softwaretechnik / software-engineering

lecture 5: procedure & process models

2016-05-09

Prof. Dr. Andreas Podelski,
Dr. Bernd Westphal
Albert-Ludwigs-Universität Freiburg, Germany

topic area project management: content

• VL 2 software metrics
  • properties of metrics
  • scales
  • examples
  • cost estimation
  • "(software) economics in a nutshell"
  • expert's estimation
  • algorithmic estimation

• VL 3 project management
  • project
  • process and process modeling
  • procedure models
  • process models
  • ..
  • ..
  • process metrics
  • CMMI, Spice

• VL 4 ...

• VL 5 ...

procedure vs. process models

(Ludewig and Lichter, 2013) propose to distinguish:
• process model and procedure model.

• a process model ('prozessmodell') comprises:
  i. procedure model ('vor gehensmodell') e.g., "waterfall model" (70s/80s).
  ii. organisational structure — comprising requirements on:
    • project management and responsibilities,
    • quality assurance,
    • documentation, document structure,
    • revision control.
    e.g., V-Modell, RUP, XP (90s/00s).

• in the literature, process model and procedure model are often used as synonyms;
  there is not universally agreed distinction.

procedure models

...
Knowing that the biggest risks are eliminated gives a good feeling.

We know early if the project goal is unreachable.

...are not risks, but environment conditions to be dealt with ...

Advantages:

• Requires: “Average Probability per Flight Hour for Catastrophic Failure Conditions”

• A concrete process using the Spiral Model could look as follows:

1. Conduct risk analysis.
2. Investigate goals, alternatives, side conditions.
3. Assess a project’s process (cost, project progress).

Note: Risk and risk value.


The Spiral Model (Boehm 1988).

Waterfall or Document-Model (Rosove 1967).

The (In)famous Waterfall Model (Boehm 1967).
Linear vs. Non-Linear Procedure Models

- **Linear**: the strict Waterfall Model (no feedback)
- **Non-Linear**: basically everything else (with feedback between activities)

Classification By Treatment of (Software) Artefacts

- **Prototyping**:
  - **req. prototype**: prototype results develop

- **Evolutionary**:
  - **req. iteration 0**: iteration 1... iteration n

- **Iterative**:
  - **req. plan spec. 1 spec. 2 spec. 3 iteration 1 iteration 2 iteration 3**

- **Incremental**:
  - **req. 1 project 1**
  - **req. 2 project 2**

- **Staircase**: pipelined incremental

(Rapid) Protoyping

- **req. prototype**:— A preliminary type, form, or instance of a system that serves as a model for later stages or for the final, complete version of the system.

IEEE 610.12 (1990)

- **prototyping**:— A hardware and software development technique in which a preliminary version of part or all of the hardware or software is developed to permit user feedback, determine feasibility, or investigate timing or other issues in support of the development process.

IEEE 610.12 (1990)

- **rapid prototyping**:— A type of prototyping in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process.

IEEE 610.12 (1990)

Kinds of Prototypes and Goals of Prototyping

- **demonstration prototype ('Demonstrationsprototyp ')**:
  - 'throw away products ' to demonstrate look-and-feel or potential usage of proposed product; can be "quick and dirty"

- **functional prototype ('Funktionaler Prototyp ')**:
  - usually regarding (graphical) user interface; maybe many separate prototypes for specific questions

- **lab sample ('Labormuster')**:
  - addresses open technical questions, proof-of-concept; need not be part of the final system

- **pilot system ('Pilotsystem ')**:
  - functionality and quality are at least sufficient for a (temporary) use in the target environment

Floyd Taxonomy:

- **explorative prototyping** — support analysis
- **experimental prototyping** — develop new technology
- **evolutionary prototyping** — the product is the last prototype (categories may overlap)

Prototyping Procedure Model

- **Approach**: Clarify
  - which purpose does the prototype have, what are the open questions?
  - which persons (roles) participate in development and, most important, assessment of the prototype?
  - what is the time/cost budget for prototype development?

question prototype specification operation environment prototype assessment

prototype determines operation environment influences prototype assess modifies

basis of prototype development

Ludewig and Lichter, 2013
Evolutionary and Iterative Development

Evolutionary software development—an approach which includes evolutions of the developed software under the influence of practical/field testing. New and changed requirements are considered by developing the software in sequential steps of evolution.

Ludewig & Lichter (2013), flw. (Züllighoven, 2005)

Iterative software development—software is developed in multiple iterative steps, all of them planned and controlled. Goal: each iterative step, beginning with the second, corrects and improves the existing system based on defects detected during usage. Each iterative steps includes the characteristic activities analyse, design, code, test.

Ludewig & Lichter (2013)

Incremental Development

Incremental software development—The total extension of a system under development remains open; it is realised in stages of expansion. The first stage is the core system. Each stage of expansion extends the existing system and is subject to a separate project. Providing a new stage of expansion typically includes (as with iterative development) an improvement of the old components.

Ludewig & Lichter (2013)

• Note: (to maximise confusion) IEEE calls our "iterative" incremental:

IEEE 610.12 (1990)

• One difference (in our definitions):
  • iterative: steps towards fixed goal,
  • incremental: goal extended for each step; next step goals may already be planned.

Examples: operating system releases, short time-to-market (→ continuous integration).

Another Characterisation of Approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Has Complete Plan</th>
<th>Has Defined Steps</th>
<th>Preliminary Results Used</th>
<th>Results Used on Target System</th>
<th>Used for Analysis of Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>RapidPrototyping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evolutionary Development</td>
<td>yes</td>
<td>to some amount</td>
<td>to a low amount</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iterative Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incremental Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Staircase Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Procedure and Process Models

• Procedure Model Examples
  • The (in)famous Waterfall model
  • The famous Spiral model

• Procedure classification
  • linear / non-linear
  • prototyping
  • evolutionary, iterative, incremental

• Process Model Examples
  • V-Modell XT
  • Agile
  • Extreme Programming
  • Scrum

Process Metrics

• CMMI
• Spice

From Procedure to Process Model

A process model may describe:
• steps to be conducted during development, their sequential arrangement, their dependencies (the procedure model);
• organisation, responsibilities, roles;
• structure and properties of documents;
• methods to be used, e.g. to gather requirements or to check intermediate results;
• project phases, milestones, testing criteria;
• notations and languages;
• tools to be used (in particular for project management).

Process models typically come with their own terminology (to maximise confusion?), e.g. what we call artefact is called product in V-Modell terminology.
You may hear about “light” and “heavyweight” process models. Sometimes: heavier means higher number of rules. Sometimes: heavier means less flexible, adaptable process. Clear: “lightweight” sounds better than “heavyweight”. In the end, a process model is too “light” if it doesn’t support you in doing things which are useful and necessary for your project; a process model is too “heavy” if it forces you to do things which are neither necessary nor useful for your project. Thus, following (Ludewig and Lichter, 2013), we will not try to assign the following process models to a “weight class”.

The Phase Model

The project is planned by phases, delimited by well-defined milestones. Each phase is assigned a time/cost budget. Phases and milestones may be part of the development contract; partial payment when reaching milestones. Roles, responsibilities, artefacts defined as needed. By definition, there is no iteration of phases. But activities may span (be active during) multiple phases. Not uncommon for small projects (few software people, small product size), small companies.

V-Modell XT

There are different V-shaped (→ in a minute) process models, we discuss the (German) “V-Modell”. “V-Modell”:

- developed by company IABG in cooperation with the Federal Office for Defence Technology and Procurement (‘Bundesministerium für Verteidigung’), released 1998
- (German) government as customer often requires usage of the V-Modell
V-Modell XT: Example Building Block & Product State

SW-Development ('SW-Entwicklung') vs. coding

V-Modell XT: Disciplines and Products

V-Modell XT: Roles (even more?!)

Project Roles:

Organisation Roles:
Akquisiteur, Datenschutzbeauftragter (Organisation), Einkäufer, IT-Sicherheitsbeauftragter (Organisation), Qualitätsmanager.

V-Modell XT: Activities (as many?!)

V-Modell XT: Decision Points

V-Modell XT: Process Details (a mini map)
V-Modell XT: Project Types

V-Modell XT considers four different project types:

- **AG**: project from the perspective of the customer (create call for bids, choose developer, accept product)
- **AN**: project from the perspective of the developer (create offer, develop system, hand over system to customer)
- **AG/AN**: customer and developer from the same organisation
- **PM**: introduction or improvement of a process model

Project type variants: one/many customer(s); development/improvement/migration; maintenance

Recall the idea of the "V shape":

- Requirements fixed
- Requirements fixed
- Acceptance
- Acceptance
- System specified
- System specified
- System delivered
- System delivered
- Architecture designed
- Architecture designed
- System integrated
- System integrated
- Modules designed
- Modules designed
- System realised
- System realised
- Verification & validation

V-Modell XT mainly supports three strategies, i.e. principal sequences between decision points, to develop a system:

- Incremental
- Component based
- Prototypical
requirements
fixed
acceptance
system specified
system delivered
architecture designed
system integrated
modules designed
system realised
verification & validation
incremental component based prototypical

Advantages:
• certain management related building block are part of each project, thus they may receive increased attention of management and developers
• publicly available, can be used free of license costs
• very generic, support for tailoring
• comprehensive, low risk of forgetting things

Disadvantages:
• comprehensive, tries to cover everything; tailoring is supported, but may need high effort
• tailoring is necessary, otherwise a huge amount of useless documents is created
• description/presentation leaves room for improvement

Needs to prove in practice, in particular in small/medium sized enterprises (SME).

Agile
The Agile Manifesto
We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan that is, while there is value in the items on the right, we value the items on the left more.

Agile Principles
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
Business people and developers must work together daily throughout the project.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
Working software is the primary measure of progress.
Simplicity — the art of maximizing the amount of work not done — is essential.
Continuous attention to technical excellence and good design enhances agility.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
The best architectures, requirements, and designs emerge from self-organizing teams.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Similarities of Agiles Process Models
• iterative: cycles of a few weeks, at most three months.
• Work in small groups (6–8 people) proposed.
• Dislike the idea of large, comprehensive documentation (radical or with restrictions).
• Consider the customer important; recommend or request customer's presence in the project.
• Dislike dogmatic rules.

(Loos and Lichter, 2013)
description of obstacles/problems during sprint

• sprint-burndown report
  see
  • which requirements (not) realised in last sprint,
  • release-burndown report
  • assess how well the scrum process was implemented; identify actions for improvement (if necessary)
  • sprint report
  • sprint retrospective
  • should be able to assess techniques and approaches
  • how many sprints, which major requirements in which sprint,
  • assess results of sprints
  • up-to-date, product backlog
  • environment needs to support communication and cooperation, e.g. by spatial locality
  • assess amount and quality of realisations; product owner accepts results

• should decrease linearly, otherwise remove tasks from sprint backlog,
  • based on initial version of product backlog,
  • collects tasks to be conducted,
  • (passive) participant of 
  • plans and decides which requirement(s) to realise in next sprint,
  • maintains requirements in the
  • represents customer,
  • responsible for keeping:
  • sprint review
  • completed/open tasks from sprint backlog,
  • product backlog
  • looks for adherence to process and rules,
  • decides how and how many requirements to realise in next sprint,
  • members capable of developing autonomously,
  • sprint backlog
  • comprises all requirements to be realised,
  • product owner

• involved:
  • scrum team
  • three roles—scrum master, product owner (if possible)

• inspired by
  • nonaka and takeuchi
  • first published 1995 (yes, the "hooligan's game played by gentlemen")
  • rugby
  • scrum

• inspired by
  • extreme programming (xp)
  • extreme programming (xp) →
  • planning game
  • continuous integration
  • pair programming
  • central metaphor
  • acceptable workload
  • joint responsibility for the code
  • integral team (including customer)
  • coding conventions
  • test driven development
  • refactoring
  • coding respect, courage, communication, feedback, simplicity

• extreme programming (xp)
  • xp values
  • xp practices
Scrum: Discussion

• Has been used in many projects, experience in majority positive.
• Team size bigger 7–10 may need scrum of scrums.
• Competent product owner necessary for success.
• Success depends on motivation, competence, and communication skills of team members.
• Team members are responsible for planning, and for adhering to process and rules, thus intensive learning and experience necessary.
• Can (as other process models) be combined with techniques from XP.

Process Metrics

Assessment and Improvement of the Process

• Idea (for material goods): The quality of the (production) process influences product quality.
• Plan: Specify abstract criteria (metrics) to determine good production processes (e.g., to choose manufacturer).

Industry in general (production!):
• ISO 9001, ISO/TS 16949 (automotive), . . .

Software industry (development!):
• CMM(I), SPICE

Note: a good process does not stop us from creating bad products; (the hope is, that) bad products are less likely when using a good process, i.e. that there is a correlation:

<table>
<thead>
<tr>
<th>process quality</th>
<th>product quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>false positive</td>
<td>×</td>
</tr>
<tr>
<td>true positive</td>
<td>×</td>
</tr>
<tr>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>low</td>
<td>false negative</td>
</tr>
<tr>
<td>true negative</td>
<td>×</td>
</tr>
</tbody>
</table>

Improving processes for developing better products and services

1991: Capability Maturity Model (CMM), DoD/SEI/CMU; superseded by
2017: Capability Maturity Model Integration (CMMI) (Team, 2010);
constellations: CMMI-DEV (development), CMMI-ACQ (acquisition), CMMI-SRV (service)

Goals:
• applicable to all organisations which develop software,
• make strengths and weaknesses of the real process visible, to point out ways for improvement,
• neutral wrt. technology employed in project,
• levels: higher levels have lower levels as premise,
• be consistent with ISO 15504 (SPICE)

Assumptions:
• better defined, described, and planned processes have higher maturity,
• higher maturity levels require statistical control to support continuous improvement,
• higher maturity level yields:
  • better time/cost/quality prediction;
  • lower risk to miss project goals;
  • higher quality of products.

CMMI Levels

<table>
<thead>
<tr>
<th>level</th>
<th>level name</th>
<th>process areas</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>initial</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>managed</td>
<td>REQM, PP, PMC, MA, PPQA, CM, SAM</td>
</tr>
<tr>
<td>3</td>
<td>defined</td>
<td>+ RD, TS, PI, VER, VAL, OPF, OPD, OT, IPM, RSKM, DAR</td>
</tr>
<tr>
<td>4</td>
<td>quantitatively managed</td>
<td>+ OPP, QPM</td>
</tr>
<tr>
<td>5</td>
<td>optimising</td>
<td>+ OID, CAR</td>
</tr>
</tbody>
</table>
Scrum proposes methods and approaches.

XP can be tailored in various ways.

Agile approaches may serve as inspiration for, e.g., definition of roles, quite comprehensive.

CMM(I) assumptions are based on experience in specific projects; may not be present for all kinds of software, slightly different vocabulary.

Some criticism points to CMMI. SPICE / ISO 15504 provides "process reference models" (in particular specific ones for automotive, aerospace, etc.) that may guide selection of sub-contractors (a certificate at least proves that they think about their process) by certain (U.S) government customers; required CMMI certificate is "merely for the process", sometimes per level.

CMMI as such may only be required, e.g., for very large companies, for important aspects of process models w.r.t. product quality.

CMM(I) requires a process model (in the sense of the course) is not well-known, very abstract, of limited practical use.

CMMI certificate is similar to CMM(I): maturity levels, assessment, certificates, required termination of iteration on process maturity.

There are examples, but no particular techniques or approaches.

SPICE is a european development: standardised in ISO/IEC 15504 (2003), a process model (in the sense of the course) is not already in level N and not already in level N+1.

Choose levels somewhat arbitrarily: "why is an area in level X while another area is not?" Changed processes may require new (expensive) appraisal, in this sense CMMI certification may hinder innovation, cannot be tailored in various ways. CMMI certification applies to one particular state of process management; may serve as inspiration for, e.g., definition of roles, quite comprehensive.

CMM(I) terminology: the process is not consciously designed, just evolved.

CMMI: Discussion

CMMI: General/Specific Goals and Practices

There are three levels of review methods A, B, C; A is most thorough (and expensive).

Example: CMMI General/Specific Goals and Practices

<table>
<thead>
<tr>
<th>Area</th>
<th>Maturity Level</th>
<th>Goals</th>
<th>Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD</td>
<td>1</td>
<td>1.0</td>
<td>OPP, QPM</td>
</tr>
<tr>
<td>TD</td>
<td>2</td>
<td>2.1</td>
<td>OPP, QPM</td>
</tr>
<tr>
<td>TS</td>
<td>3</td>
<td>3.0</td>
<td>OPP, QPM</td>
</tr>
<tr>
<td>PI</td>
<td>4</td>
<td>4.0</td>
<td>OPP, QPM</td>
</tr>
<tr>
<td>VER</td>
<td>5</td>
<td>5.0</td>
<td>OPP, QPM</td>
</tr>
<tr>
<td>VAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSKM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each area is prescribed to responsible people; each project may have its own process.

Example: CMMI General/Specific Goals and Practices

<table>
<thead>
<tr>
<th>Area</th>
<th>Maturity Level</th>
<th>Goals</th>
<th>Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD</td>
<td>1</td>
<td>1.0</td>
<td>OPP, QPM</td>
</tr>
<tr>
<td>RD</td>
<td>2</td>
<td>2.1</td>
<td>OPP, QPM</td>
</tr>
<tr>
<td>RD</td>
<td>3</td>
<td>3.0</td>
<td>OPP, QPM</td>
</tr>
<tr>
<td>RD</td>
<td>4</td>
<td>4.0</td>
<td>OPP, QPM</td>
</tr>
<tr>
<td>RD</td>
<td>5</td>
<td>5.0</td>
<td>OPP, QPM</td>
</tr>
<tr>
<td>RD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...CMMI: General/Specific Goals and Practices...


