



Making Agile Processes Scalable

Harald Störrle

University of Munich & FJA GmbH

3.5.2003
International Workshop on
Process Simulation and Modelling
Portland, Oregon

[http://www.pst.informatik.uni-muenchen.de/
personen/stoerrle/Veroeffentlichungen](http://www.pst.informatik.uni-muenchen.de/personen/stoerrle/Veroeffentlichungen)

ProSim'03, Portland, OR © 2003, Dr. Harald Störrle, stoerrle@informatik.uni-muenchen.de

Two competing paradigms *comprehensive and adaptive*

- Currently, we see two competing paradigms in modeling and managing software:
 - traditional approaches like VM, CMM, SPICE;
 - so called „agile“ approaches like XP, SCRUM, FDD.
- These paradigms have different focus, benefits and problems.
- I call these two *comprehensive* and *adaptive*, since these are their main goals, respectively.

ProSim'03, Portland, OR © 2003, Dr. Harald Störrle, stoerrle@informatik.uni-muenchen.de

Note that both adaptability and comprehensiveness may refer to different aspects. For instance,

- the goal of covering the **complete software lifecycle** („cradle to grave“), is the main motivation for ISO 12207 and stated explicitly in the preamble;
- the VM deals with **all types of projects**: software, hardware, and co-development;
- the CMM and SPICE deal with **all levels of capability**, and **all phases in the process lifecycle**, and so on.

Looking at adaptability,

- XP is set up to adapt to changes in **requirements contents and priority**,
- SCRUM is set up to take into account for ensuing steps **knowledge obtained** in the previous step, i.e. it is more like a *learning* method.

At this point, let me deliver my standard disclaimer: the observations I am presenting here are, of course, somewhat streamlined - there are counterexamples, exceptions and possibly even contradictions. However, I claim, the trend and the conclusions are valid.

I shall now discuss the characteristics of both of them briefly.

The comprehensive paradigm Characteristics

- Comprehensive („traditional“) models provided a conceptual framework to understand/model processes in the first place, and manage them at all, for the first time.
- Comprehensive processes are
 - + efficient under stable conditions
 - + applicable for very large projects
 - + good understanding of process improvement
 - very large models (steep learning curve, top-down req.'d)
 - require tailoring
 - too inflexible at times
 - large overhead for small projects and small organizations

ProSim'03, Portland, OR © 2003, Dr. Harald Störrle, stoerrle@informatik.uni-muenchen.de

The traditional waterfall model emerged from the first few really big projects, such as OS/360, and it tried to replace chaos by order, eradicating change.

I believe that for many, if not all large projects this is still de rigeur, simply because this is your only chance to achieve *some* manageability for the overall development effort.

From there on, process technology expanded on this Leitmotiv, covering more and more areas (reuse, reengineering, maintenance, acquisition, ...) and concerns (management, modeling, introduction, improvement, simulation).

However, in doing so, these models became too large, and thus less applicable to small projects, or projects with changing requirements.

The adaptive paradigm Characteristics

- Adaptive („agile“) models improved management under changing environment conditions (e.g. requirements, technology), but gave away some benefits.
- Adaptive processes are
 - + very responsive to changing environment
 - + easy to spread („sexy“, simple)
 - unclear notion of improvement
 - very limited scalability
 - often just an excuse for hacking

ProSim'03, Portland, OR © 2003, Dr. Harald Störrle, stoerrle@informatik.uni-muenchen.de

What makes adaptive approaches so adaptive?

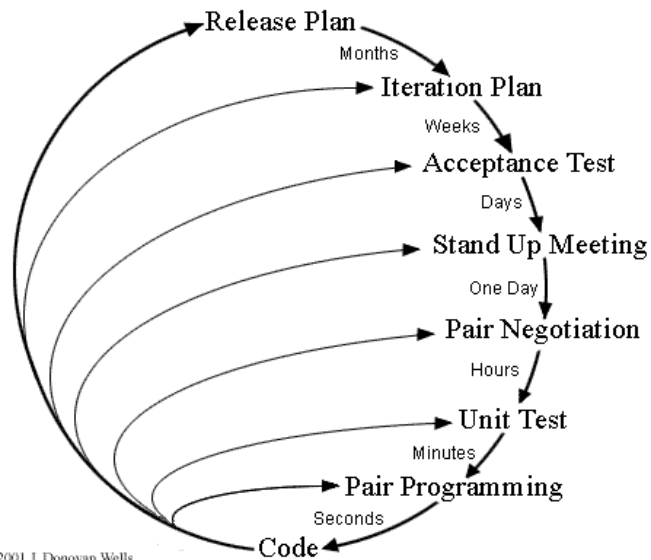
They have (many) layers of feedback or short iterations

Let's look at two examples of this, the two agile approaches probably best known: XP and Scrum.

The adaptive paradigm

Feedback in XP

Planning/Feedback Loops

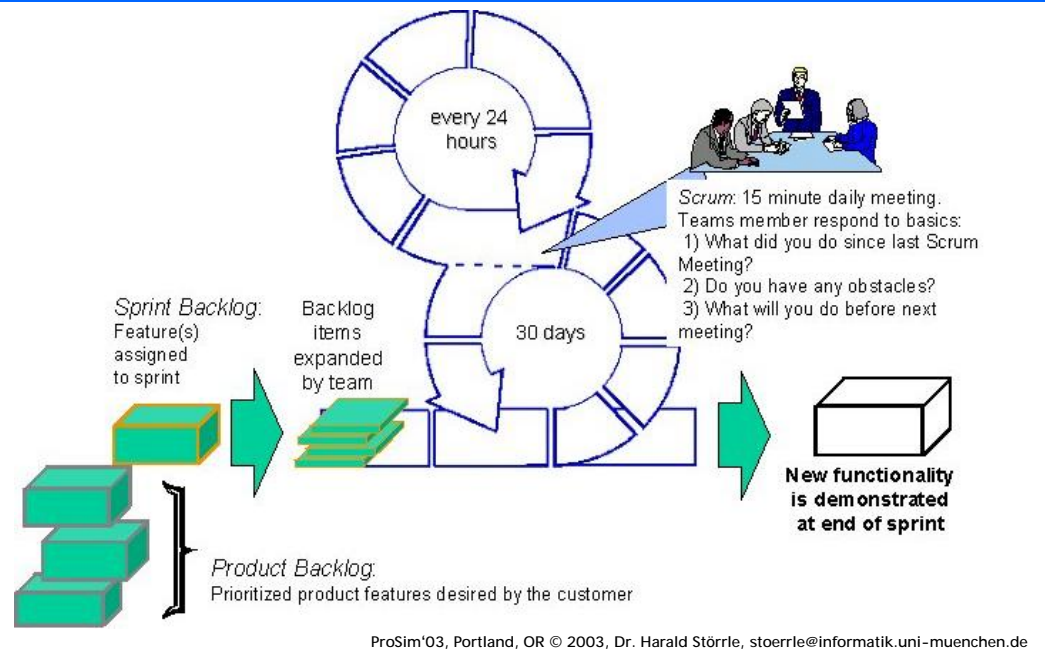


Copyright 2001 J. Donovan Wells

ProSim'03, Portland, OR © 2003, Dr. Harald Störle, stoerle@informatik.uni-muenchen.de

The adaptive paradigm

Feedback in Scrum



Now, having examined these two paradigms and their respective characteristics.

Blending the two paradigms

- Can we have the best of both paradigms?
- In particular, can we have an adaptive process
 - that works for non-miniscule projects,
 - that has a useful notion of process improvement,
 - that allows selective usage of different approaches,
 - and thus offers a better cost/benefit-ratio?
- First of all, we need a common framework that is capable of encompassing both paradigms.

ProSim'03, Portland, OR © 2003, Dr. Harald Störrle, stoerrle@informatik.uni-muenchen.de

Yes, I think we can. For I believe, the two paradigms are not mutually exclusive, but complementary: each has its application domain, i.e. a set of context conditions, where it applies, and where the other one does not apply equally well.

Now, the problem is: what do we do with projects that do not cleanly fall in either of these categories, but exhibit features of both, or none (and that is the vast majority)? As a first step towards a solution, we need a common way of describing **all** solutions, a common descriptive framework: how can we express them both on equal footing?

I propose to structure process descriptions by patterns, that is, use process patterns. A process pattern is very similar to a design pattern, only that its about the process rather than the product.

A pattern (of any kind) is experience condensed into succinct rule, or more traditionally [->]

Process Patterns

Rationale & Characteristics

Definition

„A pattern is a practically proven solution to a recurring problem.“

Characteristics

- good cost/benefit-ration (Pareto-principle)
- easy to integrate existing knowledge
- supports internal marketing
- patterns are (as a concept) scale invariant
- conceptual framework to blend both approaches
- patterns always come in families

ProSim'03, Portland, OR © 2003, Dr. Harald Störrle, stoerrle@informatik.uni-muenchen.de

I suggest to also use a descriptive schema very similar to that known from Design Patterns.

Thus, there can be a discussion of why would want to use a particular pattern in a given situation, what the problem, and what the solution are, what interactions there are with other patterns, how it may be classified, who collaborates in reifying the pattern and so on.

One point I want to elaborate a bit for you is classification. In design patterns, there is a distinction into several „levels“, namely idioms, proper patterns and architectural styles. I found this distinction extremely helpful, both for capturing and for spreading knowledge, so, for process patterns, I propose to distinguish between

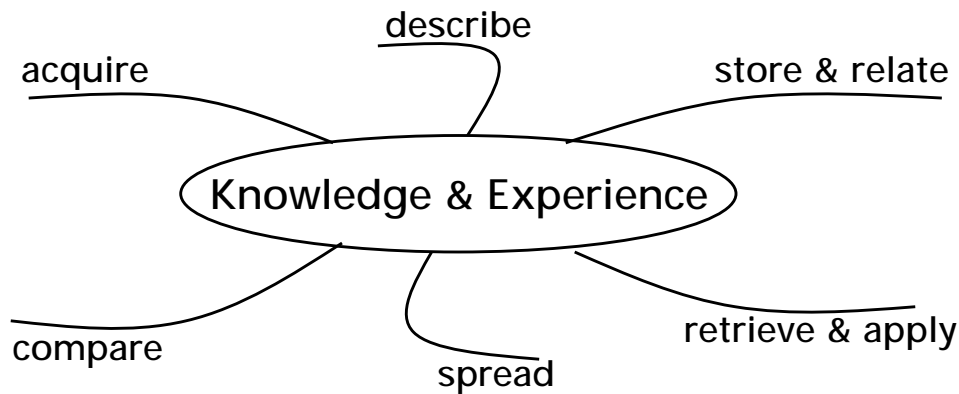
- 1) best practices („Anleitungen“: XP practices, QuickCards);
- 2) task guide („Aufgabenbeschreibung“: change mgmt [see paper],); and
- 3) project process („Projektvorgehen“: XP, RUP, tailored VM instance).

I am well aware of the terminology overloading here, but that doesn't matter in real life.

Comming back to the notion of process patern, one might also say: a pattern is a pragmatic solution to a practical problem with limited resources, abstracted from experience. And that is exactly the definition of engineering - which is, I reckon, why engineers love it.

(Another obvious interpretation is that of problem solving or decision making with bounded rationality, and here we are back in the 80ies, doing Artificial Intelligence.)

Dealing with Software Processes is Knowledge Management



ProSim'03, Portland, OR © 2003, Dr. Harald Störrle, stoerrle@informatik.uni-muenchen.de

One thing that engineers tend not to like quite as much is the fact, that dealing with software processes is really a kind of knowledge engineering, in a pretty immediate way, and that means, dealing with people, for knowledge is, by any useful definition, something people may or may not possess, but never machines.

Now, I have a little exercise for you: take whatever is your favourite process model or descriptive framework or paradigm or whatever you call it, and match it against these requirements here: does your pet approach satisfy them? Does it satisfy them better than process patterns? In particular, take the paradigms postulated before (comprehensive & adaptive) and compare their score with that of a pattern approach here. Now, this is just for you to ponder about.

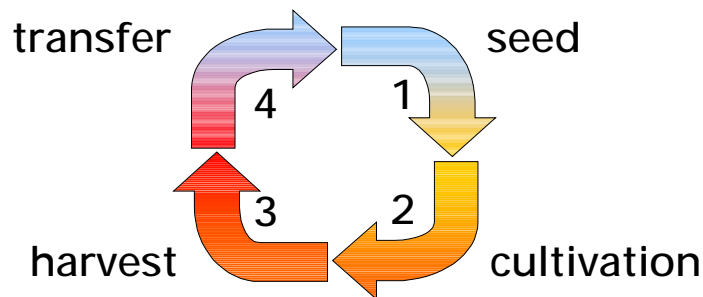
Note again, that in putting the focus on knowledge rather than models, I emphasize the people-dimension of software process, that is, things like staffing procedures and career development, which I think must be an integral part of a process model, at least in the western world. It may not be as true when it comes to delegating specific development activities to other countries where avg. qualification and labour prices are lower.

So, in our context, how can we further software processes? Traditional process technology relies on management action to introduce and improve processes. Mounting and pursuing a CMM programme, for instance, is a gargantuan task that must be driven and fed for years. Such efforts can only succeed if management is very strongly committed, they require enormous discipline. So, instead of comprehensive, I might have also called it the top-down, or the calvinist paradigm, if I wouldn't want to avoid allusions to a broader cultural context.

Focusing on knowledge and participation rather than procedures and obedience, I would like to introduce an agricultural metaphor.

[Explanation according to paper]

Introduction, Improvement & Spreading of Process Patterns



ProSim'03, Portland, OR © 2003, Dr. Harald Störrle, stoerrle@informatik.uni-muenchen.de

On the contrary, adaptive approaches start at what individuals, or small groups can do - without a lot of commitment indeed on behalf of many layers of hierarchy above them.

So, I could have also dubbed this paradigm the grassroots or hedonistic paradigm, again, if I wouldn't want to avoid allusions to a broader cultural context.

To be sure: I do believe that building software is to a much greater level related to cultural and sociological currents than, say, the production of cars - and we all know the role cultural and sociological facts played even there, just look at the introduction of Taylorism in car manufacturing, and its eventual demise, being replaced by what we today call lean manufacturing, and which comes from Japanese and Swedish sources.

However, I'm not a sociologist and lack both methodology and understanding to properly deal with such subjects. Thus I shall end my little digression here, and return to the introduction and improvement of software process - knowledge (!).

What else, what next?

- **Modularity**
 - Binding within (process) patterns
 - Coupling between (process) patterns
 - Compositionality - treatable in isolation
- **Simulation**
 - Could it be done?
 - Would it make sense?

Wrap up

- Process Patterns are
 - structured just like design patterns;
 - good to capture existing knowledge;
 - a unifying framework for both paradigms;
 - help spread knowledge at grassroots level.



www.pst.informatik.uni-muenchen.de/~stoerrle

ProSim'03, Portland, OR © 2003, Dr. Harald Störrle, stoerrle@informatik.uni-muenchen.de