2002

Managing Incremental Development: Combining Flexibility and Control

Jacob Norbjerg
Copenhagen Business School, jno.itm@cbs.dk

MANAGING INCREMENTAL DEVELOPMENT: COMBINING FLEXIBILITY AND CONTROL

Jacob Nørbjerg

Department of Informatics, Copenhagen Business School, Howitzvej 60,
DK-2000 Frederiksberg, Denmark
ph: +45 3815 2478, fax: +45 3815 2401
e-mail: jacob@cbs.dk

ABSTRACT

The current demand for flexible software development makes software development organizations consider iterative and incremental development approaches as alternatives to the classical waterfall software development model. This, however, may jeopardize process visibility and manageability as well as product quality. Therefore, modern software development organizations need to find ways to install flexible development processes without sacrificing project overview and control. This paper reports experiences from a real life project that used timeboxing *as the basic organizing principle in an incremental and iterative design and construction process. The project's experiences show that such a process is indeed both flexible and manageable but that it requires periodic planning and re-planning, explicit concern for coordination and synchronization activities, high process discipline and organizational readiness to accept fluctuating requirements.*

1. INTRODUCTION

The traditional waterfall life cycle model for software development project planning and management has been challenged for several years [Boehm 1988]. The model is criticized for its rigidity and poor ability to deal with unstable and/or uncertain requirements, the late identification and resolution of technical risks, and the accumulation of delays and quality problems until the final test and integration phase. In recent years, therefore, several alternatives to the waterfall model have emerged [c.f. Booch et. al 1999, Beck 1999, Cusumano and Selby 1997, DSDM 1998, Rising and Janoff 2000]. The alternative models are based on, e.g. prototyping, short development cycles, several releases and continuous negotiation of requirements.

However, the alternatives to the wtaerfall model introduce other risks related to poorer product quality, late requirements specification (if at all), planning uncertainties, poor process visibility and lack of documentation [Boehm et. al. 1984, Mathiassen et. al. 1995, Sommerville 2001]. Furthermore, although software project teams may not *follow* the waterfall model, the model still dominates software engineering textbooks as well as many organizations' official quality handbooks [Sommerville 2001]. Thus, there is a need for research into the practical use of more flexible process models, especially into how to apply such models without sacrificing process overview and product quality.

This paper reports how an industrial development project used risk management and timeboxing [Stapleton 1997] to successfully plan and control an iterative and incremental development project. The project, a combined software and hardware development project, took place in a company that manufactures industrial process monitoring and control systems.

The most important experience from the project is that it is possible to maintain process transparency and project control in iterative and incremental development projects. There are however costs and risks to consider:

- re-planning, redesign, and integration of evolving products shall take place at regular and planned intervals,
- the development organization and its customers must accept fluctuating requirements,
- quality plans must balance between the need for high quality (in the end product) and the constant production of executable – if not perfect – intermediate products,
- potential staff burnout due to frequent delivery deadlines.

The next section elaborates on the discussion of software process models and describes – in some detail – the timeboxing concept used in the experiment. Section 3 describes the research method, and section 4 describes the development project. The experiences from the project are summarized in section 5 and section 6 discusses the experiences. Section 7 concludes the paper.

2. CONTROLLING FLEXIBILITY

Software process – or life cycle – models support project planning and control by defining major activities, their sequence, deliverables and milestones in software development projects. The models usually describe the process at a high level of abstraction, leaving it to more detailed process descriptions to explain *how* to carry out activities and document results.

The waterfall model [Boehm 1988, Royce 1970] has for a long time been the dominant reference model for software processes. This model emphazises a clear separation and sequential ordering of activities; e.g. requirements specification, design, construction etc. The model is easy to understand and offers – at least in theory – a manageable process with well-defined checkpoints and early requirements freeze as a foundation for estimation and planning. It is, however, well known that the model does not work so well in practice [Boehm 1988]. It has especially proven difficult to produce precise and stable requirements at the early stages of a project. Discovery of serious technical problems may furthermore be delayed until the final integration and testing phases where the cost of solving the problems can be very large.

The increased emphasis on flexibility and short development cycles further exposes the shortcomings of the waterfall model. Baskerville et. al. report how web-development companies negotiate quality, development time, requirements, and quality to deal with short development cycles and fluctuating requirements [2001]**.** Truex et. al argue that, as organizations become less stable and more emergent we need to reorient systems development towards flexible project organizations, continuous development, prototyping, and acceptance of incomplete and ambiguous specifications [1999].

To meet these challenges, several alternative software development models that build on variations over incremental and/or iterative development, have been suggested. In a now classic article Boehm proposed a generic "spiral model" based on risk analysis and repeated process adjustments [1988]. The spiral model is generic and very abstract but several other methods with more operational process models have been suggested, e.g. DSDM , MSF, The Unified Software Development Process, Scrum and xP [Beck 1999, Boch et. al. 1999, Microsoft 2001, Rising and Janoff 2000, Stapleton 1997].

The alternatives to the 'waterfall model' all share a number of basic ideas:

- early demonstration of central system properties through prototypes,
- repeated, short development cycles aiming to quickly produce an executable system,
- the end-product is delivered as a series of separately released sub-products (increments),
- evolving requirements.

This type of development process introduces new risks and problems, such as the absence of reliable estimates and plans, fluctuating and difficult to manage requirements, and risk of poor product quality [Boehm et. al. 1984, Mathiassen et. al. 1995, Sommerville 2001]. We do not know, however, to what extent present day companies actually use this type of process models to plan and control projects, and not much research reports from their experiences.

The project discussed in this paper defined its own iterative and incremental development process and applied the timeboxing concept [DSDM 1995, Stapleton 1997] to manage the process. The next two sections brriefly describe the timeboxing concept and explains the use of *iterative* and *incrementa*l within the project.

2.1 Timeboxing

Timeboxing as a project management technique has been associated with the Rapid Application Development method (RAD) [DSDM 1995, Stapleton 1997] but similar approaches to project management can be found elsewhere; e.g. *Sprints* in the Scrum method [Rising and Janoff 2000] or Microsoft's daily builds [Cusumano and Selby 1998].

Timeboxing focuses on the time allocated to a task, rather than on the task itself. That is, a timebox describes a set of *related* and *prioritized* tasks to be completed within a given deadline. The timebox results in a product; e.g. a requirements specification, a prototype, an increment, an executable sub-system a test plan etc., that is delivered *and* reviewed within the time allocated.

The timebox is assigned to a small group of developers – the timebox team – who are responsible for meeting the timebox objective with little or no outside interference. The team members can decide to reduce the scope – the number of tasks completed – of the timebox if they realize they cannot meet the objective. They can, for example, implement fewer features in a sub-system and/or release it in spite of known quality problems, they may specify fewer requirements, execute fewer test cases, etc.. To manage (inevitable) scope adjustments is the timebox' objective broken down into a prioritized set of tasks and a „minimum" acceptable scope or *success criteria* is stated.

Timeboxing is in principle applicable to any development activity but it appears to be mostly applied to activities that result in a tangible and verifiable product such as a prototype, an executable (sub)system and – in some cases – a requirements specification or a systems design. In the software development project described below, timeboxing was mainly used during design and construction to produce prototypes and increments, but the technique was also successfully used to manage other activities, such as sub-contractor management, construction of a quality plan for the project, and most importantly: to periodically integrate and stabilize sub-systems.

The project team made a number of adjustments and additions to the timeboxing concept in the course of the project. The most important of these were the introduction of s*ynchronizing periods* between successive timeboxes and the use of *mid way meeting* to discuss, and if necessary adjust, the process in the timebox. Both of these will be further described below.

2.2 Iterative and incremental development

There is some confusion over the precise meaning of the terms *iterative* and *incremental*. The developers responsible for the project described here, used the terms in a way consistent with the Unified Process [Booch et. al. 1999]. This paper adopts their definitions: *Iteration* refers to the process; i.e. the same activities, or sequence of activities, are repeated several times during the project. *Incremental* refers to the gradually growing product. Thus, an increment may involve changes (code clean-up, redesign, removal of errors) as well as additions (new functionality) to a previous release

3 RESEARCH METHOD

The research reported here was part of a three year action research project in a medium sized Danish software organization called *Meters Inc.* The researchers participated in Meters' Software Process Improvement (SPI) initiative, aiming to identify and remove problems in the company's software development processes. Meter's SPI initiative was organized around an SPI group consisting of outside researchers – among these the author of this paper – and employees from Meters. The SPI project at Meters was part of the Danish Center for SPI, a large scale action research project involving four different software organizations [Mathiassen et. al. 2002].

An initial diagnosis of Meters' software process problems had shown that the company's official waterfall process model did not meet the needs of the development projects. Members of Meters' SPI group therefore formed a support group which entered into close collaboration with the *Monitor project group* (see below) to define and gain experience with a new development process model. The author of this paper was a member of the support group. The support group and the project group defined a first version of a software process model when the Monitor project was initialized. This first version of the model was used for the project's initial planning and development activities and subsequently adjusted and refined based on the experiences gained from its use.

The support group met with the Monitor project team about twice monthly to collect experiences, adjust the model, and coach or guide the project team as needed. In addition the support group conducted an interview with Monitor's project manager towards the end of the project in order to document his experiences with the process model.

This paper is based on minutes and notes from the interview with the project manager and from the ongoing meetings between the support group and the project group. Furthermore, it is based on the material (descriptions of the development model, plans, internal reports, instruction material and other documentation), which was produced continuously by the Monitor project and/or the support group.

4. THE MONITOR PROJECT

Meters Inc. manufactures leading edge measurement instruments and systems. A typical product consists of one or more sensors – microphone, thermometer, accellerometer or other – connected to a PC with analysis and presentation software. Most projects have both a hardware and a software part but they are run as integrated projects under a common project manager.

In late 1997 Meters' *Process Control Systems* (PCS) division launched the Monitor project to develop a new version of a machine and site monitoring system (called the *Monitor* system in this paper). The project should add functionality to the system and move it to a different hardware and operating system platform. The new product was to be delivered in three releases over a two year period according to the initial project plan. The first release should develop a new group of features on a new hardware platform, and the next two releases should gradually move all functionality from the old to the new platform. The original plan was based on significant reuse of hardware and software from other Meters products, but the selling of PCS to another company created legal and financial barriers for the reuse and the plan was revised. The second plan estimated about 18 months for the first release, with the others to follow.

The following concerns only work on the first release which took place from early 1998 to May 1999 when the project was cancelled due to changed product strategies in PCS' new parent company. During this period the size of the project group varied between 10 and 13 software and hardware developers, including the project manager.

At formal project launch in early 1998 the support group and the project group defined an outline software development model for the project. The model had three stages:

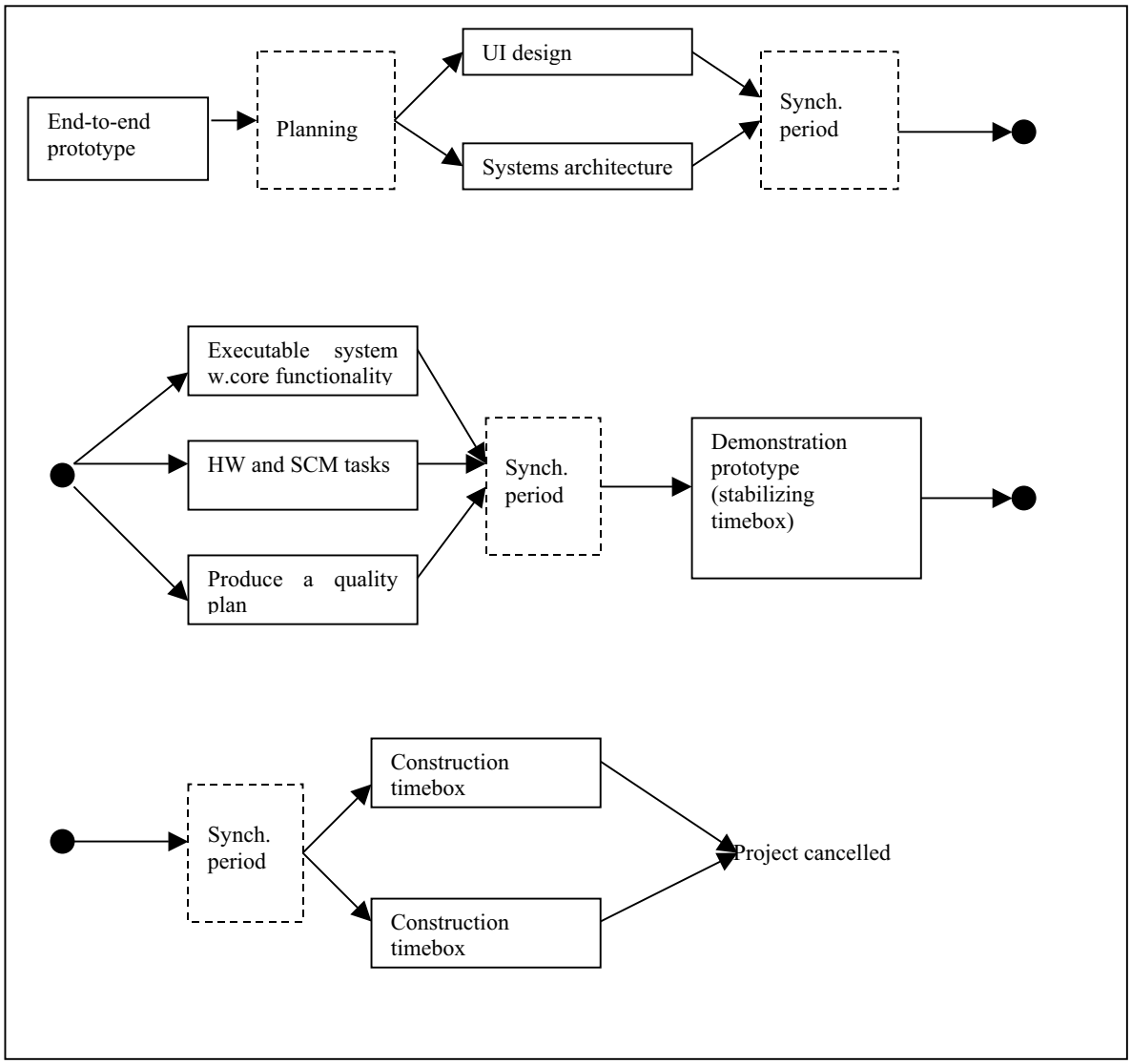

Figure 1. The sequence of timeboxes in the Monitor project

- *clarification* – identification of requirements, technical risks, architectural design, project plan, and training;
- *construction* – development of the product functionality to a „reasonable“ quality level
- *stabilization* – final integration and quality assurance to release level.

The project team decided to use iterative and incremental development during the construction stage and manage the process through timeboxing. The model was further refined and adjusted during the Monitor project. The following focuses on the use of timeboxing during construction.

Project initiation was delayed due to the above mentioned reuse discussions and hiring and training of new staff, but around April 1998 the project manager and some of the developers began to produce a first list of requirements. In May-June five developers succesfully completed the projects' first timebox (see figure 1) where they developed a small end-to-end prototype. The purpose of the prototype was to resolve risks associated with the new hardware- and software platforms, and with the new development environment.

Over summer the project manager produced the first timebox plan with design and construction activities distributed across a series of timeboxes, and in September two parallel timeboxes were

launched to produce a user interface design, resp. a systems architecture and prototype. The first team successfully completed and tested a number of (central) user interfaces, while the other group only managed to produce a design specification. Apparently this team had been „going round in circles“ because of lack of sufficient domain knowledge. This knowledge was collected and used by the user interface team, however, but it was not communicated to the other team. As a result, the architecture became much more generic and complex than what was actually needed, and it was greatly simplified in a later timebox.

The project manager revised the project plan based on the results from these two timeboxes and subsequently launched three parallel timeboxes with the following tasks:

- construct a running system able to collect, store and display a limited set of measurement data.
- produce a quality plan adapted to the development model.
- perform a bundle of hardware-related tasks, including review of products delivered from a (hardware)sub-contractor

The first two of these timeboxes met their objectives, while the third was only partially successfull The hardware-related tasks were completed, but the sub-contractor was delayed and the team was therefore unable to review his products.

The partial success of the timebox demonstrates however, that this type of activity can be managed by means of timeboxes. It turned out to be extremely effective to put the few, otherwise isolated, hardware developers on a team even though their tasks were only partially related.

The goal of the next timebox was to integrate and stabilize a demonstration prototype for the division’s new parent company. The timebox was successful and the prototype gave the project manager a clear overview of the project’s status, and confidence in the construction schedule. The schedule had to be revised, however, due to staffing problems.

At this point (February – March 1999) there was increasing insecurity within and around the project: The status of the Monitor system’s within the new parent company’s product strategy was unknown, and at the same time the project manager and several team members decided to leave the company. The reduced team launched further timeboxes based on the demonstration prototype but these were not very successful due to decreasing motivation and several disturbances from outside.

By May 1999 management finally decided to stop independent development of the Monitor system and merge parts of it with a similar product from the parent company.

Interviews with both the project manager and the team showed, however, that they found the approaches and ideas employed in the experiment useful in spite of the problems encountered and the premature cancellation of the project. The technical director of PCS was also appealed by the approach and would like to use it in future projects.

5. EXPERIENCES

The experiences from the Monitor project show that combining timeboxing and iterative and incremental development enables a project group to progress systematically towards producing a deliverable (if not complete) product while maintaining process transparency and project control.. The success depends on periodic planning and replanning, injection of synchronizing periods between timeboxes, systematic review and adjustment of team processes, and high discipline within timebox teams. Also, at a more general level, the development organization and its customers must accept flexible requirements as well as frequent re-scoping and re-planning. The experience from the Monitor project finally indicates a number of new risks and uncertainties pertaining to product quality and staff pressure.

5.1 Visibility and product focus

The project manager reported that the combination of incremental development and timeboxing gave him a reliable overview of the project's progress and status. By the end of each timebox he knew which production tasks were finished and which would have to be either postponed to a later timebox or abandoned altogether. A consequence of this visibility was that by the end of each timebox, the project manager had a clear indication of whether the project was on track or not. By January 1999, he was, for example aware that the original release date would have to be changed or the requirements adjusted. As a result, the project was re-scheduled, and the release date postponed one month.

Furthermore, timeboxing and incremental development made the team focus on implementation of functionality instead of completing modules or perfecting details. The approach also forced the team to focus on the most important requirements, while ignoring or postponing the less important. In this way they were able, at a relatively early stage in the project, to produce a demonstratable system version which could have been stabilized to release level with a reasonable effort.

Similar experiences have been reported from other development projects employing short, incremental development cycles [Baskerville et. al. 2001, Beck 1999, Cusumano and Selby 1998].

5.2 Planning discipline

The approach demanded high planning and process discipline from the project manager and the team members. After each timebox, the project manager needed time to take stock of the project's status and revise the project plan. In this process he relied on the product of the timeboxes, as well as on a list of *outstanding issues* from each timebox. The list summarized unfinished tasks as well as known problems with the delivered product; e.g. known errors, poor designs or intermediate solutions that would have to be improved before release of a final product. Based on this information the project manager revised product requirements and the project plan.

The timebox team on their side had to maintain a high degree of discipline to keep the timebox on track and deliver a product. The most important control device was the prioritized task list and the *success criteria*; i.e. a statement of the purpose of the timebox. The task list showed the team which tasks they could postpone if needed and the succes criteria helped the developers focus on the overall purpose of the time box instead of using needless time on details or goldplating.

5.3 Process monitoring: the mid-way meeting

The timeboxing idea depends on autonomous project teams [Stapleton 1997] who work towards meeting the timebox' goals in whatever way they find feasible. Experience from the Monitor project shows, however, that even highly competent and disciplined developers can loose focus when encountering a difficult or teasing problem; e.g. a module that is harder than expected to program, a delayed delivery from a sub-contractor or unclear design criteria. In these cases developers on the Monitor project would jeopardize the success criteria by spending (too) much time solving the problem instead of finding a way to work around it. To amend these problems, the support group introduced the *mid-way* meeting about one-third to half-way through a timebox. The purpose of the meeting was to discuss, and if necessary adjust, the *process* in the timebox, not to measure progress or discuss the task list. Experience from the Monitor project indicates that the meetings should be organized and facilitated by ‚outside' process consultants; e.g. from a quality or process group. The project manager shall, as a general rule, not be involved in the meetings since his presence tends to turn their focus away from the process towards evaluation of project and product status.

5.4 Synchronizing periods between timeboxes

The work within timeboxes is very intensive and product oriented and does not leave room for other project tasks. The Monitor project therefore introduced so-called *synchronizing periods* (see figure 1) of between one and two weeks between timeboxes. The periods were typically used for

- status review and re-planning
- integration of products from parallel timeboxes and production of a new baseline
- education and training
- relaxation and recharging

The synchronizing periods are, by nature, more flexible and less structured in terms of the tasks to complete than the timeboxes, but we found that they too require some planning and monitoring. At one point was a synchronizing period allowed to ‚slip' two weeks because of unexpected integration problems. The slip caused further delays and problems in the subsequent timebox which could have been avoided if the project manager, upon realizing the problems, had revised the project plan and scheduled a *stabilizing timebox* (see next section).

5.5 Quality concerns

Incremental and/or prototyping approaches to software development may result in less stable and robust systems [Mathiassen et. al. 1995]. In the Monitor project we saw how, in a similar way, timeboxing moved focus from producing a stable and complete to a satisfactory product. This created a risk of pushing quality problems such as; e.g. unstable solutions, unfinished interfaces, and uncoordinated designs or subsystems ahead of the process, causing increased pressure on the final stabilization phase.

Thus, an incremental development project like the Monitor project faces two somewhat contradictory quality requirements: The gradual completion of the product means that it is inefficient or impossible to test and stabilize to release quality before the final stabilization phase. On the other hand should sub-products be stabilized and integrated as early as possible to reduce the pressure on that phase.

The Monitor project team attempted to deal with these issues in several ways: First, the timebox teams maintained a list of outstanding issues as described above. Second, the project introduced the idea of periodic stabilizing timeboxes to resolve larger or more complex integration and quality issues, and produce a reasonably stable version of the evolving system. Experience from the project indicates that these *stabilizing timeboxes* should be injected for every 2-3 construction timeboxes and that they shall be planned and managed in the same way as other timeboxes. Finally, the project manager, the project's quality manager and the head of PCS' QA department defined a quality plan adapted to the development model. The plan specified three quality levels and corresponding quality assurance activities to be used in normal construction timeboxes, the so called stabilizing timeboxes, and the final stabilizing phase, respectively.

The premature closure of the Monitor project means, however, that we do not know if these measures were sufficient to resolve the quality risks discussed above

5.6 The length of a timebox

A timebox period for about 6 weeks has turned out to be appropriate. During this time, the group can produce a satisfactory product that implements a limited and meaningful functionality.

The time within the time box is distributed as follows:

- Planning: 1 week
- Construction: 4 weeks
- Demonstration, review, documentation and cleaning up: 1 week

Shorter timeboxes give too little time for construction, while longer are harder to monitor and increase synchronization and coordination problems across parallel timeboxes.

5.7 Staff pressure

Timeboxing has been reported to result in burnout due to the frequent deliveries and unbreakable deadlines [Van Camp 1996]. The developers in the Monitor project did report of an occassional high work load but the synchronizing periods allowed them to relax and recharge between timeboxes. Interestingly enough, the project manager reported of inverse relations between periods of pressure and slack: Reviewing and replanning the project put him under extreme pressure during synchronizing periods, while the timebox periods were more quiet and allowed him to deal with other issues such as negotiations with customers, long-term planning, etc.

The developers also reported that the timeboxing approach resulted in an interesting combination of satisfaction and frustration: They were satisfied with the frequent deliveries of visible and useful parts of the final system but on the other hand they became frustrated because the product never reached an acceptable – from their professional viewpoint – quality level.

6. MANAGING COMPLEXITY AND UNCERTAINTY

Mathiassen and Stage proposes that systems development projects face two, somewhat contradictory requirements [1992]: On the one hand, must a systems development project deal with the *complexity* of understanding and structuring requirements, organizing a project, evaluating solution alternatives etc. On the other hand are projects characterized by considerable *uncertainty* regarding the properties of the system to be produced, the most feasible way of producing it, staff availability and capability etc. To reduce complexity a project applies an analytical, specification oriented approach that allows it to systematically create useful, less complex abstractions out of available information. To reduce uncertainty, on the other hand, the project must adapt a more experimental, prototyping approach that allows it to try out different options and explore the capabilities of technology and staff.

The two approaches are intrinsically related. The simplifying abstractions resulting from an analytical approach creates new uncertainties concerning the relevance and validity of the abstractions and specifications. An experimental approach, on the other hand, creates more information and new options that must be analyzed and sorted out before the project can proceed. Mathiassen and Stage therefore recommend that systems development projects apply mixed strategies that combine analytical and experimental approaches to software development in a systematic way. The Monitor project gives one example of such a systematic mix: In the terminology used by Mathiassen and Stage the construction timeboxes deal with uncertainty by testing options and producing and demonstrating versions of a running system. This gives both the project team and the customers a clear view of the project's status and the properties of the final product. The activity increases complexity however by adding to the list of unsolved problems and by creating new sub-products and versions of the system. The complexity is dealt with in the synchronizing periods through replanning, sub-system integration, baselining, and – if need be – by injecting stabilizing timeboxes.

It is perfectly true that the waterfall model similar planning and replanning activities can and should take place within waterfall development projects as well. As a minimum at phase transitions at prefereably at shorter intervals. The approach to planning adopted in the Monitor project differed from

this in several ways: First, the completion of a set of parallel timeboxes provided a natural time space for planning activities at regular intervals. Second, the prototypes and executable system versions resulting from a timebox provided, according to the project manager, a very reliable project status, including knowledge of hard implementation problems pending quality problems etc. Similar information would not have been available until very late in a project managed according to the waterfall model.

One uncertainty remains however: the final properties of the system are unknown until very late in the project. At the end of each timebox the project team has to reconsider system requirements in the light of what was achieved in the timebox – redefining and reducing requirements if needed to meet the project deadline. Therefore, the initial list of requirements represent only an intention which will be continually adjusted as the project progresses. This observation corresponds to the recommendations given in the xP approach where customers and developers regularly negotiate what features to implement, given time and manpower constraints [Beck 1999] and to the properties of web-development reported in [Baskerville et. al. 2001]. It also corresponds to the idea of emergent systems discussed by Truex et. al. [1999].

8 CONCLUSION

The introduction to this paper raised the question of how a development project may address the problems inherent in waterfall like approaches to software development without loosing project control and jeopardizing quality. The Monitor project is one example of how this may be done. The approach adopted in the project is similar to recommendations and descriptions from other (current) development models like xP, RAD, and MS development framework, and the experiences from „internet development“: That is, a focus on rapid production of an executable and potentially marketable – if not quite stable – product, relatively short construction cycles, and flexible/evolving requirements. The experience reported in this paper gives us more insight into how such development approaches work out in the „trenches“, how they can be managed, and of the potential risks involved.

Among the particular results from the experiment are the importance of regular planning and replanning between construction activities, injection of synchronizing periods between timeboxes, the distinction between construction timeboxes and stabilizing timeboxes, and the use of mid-way meetings to review and adjust team processes during a timebox.

The Monitor project applied particular measures to resolve risks concerning the quality of the final product and potential staff burnout due to high pressure and other frustrations. The ill fate of the project caused by external events does not, however, allow us to conclude anything about the effectiveness of these measures.

Thus, there is a need of further research into the practical application of timeboxed and iterative approaches to systems development to resolve these and other questions.

REFERENCES

Baskerville, R., Levine, L. Pries-Heje, J., Ramesh, B. and Slaughter, S. (2001): How Internet Software Companies Negotiate Quality, *IEEE Computer*, **34**, (5), pp. 51-57 (May).

Beck, K. (1999): Embracing Change with Extreme Programming, *IEEE Computer*, **32** (10), pp. 70-77 (October)

Boehm, B.W, Gray, T.E. and Seewaldt, T. (1984): Prototyping vs. Specifying: A Multiproject Experiment, *IEEE Transactions on Software Engineering*, **10** (3), pp. 290-303 (May).

Boehm, B.W. (1988): A Spiral Model of Software Development and Enhancement, **21** (5), pp. 61-72, (May).

Booch, G., Jacobson, I. and Rumbaugh, J.: *The Unified Software Development Process*, Addison-Wesley, 1999.

Cusumano, M. A. and Selby, R. W. (1997): How Microsoft Builds Software, *Communications of the ACM*, **40**, (6), pp. 53-61, (June).

Cusumano, M.A. and Selby, R.W. (1998): Microsoft Secrets, Tochstone, N.Y.

DSDM Consortium: *Dynamic Systems Development Method, DSDM Manual version 2*, Tesseract Publishing, 1998.

Mathiassen, L. and Stage, J. (1992): The Principle of Limited Reduction in Software Design, *Information Technology and People*, **6**(2-3), pp. 171-185.

Mathiassen, L., Seewaldt, T. and Stage, J. (1995): Prototyping and Specifying: Principles and Practices of a Mixed Approach, *Scandinavian Journal of Informaiton Systems*, **7**, (1), pp. 55-72.

Mathiassen, L., Pries-Heje, J. and Ngwenyama, O. (2002): *Improving Software Organizations*, Addison-Wesley.

Microsoft (2001): *Microsoft Solutions Framework overview*, Microsoft (http://www.microsoft.com/business/services/framework.asp)

Rising, L. and Janoff, N. S. (2000): The Scrum Development Process for Small Systems, *IEEE Software,* **17** (July/August), pp. 26-32.

Royce, W. W. (1970): Managing the development of large software systems, *IEEE Wescon, pp. 1-9.*

Stapleton, J. (1997): *DSDM. Dynamic Systems Development Method*, Addison Wesley.

Sommerville, I. (2001): *Software Engineering*, 6th ed, Addison-Wesley.

Truex, D. P., Baskerville, R. and Klein, H. (1999): Growing Systems in Emergent Organizations, *Communications of the ACM*, **42**(8), pp. 117- 123 (August).

Van Camp, K. (1996): Why RAD is BAD, *Object Expert*, pp. 68-70 (August).