of people involved in the project, the criticality of errors, and where the priority of the project lies (e.g. productivity, legal liability, etc.). The more people are involved in a project, the more formal the communication to coordinate efforts has to be. If a system is life-critical, verification and validation practices must be extensive and rigid, and so on.

These are views we are looking to incorporate into our framework in the future. One early temptation has been to create a list of best practices from different sources. The list would then be used as a source to pick practices when needed. Unfortunately, practices are often interrelated and picking one may require picking a bunch of others for consistency. That is why the idea of families of practices or methodologies for different situations is appealing.

An interesting question for further work is the scalability of the framework. Cockburn's ideas of methodology families are appealing and we want to look into what it might mean in terms of our framework. This might also solve the issue of prioritizing and sorting the details.

Currently we are working on developing and implementing a product roadmapping process for strategic release management into two of the companies we work with. We are also looking closely on testing and defect tracking.

References

[1] Bays, M.E., *Software Release Methodology*, Prentice Hall, Upper Saddle River, 1999.

[2] Beck, K., *eXtreme Programming eXplained,* Addison-Wesley, Boston, 2000.

[3] Brodman, J.G. and D.L. Johnson, "What Small Businesses and Small Organizations Say About the CMM", In *Proceedings of ICSE-16*, 1994.

[4] Brooks, F.P. Jr., *The Mythical Man-Month: Essays on Software Engineering, 20th anniv. ed.,* Addison-Wesley, Reading, 1995.

[5] Carnegie Mellon University / Software Engineering Institute, *The Capability Maturity Model: Guidelines for Improving the Software Process*, II. Series, Addison-Wesley, 1995.

[6] Cockburn, A., *"Designing a light methodology"*, Presentation 1998, http://members.aol.com/humansandt/crystal/tutorial/methodology2.ppt, Cited 17.3.2001.

[7] Cooper, R.G., *Winning at New Products*, 2nd ed, Addison-Wesley, Reading, 1993.

[8] Cooper, R.G., S.J. Edgett and E.J. Kleinschmidt, *Portfolio Management for New Products*, Addison-Wesley, Reading, 1998.

[9] Cusumano, M.A. and R.W. Selby, *Microsoft Secrets*, The Free Press, New York, 1995.

[10] Cusumano, M.A. and D.B. Yoffie, "Software Development on Internet Time", *IEEE Computer*, Vol. 32, No. 10, 1999, pp. 60-69.

[11] DeMarco, T. and T. Lister, *Peopleware: Productive Projects and Teams, 2nd ed*, Dorset House Publishing, New York, 1999.

[12] Englund, R.L. and R.J. Graham, "From Experience: Linking Projects to Strategy", *Journal of Product Innovation Management*, Vol. 13, No. 1, 1999, pp. 52-64.

[13] Highsmith, J.A. III, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House Publishing, New York, 2000.

[14] Jacobson, I., G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, Reading, 1999.

[15] Kerssens-van Drongelen, I.C. and A. Cook, "Design Principles for the Development of Measurement Systems for Research and Development Processes", *R&D Management*, Vol. 27, No. 4, 1997, pp. 345-357.

[16] MacCormack, A., R. Verganti and M. Iansiti, "Developing Products on 'Internet Time': The Anatomy of a Flexible Development Process", *IEEE Engineering Management Review*, Vol. 29, No. 2, 2001, pp. 90-104.

[17] McCormick, M., "Programming Extremism", *Communications of the ACM*, Vol. 44, No. 6, 2001, pp. 109-111.

[18] Rautiainen, K., M. Nissinen and C. Lassenius, "Improving Multi-Project Management in Two Product Development Organizations", In *Proceedings of HICSS-33*, 2000.

[19] Shenhar, A.J., "From Theory to Practice: Toward a Typology of Project-Management Styles", *IEEE Transactions on Engineering Management*, Vol. 45, No. 1, 1998, pp. 33-48.

[20] Spenser, L.M., S.M. Spenser, *Competence at Work: Models for Superior Performance*, John Wiley & Sons, New York, 1993.

[21] Wheelwright, S.C. and K.B. Clark, *Revolutionizing Product Development*, The Free Press, New York, 1992.

[22] Wiegers, K.E., *Creating a Software Engineering Culture*, Dorset House Publishing, New York, 1996.

[23] Williams, L., R.R. Kessler, W. Cunningham and R. Jeffries, "Strengthening the Case for Pair Programming", *IEEE Software*, Vol. 17, No. 4, 2000, pp. 19-25.

[24] Yourdon, E., *Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects*, Prentice Hall, Upper Saddle River, 1999.