Software Processes in an Agile World

Horst Lichter
RWTH Aachen University, Research Group Software Construction
lichter@swc.rwth-aachen.de

Abstract

In this paper we relate classical software process models to new agile development processes and software process improvement. We argue that there is no single process model that always fits and that organizations have to re-use the best out of classical and agile processes. Furthermore we question "classical" software process improvement because it is often done isolated from people and technology issues. Finally, we present ten propositions about software process models and software process improvement.

Keywords*: Software Process Model, Software Process Improvement, Agile Development*

1. Introduction

Software development is a complex, challenging, and creative task. Although we have more than 50 years of software development experience and we are living in world where software is playing an outstanding and important role, we do not have one single completely sound Software Engineering approach that always leads to high quality software. But there are plenty of methods, languages, and techniques that have proven to be successfully applicable in industrial software development (nevertheless, they are not applied always and everywhere). Software development processes play an important role in software development. The IEEE Software Engineering Glossary [3] defines this term as follows:

> **software development process** — The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use.
> Note: These activities may overlap or be performed iteratively.

We always should clearly distinguish the concepts *software process model* and software process. A software process model generically describes tasks and artifacts of software development. It serves as a template that can be instantiated many times in different projects; we call a concretely instantiated software process model a software process [9].

There are two different kinds of software process models: abstract and concrete ones. Abstract software process models describe as a matter of principle how to organize software development (e.g., Software Life Cycle, Prototyping, and Spiral Model). Hence, abstract process models cannot be instantiated in a project off the shelf. In contrast, concrete software process models, e.g. the Unified Process [6], define all aspects that are needed to instantiate the process model in a project (usually after some tailoring). That means a concrete software process model defines all activities, all roles and all artifacts that have to be established in a project.

Software Process Improvement (SPI) has been introduced in 1990th by the Software Engineering Institute (SEI) as a systematical and continuous approach to improve the maturity of software process models based on the Capability Maturity Model (CMM). Since then, the CMM has evolved to the CMMI [4] and SPI has become very popular in the software industry (see e.g. [15]).

In this paper we want to relate classical software process models to new agile process models and SPI. The paper is organized as follows: At first we present a brief historical overview and describe the commonalities of agile development approaches. Then we focus on how to get the right software process model and present our concerns regarding classical SPI. We conclude by formulating ten propositions about software process models and SPI.

2. Software Process Models – A Brief Historical Overview

Software process models have been introduced in Software Engineering in the 1970th when developers encountered that the problems that were solved by means of software had become so complex that software could no longer be developed by a single person in a single step (namely programming). The historical paper of Kron and DeRemer [2] who introduced the term *Programming in the Large* reflects this situation pretty well. Hence, new software development organization rules were needed. An important milestone in the development of process models is the so called *Waterfall Model* which was introduced by Walter Royce [1]. For the very first time, this model identifies all major software development activities or steps. Although there is lots of criticism of this model and today we know that it does not fit for innovative developments, it has influenced and impelled Software Engineering a lot. To overcome the drawbacks of the Waterfall Model, Berry Boehm introduced the so called *Spiral Model* [5]. It focuses on development risks and arranges the development process in a way that the top level risks are solved first and so forth. With this in mind, risk driven development as proposed by the Spiral Model is another important contribution of the early process models to Software Engineering.

In the 1990th it was encountered that a sequential process model does not appropriately reflect typical development scenarios where developers are not able to capture all requirements at the beginning and the operation of a software systems changes its own requirements. To cope with changing requirements iterative and incremental software process models have been introduced. The core idea of both *iterative* and *incremental* software development is (1) to create a solution quickly and to revise it systematically based on user feedback until it fits to the user's needs and (2) not to develop the complete functionality in one big project but to evolve the functionality step by step. Typically, iterative and incremental process models include *Prototyping* as a means to cheaply build an early version of the system that can be assessed by the users [10]. The Unified Process [6] and the V-Model XT [14] are well known iterative, prototyping-based, and incremental process models. For instance, in UP each increment development is divided into four development phases; inside each phase the work is done iteratively based on prototypes.

3. Agile Software Development

In response to software process models aiming to organize software development completely by defining lots of roles, activities and documents (e.g. UP) a group of important software engineers have published the so called *Agile Manifesto* [8] in 2001. Agile approaches expect that during the implementation users discover the possibilities of the system, adjust requirements, and want to profit immediately from the system. The aim is to deliver operating functionality of the software as fast as possible, starting with the components that are most important for the user.

Hence, the main idea of agile software development is to concentrate on the users' needs and to develop solutions quickly without unnecessary overhead (i.e. documents). Since then, a series of agile software process models has been proposed. Some of them are more or less abstract frameworks for software development like Crystal; others have been used intensively in industry like Extreme Programming or Scrum. A detailed review can be found in [7]. Although these process models are very diverse they share the following common characteristics [9]:

- Development is done iteratively; the development cycles are not longer than three month (preferably much shorter).
- The size of the project team is small (typically six to ten people); the team usually shares one or two offices (i.e., the development is not distributed).
- The client is integrated into the team and most of the time available.
- The work is organized and assigned by the team itself.
- Documentation is reduced to a minimum; the most important result is a running system.

Agile process models are often called *light-weight* in order to clearly distance them from documentation-centric or *heavy-weight* process models, e.g. UP.

All in all, nowadays software organizations have at command a tool-box filled with a couple of very different process models. Therefore, software organizations have to decide which process model or

which portfolio of process models to apply. This decision is not a simple one because the applied process model heavily influences the organization, its projects, its culture and especially its developers.

4. The Search for the Best Software Process Model

The choice of the "right" software process model is crucial for each software development organization. Based on published and own experiences it can be stated, that there is no single software process model that fits for all organizations. But, what are the most important criteria that influence the choice of a software process model? Again, there is no single answer. The following aspects and questions should be taken into consideration when choosing a software process model:

- What kind of software do we develop (e.g., embedded software, information systems, safety critical software)?
- Do we always develop similar software of the same application domain?
- What is the size of our typical projects and project teams?
- Do we develop software at one site or is the development distributed over different sites?
- What are the skills of our developers (e.g. programming, quality assurance, etc.)?
- Do we have close relationships to the customer?
- Are we the principal contractor or are we a sub-contractor?
- Are we free to choose a software process model or do we have to apply the one that the customer requires?

This list of aspects is far from being complete. But, it demonstrates that there are lots of impact factors regarding the choice of a software process model. These factors might be conflicting and might not be assessed in isolation. Thus, an organization has to make compromises. A large organization that runs many different projects in different domains in parallel cannot expect that one single process model is sufficient. It needs a portfolio of process models with defined tailoring options in order to cover its project variety. One the other hand, a small software organization that is specialized in one specific domain may be able to manage its projects based on one or two very similar process models. Sometimes the discussion whether to apply classical or agile software processes is a "religious" one. But, there is no single truth. Obviously, organizations have to combine the best of classical process models and agile ones. It would be careless not to re-use everything that has been proven to effectively support software development. Fortunately, we do have choices and we should base our decisions on explicit impact factors and on all available software process models.

To sum up, each organization is responsible for selecting and applying those process models that support their business goals and that fit best to their specific project constraints. A selected software process model has to support the project team; it should define all those aspects that are really needed to successfully complete projects. Everything else is "dead freight" that costs resources and does not contribute to project success. Organizations should never apply process models for their own sake!

As the world is changing and software development is changing as well, each organization has to monitor its software processes continuously in order to know whether they still fit or whether they have to be revised or exchanged by new ones.

5. Process Improvement – Valuable Investment or Waste

Software process improvement (SPI) has been promoted by SEI and some other protagonists for many years. Based on reference models like CMMI, SPICE or ITIL process assessment and improvement has become a big business. Beside this economical perspective, each organization has to answer the following questions:

1. How much SPI is needed?
2. What SPI measures have to be performed?
3. How to organize and manage SPI in the organization?

As there are many publications addressing these issues (e.g. [11], [12], [13]) we want to focus on other important aspects that are related to SPI. We see the following three close connections (see Fig. 1):

- First, the applied software process has to support the project and has to be flexibly customizable to the project's needs. A process model is only beneficially if all involved persons (i.e. developers and managers) accept it; i.e., they have to be convinced that the process model supports their work and contributes to the project's success.
- Second, all technologies (i.e. languages, methods and tools) applied in a project have to be selected in a way, that they enable the team to develop the software efficiently and effectively.
- Third, all team members have to have all skills needed to apply the software process model as well as the technologies.
-

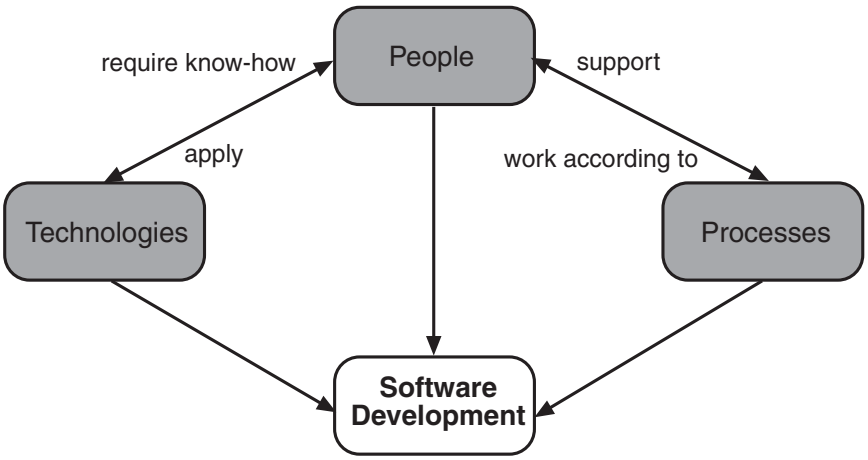

Figure 1. Impact factors of software development

Hence, a software process model should not be considered and improved in isolation; it is always embedded in an organization and linked to people and technologies. This implies that each SPI measure should always take into consideration both the people and the technology dimension. Therefore, each SPI measure should not only define how to improve a special process (e.g. test process) but also include a training and technology component (e.g. training on test methods, selection of appropriate test tools). If an organization does not improve process, people and technology in sync, SPI is waste, otherwise is might be a valuable investment. A last remark: Very often organizations decide to go the CMMI way. And sometimes this leads – besides high costs – to frustrated developers with decreased motivation. If SPI is implemented wrongly it is counterproductive and finally produces "scorched earth".

6. Conclusion

In this paper we argued that the application of process models is a prerequisite to successfully complete software development projects. But, as there are very different project constraints and process models, each software organization has to choose those process models that best fit. The same holds for software process improvement. There is no silver bullet and no solution that fits always and everywhere.

Therefore we want to conclude this paper with ten propositions:

1. Organizations have to apply process models in order to successfully run development projects!
2. There are many process models that have proven to support software development!

3. There is and there will be no process model that fits to all project conditions!
4. Organizations have to choose actively and explicitly process models that best fit to their specific environment and conditions!
5. A process model has to support and not to burden the project team!
6. A process model is beneficial if and only if it is accepted by the developers!
7. Applied process models have to be evolved or exchanged by new ones, because the world is changing!
8. Organizations have to combine the best out of classical and agile models!
9. SPI has to consider the people and technology dimensions as well!
10. SPI has to be implemented in a way that all involved persons are convinced of the benefits. It should not been done for its own sake!

7. References

[1] Royce, W.W. , "Managing the development of large software systems". IEEE WESCON, Los Angeles, CA, 1-9; re-printed in Proceedings of 9th ICSE, Monterey, CA, IEEE Computer Society Press, 328-338, 1970.
[2] DeRemer, F., H.H. Kron, "Programming-in-the-Large Versus Programming-in-the-Small". IEEE Transactions on Software Engineering, Vol. 2 (5), 80-86, 1976.
[3] IEEE Std 610.12, "IEEE Standard Glossary of Software Engineering Terminology". IEEE Standards Association, 1990.
[4] CMMI, "CMMI for Development, Version 1.2". CMU/SEI-2006-TR-008, ESC-TR-2006-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.
[5] Boehm, B.W., "A Spiral model of Software Development and Enhancement". IEEE Computer, Vol. 21 (5), Mai 1988, 61-72, 1988.
[6] Jacobson, I., G. Booch, J. Rumbaugh, "The Unified Software Development Process". Addison-Wesley, Boston, MA, 1999.
[7] Abrahamsson, P., O. Salo, J. Ronkainen, J. Warsta, "Agile software development methods. Review and analysis". Espoo 2002, VTT Publications 478, 2002.
[8] Manifesto for Agile Software Development. http://www.agilemanifesto.org/
[9] Ludewig, J., H. Lichter, „Software Engineering – Grundlagen, Menschen, Prozesse, Techniken". 2. Aufl. (in German), dpunkt.verlag Heidelberg. ISBN 9783898646628, 2010.
[10] Bäumer, D., W. Bischofberger, H. Lichter, H. Züllighoven, "User Interface Prototyping – Concepts, Tools, and Experience". Proceedings of 18th ICSE, Berlin, IEEE Computer Society Press, 532-541, 1996.
[11] Pricope, S., H. Lichter, "A Model Based Integration Approach for Reference Models". Second Proceedings of 12th International Conference on Product Focused Software Development and Process Improvement, Danilo Caivano et al., Torre Canne, Italy, June 20-22, ACM, New York, NY, USA, ISBN: 978-1-4503-0783-3, pp 6-9, 2011.
[12] McFeeley, R., "IDEAL: A User's Guide for Software Process Improvement". CMU/SEI-96-HB-001, ADA 305472, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.
[13] Suwanit Rungratri, Sasiporn Usanavasin, "Semantic based Approach Supporting CMMI Gap Analysis Process", JCIT, Vol. 7, No. 20, pp. 127 -137, 2012.
[14] Fundamentals of the V-Model: XThttp://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/1.3/Dokumentation/V-Modell%20XT%20HTML%20English/
[15] Fatemeh Kafili Kasmaee, Ramin Nassiri, Gholamreza Latif Shabgahi, "Achieving CMMI Maturity Level 3 by Implementing FEAF Reference Models", IJACT, Vol. 2, No. 4, pp. 115-122, 2010