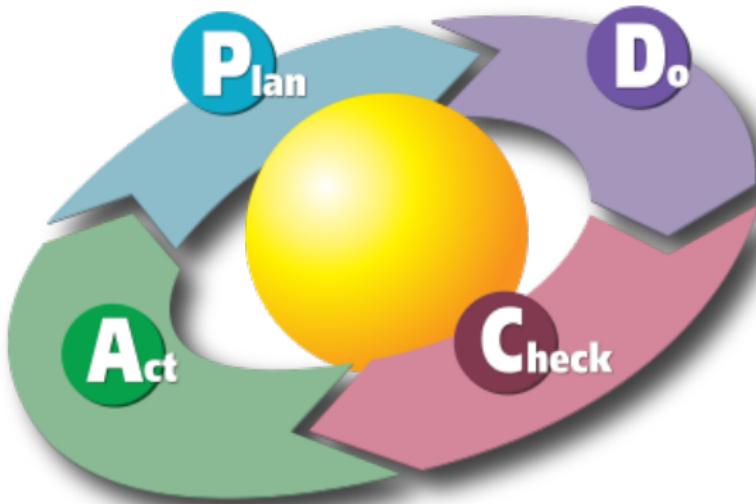


# CMMI: Project Monitoring and Control



# Outline

The WHAT: Project Monitoring and Control, by Tobias Stoll

SG 1: Monitor the Project Against the Plan

SG 2: Manage Corrective Action to Closure

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

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

## Introductory information

- ▶ is at maturity level **2**
- ▶ important to detect deviations
- ▶ corrective actions are taken to fulfill objectives

## Related Process Areas

- ▶ Project Planning
- ▶ Measurement and Analysis

# Project Monitoring and Control

## SG 1: Monitor the Project Against the Plan



### Goal

- ▶ identify **actual** progress and performance against plan
- ▶ **contrast** results with project plan

# SG 1: Monitor the Project Against the Plan

## SP 1.1: Monitor Project Planning Parameters



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- ▶ measure **actual** project planning parameters
- ▶ identify **significant** deviations to the estimates in the plan
- ▶ record results

### Sub-practices

- ▶ Monitor progress against schedule
- ▶ Monitor the projects cost and expended effort
- ▶ Monitor the attributes of the work products and tasks
- ▶ Monitor resources provided and used
- ▶ Monitor the knowledge and skills of the project personnel
- ▶ Document the significant deviations in the project planning parameters

## SG 1: Monitor the Project Against the Plan

### SP 1.2: Monitor Commitments



- ▶ look at commitments
- ▶ compare them to identified commitments in the plan

#### Sub-practices

- ▶ Regularly review commitments
- ▶ Identify commitments that have not been satisfied
- ▶ Document the results of the commitment reviews

## SG 1: Monitor the Project Against the Plan

### SP 1.3: Monitor Project Risks



- ▶ Monitor risks against those in the plan

#### Sub-practices

- ▶ Periodically review the documentation of the risks
- ▶ Revise the documentation of the risks
- ▶ Communicate risk status to relevant stakeholders

# SG 1: Monitor the Project Against the Plan

## SP 1.4: Monitor Data Management

- ▶ check data management activities
- ▶ ensure compliance of data management plans

### Sub-practices

- ▶ Periodically review data management activities
- ▶ Identify and document significant issues and their impacts
- ▶ Document the results of data management activity reviews

## SG 1: Monitor the Project Against the Plan

### SP 1.5: Monitor Stakeholder Involvement



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- ▶ detect involvement of identified stakeholders
- ▶ ensure that the right interactions occur

#### Sub-practices

- ▶ Periodically review the status of stakeholder involvement
- ▶ Identify and document significant issues and their impacts
- ▶ Document the results of the stakeholder involvement status reviews

## SG 1: Monitor the Project Against the Plan

### SP 1.6: Conduct Progress Reviews



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- ▶ project reviews to inform stakeholders
- ▶ may not be specified in project plan

#### Sub-practices

- ▶ Regularly communicate status on assigned activities and work products to relevant stakeholders
- ▶ Review the results of collecting and analyzing measures for controlling the project
- ▶ Identify and document significant issues and deviations from the plan
- ▶ Document change requests and problems identified in any of the work products and processes
- ▶ Document the results of the reviews
- ▶ Track change requests and problem reports to closure

## SG 1: Monitor the Project Against the Plan

### SP 1.7: Conduct Milestone Reviews



- ▶ review of results at selected milestones
- ▶ typically formal and planned

#### Sub-practices

- ▶ Conduct reviews at meaningful points in the projects schedule
- ▶ Review the commitments, plan, status, and risks of the project
- ▶ Identify and document significant issues and their impacts
- ▶ Document the results of the review, action items, and decisions
- ▶ Track action items to closure

# Project Monitoring and Control

## SG 2: Manage Corrective Action to Closure

### Goal

- ▶ react with significant deviations from plan
- ▶ take corrective actions

## SG 2: Manage Corrective Action to Closure

### SP 2.1: Analyze Issues



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- ▶ analyse issues
- ▶ determine necessary corrective actions

#### Sub-practices

- ▶ Gather issues for analysis
- ▶ Analyze issues to determine need for corrective action

- ▶ produce action plan with correctiv actions

#### Sub-practices

- ▶ Determine and document the appropriate actions needed to address the identified issues
- ▶ Review and get agreement with relevant stakeholders on the actions to be taken
- ▶ Negotiate changes to internal and external commitments

- ▶ have a look on the action results

#### Sub-practices

- ▶ Monitor corrective actions for completion
- ▶ Analyze results of corrective actions to determine the effectiveness
- ▶ Determine and document appropriate actions to correct deviations

# Outline

The WHAT: Project Monitoring and Control, by Tobias Stoll

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

## Classical approaches

- ▶ **waterfall model** does one step after the other
- ▶ problem: each step is considered to be **finished** after moving to the next step
- ▶ ⇒ 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

# industrial practices

## Scrum - what it is

### 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”



### 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 (⇒ 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



### 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 (⇒ 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

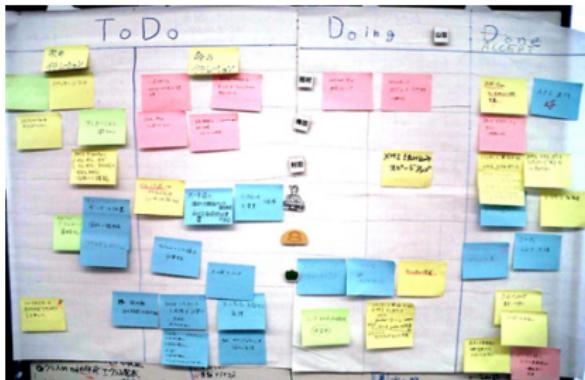


Figure : Scrum Taskboard

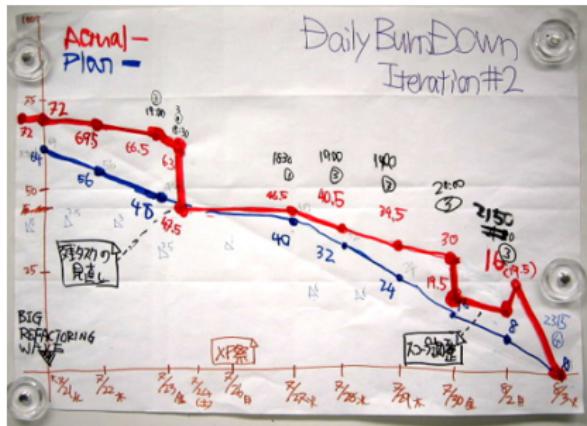


Figure : Scrum Burndown Chart



### Overview

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



### Overview

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

### Differences to Scrum

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



### Through the engineering process

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

|  |  |   |
|--|--|---|
| <p><b>1. exploration phase</b><br/>create user stories/split them into tasks</p> | <p><b>2. commitment phase</b><br/>commit to functionalities/assign tasks</p> | <p><b>3. steering phase</b><br/>adjust plan/perform tasks, match result to plan</p> |
|--|--|---|
- ▶ **Test-driven Development:** all productive code is written to make failing unit tests pass → 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



### 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**

|  |  |   |
|--|--|---|
| <p>1. <i>exploration phase</i><br/>create user stories/split them into tasks</p> | <p>2. <i>commitment phase</i><br/>commit to functionalities/assign tasks</p> | <p>3. <i>steering phase</i><br/>adjust plan/perform tasks, match result to plan</p> |
|--|--|---|
- ▶ **Test-driven Development:** all productive code is written to make failing unit tests pass → 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**

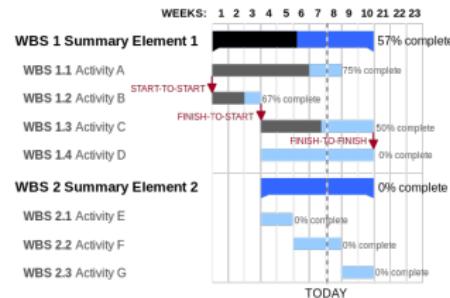
it is the **combination** of the 12 principles that makes XP work

# industrial practices

## Methodology-independent representations of progress

### Gantt chart

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

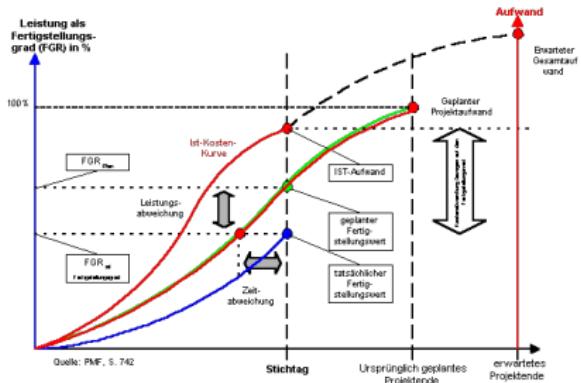


### Network diagram

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



Figure : gantt chart, network diagram



### 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**

Figure : combined values in EVA



### 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:  $EV = \text{Budget} \cdot \% \text{Progress}$

### Derived values

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



### Your task

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

### Measured values

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

### Metrics

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

### Derived values

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



### Your task

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

### Measured values

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

### Metrics

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

### Derived values

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

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

# Outline

The WHAT: Project Monitoring and Control, by Tobias Stoll

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

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

at openLearnWare

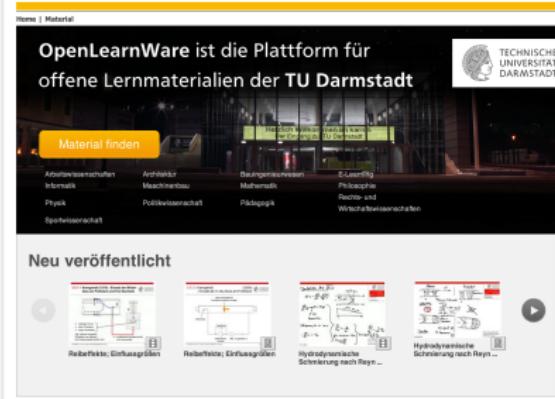
at dimetis GmbH

at a major-client software-engineering company

# real-life examples at openLearnWare - project overview

## 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



The screenshot shows the homepage of the OpenLearnWare platform. At the top, there's a banner with the text "OpenLearnWare ist die Plattform für offene Lernmaterialien der TU Darmstadt". Below the banner, there's a search bar labeled "Material finden". Underneath the search bar, there are several categories: Arbeitswissenschaften, Informatik, Physik, Architektur, Maschinenbau, Politikwissenschaft, Bauingenieurwesen, Mathematik, Pädagogik, e-Learning, Philosophie, Sprache und Wirtschaftswissenschaften. Below these categories, there's a section titled "Neu veröffentlicht" featuring three thumbnail images of recently published materials: "Reibefaktor; Einflussgrößen", "Reibefaktor; Einflussgrößen", and "Hydrodynamische Scherung nach Reay...".

Figure : [tu-darmstadt.de/olw](http://tu-darmstadt.de/olw), 7.1.13

# real-life examples at openLearnWare - project structure

## 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



## 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)

Figure : former product backlog



## 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

## 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

# real-life examples at dimetis GmbH

## 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

## 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

## 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 occurring bugs to developers or reports them in the issue tracker

---

**Thank you for your attention!**

---



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

---

**Thank you for your attention!**



---

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

**Start now!**

# Bibliography

-  **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

## Shared understanding

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

## Continuous process

- ▶ continuous integration
- ▶ refactoring/design improvement
- ▶ small releases

## Programmer welfare

- ▶ sustainable pace