Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Information and Software Technology 47 (2005) 587–608
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

How to steer an embedded software project: tactics for selecting the software process model

Petri Kettunen*, Maarit Laanti[1]

Nokia Corporation, P.O. Box 301, 00045 Nokia Group, Finland

Received 19 May 2004; revised 7 November 2004; accepted 7 November 2004
Available online 25 December 2004

Abstract

Modern large new product developments (NPD) are typically characterized by many uncertainties and frequent changes. Often the embedded software development projects working on such products face many problems compared to traditional, placid project environments. One of the major project management decisions is then the selection of the project's software process model. An appropriate process model helps coping with the challenges, and prevents many potential project problems. On the other hand, an unsuitable process choice causes additional problems. This paper investigates the software process model selection in the context of large market-driven embedded software product development for new telecommunications equipment. Based on a quasi-formal comparison of publicly known software process models including modern agile methodologies, we propose a process model selection frame, which the project manager can use as a systematic guide for (re)choosing the project's process model. A novel feature of this comparative selection model is that we make the comparison against typical software project problem issues. Some real-life project case examples are examined against this model. The selection matrix expresses how different process models answer to different questions, and indeed there is not a single process model that would answer all the questions. On the contrary, some of the seeds to the project problems are in the process models themselves. However, being conscious of these problems and pitfalls when steering a project enables the project manager to master the situation.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Software project management; Software process models; Risk management; Embedded systems; New product development

1. Introduction

Managing modern industrial software projects successfully requires situation-aware control with the possible and oncoming troubles, taking the anticipated and even unexpected situational conditions into account. A powerful tool any project manager might then have is the power of initially choosing and—if necessary—later revising the software process model.

In this paper we present a systematic approach when it would be wise to use a certain software process model under certain project conditions, and why. Specifically, we are interested in investigating how different process models cope with different project problems. The purpose is to provide pragmatic aids for practicing project managers by combining and distilling knowledge from a variety of literature sources coupled with our practical experiences.

Obviously there are many other ways of directing the project course than just selecting and adjusting the software process. The project management tools in this sense form a very wide arsenal. Some of these belong to the area of organization psychology; some belong to the area of financial control. Our purpose here is certainly not to cover all different areas of effective project management.

In general, there is a wide range of software development project types ranging from large contract-driven IT/IS systems to small in-house developments. While they share many common characteristics, each project type and context induce particular considerations. In this study we focus on one specific type of software projects, namely market-driven development of embedded software for telecommunications products (e.g. mobile phones, radio network elements).

* Corresponding author. Tel.: +358 50 382 3672.
E-mail addresses: petri.kettunen@nokia.com (P. Kettunen), maarit.laanti@nokia.com (M. Laanti).
[1] Tel.: +358 40 530 8056.

0950-5849/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2004.11.001

Even within this category there are many different project types, such as completely new product development, new features development for existing products and derivatives, and platform developments. Here we limit ourselves to the first type, i.e. the software development for a whole new product. We have made that limitation to new product development, since we feel that there is much more freedom for the project manager to choose the initial software process used, than to make changes to a one that has already been established and used for many past software releases. Neither of the two limitations is though exclusionary nor definitive for the usage of our guidelines—the reader is encouraged to explore the suitability to her own application area.

The rest of the paper is organized as follows. Chapter 2 explores the background and related work, and sets the exact research question. Chapter 3 then describes our solution ideas, while Chapter 4 evaluates them. Finally, Chapter 5 makes some concluding remarks, and outlines further research ideas.

2. Many software process model alternatives

2.1. Software process models and project problems

In this paper, we define 'software process model' broadly so that it includes all the project life-cycle activities of project planning, tracking, and requirements management as well as the actual software construction and release. The process model defines the overall flow and order of the project work. This definition covers also the new agile software development methods.

Over the years, there have been many different software process models around. Many research investigations and numerous software engineering guidebooks compare and contrast the different models, see for example [2–3,9–10, 14–15,29,33,36,44–45,47,49]. There are in addition various handbooks, 'checklists', and even standards available. For instance, the ISO/IEC standard 12207 has an accompanying guide showing the key differences between the waterfall, incremental, and evolutionary models [53].

Those different investigations use various different comparison viewpoints of the process models, such as

- ease of management [47]
- suitability for different development types [45]
- suitability when poorly understood, unstable requirements [33]
- means for managing different software risks (uncertainty) [36]
- size, criticality, project's priorities [14,15]
- primary objectives (e.g. rapid value vs. high assurance) [9]
- universal prescription vs. situational adaptation [2,3]
- people factors [49], and
- multidimensional home ground profiles [10]

In modern software product development environments the basic premises and assumptions of the traditional process models have been stretched so much that many such models have become partially unsuitable. In addition, the growing understanding of innovation patterns and organizational learning has influenced software engineering management (knowledge management) [26]. Because of many unknowns and uncertainties coupled with ambitious time-to-market goals, basic serial document-driven development is often not feasible. The modern business pressures and technology advances often require responsive last-minute changes in the product contents [35]. New agile software process models address such aspects.

The current trend in software process model development advocates more adaptable and flexible ways of working, i.e. moving from rigid all-defining huge organizational processes towards sketched, tailorable, agile processes. Typical way of working is to give only a few most essential practices to the project—more like a process skeleton—where the practices can gradually be added. The mental model here is mere to let the project decide about the practices whenever ready to take those into use rather than having well-set rigid model to follow [1,21,22(Ch. 8),25].

Embedded systems have in addition certain intrinsic software project problems [48]. The software developers must often understand interdisciplinary product application domain knowledge [16]. Systems engineering is then a key activity [51]. In industrial new product development environments, there are also many limiting business constraints to be taken into account [30]. Note that in complex product systems (e.g. mobile phones) there are often many profoundly different types of embedded software subsystems ranging from real-time hardware drivers to sophisticated man–machine interfaces. The most recognized software models are pure models in a sense that only software is focused into. Models used in embedded software development are often variations of these. However, most existing process models can to some extent be tuned to real-time embedded software projects by taking into account the systems engineering and hardware dependencies.

The question is now for a practicing software project manager to choose an appropriate process model for her particular project, taking into account the current and anticipated problems of the project. To the best of our knowledge none of the published investigations cited above provide comprehensive guides for such purposes from that point of view. This is what we want to address.

2.2. Research question

Based on the background in Ch. 2.1, we now set the following specific question:

- How can the project manager avoid typical project problems by selecting an appropriate software process model, based on the project situational factors?

The challenge is for the software project manager to find an appropriate process model among the many different alternatives, knowing how the selected model works under given project problem conditions [19]. For example the ISO/IEC standard 12207 simply states that the user is responsible for selecting a life cycle model, although some informative guidelines are suggested [53]. Our aim here is to offer pragmatic aids for doing this in a systematic way, preventing the basic problems of selecting a fundamentally wrong model ('Lifecycle Malpractice'), using an overly bureaucratic process ('One Size Fits All'), or even not choosing any definite process model at all [13, 33(Ch. 7)]. By making conscious choices, the project manager can also avoid any inherent disadvantages of the process model.

The rest of this paper proposes answers to that question. The research method for the question is quasi-formal comparison based on distilling features [46]. As stated in Ch. 1, our special focus is embedded software development for new telecommunications products. In addition, we concentrate on large-scale projects, requiring tens of man-years of work effort.

Our underlying premise is that the process model is a significant productivity and quality factor for large software development projects. However, we do not argue that it is the most important success factor. Often, people factors tend to be ultimate keys [49]. Nevertheless, an efficient project management and development process has been recognized to be one typical characteristic of successful projects [24].

3. Tactics for selecting the software process model

3.1. Software process model selection matrix

There are many process models available, each having different characteristics and areas of suitability. The problem is then to find good matches with the actual project situations. There are no standardized solutions for this.

To help this, we have composed a process model selection matrix. Table 1 shows that structure. Table 2 is a sample excerpt of the actual matrix (top left-hand corner). See Appendix A for the complete matrix.

This matrix (Appendix A) is basically a comparative analysis of different software process models. A notable feature of the matrix is that we have based the comparison on how well each process model tackles typical problems of large embedded software projects. The reader is assumed to be familiar with the basics of the models in order to be able to understand the analysis points.

Table 1
Software process model selection matrix (Appendix A) structure

	Software process model
Project problem, risk, failure factor	*How does this process model prevent that particular problem from happening, or helps mitigating it (in the context of large embedded software projects)?*

Note that the matrix (Appendix A) is by no means an all-encompassing directory of software process models or potential project problems. The matrix has in principle been composed as follows. We have selected the process model alternatives based on a literature survey (see Ch. 2.1) as well as on our own experiences with large embedded software development projects. The idea is to cover a wide range of models, including both traditional and modern ones. Currently our matrix includes the following process models (columns): waterfall, incremental development, Spiral model [7], RUP [27], FDD [37], ASD [22], XP [54], and 'hacking'. However, we are fully aware of the fact that for instance many other agile methodologies have been proposed [2].

Similarly, we have distilled distinct project problem areas and risk factors based on well-known investigations (for example [8,11–13,16–18,24,32–34,40–43,52]) coupled with our own large embedded software project experiences. Currently our matrix includes some 50 problem items (rows). They incorporate for example the well-known Boehm's risk list [8]. The rows are grouped according to the project life cycle: project initiation, execution, completion (see the leftmost column of Table 2). The idea here is to cover such essential factors, which make a clear difference between the models in the context of large embedded software projects. Again, we acknowledge that other factors could have been included.

In addition, the matrix includes a key point section of each process model's home ground, drawbacks, and typical pitfalls. Table 3 is the outline of that part (bottom left-hand corner in Appendix A). Assuming that the reader is familiar with each process model in general, this summary serves as a quick reminder of notable remarks. Considering embedded systems, it summarizes the applicability of each process model for large embedded software projects. Notably current agile process models do not specifically address embedded software development [39].

3.2. Using the selection matrix

A project manager can use the matrix (Appendix A) described in Ch. 3.1 in two basic ways:

(a) By columns: selecting the project's process model by comparing the basic alternatives according to the prevailing or anticipated project problem situation, i.e. by reflecting presented problem areas to her own known problem areas, and optimizing the best solutions, selecting a process model which supports it best.
(b) By rows: evaluating how specific project problems can be tackled with different process model alternatives.

One could even give ratings of problems and the solutions each method would provide—and calculate

Table 2
Software process model selection matrix (Appendix A) example

Project problems, failure factors	Software process models		
	Plan/specification-driven models		Evolutionary models
	Waterfall (serial development)	Incremental development models	Spiral model (risk-driven iteration)
Project initiation			
Unclear project objectives (lack of a project mission)	Waterfall model does not tackle especially this problem. You should stay on the specification phase, until your project objectives are clarified	Can start working on the known increments, and clarify the rest later. Note! May arise other problems later, if project is not well defined or if the definition changes much later	…
Overplanning/underplanning (e.g. 'glass case' plan)	If you can do the planning reasonably well up-front, there is less overhead than with the iterative/incremental models. However, in the case of major uncertainties…	…	…
Lack of resources (people)	…	…	…

averages or weighted averages for each method, and make analytical decisions on that basis.

4. Evaluation and discussion

4.1. Validation

At the time of this writing we are not ready to publish empirical case study data of using our process model selection matrix presented in Ch. 3 (Appendix A). However, the following examples based on certain past real-life projects within Nokia Group test some main points. Note that the examples have been sanitized for confidentiality reasons.

4.1.1. Example 1

Problem. There is a project case, where some of the requirements are known a lot earlier than other set of requirements. This kind of a case could be a project, where the underlying embedded system hardware is known, but other requirements (such as customer and user interface requirements) will only be found out later.

Suggestions. The project manager could decide, based on the rows *Unclear project objectives*, *Incomplete requirements/specs* of the matrix, knowing that there might also be *Poor requirements management (uncontrolled requirements changes)*, that a typical waterfall model used previously in this organization does not provide an optimal fit in her case. Instead, she decides to apply incremental development in such a way, that the first increment is a generic-type of solution supporting a newest version of her computing hardware, and the two succeeding increments will consist of partially customer and partially user interface requirements.

4.1.2. Example 2

Problem. An embedded software project implements a new network system algorithm, based on a recent international telecommunications standard. No other network element vendor has yet implemented it. The algorithm is complex, and the standard specifications leave some room for interpretations. Therefore, the specification work is expected to be a problematic area.

Suggestions. Considering the row *Research-oriented development*, we can see that either the Spiral model or ASD is an appropriate choice to begin with. Both emphasize resolving the major uncertainties iteratively from the beginning. Those process models do not prescribe how exactly this could be done, but for example some simulation studies or prototypes could in practice help clarifying the specification details. After resolving the specifications uncertainties, the rest of the project implementing the specifications could be run incrementally, if the program size is considerable (see row *The project is too big for 'one shot'*).

4.1.3. Example 3

Problem. The product systems design is based on complex ASIC circuits and embedded software cooperation. The product development program is initially based on

Table 3
Software process model selection matrix (Appendix A) structure (cont)

	Software process model
Home ground	*Most applicable project environment(s)—'sweet spot'*
Consequences, Side-effects, Drawbacks:	
Scope	*Coverage of the model (project life-cycle activities)*
Nature	*Methodological characteristics*
Advantages	*Key benefits*
Constraints	*Limitations and disadvantages, prerequisites*
Cautions!	*Significant risks and pitfalls*
Notes	*Miscellaneous remarks*
Embedded systems	*Particular considerations for embedded software projects*

a waterfall model. However, at a late stage, when the first ASIC prototypes become available, a subtle ASIC design fault is discovered. Because of the tight product release schedule target, there is no time to redesign the ASIC. Instead, a non-trivial software workaround algorithm is specified, requiring considerable additional software design and testing efforts.

Suggestions. The initial choice of the waterfall model may have been wrong, if such a risk has been foreseeable from the beginning (see row *Underestimation of project size, complexity*). None of the process models covered address such external dependency failures directly (row *Project external dependencies late and/or imperfect*), but such a change makes it difficult to continue with the waterfall model (row *Project redirected*). Adaptive replanning is needed. One way of tackling this problem could be to have a new concurrent feature team for working on the additional functionality (FDD).

4.1.4. Example 4

Problem. A framework project implements a new Operations and Maintenance (O&M) framework by using a spiral model. Work is well split on increments, and each increment is specified before the software units are implemented and tested. The increments are typically completed in ahead of schedule. The product specific O&M software is implemented by another team at the same time with the framework. The plan is, that the product specific O&M should utilize the new O&M framework. The product specific O&M is following a waterfall model, and it is having huge difficulties on meeting the deadlines. Later on, when the framework integration with the product specific O&M software starts, major flaws are revealed. Part of the code is written twice (once by the framework and once by the actual O&M) and part of the code seems to be missing. Also it seems, that the split to framework and product specific part was initially vaguely done. After huge struggle, the remaining framework project is stopped, and all the project personnel from the framework project and product specific O&M project is moved to one big project which goal is just to make a working O&M solution before the first product launch—by using hacking.

Suggestions. In any two tightly coupled projects, communication and interface design is always an issue. The biggest problems here were that the product specific O&M project was lacking interface specifications, and the initial set-up for both of them was vague. Actually, there was not anything wrong with the process model selected for the framework project (the increments were completed on time with the set of specified features) but the process implementation may not have been that thorough (major risks unmanaged, major documents missing). The product specific O&M project might have benefited, if it had identified its major problem (row *Incomplete requirements, poorly defined parts*) and worked with some other than waterfall process model that had better tackled this problem. Ideal choice would have been a spiral model, which cycles had been closely tight with the framework cycles. However, the vague initial set-up is so fundamental problem that it cannot be solved by any process selection.

4.2. Answering the question

In Ch. 2.2, we set a research question. We now evaluate our proposals presented in Ch. 3 against that question with respect to the literature reviews (Ch. 2.1).

What is problem-conscious project management? One part of this steering is to select the project's software process model. What are the possible pitfalls of the selected model? Could these pitfalls be avoided by careful planning or by some other means? We have addressed this in the context of large embedded software projects by composing a software process selection matrix (Appendix A).

Based on the limited set of project use cases examined in Ch. 4.1, we can conclude, that the process selection matrix works reasonably well on at least some typical embedded software project problem scenarios. However, it is certainly not a silver-bullet problem solver, and there are probably many situations in which the matrix cannot help so much. The usefulness depends much on the experience and assessment capabilities of the project manager, as illustrated in Example 4.

Our selection matrix does not provide new information about any process models nor project problem items, but the value of the matrix is in its systematic composition. The matrix contains distilled advice about the selected process models in a concise form. Notably none of the reviewed investigations (Ch. 2.1) uses the viewpoint of the comparison based on project problem factors. Ould has used a rather similar viewpoint but with a much more limited scope [36(Ch. 4)]. A recent work by Boehm and Turner includes an extensive comparison of many generic process models based on project's risk factors profile [10]. Typically, software process model comparisons are more coarse-grained, only indicating in general the circumstances when certain model is suitable or not. We have elaborated this onto a more specific level.

Finally, embedded software development puts emphasis on certain process areas as described in Ch. 2.1. The software process activities must then be focused accordingly [48]. We have highlighted this in the selection matrix by including a dedicated summary section for embedded software use (see Table 3). However, even more thorough analysis could be done. Ronkainen and Abrahamsson have made a limited investigation towards that direction [39].

4.3. Application possibilities

The main idea of using the selection matrix (Appendix A) is to first select the process model based on the problem issues (see Ch. 3.2). However, there is no reason why the matrix could be used in other ways, too.

Another use of the matrix is to evaluate an ongoing project in the case the process model has already been fixed (for external reasons). The project manager can then use the matrix to see, how the process model behaves under certain problem conditions. In case there seem to be some weak points, she can start thinking about potential future mitigation strategies. The matrix helps thus staying alert to those problems.

One can also use the matrix for training purposes. Although the matrix does not explain the basics of the process models, systematic reading of it may raise new thoughts about the project's potential risks and problems, or possibly useful new practices.

4.4. Limitations

Our process model selection matrix (Appendix A) provides alternative ways (heuristics) to manage a large embedded software development project. It does not show any one best way of running a project—there is no one-size-fits-all methodology [14]. Note that typically there is more than one way to tackle a certain problem. Also there are often some trade-offs. All in all this is about advanced software process competence (Level 3 competence according to Turner and Boehm [49]).

There is an ongoing discussion, whether new agile software development models are (or should be) CMM-compatible or not [9–10,13(ch. 4),36,45]. The original idea of many agile methods is to avoid heavyweight process rigidity, thus making them less compatible. The home ground and scope are different. However, often such agile methods can be extended to become closer to the CMM(I) framework (e.g. [28,31]). In this paper our intention is not to specifically stress CMM-compatible solutions nor to object them, but to emphasize situation-specific flexibility.

One must also notice, that no process model can ever fix all the possible problems in product creation. For some of the problems, tailoring the process might simply be the wrong measure used. Fig. 1 describes the fact that process methods provide just one viewpoint to the problems there might be. It would not help, for example, to tailor the process if the selected technology is too new and immature to the project on hand. Respectively, with process methodologies only people management issues can be tackled, leadership issues typically falls to other scope. Product issues fell in domain of business strategies and trade than anything else. Such implications for software process models have been raised for example by Curtis et al. [16].

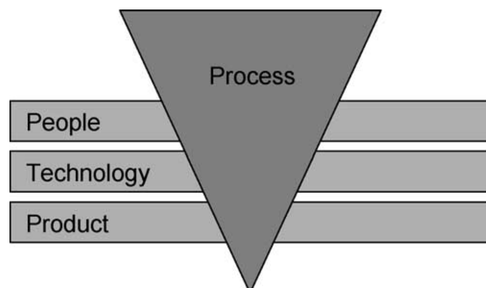

Fig. 1. Product process provides just one-angled view to the problems.

5. Conclusions

Even good managers cause projects to fail, when they don't understand the business ecosystem in which their projects must live, and the need in complex situations to know their options and to be flexible [22(Ch. 7)]. In this paper we have developed some pragmatic aids for a good project manager to cope with such challenges.

We have made a comparative analysis of a range of software process models, including agile software methods. Our specific viewpoint is to compare the models with respect to their characteristics under typical project problem conditions. The outcome of this comparison is not any particular process model recommendation, but the idea is that a project manager can use the comparison matrix (Appendix A) to support her own selection of the particular process model. Each process model amplifies certain characteristics of the project. The key is then to match the current project situation with the process model alternatives.

For a large, complex project often no single model is the best one. Instead, a hybrid model blending and balancing the features of different models is often the choice [9,10,13 (Ch. 4), 36, 45]. This depends on the varying characteristics of the different parts of the product. For example, while the user interface part may benefit from agile modelling, more stable core parts of the product may follow the waterfall.

The world of software engineering is in the state on continuous flux [5(Ch. 18)]. As the products become more complex, the project complexity increases, making the projects subject to more complex problems. Companies try to fight this complexity by hiring experienced managers (personal competence) as well as building knowledge inside the organization (such as building detailed process models). In this paper we have summarized some first-hand information to a structured form, giving the software fellows a fresh viewpoint to process models.

Our special focus has been in large embedded software projects. None of analyzed process models is specifically intended for embedded software development, but most of them are applicable to some extent. The special concerns of large embedded software projects are not so much in

Table 4
Software techniques and practices selection chart structure

	Software development method, technique, practice
Software process model	*How can this method, technique, or practice be used with that process model (in the context of large embedded software projects)? Does the process model advocate it specifically or not?*

Table A1

Project Problems, Failure Factors	Referen-ces	Software process models							
		Plan/specification-driven Models		Evolutionary models		Agile methodologies			Ad hoc
		Waterfall (serial development)	Incremental development models	Spiral model (risk-driven iteration)	Rational unified process (RUP)	Feature-driven development (FDD)	Adaptive software development (ASD)	Extreme programming (XP)	No discipline (chaotic 'hacking')
Project initiation									
Unclear project objectives (lack of a project mission)	[43/#1, #2]	Waterfall model does not tackle especially this problem. You should stay on the specification phase, until your project objectives are clarified	Can start working on the known increments, and clarify the rest later. Note! May arise other problems later, if project is not well defined or if the definition changes much later. Rule of thumb is: 80% of the requirements should be known in the beginning. Make a project priority chart, and plan the increments accordingly. Sometimes the priorities must be changed during the project	Can start working on the known parts, and clarify the rest later. Prioritize the work according to the project's major risks (time-to-market, defect reduction, response time, etc)	The Inception phase produces the project's Vision document defining the objectives (scope and constraints). The phase completes with a Lifecycle Objective (LCO) milestone, which criteria include a stakeholder agreement on the scope and the main requirements (features)	FDD does not cover the project initiation phase nor the customer requirements elicitation. However, a part of the Domain (Object) Model development is to understand, what the system is supposed to do. The model and the Features List are recommended to be agreed with the customers (stakeholders). With FDD, staged delivery is often recommended, thus the known/specified features can be made/shipped first	The Adaptive Life Cycle defines a project initiation phase, which covers explicit project mission artifacts: Project Vision (Charter), Project Data Sheet, and the product specification outline	Strong connection with the customer. The customer should be present on weekly planning sessions, and check that what is planned is consistent to what is expected. If the project objective is not clear to the customer either, it is very unlikely that the project will deliver anything useful at all. Requirements elicitation is mostly done by the on-site customer	This is often the reason why projects resort to hacking. However, hacking with unclear project objectives may lead to prototypism: you think you have a ready product when you are just the half-way there
Overplanning/ underplanning (e.g. 'glass case' plan)	[13/Planning 911] [43/#13, #15]	If you can do the planning reasonably well up-front, there is less overhead than with the iterative/incremental models. However, in the case of major uncertainties it is difficult to plan the project fully in advance. You must really proceed with the development to understand it better for realistic planning	Incremental development may make adjusting the planning easier, but planning the increments requires additional effort. If you fail to split the functionality into reasonable, prioritized increments, you may loose the benefits	The spiral model emphasizes risk-driven planning. The focus is always on reducing the (next) major risks	There are two types of plans: a coarse-grained Phase Plan, and a more detailed Iteration Plan (for the current iteration). Excessive planning beyond the current horizon is not favored. The plans have evolving levels of detail. Generally, no work should be done outside the iteration plans	Overall project planning is not covered. However, FDD emphasizes systematic up-front planning of the feature list. The feature development plan is then based on that. FDD does not really emphasize estimation. It relies more on systematic monitoring of the progress of each feature. The reasoning here is that the features are small (no more than 2 weeks)	ASD recognizes the fact that in uncertain environments the initial plan is merely a speculative outline, which will be revised after each development cycle	XP is based on continuous planning ('planning driven'). The plans are continuously adjusted based on latest achievements/metrics and the customer's changes. The recommended planning horizon is two iterations (2–3 weeks/ iteration)	Underplanning is definitely a risk here

(*continued on next page*)

Table A1 (*continued*)

Project Problems, Failure Factors	Referen-ces	Software process models							
		Plan/specification-driven Models		Evolutionary models		Agile methodologies			Ad hoc
		Waterfall (serial development)	Incremental development models	Spiral model (risk-driven iteration)	Rational unified process (RUP)	Feature-driven development (FDD)	Adaptive software development (ASD)	Extreme programming (XP)	No discipline (chaotic 'hacking')
Lack of resources (people)	[8/#1]	The waterfall model does not tackle especially this problem	If possible to start with the first increment(s), more resources may become available later. The first resources should though be competent, otherwise it might be that the first increment is not usable at all	The spiral model does not tackle especially this problem	RUP does not cover resource management issues (hiring). However, a part of the iteration management is the 'acquiring' of staff. A balance must somehow be found between the resources, effort, and schedule for each iteration	FDD does not cover resource management. Prioritize the features, and concentrate on the most important ones. Make effort estimation analysis and adjust the plans to what is reasonable with your resources	Each project should have an Executive Sponsor controlling the resourcing. The project team and the sponsor should agree on the project targets and the resource needs during the project initiation phase. After each cycle, re-evaluation should be done	XP needs a few, skilled resources. If you have lack of resources, you should not try XP at all	Too few resources is the main reason, why hacking is usually taken as a project practice. You have to notice, though, that some things (like documentation) is typically left undone. This may save some resources, but the longer-term consequences can be severe
Lack of competence (personnel shortfalls)	[8/#1, 38/#1, 43/#29]	Cannot move to the next phase until the previous one is completed. This requires learning the needed competence as a whole. Sometimes this is a typical pitfall, delaying the progress. This may be a risk for competitive time-to-market goals	May be able to learn by doing the first increments. This may, however, lengthen the project schedule. There is a risk that the first increments may have to be reworked	Make first simpler versions/iterations. As the skills/knowledge is grown, make more sophisticated versions/algorithms	RUP does not cover resource management issues (training). However, it recommends defining not only the number of staff, but also their skills, experience, and 'caliber' while staffing the project. Role descriptions guide this	FDD does not tackle especially this problem (staffing). There are six key project roles defined with certain qualifications. The features are prioritized based on customer needs/expectations, so the implementation can be technically demanding already in the beginning of the project	The Adaptive Development Model encourages intensive team collaboration and learning by developing the product iteratively. In addition each member should develop his/her personal software engineering competence. However, you may not want to run an extreme project with a junior team	The XP expects the majority of the people to be basically on the expert-level. Only few novices can be trained aside the project. If you have new project personnel, you should not try XP. For example, a 'programmer' must know in addition integration, configuration management, etc. special competences	Lack of competence is directly reflected as poor quality when hacking. Professional people are usually reluctant to do any hacking whatsoever. Learning is usually not improved by hacking
Underestimation of project size, complexity, novelty	[43/#7, #10, #12, #17]	The waterfall model does not tackle especially this problem. Replanning of the whole project is needed. May even require restarting	The incremental models do not tackle especially this problem. Replanning of the project may be needed	The spiral model tackles the most uncertain areas first. Each new cycle is assessed. New estimate of the project-complete day is needed	The purpose of the use-case modeling is to clearly understand what the software must do. It is used as the basis for the project estimates. The highest risks should be tackled early	New estimate of the project-complete day is needed, if features are just bigger and more complex than estimated. The planned features should be small (no more than 2 weeks effort)	Extreme projects are by nature uncertain. Everybody must understand that from the beginning. Re-evaluation and replanning will be done after each cycle when more is learned	New estimate of the project completion day is needed. Replanning is a part of XP. The customer is always involved	The problem is that there are probably no estimates at all. New estimate of the project completion day is needed

Research-oriented development (unprecedented, either the project ends or the means of meeting them are very much unknown)		Strictly serial waterfall is hardly ever applicable for such exploratory situations, where detailed preplanning and specification are not reasonable by nature	This is not really addressed by incremental development, but early increments may help resolving some uncertainties earlier. However, planning reasonable increments could be difficult in this case	Spiral model could possibly be applied by focusing on the unknowns and uncertainties	RUP is directed more towards orderly engineering projects. In research-oriented collaborative environments, the Vision document is more important than predefined requirements	By definition, a feature is a 'client-valued function'. In research-oriented development it may be difficult to plan such items in advance	Problem-solving is by nature an emerging activity requiring flexibility. ASD absorbs this	The primary goal in XP is to put out a good quality product within reasonable time. There is a mismatch with typical research goals	This may even make some sense, since research work is by nature 'chaotic'. However, even then totally undisciplined way of working is hardly acceptable
New, immature software technology	[43/#3]	Warning! You should not try waterfall with new, immature software technology. There is a high risk your project will be cancelled. Waterfall assumes a mature, stable environment	This is not really addressed by incremental development, but it may help resolving some uncertainties earlier. The first increments could focus on clarifying the technology uncertainties	Focus on the feasibility risks first	Recommended to put more efforts on the Elaboration phase	FDD does not cover this area	This is one source of project uncertainty. ASD emphasizes gaining better understanding by iterative development cycles	Often this does not match well with the XP philosophy of 'quick planning' and 'simple design'. The infrastructure is assumed to be doable on the fly	Some ad hoc experiments may even be justified
The march order: what should be done first and what after that (phasing)	[43/#13, #20]	The march order is exactly what waterfall model is defining accurately. This is a way to make a large group of people to work towards a common goal, with clearly defined milestone synchronization gates	Plan the increments and the milestones well. Note that there is a probable pitfall here, if the milestones are not properly followed up	The planning, implementation and testing cycles follow each other. Note that the amount of cycles needed might be hard to estimate	A project comprises four phases (Inception, Elaboration, Construction, Transition). Each phase concludes with a defined milestone. The iterations of each phase are primarily ordered based on the risks	The march order follows normal specify-implement-test cycle, the features can just be on different stages at the time. Be aware though that the stages are well defined. Notably FDD does not care about the feature start dates (just the completion)	The Adaptive Planning Cycle includes assigning the tasks into the development cycles. It encourages concurrent engineering (for high speed), which may be more difficult to manage, though	Planning sessions followed by implementation rounds followed by automated testing. The iterations are recommended to be short (some 2 weeks)	The march order is typically decided by the key designer. A lot is depending on his/her competence and communication skills
The project is big of a size (maybe even a mega project), i.e. the project will require many (even hundreds of) man-years of work to complete	[43/#6]	This is were waterfall is as it best: it suits well to bigger projects (which need more formalism than smaller projects)	Use bigger (or more) increments. However, there is a limit here. Too big increments spoil the very idea of incremental development	The spiral model suits well to large, complex system projects. However, you must control the iterations carefully in bigger projects. With bigger projects the work should be split into reasonable tasks. Managing task interfaces is an issue here, because different iterations might change the task boundaries	The iterations of a larger project are longer, because the coordination of many people is more complicated	This is were FDD is at its best. FDD was originally developed to answer the problem of rather big development projects. Feature-based allocation may help to manage	In a larger project, increase the rigor and discipline. Define and monitor component dependencies systematically	There should not be more than 10 programmers in an XP project, so you can't do anything too big with it. It might be possible to have multiple concurrent XP teams, each working on their own stories	Hacking in a bigger project leads to chaos, and bad usage of the available resources (part of the project personnel may not know what they should do). You simply cannot coordinate and synchronize a large project with hacking

(*continued on next page*)

Table A1 (*continued*)

Project Problems, Failure Factors	References	Software process models							
		Plan/specification-driven Models		Evolutionary models		Agile methodologies			Ad hoc
		Waterfall (serial development)	Incremental development models	Spiral model (risk-driven iteration)	Rational unified process (RUP)	Feature-driven development (FDD)	Adaptive software development (ASD)	Extreme programming (XP)	No discipline (chaotic 'hacking')
The project is too big for 'one shot' (problem size)	[43/#13]	Waterfall model was originally intended for managing large scale software systems. However, a very complex project usually requires additional means for managing the interdependencies, in particular when there are many uncertainties	Split the project into increments. Build functionality gradually. However, this requires that you understand the 'big picture', so that the increments converge towards a complete solution	In this case the spiral model attempts to tackle the problem definition risks first	There is no specific upper limit for the size. A larger project uses longer iterations	Split the project into features, and develop the features in stages. A very long project can be sectioned with time-boxing. Each feature should not take more than 2 weeks. Split any bigger features to smaller ones	ASD does not address especially this problem	XP works only with small-size projects. The project team should not exceed 10 developers by definition, so you cannot handle very big developments with XP alone	Hacking in a bigger project leads to chaos, and bad usage of the available resources (part of the project personnel may not know what they should do)
Unrealistic schedule target	[8/#2, 38/#2, 43/#5]	Waterfall model does not tackle especially this problem	Agree on the first increment(s), (re)negotiate the delivery later. Incremental delivery may help to manage	The schedule risk becomes apparent early. On the other hand, some progress can be shown on a very early phase of the project, which might make the customer more eager to wait for the final product (or redefine the project)	A realistic understanding of the project targets should be developed in the Inception and Elaboration phases. If this fails, the milestones are not passed, and the project should not move to the Construction phase	Adjust the contents, i.e. keep the targets but deliver less features. During the project Planning phase, the feature sets completion dates are estimated (measured in months). That plan is recommended to be reviewed with the stakeholders, possibly revising the project goals	The project initiation phase includes the determination of the project time-box boundary (target date). However, the project team commits to their planned date	XP is optimized towards rapid development. However, one needs to balance the costs (working with expert team, making customer available). The team has its natural velocity. The customer and the project team should agree on the realistic schedule target	Unrealistic schedule target is the other main reason, why hacking is applied to. You have to notice, though that some things (like documentation) is left undone. With excessive overtime, you may even be able to meet the schedule (but with a corresponding high cost of attrition, etc)
Extreme project (high speed, high change)	[22]	Strictly serial waterfall is hardly ever applicable for such situations. More flexibility is required	By splitting the project to incremental sections, there might be some limited flexibility in rearranging them. Some increments could possibly be developed concurrently	There is no particular emphasis on such circumstances, but reducing the major risks early should help avoiding delays caused by later reworking	RUP does not embrace such projects by nature	FDD is not so much intended for extreme cases, but a reasonable amount of changes can be absorbed. Concurrent development of some features may speed up the project	ASD is targeted for extreme projects	XP is targeted for extreme projects	Warning! Hacking is often used with extreme projects. This may lead to burn-out of the key personnel. You may be able to stretch your capabilities, but after a certain limit it simply would not work

Death March project; This is a compound problem: a project whose 'project parameters' exceed the norm by at least 50%. A death march project is one for which an unbiased, objective risk assessment determines that the likelihood of failure is >50%	[52]	Waterfall model does not tackle especially this problem. Rather the opposite: the work-product is visible only in the end of the project. In extreme projects, you need more flexibility	Staged delivery demonstrates some visible progress. Maybe more time/money/confidence can be won in that way	Staged delivery of the first iterations can be used to demonstrate some partial functionality (like some brute-force algorithms), which can be improved with the later cycles	The iterative approach may provide some aids for balancing the edge of chaos	FDD does not tackle especially this problem. Splitting the work into smaller chunks makes the project easier to manage, but does not necessarily ease the effort. There may not be intermediate work products to show, either	This an extreme case of an extreme project. However, there is always some limit for 'stretching'. Basically ASD encourages realistic planning, and not committing to arbitrary targets. Rational extreme projects are not death marches	The customer is in close contact with the project team all the time, so progress is made very visible. This is usually enough to make the customer wait for the product she wants	Warning! Attempting hacking in a death march project is very high-risky. You are likely to end up with a high cost of attrition, etc
Project Execution									
Incomplete requirements/specs (poorly defined parts), lack of user input		Waterfall model does not tackle especially this problem. By model definition you should have stayed on the specification phase, until the requirements and specs were clarified	Can possibly start working on the known requirements, and clarify the rest for the subsequent increments	Risk-driven specification focuses on the uncertain areas	RUP is Use-Case-driven. The use-case model is supposed to make it sure that all the functional requirements are handled by the system. The Vision document provides a high-level view	There is a Domain (Object) Model. The Domain Experts work together with the feature teams, helping to clarify the problem to be solved. Domain Walkthroughs are conducted to clarify any unclear details	Uncertainty and lack of initial understanding are seen natural. The idea is to learn more with iterative development cycles providing frequent feedback. The key is to progress to the right direction	Strong connection with the customer is a prerequisite of XP. The customer should be present on weekly planning sessions, and check that what is planned is consistent to what is expected. If the project objective is not clear to the customer either, it is very unlikely that the project will deliver anything useful at all	It is typical for projects using hacking to skip or run though the requirement phase. The changes cause more hacking
Unstable (volatile) requirements, continuous requirements changes	[38/#3, #5,43/#8]	Waterfall model does not tackle especially this problem. By the model definition, you should have stayed on the specification phase, until the requirements were clarified. Frequent and/or late changes are not welcome	Freeze the requirements only for the current increment, allowing changes to the later increments. Increments provide feedback about the changing needs. However, excessive change-rate can still be a problem	Identify the most volatile (= risky) areas. Changes can be incorporated for next cycles. Allow some adjustment in project timeframe	Basically you should mostly be able to agree on the major requirements (features, use cases) during the first phases of the project. Controlled change management is advocated	A feature can be replaced by another feature, with more advanced functionality and enlarged specifications. (Like replacing navigation system with more precise one). The requirements (features) are recommended to be prioritized somehow systematically. Up to 10% of change is supposed to be absorbable without extra actions	The development cycles are time-boxed, 'forcing' to make trade-off decisions gradually. Unhealthy oscillation could be avoided by focusing on the project mission and the problem definition early. Shorter cycles should be used for areas of high uncertainty	Welcoming changes is the true nature of XP. The project is redefined on weekly basis	You may be able to accommodate a certain amount of changes, provided that the project key personnel is not changing

(*continued on next page*)

Table A1 (*continued*)

Project Problems, Failure Factors	References	Software process models							
		Plan/specification-driven Models		Evolutionary models		Agile methodologies			Ad hoc
		Waterfall (serial development)	Incremental development models	Spiral model (risk-driven iteration)	Rational unified process (RUP)	Feature-driven development (FDD)	Adaptive software development (ASD)	Extreme programming (XP)	No discipline (chaotic 'hacking')
Poor requirements management (uncontrolled requirements changes, requirements creep)	[8/#6, 43/#18, #25]	There should not be many changes at all since the uncertainties are supposed to be resolved at the first stages	Increments allow determining the requirements piecewise	If there is such a risk, the focus should be on taking the requirements baselining under control	The use-case model is the basis for the development. Controlled change management is emphasized (CCB). Requirements management tools are advocated. Unified Change Management has been proposed	Requirements are allowed to be changed, but FDD emphasizes controlled change management. Requirements (features) source traceability is emphasized	Short, time-boxed delivery cycles freeze the requirements piece by piece	This is a part of the Planning Game. However, because of the nature of XP development, there is not much formal change management	Typically there are no formal requirements to be managed
Gold plating (developers adding unnecessary functionality)	[8/#5]	The development follows the accepted specifications. Basically no additions are allowed later	Incremental development does not actually solve this problem, but the consequences become visible earlier	The Spiral model does not directly address such details, but this is an additional source of risk to be reduced	The use-case model sets the boundaries and keeps it focused	The Features List focuses the development	Time-boxed cycles limit	The customer decides the features to be implemented during the Planning Game	This is a natural consequence. It may even work within small limits, but definitely not on larger projects
Constantly changing schedule target		This contradicts with the assumptions of waterfall development, which relies on agreed plans. Continuous replanning is not well accommodated. You can also try to cut your features, but then you are not following pure waterfall model any more	Increments make it possible to release the product step by step, thus allowing some adjustments of the schedule targets within reasonable limits	If the objective is to able to accommodate frequent rescheduling, the Spiral model concentrates on flexible development approaches	The iteration plan defines the start and end dates, and the delivery date. You should not change the current iteration much. However, the next one could be replanned	Features are recommended to be very small (no more than 2 weeks of effort). Feature sets could occasionally be reassigned between the teams (but not too often)	The project is time-boxed. The cycle dates are not changed. If the original schedule turns out to be wrong, it can be renegotiated in cycle replanning	The iterations are time-boxed. The releases are small. New releases can be scheduled, if the customer wishes that	The schedule depends very much on the key persons. They may or may not be able to make it, but it is hard to tell that in advance (poor predictability)
Poor software architecture design quality	[43/#30]	There could be a separate architecture design phase with a milestone review. However, if the architecture later turns out to be deficient, major redesign is difficult to manage	Incremental development requires a solid architecture. Otherwise it may be difficult to incorporate new functionality. Parts of the architecture could possibly be refactored for some increment	Architectural risks can be iterated to some extent	RUP is an architecture-centric process emphasizing, evolutionary, component-based architecture work with visual modeling (UML)	FDD does not tackle especially this problem. It may be very hard to make any corrections to the architecture in the middle of the project, when half of your features are already ready	ASD does not cover architecture design details	Small changes to architecture can be implemented easily. However, XP does not offer much support for system architecture design (just 'metaphors' and 'simple design')	This is definitely a risk, since typically there is no systematic architecture design at all. 'Quick-and-dirty' solutions are typical

Wrong architecture solution selected in the first phase (inadequate systems engineering)	[8/#10, 43/#19]	There could be a separate architecture design phase with a milestone review. However, the architectural solutions must be committed early in the project. If they are based on false assumptions, later redesign may be difficult. The waterfall model assumes that the architecture solution can be understood early	Incremental methods do not tackle especially this problem. However, you can probably see the problem earlier	If there is a risk of making wrong choices, the architecture selection could be emphasized first. You could always throw out some code and start from the beginning. (But will you loose your faith to the project on the same?)	The architecture choices are based on architecturally significant use cases. An (evolutionary) architectural prototype is recommended. Thus, no totally wrong solutions should result (architecture first)	FDD does not tackle especially this problem. It may be very hard to make any corrections to the architecture in the middle of the project, when half of your features are already ready	The problem definition done during the project initiation guides the architecture selections. Iterative development cycles support learning more about the architectural choices	XP does not tackle especially this problem. It might be hard to convince the customer to buy the development cost for the better architecture (when the customer is actually expecting progress in form on some new features). Refactoring could help to some extent, but fundamentally wrong solutions cannot be salved	This is an obvious risk for any longer-term development
Inappropriate design methods	[38/#7, 43/#21]	Waterfall model does not tackle especially this problem	The methods could be changed for some increment. On the other hand, one essence of the incremental development is to test the tool-chain early, so you will loose these benefits	If there is a risk of selecting an inappropriate method, the first risk reduction cycle could concentrate on testing the suitability of the method	RUP advocates certain design methods which are supposed to be generally applicable (such as Use Cases, UML, components)	Rework features to some later release with better tools	ASD does not cover design details	Replan and re-schedule your project. If the customer accepts this, can be done. In general, do not try to use totally new design methods with XP	We can try to change them on the fly
Unsuitable or low-quality tools		Waterfall model does not tackle especially this problem	Test the tools during the early increments. Consider replacing the problem tools for the later increments. On the other hand, one essence of the incremental development is to test the tool-chain early, so you will loose these benefits	If there might be a risk with some new tools, focus on them first	RUP is very much tool-oriented. There is a wide set of commercially available tools	FDD does cover any tool issues	ASD does not cover any tools details	XP does not cover any tool details. But there would not be any sense of buying the best experts on the field and equip them with poor tools. In general, do not try to use totally new tools with XP	We can try to change them on the fly
Integration difficulties	[43/#28, #32]	Waterfall model directs to integrate the whole system on one shot, which often leads to integration difficulties (‘big bang’). So the model rather creates this problem than prevents it	Increments force the integration early, discovering the possible breakage, while there is still time to correct it	There should not be big integration in the end of the project, if the spiral model has been properly followed up—but the product has been integrated and tested along the way	RUP encourages almost continuous test and integration (executable releases for each iteration). Any breakage should thus become visible early. Early architectural risk reduction is emphasized	FDD does not define integration in any exact way. However, the Chief Programmers are responsible for testing their features. FDD used with staged delivery makes the integration steps smaller, and thus easier. A regular build schedule is recommended (supported by solid configuration management)	There is no particular emphasis on integration, but each cycle should end with valid results	You have the whole software team to back-up the integration. However, this requires that everybody knows how to do the integration. Note also that it may be difficult to manage the integration of a large complex system without rigorous up-front planning	Hacking is likely to lead to undocumented code and unspecified interfaces, which make the integration step extremely difficult

(*continued on next page*)

Table A1 (*continued*)

Project Problems, Failure Factors	References	Software process models							
		Plan/specification-driven Models		Evolutionary models		Agile methodologies			Ad hoc
		Waterfall (serial development)	Incremental development models	Spiral model (risk-driven iteration)	Rational unified process (RUP)	Feature-driven development (FDD)	Adaptive software development (ASD)	Extreme programming (XP)	No discipline (chaotic 'hacking')
Low visibility to progress	[43/#22]	One of the typical pitfalls when using the waterfall model. Not any working software is available until at the final stage. The only sign of progress is the documentation (which may be enough in some cases, though)	Regular increments demonstrate the progress. The duration of each increment should not be too long to retain the visibility	Already the first iterations show some view what the final product could be. Risks are shown to be gradually reduced with every spiral cycle. A new cycle should not be started before the objectives, risks, and constraints are understood	Progress is measured in terms of use cases (features) completed, test cases passed, performance requirements satisfied, and risks eliminated. Regular, demonstration-based assessment is emphasized. Iteration Assessments are conducted after each iteration (e.g. revalidating the requirements)	FDD provides good visibility to progress, because delivery of each feature can be monitored. The progress reporting is recommend to be done based on feature completeness. If the project takes longer than some 3 months, formal monthly progress reviews are recommended	ASD does not improve the traditional project visibility since it relies on intense collaboration (tacit knowledge). The documents evolve during the whole development. Only the results matter	This should not be a problem at all with XP. The customer sees the progress weekly	You may be able to show some progress by demonstrating the software. However, typically the quality tends to be unpredictable. The progress is often variable due to unplanned design
Vague milestones		The basic premise of the waterfall model is strict milestone gating. Waterfall model with vague milestone definitions changes easily to a unmanaged project, living its own life without control	Increments are major milestones. For each increment, there should be a clearly defined purpose	Each cycle completion is a clear milestone (provided that the cycle objectives have been planned clearly). Three project 'anchor point' milestones have been developed (like RUP): Life Cycle Objectives, Life Cycle Architecture, Initial Operating Capability	The phases are defined with given major milestones (generically defined). The minor milestones depend on the iterations. Each iteration should have a clear objective. Change the plans if the phase milestones are not passed	For each feature, there are six sharp milestones defined: Domain Walk-through, Design, Design Inspection, Code, Code Inspection, and Promote to Build. The whole sequence should not take more than some 2 weeks	Each short cycle (6–10 weeks for a long project) has a definite end-result. A milestone is reached when the artifacts are determined to be in the planned state	The progress is determined by the stories (features) completed. Weekly meetings with the customer to verify them serve as milestones. However, you must be able to agree on what exactly it means to complete a story (without detailed documentation)	Typically there are no predefined milestones at all
Communication gaps (project internal)	[38/#9, 43/#9]	The serial development relies much on passing the documentation between the phases. This may not be enough for carrying all the necessary information (tacit knowledge)	Incremental development does not really solve this problem. However, more feedback information become available with early releases	This problem is not specifically addressed	RUP emphasizes tool-based artifacts for sharing the information	The Domain Experts work together with the feature teams. This should improve the communication	ASD emphasizes rich and intense collaboration, even with virtual teams. However, this requires considerable attention. Customer focus-groups and software inspections are specific techniques for learning	XP is based on open and frequent communication. Inside the team, the communication gaps are fatal. From team to other parties (where the team might have loose connection) these could do serious damage (as the documentation is often plan one and throw away-of type)	With little formal documentation, the communication relies on the tacit knowledge shared face-to-face

Excessive documentation (overhead)		Excessive documentation is a classic problem when using the waterfall model. The problem could grow even bigger if rework makes intermediate documents obsolete	This may be a problem with incremental development. Planning and managing the increments require some additional documentation. However, some documentation rework could be avoided if the increments match well	Risk-driven documentation: Concentrate on those parts in which incomplete documentation is risky. Complete specifications are not to be insisted prematurely	RUP prefers tool-based models to paper documents	FDD is not so much document-driven. It leaves the documentation details open to be decided by the project manager according to the current needs. Intranet-based hyperlinked documentation tools are recommended. Good user documentation is emphasized	ASD is not document-driven. Instead it relies on tacit knowledge and intense collaboration	XP emphasizes working software over documentation	Often there is no documentation whatsoever—i.e. there is certainly no risk to end up with too much documentation
Project external dependencies (including subcontracting) late and/or imperfect (e.g. system specs)	[8/#7, #8, 38/#4, 43/#16, #31]	Waterfall model does not directly address such issues, but for example if some input spec is late, the specification phase cannot be concluded	The dependency risks can be addressed by different increments	If there is such a risk, the Spiral model considers possible alternatives	There is no particular support for this, but you should monitor those risks from the beginning, and plan the iterations accordingly. RUP does not cover Systems Engineering	This is not really addressed by FDD, but such dependencies could be taken into account while planning the feature development order	The project vision document identifies the dependencies. The dependencies are revalidated in each cycle review	You end up with the team waiting. The customer must be involved	Such risks are usually not controlled. Perhaps some ad hoc workarounds are possible
Geographically dispersed teams		Waterfall model does not cover such issues	Incremental development does not really tackle this problem	This problem is not specifically addressed	A tool-based process implementation may help in lessening the problems. However, in general this complicates the Construction phase	FDD does not address this issue	ASD considers virtual teams as a natural mode of operation	XP relies on a co-located team	This may be a big problem with little external documentation. Depends on the key persons
Loss of (key) staff (either because they leave or get transferred)	[8/#1, 38/#8, 43/#23]	Waterfall model does not cover staffing. However, comprehensive documentation helps accomodating staff changes	The increments help limiting the consequences. May be able to adjust the later increments (if the consecutive increments are independent)	Such issues are not directly addressed. If there is a risk of loosing some key staff, possible alternatives should be considered. Staff changes may be a problem unless the previous spiral cycles are well documented	The Iteration Plan must be adjusted accordingly for the next iterations	It may be difficult to replace some class owners quickly. Some feature teams may have to be replanned	Each cycle review reassesses the resourcing situation against the targets	Replanning when the team changes (velocity). Sudden loss of key persons may be a serious problem, since the source code is the main tangible piece of information	Such risks are not managed. Usually no continuation if the key persons leave
Low morale, motivation		Waterfall model does not cover this. Working on the documentation long before seeing any working software may be demotivating	Regularly released working increments typically boosts the morale	Focusing on the risks may help convincing that the project proceeds in a sensible way	The iterative approach lets the developers see working software earlier. This may help keeping the spirit	The feature-based tracking may help. So-called Feature Kills sessions may be uplifting. Public, colored feature tracking charts are advocated	Building 'great groups' is one of the cornerstones of ASD. Given the right environment, people motivate themselves	XP pays special attention to developer morale and motivation. A sustainable 40-hour week is emphasized as a norm. This may help people keeping the spirit high. Pair programming may be enjoyable	Some individuals may like the apparent freedom of totally unconstrained working

(*continued on next page*)

Table A1 (*continued*)

Project Problems, Failure Factors	References	Software process models							
		Plan/specification-driven Models		Evolutionary models		Agile methodologies			Ad hoc
		Waterfall (serial development)	Incremental development models	Spiral model (risk-driven iteration)	Rational unified process (RUP)	Feature-driven development (FDD)	Adaptive software development (ASD)	Extreme programming (XP)	No discipline (chaotic 'hacking')
'Crunch' mode (tight schedule, just achievable with extra-ordinary measures)		Waterfall assumes a stable environment. No 'crunch' mode of operation is really expected	With careful planning of the increments, you could be able to deliver at least some partial functionality on time. That is often better than delivering the full functionality but late	This problem is not specifically addressed. In general, the Spiral model attempts to avoid such extremes by resolving the related risks early	This is not in line with the philosophy of RUP. With sensible iteration plans, no such thing should happen	Basically this is not in line with the FDD philosophy. With orderly planning and monitoring of the features, there should not be any need to operate in such a mode	ASD is designed for high speed, high change circumstances	XP emphasizes steady, good (∼high) output level. The productivity of the team (velocity) is a key planning parameter. The customer and the project team agree on what is reasonable	With no defined process, you are basically free to do whatever it takes. But there is always a limit
Project redirected (profound changes of the schedule/ functionality/ resources)	[43/#34]	The pure waterfall model cannot adapt well to major mid-course changes. The lifecycle must usually be restarted	You may have to renegotiate the remaining increments, but the already delivered ones are anyway available	The next cycle of the spiral restarts the planning	Continuous refinement of the plans is underlined	There are three ways to balance this: (a) lower-priority features are cancelled; (b) the project schedule is extended; (c) new feature teams (people) are added to work concurrently; If the overall project plan is changed drastically, a new project initiation should be considered, however	Basically even major changes can be accommodated in the cycle reviews	The customer can present new specifications (new user stories) on the weekly meetings	This is really a part of the approach. It may even work within some limits, but eventually you may end up into a havoc
Project cancelled		The project cannot show any results (except documentation) since no working software is available before the integration stage	If some increments have been delivered, the project managed to release something tangible	The Spiral model incorporates such a possibility. For each cycle there is a hypothesis. If it fails, the spiral is terminated. The idea is to resolve the major risks early, so the probability of a completely surprising cancellation should become lower while the spiral proceeds	After the Inception and Elaboration phases, there is supposed to be a clear understanding about the feasibility of the project (for GO/NO-GO decision). Later, in case of a mid-project cancellation, you may be able to deliver some of the interim releases produced so far	This beyond the scope of FDD. However, the features completed so far could be somehow useful	You may agree on completing the current cycle so that the termination status is clear. Since you have completed the earlier cycles, the project succeeded in producing some results anyway	The customer can cancel the project any time on her will. What has achieved to that point can be taken into use	This is a considerable risk, if already the project setup was ad hoc. Typically the project cannot deliver anything usable
Project completion									

Trouble validating the system (acceptance test)	[43/#37]	Waterfall model does not provide an answer to this problem. The system tests are done according to the predefined plans and specs. If they are wrong, this is a problem	Incremental delivery allows early feedback	When iterations are used, the system is often tested on the way. Thus it is very unlikely, that this problem would exist in a project using this method	The use-case model defines the expected functionality. Continuous integration, prototyping, and demonstrations are encouraged	The systems consists of features. Validate the features separately	A healthy project converges. By the time of the last cycles, no major surprises should not happen	If you implement XP properly, you should make automatic test cases that are repeated continuously. This leads to overall better quality, and thus the end-product should be more easier to integrate. The customer defines and runs the acceptance tests	Typical the acceptance criteria is ad hoc. The outcome may be totally different from the original idea. In addition, hacking may leave to undocumented code which is hard to maintain and modify. This may mean problems when the code should be modified to pass the acceptance test
Unstable or poorly performing software release	[8/#9, 38/#6, 43/#30, #33]	The integration stage does not end until the software is fully tested	Each increment should be a stable subrelease	If there is such a risk, the iterations should plan measures for reducing potential quality problems (e.g. early performance analysis). Note that a long sequence of iterative refinements may lead to an undefinitely performing system (like prototypism)	The software is incrementally integrated for each iteration. Thus any breakage should be detected early	FDD advocates design and code inspections, and some kind of unit testing for quality assurance	The technical quality is maintained during the development, in part, with software inspections	This should not happen in XP, since it advocates for making (even small) pieces of working software from the beginning	This is a serious risk
Unattractive software release (wrong, obsolete or missing features)	[8/#3,#4, 43/#27, #40]	The release is built according to the initial requirements phase. If that phase was conducted poorly, the resulting release is likely to be unattractive	The customers can see the growth of the software with every increment. There should not be any big disappointments in the end	If there is such a risk, the iterations should include some dedicated activities for reducing the uncertainty (e.g. prototyping)	The features are agreed with the customers (and other stakeholder) with the Vision document (business case)	Feature list planning ensures focusing on the right features. Ideally, the list is accepted by the customers (stakeholders) prior to the construction. Regular pre-releases of features demonstrate the progress	Customer Focus-Group (CFG) reviews help getting timely feedback about the product features	This is tackled with the Planning Game. The customer selects the features to be implemented	Unpredictable
How to make a good starting point for the next project (e.g. updating the documentation)?	[15, 43/#38, #39]	By definition, this is the last stage of the waterfall	This should be a part of the last increment, or there might be an extra finishing increment	Each iteration cycle completes with a well-defined evaluation. This should make a clear starting point for the subsequent project cycles	RUP embraces tool-based engineering artifacts. Subsequent project cycles may be partially overlapping	The Domain (Object) Model is a useful asset for extending the product. The feature tracking charts provide high-level information about the completed functionality. The user documentation could be required as a part of each feature completion	ASD encourages 'finishing strong', leaving a good trail	A working software release is always a good starting point, but not necessarily enough. XP relies much on tacit knowledge. This may be a serious problem, in particular if the project team changes	Hacking leads to bad maintainability and poor documentation

(*continued on next page*)

Table A1 (*continued*)

Project Problems, Failure Factors	Referen-ces	Software process models							
		Plan/specification-driven Models		Evolutionary models		Agile methodologies			Ad hoc
		Waterfall (serial development)	Incremental development models	Spiral model (risk-driven iteration)	Rational unified process (RUP)	Feature-driven development (FDD)	Adaptive software development (ASD)	Extreme programming (XP)	No discipline (chaotic 'hacking')
Unclear project end-criteria		The end-criteria definition is a part of the planning phase	The end-criteria should be planned as a part of the increments planning	This can be a potential problem with iterations: there is always room to improve. When do we say it is final? The cost increases with every new spiral cycle	The use-case model serves as a 'con-contract' between the customers and developers. There is a Project Acceptance Review. The end criteria should be defined during the Inception	Basically the project ends, when all the planned features have been built according to the Features List. The list should have been accepted by the project stakeholders somehow (not covered by FDD)	According to the ASD philosophy it is normal that the actual end state is different from the initial plan. The project time-box sets the schedule boundary	The project is ended when the paying customer is happy with the end-product or cancels the development. The customer opinion is checked weekly	Undetermined
References:		[33(Ch. 7.1), 41(Ch. 1)]	[33(Ch. 7), 34, 41]	[7]	[27]	[37]	[22]	[6]	[33(Ch. 7.2)]
Home ground:		Low speed, low change [22]. Primary objective: high assurance, predictability. It works well on stable parts in which you can commit to the requirements and resolve the uncertainties early. Works well on complex projects by adhering rigid controls and ordering	Basically any project that has some advantage in building and delivering (externally or internally) the software gradually in slices rather than completely at the end	Low speed, high change [22]. Evolutionary, iterative development is a natural approach with volatile parts requiring exploration (e.g. complex user interfaces). Suits well for very large, complex, and ambitious projects (research-oriented)	RUP is a generic process framework intended to be tailored for different project types (development case). However, not being a light-weight methodology per se, it is more suitable for larger, complex projects 'out of the box'	Applicable to a wide range of general-purpose business systems. Can be applied to 'greenfield' development as well as new feature development for an existing product. The project size can be much more than 10 people	High speed, high change ('extreme' projects)	Primary objective: rapid value. Typically suitable for small projects with a familiar application area and low risks. XP is suited for projects in the C4 to E14 categories [15]. Not recommended for very large, complex application systems as such	No place in large-scale professional software development! Some small off-line demos or feasibility studies might just be acceptable
Consequences, Side-effects, Drawbacks:									
Scope		Generic software development starting from the system requirements spanning to the operation and maintenance	Generic software specification and construction. Incremental models: - incremental development - incremental delivery (internal/external), e.g. Staged Delivery	The Spiral model is actually a meta-model, basically encompassing all process models. For example, if the schedule predictability is a high risk, the model unwinds to the waterfall	RUP covers software project work widely starting from the project initiation ranging to the product deployment. Also many support activities are addressed (like SCM)	FDD addresses only the software construction process. Initial user requirements elicitation and system tests are beyond the scope	ASD is primarily a management approach. It does not offer much support of how to implement the software engineering tasks in practice	XP actually focuses on the software construction. The basic project management activities (like planning, change management, tracking) are incorporated, although mostly informally	This is not really a process model at all

Nature	Waterfall model is workflow-oriented [22]	The basic idea is to split a large project into smaller sections. Working aggressively, the increments could possibly be developed overlapped in parallel. The basic form of incremental development is to specify all requirements first, followed by a sequence of builds. A possible variant is to make the specifications incrementally, too	In general, advancing in the spiral reduces the risks, and increases the project cumulative cost. The basic form of evolutionary development is to let the requirements evolve with iterations	RUP is tool- and work product intensive [15]	FDD emphasizes client-valued functionality (features)	ASD is primarily work state-oriented	XP is activity intensive. XP suggests maximizing concurrency [15]	No preset rules
Advantages	Properly implemented waterfall could be the fastest way to run a project under right circumstances. It is easy to manage serial development. The serial development model is easy to learn and follow, even with inexperienced people	The increments can be delivered to the customers for early feedback. Changes can be accommodated by adjusting the increments	Focusing on the risks and considering the alternatives systematically makes the project management more robust and resilient to uncertainties	RUP is a comprehensive process framework with tool support available. It provides detailed definitions for the project milestones, artifacts, activities, and roles	Focusing on the features systematically provides a coherent view of the project	Admitting that different project situations require different solutions makes the project management inherently adaptable	The lightweight way of working can be very efficient, provided that the project home ground is right	This is very flexible in the sense that there are basically no preset rules to be followed. There is no management or documentation overhead
Constraints	Major midcourse changes are basically not favored. Waterfall is only recommended for stable project environments	Incremental development requires more management activities. (The integrity of the project must be checked consistently.) Requires more testing, because all features must be re-tested for each increment. SCM is more complicated, especially if increments are developed in separate branches that will be merged later on. Managing parallel increments is more complicated. Not every application can be delivered in increments [4]	With iterations one should be ready to throw away some versions of the working software. In a sense, this means compromise with the schedules. A significant disadvantage of iterative development is that it is often difficult to define deliverables [4]	The 'out of the box' version of RUP is intended to be an organization-wide process. The project-specific processes may need adaptations	The features must be known, and prioritized. Once the features have been selected, it is very hard to change the contents without causing serious damage to the project. FDD assumes a working configuration management system for shared access. SCM can be more complicated, if stages overlap and/or if features are selected for each release from a large base	ASD relies much on intense communication and iterative learning. How to make this work in practice may not be that easy, though. ASD recommends having a customer available for conversation each day [23]	Collective code ownership may not scale up. By XP definition the project team should be on one site. Requires an active onsite customer, who is willing to follow the rules of the process model. It may not be reasonable in practice to make a new customer release of a large system every week or so often	It may be difficult for new people to join the project (catching up), since the process is not defined anywhere. The visibility is low (no intermediate products or milestones defined)

(*continued on next page*)

Table A1 (*continued*)

Project Problems, Failure Factors	References	Software process models							
		Plan/specification-driven Models		Evolutionary models		Agile methodologies			Ad hoc
		Waterfall (serial development)	Incremental development models	Spiral model (risk-driven iteration)	Rational unified process (RUP)	Feature-driven development (FDD)	Adaptive software development (ASD)	Extreme programming (XP)	No discipline (chaotic 'hacking')
Cautions!		If there are major uncertainties at the outset, the waterfall model is often not suitable. Because the model relies on completely defined plans and specifications, later changes may cause heavy rework	The increments must be chosen wisely, so that each increment builds to the previous one. Otherwise the project may end just coding parts of the big functionality several times, because earlier implementations are not parts of the latter one. If the deployment of the system is complicated, it may not make sense to deliver new increments often	This abstract model requires careful planning. It may be difficult to apply it in practice	The commercial version of the process model relies on certain tools. It may become more difficult to use the process without those particular tools	In a large complex system, it may be difficult to find a suitable development order of the features, and organizing the feature teams, if there are many interdependencies. If you only concentrate on the business features, there is a risk to neglect internal technical features	Too much flexibility can be dangerous, too	The apparent light weight of XP means that you have to define many practices and rules on your own. If you cannot find a customer who wants to work that way you should not try XP at all [6]. XP assumes a certain amount of tacit knowledge and skill [23]	The project (or the company) becomes very dependent on the key programmers, in particular if there is not much written documentation. The project may easily slide into an unrecoverable chaos
Notes		There are MODIFIED WATERFALLS: Overlapping phases, parallel subprojects, risk reduction iterations	Some form of incremental development is a key characteristic of most (if not all) agile methods. A question to ponder (for the management): What is the distinction between an increment and a separate release?	Requires good software risk management experience	RUP is more like a heavyweight methodology. Some lighter adaptations have been proposed for smaller projects	Staged delivery causes partially same problems as incremental development (overhead in testing and content management). FDD assumes that the overall value of the features is determined early in the project and that scheduling those features should be primarily a technical decision [23]	There is a philosophy of complex adaptive systems behind	A user story = a feature	You should not really consider this model as an alternative. Hacking is a process antipattern, sometimes mistakenly justified by iterative development [4]
EMBEDDED SYSTEMS	[39]	May be suitable in case you can agree on the hardware/software specifications early. Typically, there is a common synchronization milestone with the software and hardware developments	The software increments can be synchronized with concurrent hardware development (e.g. prototype boards)	Sometimes the hardware development is best done with the sequential waterfall model, while the software development may apply the spiral model	There are some real-time software design specialities	May be suitable. Does not address embedded systems specifically. Planning the feature list with concurrent hardware development may be challenging	May be suitable. Does not address embedded systems specifically	May be suitable, especially if the hardware is already available. Does not address embedded systems specifically	Some software experiments with the target hardware may make sense

Note: The column References shows the problem item numbers used in the respective publications, e.g. [43/#1] refers to the first item of the list in [43].

software construction but in the related systems and hardware engineering issues.

This paper leaves room for further study:

(1) Empirical validation: At the time of the writing we are not yet able to present current empirical validation data about our propositions. Such data could be collected by experimenting with the matrix (Appendix A) in ongoing software projects. How useful is the matrix? However, even now it reflects some of our practical project experiences and learnings, like illustrated in Ch. 4.1. Also other authors have recognized the need for such industrial empirical evidence [30].
(2) An improvement could be to add different sort keys of the problem factors in the matrix (Appendix A). In different situations different views might be useful. For example Ambler has compared some development approaches with respect to their ability to support certain overall project requirements (e.g. 'critical features must be put into production as soon as possible') [4(Ch. 1)]. Does the project need to prioritize for example predictability, flexibility, or visibility? What are the prerequisites? In addition, certain color codes could be used in the matrix to highlight, how well each process model tackles each problem. Such a colored cell map could provide a quick overview about the whole matrix.
(3) The matrix (Appendix A) could be extended with other comparisons, for example such as suggested by Table 4 (c.f. Table 1). The different practices could be selected in particular among the generally advocated agile practices [50]. The practices could be grouped for example on collaboration, project management, and software development practices [23(Ch. 25)].
(4) Changing the comparison focus from large-scale embedded systems to some other, e.g. multisite or web site development project.

Acknowledgements

The authors would like to thank Tuomo Kähkönen (Nokia Corporation) for reviewing an earlier version of this paper.

Appendix A. Software process selection matrix

Table A1.

References

[1] I. Aaen, Software process improvement: blueprints versus recipes, IEEE Software 20 (5) (2003) 86–93.
[2] P. Abrahamsson, et al., Agile Software Development Methods: Review and Analysis, Technical Research Centre of Finland, VTT Publications 478, Finland, 2002.
[3] P. Abrahamsson, et al., New Directions on Agile Methods: A Comparative Analysis, Proceedings of the 25th International Conference on Software Engineering, 2003 pp. 244–254.
[4] S. Ambler, Process Patterns—Building Large-Scale Systems Using Object Technology, Cambridge University Press, Cambridge, 1998.
[5] P.G. Bassett, Framing Reuse: Lessons from the Real World, Prentice-Hall, Upper Saddle River, NJ, 1997.
[6] K. Beck, M. Fowler, Planning Extreme Programming, Addison-Wesley/Pearson, Upper Saddle River, NJ, 2001.
[7] B. Boehm, A spiral model of software development and enhancement, IEEE Computer 21 (5) (1988) 61–72.
[8] B. Boehm, Software risk management: principles and practices, IEEE Software 8 (1) (1991) 32–41.
[9] B. Boehm, Get ready for agile methods, with care, IEEE Computer 35 (1) (2002) 64–69.
[10] B. Boehm, R. Turner, Balancing Agility and Discipline—A Guide for the Perplexed, Addison-Wesley/Pearson Education, Boston, MA, 2004.
[11] F.P. Brooks, No silver bullet: essence and accidents of software engineering, IEEE Computer 20 (4) (1987) 10–19.
[12] F.P. Brooks, The Mythical Man-Month: Essays on Software Engineering (20th Anniversary Edition), Addison-Wesley, Reading, MA, 1995.
[13] W.J. Brown, et al., AntiPatterns in Project Management, Wiley, New York, 2000.
[14] A. Cockburn, Selecting a project's methodology, IEEE Software 17 (4) (2000) 64–71.
[15] A. Cockburn, Agile Software Development, Addison-Wesley/Pearson, Boston, MA, 2002.
[16] B. Curtis, et al., A field study of the software design process for large systems, CACM 31 (11) (1988) 1268–1287.
[17] R.E. Fairley, M.J. Willshire, Why the Vasa Sank: 10 problems and some antidotes for software projects, IEEE Software 20 (2) (2003) 18–25.
[18] R.L. Glass, Software Runaways, Prentice-Hall, Upper Saddle River, NJ, 1998.
[19] R.L. Glass, Matching methodology to problem domain, CACM 47 (5) (2004) 19–21.
[20] H. Glazer, Dispelling the process myth: having a process does not mean sacrificing agility or creativity, CrossTalk 14 (11) (2001) 27–30.
[21] M. Gnatz, et al., The living software development process, SQP 5 (3) (2003) 4–16.
[22] J.A. Highsmith, Adaptive Software Development—A Collaborative Approach to Managing Complex Systems, Dorset House Publishing, New York, NY, 2000.
[23] J.A. Highsmith, Agile Software Development Ecosystems, Addison-Wesley/Pearson Education, Boston, MA, 2002.
[24] C. Jones, Patterns of Software System Failure and Success, International Thompson Computer Press, Boston, MA, 1996.
[25] F. Keenan, Agile Process Tailoring and Problem Analysis (APTLY), Proceedings of the 26th International Conference on Software Engineering, 2004 pp. 45–47.
[26] P. Kettunen, Managing embedded software project team knowledge, IEE Proceedings—Software 150 (6) (2003) 359–366.
[27] P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley, Reading, MA, 2000.
[28] T. Kähkönen, P. Abrahamsson, Achieving CMMI Level 2 with Enhanced Extreme Programming Approach, Proceedings of the Fifth International Conference of Product Focused Software Process Improvement (PROFES), 2004 pp. 378–392.
[29] C. Larman, Agile and Iterative Development—A Manager's Guide, Addison-Wesley/Pearson, Boston, MA, 2004.

[30] P. Manhart, K. Schneider, Breaking the Ice for Agile Development of Embedded Software: An Industry Experience Report, Proceedings of the 26th International Conference on Software Engineering (ICSE) 2004; 378–386.

[31] L.V. Manzoni, R.T. Price, Identifying Extensions Required by RUP (Rational Unified Process) to Comply with CMM (Capability Maturity Model) Levels 2 and 3, IEEE Transactions on Software Engineering 29 (2) (2003) 181–192.

[32] G. May, M. Ould, Software project casualty, IEE Engineering Management Journal 12 (2) (2002) 83–90.

[33] S. McConnell, Rapid Development: Taming Wild Software Schedules, Microsoft Press, Redmond, WA, 1996.

[34] S. McConnell, Software Project Survival Guide, Microsoft Press, Redmond, WA, 1998.

[35] W. Mellis, Software quality management in turbulent times—are there alternatives to process oriented software quality management?, Software Quality Journal 7 (3-4) (1998) 277–295.

[36] M.A. Ould, Managing Software Quality and Business Risk, Wiley, Chichester, 1999.

[37] S.R. Palmer, J.M. Felsing, A Practical Guide to Feature-Driven Development, Prentice-Hall, Upper Saddle River, NJ, 2002.

[38] D. Reifer, Ten deadly risks in internet and intranet software development, IEEE Software 19 (2) (2002) 12–14.

[39] J. Ronkainen, P. Abrahamsson, Software development under stringent hardware constraints: Do agile methods have a chance?, Proceedings of the Fourth International Conference on Extreme Programming and Agile Processes in Software Engineering 2003; 73–79.

[40] J. Ropponen, K. Lyytinen, Components of software development risk: how to address them? A project manager survey, IEEE Transactions on Software Engineering 26 (2) (2000) 98–111.

[41] W. Royce, Software Project Management, Addison-Wesley/Pearson, Upper Saddle River, NJ, 1998.

[42] J.M. Smith, Troubled IT Projects—Prevention and Turnaround, IEE, London, UK, 2001.

[43] J. Smith, The 40 root causes of troubled IT projects, IEE Engineering Management Journal 12 (5) (2002) 238–242.

[44] J. Smith, A Comparison of the IBM Rational Unified Process and eXtreme Programming. White paper, IBM/Rational (http://www-306.ibm.com/software/rational/info/literature/lifecycle.jsp#White papers, January 2004)

[45] I. Sommerville, Software process models, ACM Computing Surveys 28 (1) (1996) 269–271.

[46] X. Song, L.J. Osterweil, Toward objective, systematic design method comparisons, IEEE Software 9 (3) (1992) 43–53.

[47] R. Sorensen, A comparison of software development methodologies, CrossTalk 8 (1) (1995).

[48] J. Taramaa, et al., Product-based software process improvement for embedded systems, Proceedings of the 24th Euromicro Conference 2 (1998) 905–912.

[49] R. Turner, B. Boehm, People factors in software management: lessons from comparing agile and plan-driven methods, CrossTalk 16 (12) (2003) 4–8.

[50] J. Vanhanen, J. Jartti, T. Kähkönen, Practical experiences of agility in the telecom industry, Proceedings of the Fourth International Conference on Extreme Programming and Agile Processes in Software Engineering 2003; 279–287.

[51] S.F. White, et al., Engineering computer-based systems: meeting the challenge, IEEE Computer 34 (11) (2001) 39–43.

[52] E. Yourdon, Death March—The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects, Prentice-Hall, Upper Saddle River, NJ, 1999.

[53] IEEE/EIA 12207.0-1996 IEEE/EIA Standard Industry Implementation of International Standard ISO/IEC 12207: 1995 (ISO/IEC 12207) Standard for Information Technology Software Life Cycle Processes, 1998

[54] http://www.extremeprogramming.org/, March 2004.