Towards a Process Model for Evolutionary Software Development in Reasearch Environment: Failure of Known Software Development Methods

Alexander Gran
alexander.gran@ipt.fraunhofer.de
Lothar Glasmacher
lothar.glasmacher@ipt.fraunhofer.de
Fritz Klocke
fritz.klocke@ipt.fraunhofer.de

Fraunhofer Institute for Production Technology IPT
January 2008

Abstract

Developing software in the research environment is not like developing ordinary business software. It has several uncommon requirements that render all known software development process models inapplicable. We will present the requirements in the research environment and work out the shortcomings of all commonly used process models.

1 Introduction

When developing software in the research environment, the software developer is providing a service to other research scientists. It may however aditionally be possible that the developed software is to be sold to a third party, being a research partner or industry. You will find this combination at scientific research institutions, like the German Max-Planck Gesellschaft or the Fraunhofer-Gesellschaft but also in research facilities of the industry. In most cases these are located close to universities.

The developed software is often used to simulate parts of the reality or to optimize processes that are already in place but not completely understood. It is however inevitable that the details that should be simulated are not yet known. The reason why such software is developed actually is to understand and evolve these details. When optimizing processes the situation is quite similar: You may have a rough goal (improve processing time of process XY), but its the purpose

of research to enlighten and develop the way to this goal. The results of the software help to understand the environment and as such make simulation and optimization possible in the first place Due to this, the results of the software - which are already produced during its creation of course - allow the more precise and extensive specification of the software. This vicious circle has to be supported by the software development process (SDP).

The lifecycle of the software is to be planned uncommon, too. As there is no concrete final target for the development that could be reached (the scientists are just trying to understand something), there is no point that marks the end of the lifecycle. The research scientists permanently improve and change, maybe even overthrow, their model of the reality. As such the software needs to be improved, changed and perhaps overthrown permanently. It will never reach something like a maintenance phase.

However it may be required, especially for optimization software, that a release is sold to a partner or customer. This may need a specific amount of planning and the ability to calculate a price for additional features.

The software developers in research institutes differ quite heavily from full time professionals in the software industry. As research institutes are often connected to universities, the employees may be students too, working as student trainees in industry or student assistants in the university. These are employed on part time basis and are not permanently available. They may have severe fluctuations in the available weekly working time depending on exams etc. Their level of knowledge changes a lot more rapidly and unpredictable than with normal employees that take further training every now and then. Furthermore their experience in developing software is very limited, and they will stay only for a few years on the project, whereas the softwares life may be much longer. Having a university at hand may create the possibility to get larger parts of the software being constructed by a student as master or diploma thesis. The software development process needs to support the integration of such large and concluded blocks.

The depicted research has been funded by the German Research Foundation DFG as part of the Cluster of Excellence "Integrative Production Technology for High-Wage Countries".

2 Requirements for a process model for research environment

This is a summary of the requirements for a process model to adequately support software development in research environment:

First of all the extremely fluctuating requirements and evolving specifications must be dealt with. Changing of basically *all* specifications within a short timeframe must have no horrible effects on software quality. The SDP has to have a lifecycle that does not end, but allows permanent improvements of the software. Furthermore it has to support plaiting larger modules developed independently,

and allow defining a release to be emitted to a third party. The planning and costing of this release must be possible. It may consider the creation of a spin-off to further develop the software when it hits a larger market. The structure of the employees, i.e. mostly part time staff from university has to be taken into account. Moreover the structures of a large research institution or university may be quite resistant to change.

3 Shortcomings of Known Process Models

Not only software as such, but also the software development processes are permanently being improved. Due to changes in the limitations in hardware, users and developers new software development processes are necessary all the time. Figure 1, according to [6], has an overview of the software development processes being dealt within this paper. Explaining the details of all process is not within the scope of this paper. Refer to [1] and [3] to get an nice overview.

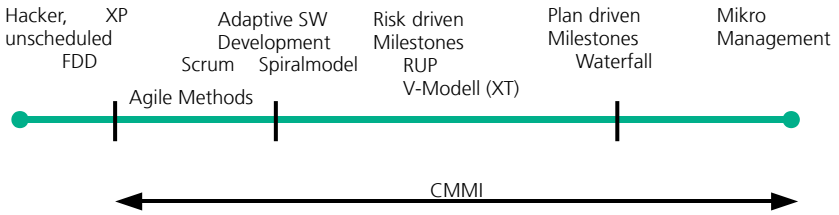

Figure 1: Overview of the most common software development processes

Unscheduled software development has a special position among the software development processes. A process model basically doesn't exist, no phases are being defined and development is just started without initial planning. This seems to be out of all reason at first, however there is still a considerably large amount of software being developed without a SDP. As there is no planning, scientific usable papers are rare about the success rate. However we surely can assume that this approach can be successful for small projects, particularly if only one developer is involved. If multiple developers are to work for a longer period, planning cannot be neglected. Obviously unscheduled development cannot be used in research environment, especially because the software developers are quite inexperienced. As we will see, even the more complex process models are insufficient, too.

Software development process can roughly be subdivided into two major groups: Agile and non agile approaches. Common non agile ones are Waterfall- and Spiralmodel as lightweight processes and V-Modell XT®(V-Modell is a registered trademark of the Federal Republic of Germany) and Rational Unified Process as heavyweight, toolsupported processes. Most prominent agile methods are eXtreme Programming, Feature Driven Development, User Centered Design, Scrum and Crystal.

The basic assumption of agile approaches of software development - change happens and change happens often - is already way better suited than rigid classical approaches. Agile methods assume that some 60 percent of the requirements change in the lifetime, whereas the old ones assume 10 percent [24]. This is still not enough for the research environment, as largely all requirements may change multiple times.

The ancient *Waterfall Model* was developed 1970 by Royce [20] to get away from the chaotic and unscheduled development that dominated before. It's definition of phases in the development process and the strict transition rules were immediately after its development regarded as insufficient. It is mainly of theoretical interest and seldom used in real world projects. Royce used it to show that a more dynamic approach is needed. As such it just does not fulfill any of the requirements of a SDP in research environment. It cannot really deal with larger changes, as it's only possible to revert by one phase at once. The lifecycle ends in a maintenance phase instead of supporting permanent improvements. External modules are unsupported, at least it is theoretical possible to plan releases using a waterfall approach. Reality however shows that this planning doesn't really work out. Employees aren't considered at all in the waterfall model. Due to its simplicity there will be no problems adapting it.

The *Spiral Model* by Boehm [4] is a more iterative, generic process model. It's two-dimensional approach supports planning of cost and progress. The iterative approach tries to reduce risk due to changes in requirements and uncertainties. However radical changes in the requirements are not expected, and the life cycle ends in a final software. External modules could be plait after each iteration, however planning realeases is not supported: On the one hand the iteration process is to rigid for this to work, on the other hand the requirements for the results of an iteration are to vague.

The *Rational Unified Process* RUP is a heavyweight SDP and a product developed and distributed by Rational (by now owned by IBM Corporation). The basic principle, the Unified Process, is also being used by the *Open Unified Process* OpenUP, Agile Unified Process, Enterprise Unified Process and Essential Unified Process. Inspecting these is out of the scope of this paper. RUP is extremely generic and needs a detailed tailoring to the specific project. This is ideally done by a RUP Process Expert, who additionally trains the developers. This is problematic in research environment, as developers may change frequently and the expert has to be paid, as ordinary developers tend to have not enough experience for his job. Furthermore the training of the developers is needed even if they are experienced with other techniques [15], only with enough time and training good results are to be expected [17]. Both experienced developers and much time is not expected with e.g. part time students. RUP requires quite a bit of documentation, which results in an unnecessary overhead if things change a lot. The permanent prototyping allows to give a release to a customer, cost planning of a release is possible due to RUPs tool support. The lifecycle is problematic as for each new set of requirements a new iteration is necessary, the iterations within the phases allow only smaller changes. The iterations are however quite long, so this may result in waste of time.

The *V-Modell* is a - at least in Germany - very popular SDP, developed by the Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (Federal Government Co-ordination and Advisory Agency for IT in the Federal Administration) based upon an idea of Barry Boehm [5]. Its XT (eXtreme Tailoring) version is an approach to become more agile, and has the product and not the process in the center of interest. The V-Modell defines work packages to be done, but does not force the user to specify their chronology. The tailoring furthermore allows to skip several parts not necessary for smaller projects. As the V-Modell distinguishes between principal and agent projects, the activities of one product are split between the two parties on two projects. This is just unnecessary when both are working closely together and introduces superfluous overhead. As the V-Modell does not force an order of the workpackages, every life cycle is possible. However no lifecyle gets specific support. Having extra releases is not supported, as is plaiting of external work. As the main phases are essentially the same as with waterfall based models it is unable to deal with heavily fluctuating requirements. It has a high initial adaption effort, that is problematic for part time employees as it is with RUP: The initial documentation of the V-Modell are some 600 DIN A4 pages.

Extreme Programming (XP) is the most prominent agile SDP. Developed in 1999 by Beck [2] at Chrysler, it has gained much attention in the last years. It is build for small teams and changing requirements. And is constructed as a collection of already known Best Practices, that are to be used extremely. It features five major values: Communication, Simplicity, Feedback, Courage and Respect. Listing all the best practices here is out of scope, refert to [8] for a detailed description. We know that pair programming produces better code and does not consume that much more time [25], but students tend to have problems using it [14]. Furthermore it is really problematic to have Pair Programming and daily Stand-Up Meetings with part time employees. It requires quite a bit of synchronization overhead if its at all possible to get everyone together. XP relies on these two practices to empower the communication, it's problematic if they are not used, as XP doesn't use further approaches to transport knowledge. It tries to produce no documentation at all. This will surely result in trouble when employees change and their knowledge has to be transmitted. The consistent speed of development required by XP is hard to keep if the weekly availability of the students changes due to exams etc. Additionally planned releases can result in a higher amount of work in other weeks. Release planning and costing is not really considered by XP. The small iterations allow a permanent controll of the conformance to the requirements and the requirements self. XP relies heavily on testing, but it may be hard to develop test cases and desired values if no one really understands the area of interest. The Best Practice On-Site Customer is easy to use as the research scientist are more or less permanently available.

As the name suggests *Feature Driven Developement* (FDD) has the feature in the center of interest. A feature is defined as "The features are small 'useful in the eyes of the client' results" [18] [11]. Additionally no feature is allowed to require more than two weeks of development time. If its more complex, it needs to be split. After each implementation of a feature, the software must be in a

functional state. That makes releasing it simple, but there is no planning which features are included in a release. It may be problematic to include larger chunks of external work in a two week timeframe, especially as you don't have some 40 working hours per developer available with part time employees. If the two week limit is risen, FDD doesn't work that well any more. FDD demands the developers to understand the details of the field of activity right at the beginning of the project. This won't work in the research environment, as not even the specialist in this area, the research scientists, have a complete understanding of the material. Adaption of FDD is a simple process, it is enough if only 20 percent of the developers are used to it [7].

Adaptive Software Development (ASD) is the SDP introduced and used by Highsmith in 2000 [12]. It tries to focus on fast changing requirements. Like FDD and XP, its a bundling of Best Practices to a new SDP. To be generic no rules are enforced, everything can be omitted. Highsmith hopes for a good team to make these decisions carefully. This is often criticized [9, 19], and extremely problematic for new developers: They may accidentally fall back to unscheduled, chaotic development. Additionally the ASD main document [13] is way to long for an introduction, and contains bulks of unimportant passages [10].

Scrum is a lightweight, iterative and adaptive SDP [21]. There are extremely positive reports of scrums success with several hundred percent of improvements in productivity [24, 22, 23, 16]. Scrums fixed iteration are hard to keep, as the developers are working on different complex parts that won't finish simultaneously. Planning releases is possible with the concept of Sprints (Scrums name for the iterations). Progress monitoring with a Burn Down Chart will help, it is however not trivial to visualize projects progress when the amount of tasks is permanently increasing. Self organizational teams are a challenging demand as they are in ASD, because the team members are generally inexperienced. Dealing with impediments won't be trivial, too. Larger research institutes or even universities tend to have very inflexible structures. A very promising aspect is Scrums lifecycle. It does not really end, this requirement is only achieved by Scrum and ASD. Scrum is a very generic model and needs a more detailed one (XP is often used) for the details of the development.

4 Conclusions

As worked out in this paper, no known SDP is able to fullfill the requirements of software development in the research environment. Table 2 gives an overview of the fulfillment. A new SDP is needed to fill this gap and make developing software more planned. Currently most research facilities developing software use unscheduled development, that is perhaps not even the worst decision, but definitely not an ideal situation.

	Extrem changing requirements	Never ending lifecycle	Release planning	Plaiting of external work	Part time employees	Unexperienced employees	Inflexible stuctures
Unscheduled development	-	o	--	o	++	++	o
Waterfall model	--	--	+	--	o	+	++
Spiral model	o	-	--	--	o	+	o
Rational Unified Process	-	o	++	-	--	--	+
V-Modell XT	-	o	++	-	--	--	++
eXtreme Programming	+	o	-	o	--	--	--
Feature Driven Development	o	+	-	o	+	+	-
Adaptive Software Development	+	+	o	o	--	--	-
Scrum	o	++	+	o	-	-	--
Evolutionary Development	++	o	-	--	+	+	--

Figure 2: Overview of fulfillment of requirements.

References

[1] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. *Agile Software Development Methods: Review and Analysis*. VTT Publications, 2002.

[2] K. Beck. *Extreme Programming Explained*. Addison-Wesley Professional, 1999.

[3] D.M. Berry. The Inevitable Pain of Software Development, Including of Extreme Programming, Caused by Requirements Volatility. In *Radical Innovations of Software and Systems Engineering in the Future*, Canada, 2002. University of Waterloo.

[4] B. W. Boehm. A spiral model of software development and enhancement. In *IEEE Computer*, volume 21, pages 61–72. IEEE Computer Society Press Los Alamitos, CA, USA, 1988.

[5] B.W. Boehm. Guidelines for Verifying and Validating Software Requirements and Design Specifications. In *European Conference on Applied Information Technology of the International Federation for Information Processing*, volume 79, pages 711–719, London, 1979. North Holland Publishing.

[6] B.W. Boehm. Get ready for agile methods, with care. *IEEE Computer*, 35(1):64–69, 2002.

[7] J De Luca. What percentage of people are required to be experienced?, 2004. Von `http://www.featuredrivendevelopment.com/node/635` besucht am 12.12.2007.

[8] R. Dornberger and T. Habegger. Extreme Programming: Eine Übersicht und Bewertung, 2004. Diskussion Paper, Fachhochschule Solothurn Nordwestschweiz.

[9] C. Eberle. Adaptive Software Development. Technical report, Institut für Informatik, Universität Zürich, 2003. Seminar: Agile vs. klassische Methoden der Software-Entwicklung.

[10] C. Eberle. Adaptive Software Development, 2003. Institut für Informatik, Universität Zürich, Seminar: Agile vs. klassische Methoden der Software-Entwicklung, Seminar Foliensatz.

[11] D. Gyger. Feature-Driven Development. Technical report, Institut für Informatik, Universität Zürich, 2003. Seminar: Agile vs. klassische Methoden der Software-Entwicklung.

[12] J. A. Highsmith. *Adaptive software development: a collaborative approach to managing complex systems*. Dorset House Publishing Co., Inc., New York, NY, USA, 2000.

[13] J.A. Highsmith and J. Highsmith. *Agile Software Development Ecosystems*. Addison-Wesley Professional, 2002.

[14] J. Karn, T. Cowling, S.L. Syed-Abdullah, and M. Holocombe. Adjusting to XP: Observational studies of inexperienced developers. In *LNCS 3092: Extreme Programming and Agile Processes in Software Engineering*, volume 5, pages 222–225. Springer, 2004.

[15] S. Madsen and K. Kautz. Applying System Development Methods in Practice – The RUP Example. *Information Systems Development: Advances in Methodologies, Components and Management*, pages 267–278, 2002.

[16] C. Mann and F. Maurer. A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction. In *Proceedings of the Agile Development Conference*, pages 70–79. IEEE Computer Society Press Los Alamitos, CA, USA, 2005.

[17] R. Motschnig-Pitrik. Employing the Unified Process for Developing a Web-Based Application-A Case-Study. In *Practical Aspects of Knowledge Management: 4th International Conference, PAKM*, Vienna, Austria, December 2002. Springer.

[18] Coad P., Lefebvre E., and De Luca J. *Java Modeling in Color with UML: Enterprise Components and Process*. Prentice Hall, 1999.

[19] D. Riehle. A comparison of the value systems of Adaptive Software Development and Extreme Programming: How methodologies may learn from each other. In *Extreme Programming Explained*, pages 35–50. Addison-Wesley, Boston, USA, 2001.

[20] W. Royce. Managing the Development of Complex Software Systems: Concepts and Techniques. In *9th international conference on Software Engineering*, Monterey, California, United States, 1970. IEEE Computer Society Press Los Alamitos, CA, USA.

[21] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2001.

[22] J. Sutherland. Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies. *Cutter IT Journal*, 14(12):5–11, 2001.

[23] J. Sutherland. Future of Scrum: Parallel Pipelining of Sprints in Complex Projects. *Agile 2005 Conference*, 2005.

[24] J. Sutherland. Scrum Tuning: Lessons learned from Scrum implementation at Google, 2006. Google Tech Talks, Video at `http://video.google.com/videoplay?docid=8795214308797356840`.

[25] L. Williams, RR Kessler, W. Cunningham, and R. Jeffries. Strengthening the case for pair programming. *IEEE Software*, 17(4):19–25, 2000.