4CC: A FRAMEWORK FOR MANAGING SOFTWARE PRODUCT DEVELOPMENT

Kristian Rautiainen, Casper Lassenius, and Reijo Sulonen
Helsinki University of Technology

Abstract

Managing software product development is challenging, especially for small companies in which a balance has to be struck between development flexibility and management control while working under tight schedule and resource constraints. While there exists several approaches to software process improvement, such as the CMM and SPICE reference frameworks, these models focus on the software process for customer projects in large organizations. Small product-oriented companies require a more holistic and practical view to software engineering management that combines business and development considerations and has a clear product focus. This article presents a general framework for managing software product development in small organizations. 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 for daily or weekly task scheduling and monitoring to get an indication of system status during development. The framework can be used both to assess the current state of development in the organization, as well as a blueprint for improving or reengineering product development management.

Introduction

It is widely understood that deploying an appropriate software process can improve the effectiveness and efficiency of software development, and several prescriptive process models exist (e.g., Royce, 1970; Boehm, 1988; Jacobson, Booch, and Rumbaugh, 1999). Several approaches for software process improvement have also been developed during the last decade, with the Capability Maturity Model for Software (SW-CMM) from the Software Engineering Institute being most well-known and used (Carnegie Mellon University, 1994).

Small companies—with less than 50 developers—however, often find it hard both to allocate resources to software process improvement and to tailor existing process models or improvement approaches to their needs. The software process and reference models provide a good basis for software process improvement, but they also provide excessive overhead if deployed in full, as shown, e.g., by a study on the perceptions of the CMM by small organizations (Brodman and Johnson, 1994). The findings indicate that small organizations find it hard to tailor such models to their needs and to motivate their personnel to use them. Since the CMM was developed with large organizations and large teams in mind, applying it to small organizations is like shooting flies with a cannon.

Another shortcoming of these approaches from the point of view of a small product-focused software company is that they fail to adequately address necessary business constraints and the fact that different processes might be needed in different situations. Furthermore, the models have their background in customer project business, which is different from product-oriented business in several ways.

In contrast to traditional approaches to software development which emphasize planning, control, and documentation, several new "agile" software development models have been proposed (e.g., Schwaber and Beedle, 2002; Beck, 2000; Highsmith, 2000). Empirical studies have shown that many companies, both large and small, in the Internet software and PC software businesses use flexible and adaptable processes (Cusumano and Selby, 1995; Cusumano and Yoffie, 1998; 1999). Such flexible approaches have also been found to lead to increased customer satisfaction although they carry a penalty due to inherent problems of rapid development (MacCormack, Verganti, and Iansiti, 2001).

About the Authors

Kristian Rautiainen received his MSc at Helsinki University of Technology (HUT) in 1996 and is currently working on his DSc (Tech) degree. He has been teaching software processes at HUT since 1998 and his research interests include software processes and software engineering management. He is currently exploring how to manage software product development in SMEs.

Casper Lassenius is an acting professor of software engineering at HUT. He received his MSc (Eng) from HUT in 1996 and is currently finalizing his DSc (Tech) degree on performance measurement in software development organizations. His research interests include software and product development in company networks, software product development management and software process modelling and enactment support.

Reijo Sulonen is professor of computer science at HUT since 1980. He received his MSc (Eng) and DSc (Tech) degrees from HUT. He is a member of board and advisory committees in a number of high-tech companies. He has been a member of the European IT Prize Committee since 1997. He has published numerous articles on different areas of computer science and its applications in industry in areas relating to electronic publishing, computer supported collaborative work, and product data management.

Contact: Kristian Rautiainen, Helsinki University of Technology, Software Business and Engineering Institute, PO Box 9600, FIN-02015 HUT, Finland; phone +358-9-451-5063; kristian.rautiainen@hut.fi

Refereed management tool manuscript. Accepted by Hans Thamhain, special issue editor. Previous version presented at 35th Annual Hawaii International Conference on System Sciences.

In order to successfully manage software product development in small companies, a holistic approach combining business and development aspects and providing a combination of control and flexibility is needed.

In this article we present a framework for managing software product development in small companies. The framework is 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, as well as a literature study covering software process models, reference frameworks, and agile development methodologies. In addition, we have used interviews, discussions, and observations made with the participating companies in our ongoing research project.

The rest of the article is structured as follows: (1) we present the research goals and methodology, (2) we present the framework, and (3) we conclude with presenting our initial deployment experiences as well as implications for further work.

Research Goals and Methodology

The research presented in this article follows a constructive research approach (Kasanen, Lukka, and Siitonen, 1993) and is the first result of a three-year research project aiming at developing tools, methods, and practices for successful management of software product development in small companies. The main focus of our research 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 them. We aim at identifying, modeling, and adopting a minimal set of practices and processes needed to successfully manage software product development. Thus, our goals are directed more toward simplicity, minimalism, practical relevance, and implementability than comprehensiveness and perfectionism.

The research has been performed in close cooperation with four Finnish software product companies, guaranteeing practical relevance and implementability of the models we develop. The companies are in mass-market types of business, meaning that the degree of customer tailoring is small.

The products, however, 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 competitive, extremely fast-paced business environment in which first-mover advantages are enormous. The companies make several releases of each product, and their 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.

The 4CC Framework

Overview. The 4CC or *Four Cycles of Control* framework combines business management and software product development, and takes both a long-term and short-term view to software product release management. The framework takes into consideration the type, timing, and content of different product releases, going all the way down to daily or weekly builds within release projects for pacing and control. The framework provides a common language and understanding of the way software product development can be organized and brings a degree of control into it, at the same time accomodating flexibility and fast response to change.

Exhibit 1. The four cycles of control framework

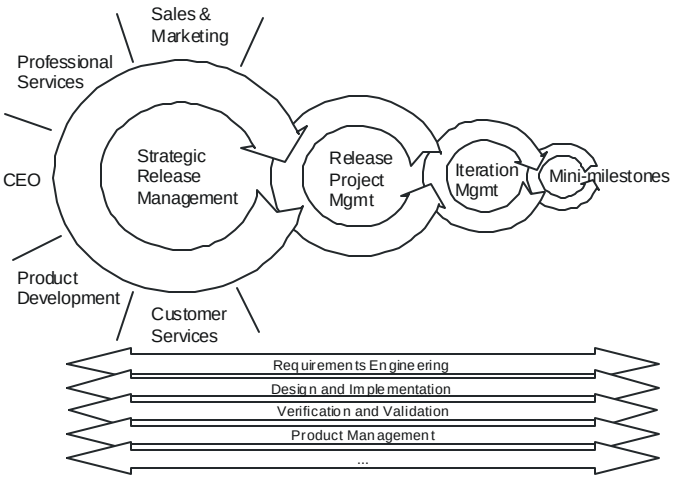


Exhibit 1 provides an overview of the framework, depicting the four cycles of control and some of the software engineering activities that span all of the cycles. The radius of a cycle symbolizes the time perspective taken, the larger the radius, the longer the time perspective.

Each cycle includes activities for planning what the cycle to its right is supposed to accomplish, thus giving constraints and goals within which to work and which to monitor at the next level. The cycle to the right then independently plans how to reach the goals within the given constraints and gives feedback on progress to the cycle on its left.

The leftmost cycle, *strategic release management*, is the interface between business management and product development and takes a long-term view to release management. The different stakeholders of the product deal with the available information and make decisions about the content, type, and timing of each individual release, the usage of the company's product development resources, and the most important technology choices. Strategic release management is an ongoing cyclic activity. The group of stakeholders can meet regularly, for instance at the end of each iteration cycle, or when a need arises, for example stemming from changes in the markets or problems in the release projects.

Release project management is concerned with the individual release projects. The products are developed in release projects in an iterative and incremental fashion. Beyond the usual project management activities, release project management has to consider the number of iteration cycles to be executed as well as the content and schedule of those cycles. The basic idea of an iterative and incremental development process is to deliver product versions early to facilitate early user feedback. 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 iteration cycle(s). The length of a release project could range from three to twelve months, depending on the nature of the business.

Iteration management concerns the individual iteration cycles. The main issue is the detailed task planning for the iteration cycle, aiming at achieving a tested and stable product at the end of the cycle. Part of the detailed task planning is to plan the

schedule of the mini-milestones that are used to pace and synchronize the development effort. Iteration cycles typically range from one to three months.

Frequent integration of the system, or *mini-milestones*, such as daily or weekly builds, are used to get a better indication of system status during development. It is a mechanism for synchronizing the effort of the development team.

The arrows at the bottom of Exhibit 1 depict the actual software engineering activities that span all cycles. The level of detail and emphasis of the activities vary in each cycle. For instance, requirements engineering is at a high level of abstraction in strategic release management, where market needs and business opportunities are identified and sets of features and functionality are elicited, prioritized, and scheduled into different release projects. The release projects take these as input and schedule them into the different iteration cycles, specifying some more detail into them and so on. In the same way, on the strategic release management level, product management can be seen as management of the different product versions in the market and as the source code control, etc., on the mini-milestone level.

Strategic Release Management. The leftmost control cycle, strategic release management, is the interface between business management and product development. It incorporates a long-term view to product and technology planning. The main purpose of strategic release management is to plan the release cycles and the preferred and prioritized content, type (e.g., major release, minor release, or service pack, emergency fix) and timing of each individual release. 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. Strategic release management allocates resources to product development and also to services requiring attention from product development, such as maintenance or integration into customer systems and tailoring. Major technology decisions, including architectural decisions, that span beyond the life cycle of a certain release project are made. These include decisions about the product platform or core components of the product. Product line decisions are also of concern here, especially when a company grows and diversifies its product offering.

Requirements engineering at the strategic release management level is concerned with eliciting, specifying, and prioritizing requirements from different stakeholders, based on market needs and business opportunities. Requirements engineering forms the main interface to the individual release projects. Exhibit 1 gives an example of some of the possible stakeholders or stakeholder representatives that might be involved. 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 vision statement used at Microsoft (Cusumano and Selby, 1995) provides a way of communicating the purpose and goals of the product. The vision statement is used to give structure to the development effort, at the same time accommodating change and flexibility during the development process. A rough effort estimation should be attached to the requirements or features to enable consideration of resource implications to the release projects.

The strategic release management cycle creates and maintains an aggregated release project plan or product and technology road map (Wheelwright and Clark, 1995). The plan shows when different types of releases are scheduled to take place, which combined with the requirements and product vision shows roughly the content of each release. The 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, or developing an entirely new product. The implication of different types of work is that it should be managed, controlled, and resourced differently, and this must be considered in resource allocation.

In a very small company a single person most likely acts in multiple roles and strategic release management is done by as few as three to four people. There should be regular meetings, for instance at the end of iteration cycles to check the situation and update the plans.

Release Project Management. The next control cycle, release project management, is concerned with individual release projects in which the actual product versions are created. The purpose of release project management is to make sure that the assigned product release gets done.

The main activity of release project management is to plan and specify the release project according to the priorities specified in strategic release management. This includes planning the length, content, and number of iteration cycles in a project. The product is built in such a way that feedback can be gained on the progress of development. Depending on whether the release deadline is more important than completing all the features of the product, release project management makes decisions about adding or prolonging iteration cycles, adding resources or dropping features. Features are dropped if they cannot be finished in time for the release. Having a prioritized list of features facilitates this. Resources can seldom be added in a small company and the risk of prolonging the project if resources are added in the middle of it is high (Brooks, 1999).

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 (MacCormack, Verganti, and Iansiti, 2001). Another consideration is perceived risk. The greater the perceived risk impact, the earlier the feature should be implemented. In this way there is enough time to react to the possibly realized risk and gain better control of the project. The prioritization of features done in strategic release management must also be considered, ensuring that the most important features— probably the ones with the highest perceived business value—are developed as early as possible.

As an example, in eXtreme Programming (Beck, 2000), 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. *Business* describes the requirements as "stories" and *Development* estimates how long the different stories will take to implement. Then Business prioritizes the stories and fixes the date or scope of the next release based on the resource budget from Development.

The output of release project management consists of plans for the iteration cycles and feedback on project progress to strategic release management. The plans contain, for instance, the iteration schedule, budget, and scope. The feedback to strategic release management contains among other things discrepancies to the plans.

As a rule of thumb, we suggest that the length of a release project should lie between 3 and 12 months, depending on the type of release and the characteristics of the business.

Iteration Management. The purpose of iterations and iteration management is to build a stable, working product in increments that can be used to get feedback, both technical and user.

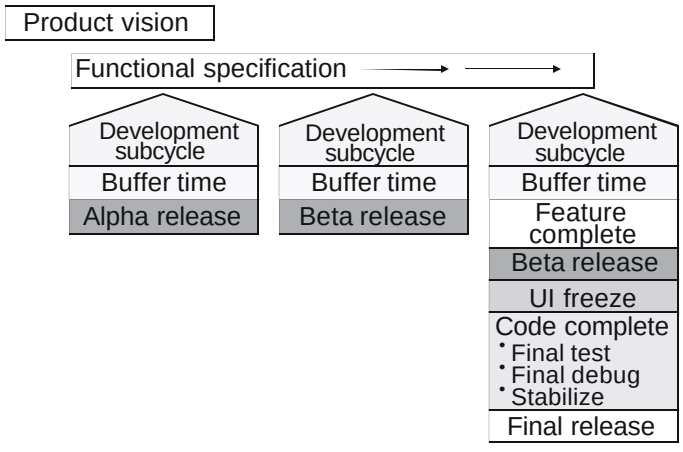
In each iteration, a set of use cases or features are identified, specified in detail, designed, implemented, and tested. At the end of each iteration there should be a working product. This facilitates revisiting strategic release management to check the market situation and decide on the focus and features to be developed during the next iteration. This approach has been found good for developing high-quality products in an environment with high uncertainty and rapidly changing requirements (MacCormack, Verganti, and Iansiti, 2001).

To gain more control of the development effort, iteration management plans and paces the development into smaller chunks, mini-milestones. These could be, for instance, daily or weekly builds including regression testing, in which the system is integrated, i.e., the efforts of the different people or teams are synchronized. In this way there is up-to-date information about system status at regular intervals to help find early warning signs that the plans are not going to be met.

An example of this approach can be found at 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 (Exhibit 2).

This way Microsoft can fall back on the previous shipment if the next cycle fails. Feature teams or individual engineers synchronize their work by building and testing the product on a daily basis. The found defects are fixed immediately. The process has been accordingly named Synchronize-and-Stabilize (Cusumano and Yoffie, 1999).

Exhibit 2. The synchronize-and-stabilize process (redrawn from Cusumano and Yoffie, 1999)



XP approaches incremental development by doing the development in very short iterations, lasting one to three weeks. Highsmith (2000) talks about time-boxing projects as a mechanism for managers to force periodic convergence of a system. All this implies that a certain amount of freedom and flexibility 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.

The output of iteration management is a detailed task plan and mini-milestone schedule. The output of an iteration cycle is a working product increment that has part of the functionality of the final product. Iteration management gives feedback to release project management about discrepancies in the plans during the iteration cycle.

Typically, an iteration cycle is one to three months in duration. Experience in the companies we work with has shown that if there is more than three months between tangible results of the product development effort, product development easily runs out of control. The shorter the iteration cycle, the fewer requirement changes are likely to happen. One of the main principles in Scrum is that during a so-called 30-day sprint (which is equal to an iteration cycle) requirements are frozen and added to the product backlog for consideration for future sprints (Schwaber and Beedle, 2002). This makes it possible for the development team to concentrate on the task at hand without constant change pressure.

Mini-milestones. The purpose of mini-milestones is to integrate and synchronize the efforts of individuals and teams in the product development project. Mini-milestones can be instantiated, e.g., in the form of daily builds, as in the case of Microsoft (Cusumano and Selby, 1995). At Microsoft the daily build-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.

XP approaches the daily build-daily test cycle more aggressively using continuous integration at a minimum of once a day and presenting the idea of "test first." The idea is to write a unit test for every production method that could possibly break. The tests are written before the code, also serving as a specification or explanation for the methods. The unit tests must be passed at all times giving confidence in changing the code, since the tests should pick up defects introduced to the system.

The output of mini-milestones is the different work products that constitute the product and feedback to iteration management on the progress of development.

Mapping Existing Process Models to the 4CC. In the previous sections we have included examples on practices that existing process models offer for the different control cycles. Exhibit 3 summarizes some of the key elements of three existing process models mapped to the concepts of the 4CC framework. The three models were chosen as representative examples of existing approaches to managing software product development.

Synchronize-and-Stabilize is the most comprehensive of the three, clearly covering all the cycles of the 4CC framework. The two other models, Scrum and XP are examples of agile process models.

The mapping shows that existing process models can to a great extent be described through the 4CC framework. The mapping

Exhibit 3. A mapping of the key elements of exisiting process models to the concepts of the 4CC framework

Concepts in 4CC process model\	Strategic release management	Release project management	Iteration management	Mini-milestones
Synchronize-and-stabilize (Cusumano and Selby, 1995)	- Multiyear product plans - Vision statement and product goals - Program reviews	- Prioritized list of desired features - Outline functional specification - Alpha and beta releases - Feedback from internal and external users	- Development subcycle - Stabilization period (feature integration, testing, problem fixing) - Final product stabilization - Buffer time	- Daily build process
Scrum (Schwaber and Beedle, 2002)	- Product backlog - Sprint planning meetings - Sprint reviews	- Release backlog	- Sprint backlog	- Daily Scrum meetings
XP (Beck 2000)		- Planning game - Metaphor - On-site customer	- Small releases	- Continuous integration

also shows that the existing models put different emphasis on the different cycles, and that strategic release management is missing from eXtreme Programming. The other models also have their limitations and none of the models have even been proposed to be used in all kinds of software projects. If detail was added to the models and the key elements of the models in Exhibit 3—such as the model's area of applicability—the table might be used as a tool for selecting engineering practices for projects in a company. The 4CC would then be used to give structure to organizing the product development management of the company showing the major decision points and organizational design and the existing models and best engineering practices could be used to provide more detail.

Lessons Learned

To date, we have partly applied the 4CC framework in four small software development organizations. This section summarizes the lessons we have learned from that work.

Deploying the framework sets the foundation for establishing a common language in the organization, which we see as one of the most valuable and tangible results of applying "process thinking." The importance of a common language concerns both the process and the product. For example, when we started developing a product roadmapping process for conducting 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.

One of the companies we have worked with reports that the implementation of the 4CC framework has helped them structure the boundaries of decision-making within the organization. The framework also made communication about the product development process to the customers easier.

Deploying the 4CC framework has encountered some resistance to change, which was expected. There never seems to be enough time for improvement activities. To battle this resistance concrete suggestions for paths of improvement should be provided.

Discussion and Further Work

We have presented a framework for managing software product development in small companies. The framework is still tentative, with several important aspects missing. As an example, measurement is not yet explicitly discussed. On that front we have been working on a toolset for the creation, management, and use of a measurement system. We plan to continue and integrate our earlier work on measurement into this framework.

The issues in the 4CC are by no means new. Management and strategy literature contains many examples on how to device a strategy for a company, how to formulate a product strategy, and so on. Also, an iterative process approach to software development was already a hot topic in the late 1980s. What has been given less attention in the literature, however, is combining these two domains, linking business management to software product development, which is one of the goals of the 4CC framework. On that front there is still a lot of ground to cover, for instance in finding and expressing the link between business models and process models.

We 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 heuristics developed for matching tools and techniques to different situations and needs. One way to do this is to describe existing process models through the concepts of the 4CC, partly like in Exhibit 3 but with more detail. Similarly, the processes of companies can be described to help discover how the engineering activities and processes could be structured and

integrated over and between the control cycles. Our intention is not to develop new process models but to apply and combine good practices from existing ones in small software product companies. This will also help in adding detail to the 4CC.

An interesting question for further work is the scalability of the framework as a company grows. Cockburn's ideas of methodology families (Cockburn, 2002) are appealing and we want to look into what it might mean in terms of our framework.

References

Beck, Kent, *Extreme Programming Explained*, Addison-Wesley (2000).

Boehm, Barry, "A Spiral Model of Software Development and Enhancement," *Computer, 21*:5 (May 1988), pp. 61–72.

Brodman, Judith G., and Donna L. Johnson, "What Small Businesses and Small Organizations Say About the CMM," *Proceedings of ICSE-16* (May 1994), pp. 331–340.

Brooks, Jr., Frederick P., *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*, Addison Wesley Longman, Inc. (1999).

Carnegie Mellon University, Software Engineering Institute, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison Wesley Longman, Inc. (1994).

Cockburn, Alistair, *Agile Software Development*, Addison-Wesley (2002).

Cusumano, Michael A., and Richard W. Selby, *Microsoft Secrets*, The Free Press (1995).

Cusumano, Michael A., and David B. Yoffie, *Competing on Internet Time*, The Free Press (1998).

Cusumano, Michael A., and David B. Yoffie, "Software Development on Internet Time," *Computer*, *32*:10 (October 1999), pp. 60-69.

Highsmith, III James A., *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House Publishing (2000).

Jacobson, Ivar, Grady Booch, and James Rumbaugh, *The Unified Software Development Process*, Addison Wesley Longman, Inc. (1999).

Kasanen, Eero, Kari Lukka, and Arto Siitonen, "The Constructive Approach in Management Accounting Research," *Journal of Management Accounting Research*, *5*:Fall (1993), pp. 243–264.

MacCormack, Alan, Roberto Verganti, and Marco Iansiti, "Developing Products on "Internet Time": The Anatomy of a Flexible Development Process," *Engineering Management Review*, *29*:2 (Second Quarter 2001), pp. 90–104.

Royce, Walter W., "Managing the Development of Large Software Systems," *Proceedings of Wescon* (August, 1970), pp. 1–9.

Schwaber, Ken, and Mike Beedle, *Agile Software Development with Scrum*, Prentice Hall (2002).

Wheelwright, Steven C., and Kim B. Clark, *Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency, and Quality*, The Free Press (1995).