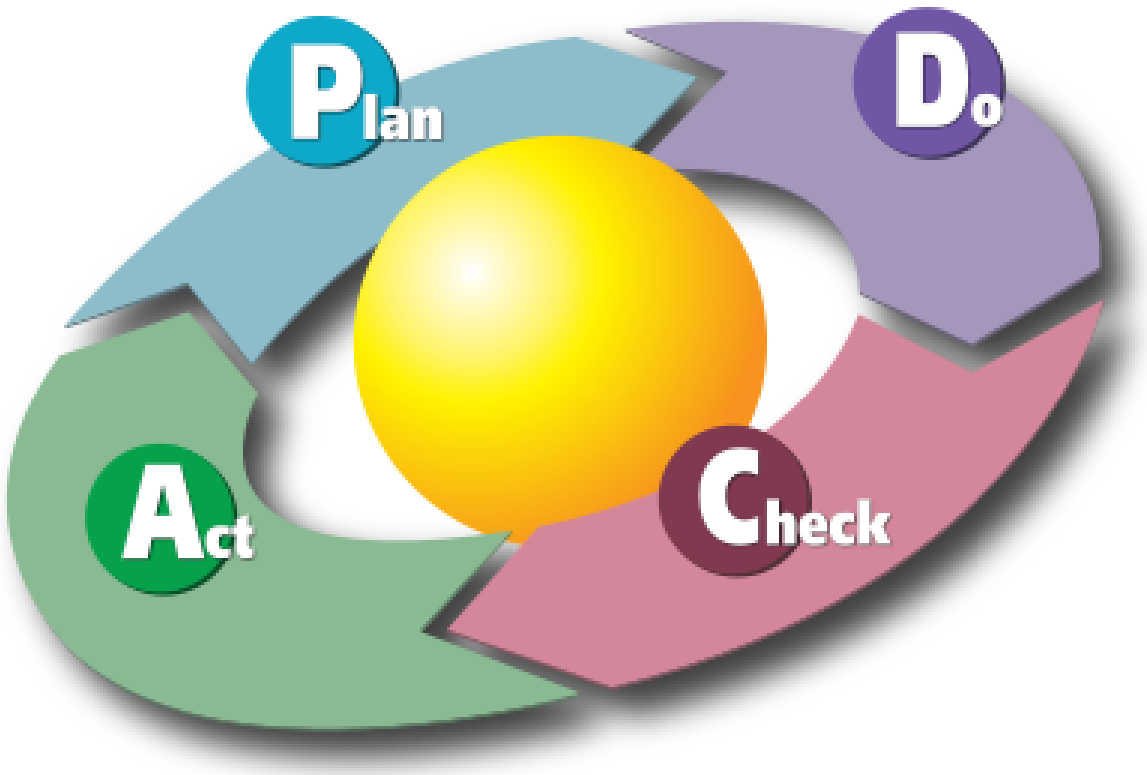
CMMI: Project Monitoring and Control

TECHNISCHE UNIVERSITÄT DARMSTADT



Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

The HOW (part 1): industrial practices, by Dominik Schreiber
- Scrum
- Extreme Programming
- Methodology-independent representations of progress
- Earned-Value Analysis

The HOW (part 2): real-life examples, by Dominik Schreiber

industrial practices

TECHNISCHE UNIVERSITÄT DARMSTADT

Classical approaches

- **waterfall model** does one step after the other
- problem: each step is considered to be **finished** after moving to the next step
- $\Rightarrow$ bad monitoring/control leads to **heavy setbacks**

Agile approaches

- iterations, small releases, . . . should minimize this risk
- setbacks are *possible* but **less heavy**
- [AB06] shows ways to implement CMMI with agile methods

TECHNISCHE UNIVERSITÄT DARMSTADT

Overview

- **agile** software-engineering process
- **iterative**: thinking in *sprints*
- **slim**: 3 *roles*, 4 *artifacts*, small set of *rules*
- **communicative**: daily meetings, planning, reviews (but less paperwork)

Figure : origin of the name “Scrum”

TECHNISCHE UNIVERSITÄT DARMSTADT

Regular meetings

- **Sprint planning meeting** (part 1: whole team):
  - clean product backlog, prioritize entries
  - choose entries for next sprint
- **Sprint planning meeting** (part 2: developers):
  - convert entries to 1-day tasks ($\Rightarrow$ sprint backlog)
  - extract sprint-goal from entries
- **Sprint Review**:
  - present product to product owner, check sprint-goal
  - give feedback for last sprint, update product backlog
- **Sprint Retrospective**:
  - concrete improvements based on
  - feedback for the last sprint

TECHNISCHE UNIVERSITÄT DARMSTADT

Regular meetings

- **Sprint planning meeting** (part 1: whole team): **SP 1.1, 1.2, 1.3**
  - clean product backlog, prioritize entries
  - choose entries for next sprint
- **Sprint planning meeting** (part 2: developers):
  - convert entries to 1-day tasks ($\Rightarrow$ sprint backlog)
  - extract sprint-goal from entries
- **Sprint Review**: **SP 1.5, 1.6, 1.7**
  - present product to product owner, check sprint-goal
  - give feedback for last sprint, update product backlog
- **Sprint Retrospective**: **SP 2.1, 2.2**
  - concrete improvements based on
  - feedback for the last sprint

industrial practices
Scrum

TECHNISCHE UNIVERSITÄT DARMSTADT

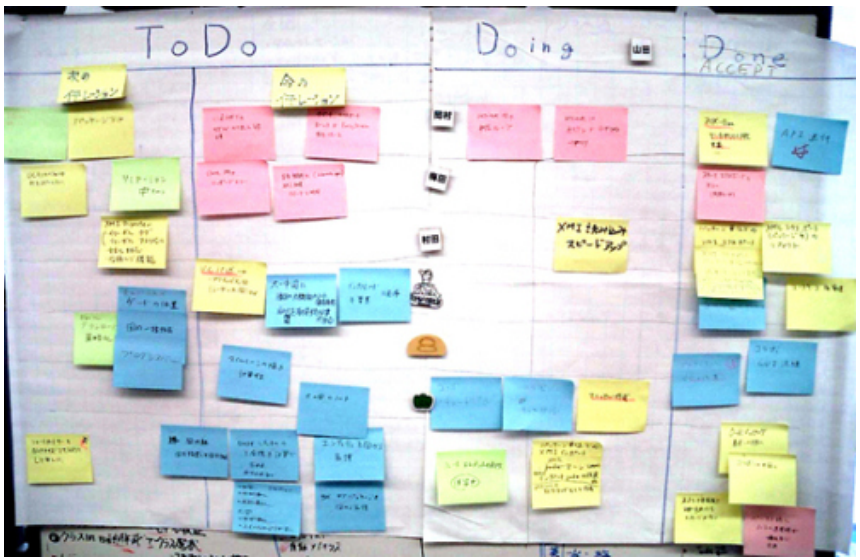


Figure : Scrum Taskboard

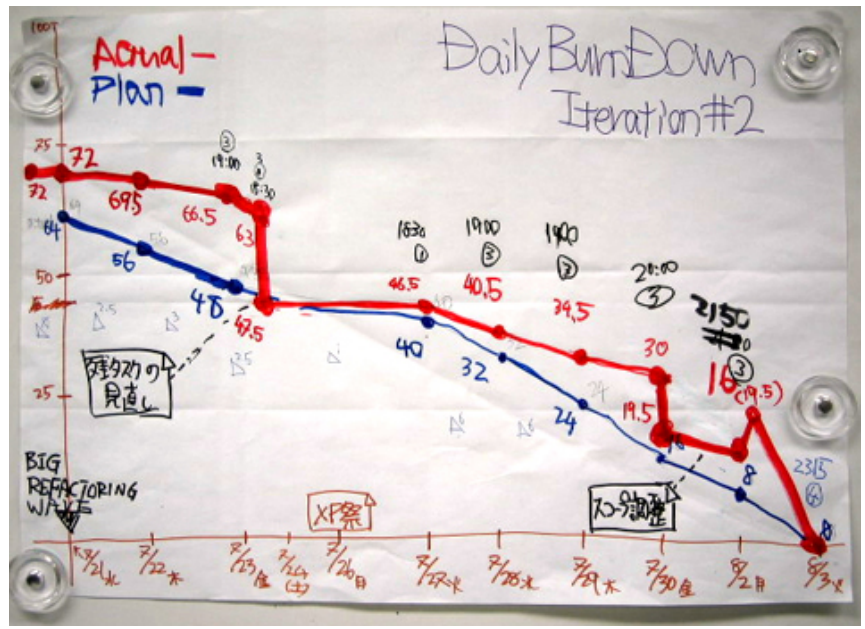


Figure : Scrum Burndown Chart

TECHNISCHE UNIVERSITÄT DARMSTADT

Overview

- **agile** software-engineering process
- strong **principles**: Pair Programming, Test-driven Development, Continuous Integration, ...

industrial practices
Extreme Programming - what it is

TECHNISCHE UNIVERSITÄT DARMSTADT

Overview

- **agile** software-engineering process
- strong **principles**: Pair Programming, Test-driven Development, Continuous Integration, ...

Differences to Scrum

- **iteration length**: week (XP) $\leftrightarrow$ month (Scrum)
- **change adaption**: always (XP) $\leftrightarrow$ not in current sprint (Scrum)
- **work order**: customer chooses (XP) $\leftrightarrow$ team chooses (Scrum)
- **engineering practices**: given (XP) $\leftrightarrow$ not given (Scrum)

industrial practices
Extreme Programming - how it supports PMC

TECHNISCHE UNIVERSITÄT DARMSTADT

Through the engineering process

- **Planning Game**: release+iteration planning match results with plan constantly, split up in 3 phases:

1. *exploration phase*
   create user stories/split them into tasks
2. *commitment phase*
   commit to functionalities/assign tasks
3. *steering phase*
   adjust plan/perform tasks, match result to plan

- **Test-driven Development**: all productive code is written to make failing unit tests pass $\rightarrow$ unit tests describe the plan
- **Continuous Integration**: automated unit tests match every commit to the plan

it is the **combination** of the 12 principles that makes XP work

industrial practices
Extreme Programming - how it supports PMC

TECHNISCHE UNIVERSITÄT DARMSTADT

Through the engineering process

- **Planning Game**: release+iteration planning match results with plan constantly, split up in 3 phases: **SP 1.1, 1.2, 1.3, 1.5, 2.1, 2.2**

1. *exploration phase*
   create user stories/split them into tasks
2. *commitment phase*
   commit to functionalities/assign tasks
3. *steering phase*
   adjust plan/perform tasks, match result to plan

- **Test-driven Development**: all productive code is written to make failing unit tests pass $\rightarrow$ unit tests describe the plan **SP 1.4, 1.6, 1.7**
- **Continuous Integration**: automated unit tests match every commit to the plan **SP 1.6, 1.7**

it is the **combination** of the 12 principles that makes XP work

industrial practices
Methodology-independent representations of progress

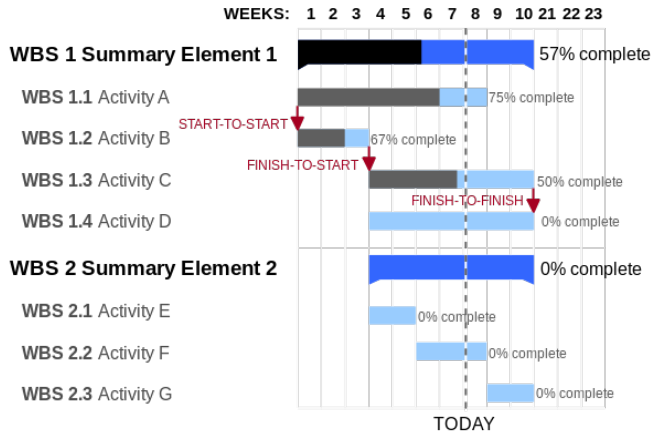
TECHNISCHE UNIVERSITÄT DARMSTADT

Gantt chart

- illustrates project **schedule**
- project is broken down into elements
- each element has **start** and **end**
- progress of each element can be illustrated

Network diagram

- shows **dependencies** in schedule
- same information as gantt chart, different representation



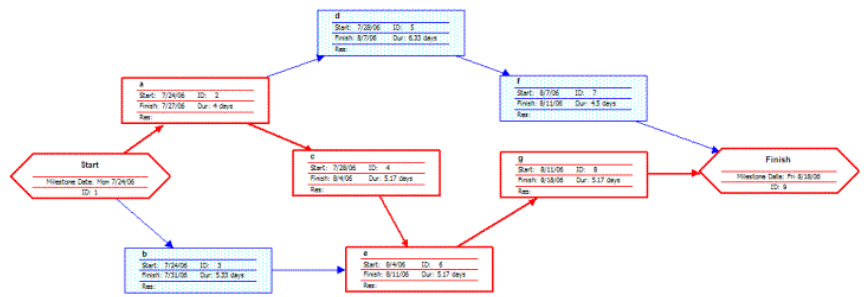


Figure : gantt chart, network diagram

industrial practices
Earned-Value Analysis - what it is

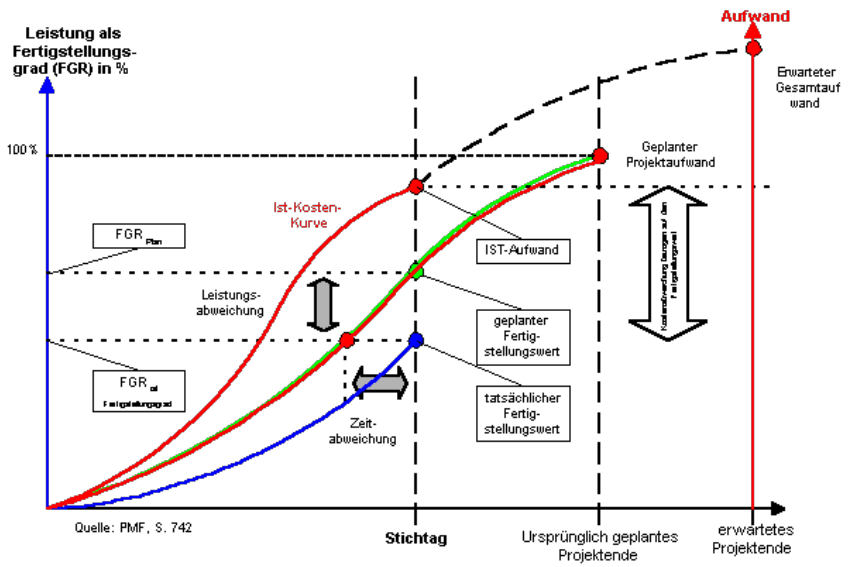


Figure : combined values in EVA

Project controlling tool

- allows **monitoring** of *scope*, *schedule* and *cost* of a project
- allows **forecasts** for the evolving project
- not restricted to software engineering, works for **all kinds of projects**

industrial practices
Earned-Value Analysis - how it works

TECHNISCHE UNIVERSITÄT DARMSTADT

Measured values

- **planned cost** (*PC*): pre-defined cost for the *whole project*
- **actual cost** (*AC*): actual cost *until now*

Metrics

- **earned value** (*EV*): actual *value* of the work until now

  simple metric: $EV = Budget \cdot \%Progress$

Derived values

- **schedule variance**: $SV = EV - PC$
- **cost variance**: $CV = EV - AC$
- **schedule performance index**: $SPI = \frac{EV}{PC}$
- **cost performance index**: $CPI = \frac{EV}{AC}$

industrial practices
Earned-Value Analysis - example

TECHNISCHE UNIVERSITÄT DARMSTADT

Your task

- list of **500** hand-written mail addresses
- insert this into **mailing list**
- you need **2h**, get **€20** (= $10\frac{€}{h}$)
- **half time!** 1h left

Measured values

- **PC** = $20€ \cdot \frac{1}{2} = 10€$
- **AC** = $10\frac{€}{h} \cdot 1h = 10€$

Metrics

- you were **fast**! Did 300 already!
- **EV** = $20€ \cdot \frac{300}{500} = 12€$

Derived values

- **SV** = $12€ - 10€ = 2€$
- **CV** = $12€ - 10€ = 2€$
- **SPI** = $\frac{12€}{10€} = 1.2$
- **CPI** = $\frac{12€}{10€} = 1.2$

industrial practices
Earned-Value Analysis - example

TECHNISCHE UNIVERSITÄT DARMSTADT

Your task

- list of **500** hand-written mail addresses
- insert this into **mailing list**
- you need **2h**, get **€20** (= $10\frac{€}{h}$)
- **half time!** 1h left

Measured values

- **PC** = $20€ \cdot \frac{1}{2} = 10€$
- **AC** = $10\frac{€}{h} \cdot 1h = 10€$

Metrics

- you were **fast**! Did 300 already!
- **EV** = $20€ \cdot \frac{300}{500} = 12€$

Derived values

- **SV** = $12€ - 10€ = 2€$
- **CV** = $12€ - 10€ = 2€$
- **SPI** = $\frac{12€}{10€} = 1.2$
- **CPI** = $\frac{12€}{10€} = 1.2$

If you stay that fast, you'll finish **20min early**. Means: you'll get **€3,30** for free.

Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

The HOW (part 1): industrial practices, by Dominik Schreiber

The HOW (part 2): real-life examples, by Dominik Schreiber
- at openLearnWare
- at dimetis GmbH
- at a major-client software-engineering company

real-life examples
at openLearnWare - project overview

TECHNISCHE UNIVERSITÄT DARMSTADT

Project: lecture material for students

- webservice for lecture material
- development started in spring 2010
- team of 2 full-time employees, 5 HiWis
- scrum-like project structure

Figure : tu-darmstadt.de/olw, 7.1.13

real-life examples
at openLearnWare - project structure

TECHNISCHE UNIVERSITÄT DARMSTADT

Team members

- “Intellectual head” – like Scrum’s **product owner**, responsible for all “non-technical stuff”
- “Technical head” – like Scrum’s **scrum master**, responsible for all “technical stuff”
- 5 HiWis, working 8-20 hours a week – the **scrum team**

real-life examples
at openLearnWare - project monitoring/control

TECHNISCHE UNIVERSITÄT DARMSTADT

Process items

- weekly **scrum meeting** – about an hour, with all team members
- weekly **planning meeting** – about 2 hours, intellectual+technical head
- **taskboard** as a mirror of the redmine *ticket system*
- **tickets** as a *sprint backlog*
- current **QSL-Request** as *product backlog*
- **Jenkins** as *Continuous-Integration Server*

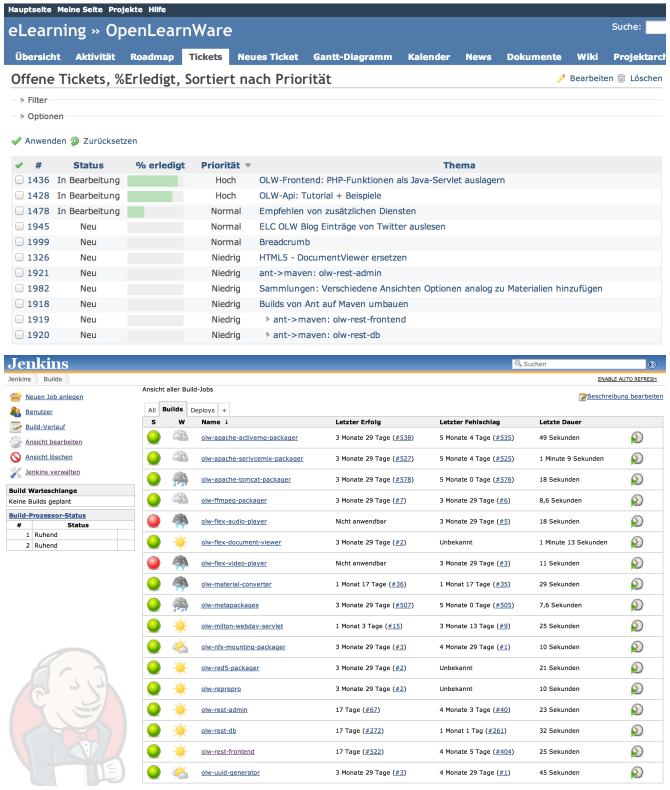


Figure : ticket system, ci-server

real-life examples
at openLearnWare - how the process evolved

TECHNISCHE UNIVERSITÄT DARMSTADT

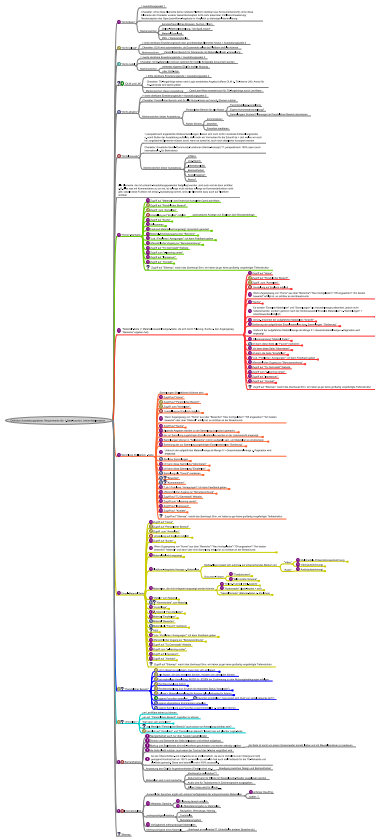


Figure : former product backlog

Change as the only constant

- no current team member from the **founder team**
- began with **giant mind-maps** as product/sprint backlogs
- had 2-3 nearly **complete restarts**
- in the beginning: **no documentation** at all (except backlogs)

real-life examples
at dimetis GmbH

TECHNISCHE UNIVERSITÄT DARMSTADT

Project: broadcasting software for major carriers

- developing with **multiple Scrum teams** on one project
- **adapted Scrum** to fit to multiple teams:
  - **Scrum master**: assigned by head of engineering, could be any developer
  - **architecture**: *raw design* is made before assigning tasks to backlogs, design changes are discussed with all scrum masters (+ overall design documentation)
  - **code ownership**: component/module-based with backup owners, code changes are discussed with owner
  - **assignments**: task→team: close related requirements to the same team, person→team: engineering head+scrum masters assign teams

real-life examples
at dimetis GmbH

TECHNISCHE UNIVERSITÄT DARMSTADT

A single Scrum team

- 2-3 Developers + QA
- 1-5 Sprints per Scrum
- 2-4 Weeks per Sprint
- same svn for all teams, branches for single team

Figure : scrum taskboard at dimetis

Conclusion at dimetis

Scrum has interesting approaches that allow reaching high cmmi-levels in theory*. In practice, the goal of putting as less* overhead *as possible on the developers stands against this.*

real-life examples
at a major-client software-engineering company

TECHNISCHE UNIVERSITÄT DARMSTADT

Project: Web-based software for the government

- up to 1.5 years **specification phase**
- teams of **<10 members**
- project head spends **1 day a week** with the customer

real-life examples
at a major-client software-engineering company

TECHNISCHE UNIVERSITÄT DARMSTADT

The process

- **specification phase**: before the project starts, it gets strongly *specified* (customer reviews this)
- **implementation**: according to system specification, features are implemented
- **internal reviews**: code is reviewed twice: once technical, once specialist
- **small releases**: working features are shipped to the customer early, loaded with unit tests/performance tests/specific tests
- **issue tracking**: customer show occuring bugs to developers or reports them in the issue tracker

Thank you for your attention!

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Thank you for your attention!

TECHNISCHE UNIVERSITÄT DARMSTADT

I want to hear at least **3 questions** from you.

Start now!

Bibliography

TECHNISCHE UNIVERSITÄT DARMSTADT

Julio Ariel Hurtado Alegria and M. Cecilia Bastarrica.
Implementing cmmi using a combination of agile methods.
CLEI Electron. J., 9(1), 2006.

Cmmi Development.
Cmmi® for development, version 1.3 cmmi-dev, v1.3.
Engineering, (November):482, 2010.

More about Extreme Programming:
The principles

TECHNISCHE UNIVERSITÄT DARMSTADT

Fine scale feedback

- pair programming
- planning game
- test-driven development
- whole team

Continuous process

- continuous integration
- refactoring/design improvement
- small releases

Shared understanding

- coding standards
- collective code ownership
- simple design
- system metaphor

Programmer welfare

- sustainable pace