Software Engineering 2
A practical course in software engineering

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science
IV. Working Together

DTU Compute
Department of Applied Mathematics and Computer Science
Working together

- Management
- Process Models
- Version Management Systems
- Collaborative Development Environments
Parts of this course are based on:


IV. Working together

IV.1. Management

DTU Compute
Department of Applied Mathematics and Computer Science
(Software) Management
Planning, organizing, leading, monitoring and controlling the software development process.
Management Goals

Define goals and make sure that they will be achieved

General goals:
- Increase productivity
decrease development costs

- Increase quality
Increase product value

In short:
Make a profit and increase it!
Software Management requires

- short-term
  (within a project or even within a phase)
  and
- long-term
  (spanning more projects)

measures
Long-term measures

Examples:
- Introduction of a new tool
- New development method/technology
- Increase reusability policy

We define “productivity” later in this course more precisely

E.g. Eclipse
E.g. EMF/GMF/ePNK

productivity
measure
time

SE 2 (02162 e16), L04
Short-term measures and long-term measures often have opposite effects:

- short-term measures result in long-term loss in productivity
  (e.g. quick hack for finishing a project; programming language)

- long-term measures result in short-term loss of productivity
  (frustration, learning curve)
Management

- **Planning**
  - set goal
  - set dates
  - define course of action
  - allocate resources
  - ...

- **Organization**
  - assign tasks
  - define organisation structures
  - assign responsibilities
  - ...

Management

- **Leading**
  - lead and motivate team members
  - improve communication
  - solve conflicts
  - ...

- **Monitoring and controlling**
  - check progress
  - identify problems (early)
  - produce relief
  - ...
Management cycle

- Planning
- Organization
- Leading
- Monitoring & controlling
IV. Working together

IV.2. Coordination

DTU Compute
Department of Applied Mathematics and Computer Science
Mechanisms

- Mutual agreement
- Instruction

- Standardization of procedures
- Standardization of the product
- Standardization of the qualification
  the team members

- Socialisation

Works for small groups of people who know each other well (from other projects)
Clear Responsibilities

- Having many discussions is very good

But, they are useless, unless

- the **results** are **put on record**; and
- **problems** are **clearly stated**,
- it is fixed
  - **what** should be done
  - by **whom**
  - by **when**
- and kept **track of**.

Put agenda and minutes for every meeting to the repository (in particular group meetings)
Coordination and leadership require communication
Four sides of a statement
(after Schulz von Thun)

- **The traffic light is green.**
- **I am in the position to tell you how to drive!**
- **It is green down there.**
- **Drive faster so that we don’t miss this green phase!**
- **I am a good driver!**
Four sides of a statement
(after Schulz von Thun)

- request
- relationship
- self-disclosure
- fact
Good Leadership

- instruct team members
- delegate responsibilities
- motivate team members
- coordinate activities
- stir and improve communication
- identify and solve conflicts
- ...

Meta-level discussions (but not too often)!
Rules for Leaders

- sets challenging (but achievable) goals
- assigns tasks and required capabilities for team members
- explains tasks and projects coherently and thoroughly
- defines (checkable) performance criteria
- identifies sources of problems so that team members can adjust

Rules after: Derschka / Balzert II
Rules for Leaders

- expresses more approval than criticism
- supports team members
- communicates high personal expectations in an individual way
- puts emphasis on human relations
- gives team members the chance to find and correct their own mistakes
- includes team members when making decisions
Rules for Leaders

- rewards and conveys team members with respect to innovation and willingness to carry risks
- regularly discusses the performance with his team members
- rewards excellent performance
Rules for Leaders

- Organizes meetings such that cooperation is encouraged
- Shows personal commitment

Makes these rules his/her rules and does not apply them schematically.
IV. Working together
IV.3. Process models

Coordination mechanism: Standardization of processes
Waterfall Model

Does software development really work this way?
Process Models

- **Process models** are the distilled experience of project plans of successful software projects.
- They are idealized and abstract from (many) details.
- This is their strength (and maybe also their weakness).
- A **project plan** is a refined, extended, modified and more concrete process model.

*The waterfall model is extremely simple. Still it has its values!*
Meta model

- Method
  - uses Notation
  - needs Phase
  - used in Role
- Phase
  - needs Document
  - is involved Role
- Document
  - produces State
  - works on Role
- Notation
  - written in Method
  - has State
- Role
  - works on Document
Project plan

- In a project plan phases are split into tasks and task might be further split into sub tasks
- Moreover more concrete information is added:
  - begin / end
  - roles of team members and assignment to concrete tasks
  - effort for each task
  - time spent of each member on the assigned tasks
Project plans

- There are many different notations for such plans

- It is easy to analyse such plans:
  - timing
  - critical paths (buffer)
  - load on team and every team member
  - planning costs
  - controlling
  - …

- Gantt diagrams
- Many tools (MS Project etc.)
Milestones

- in order to monitor the progress, the project plan defines **milestones**
- a milestone is a set of documents that must be provided at a specific time and in a specified state
- typically, the documents at the end of a phase or task or made a milestone
How to choose milestones

- **Verifiable**
  - It must be verifiable whether the milestone is reached or not
    (“some document is 75% finished” is NOT a milestone)

- **Manageable**
  - The required documents can be produced in a reasonable amount of time
    (weeks or months, not years)

- **Regular**
  - The milestones should be in regular intervals
The V-Model (rough idea)

The V-Model is much more detailed. Here the structure is the main issue.
Other models

- Prototype models
- Evolutionary or incremental models
- Spiral model
- ...

Idea:
- Stepwise development of product
- Early prototypes (“executable milestones”)
- “Maintenance as part of the development”

These models have their advantage (dependent on type and size of the project). Sometimes, however, they are used as bad excuse for not planning. (But, that is not the fault of these models.)

In our project:
Technology experiment!
IV. Working together
IV.4. Version Management

DTU Compute
Department of Applied Mathematics and Computer Science
Motivation

Situation:

- Software consists of many different documents (requirements, specification, design, implementation, documentation, handbook, ...)
- Software development is team work

That is the very definition of software.
Consequently:

- Many different people access (and possibly change) the same set of documents
- Often, different people work on the same document concurrently (independently of each other at the same time)
Motivation

**Result:** Typical problems / questions:
- Where is the up to date version?
- Who has the last working version?
- Where is the version from August 10, 2016?
- Who has the version that we presented recently at the meeting with our customer?
Result: Typical problems / questions:

- Who has the documentation that corresponds to the current implementation?
- Who has undone my changes from yesterday? And why?
- All my documents are lost!
- …
Motivation

Simple “Solutions”:
- Shared file system
- Conventions and rules for naming files (e.g. append: version number and date)
- Policies for changing documents
- Appoint persons responsible for documents
Motivation

Simple “solutions” are not:

- conventions will be violated
- coordination is inefficient and might cause long delays
- variants and different configurations need to be managed by hand
- ...

→ mental capacity should not be wasted with these “details”

Conventions need to be enforced!
Version and Configuration Management Systems solve all these problems (almost) without any extra effort

- when applied properly and
- with good policies

They have even some extra benefits (e.g. simple backups).
Version Management Systems

Version management system

documents in hierarchical structure

coordinates accesses and modifications (maintains consistency)

Access, modification, and creation of documents

user

user
Mechanisms

Pessimistic mechanisms:
A document can never be accessed and changed by two different people at the same time (→ locking / checkout / checkin).

Optimistic mechanism:
A document can be changed by different people at the same time; the system detects and integrates the different changes (→ commit / merge / update ).

Problem: This cannot always be done fully automatically.
Optimistic Approach

Repository

Version management system

Local copies of the repository (working copy)
Optimistic Approach

- **An update** updates the user’s working copy with the information from the repository.

- **A commit** brings the changes of the working copy to the repository (with a new version number).

These exist as command line commands as well as in the Eclipse GUI (Team).
Optimistic Approach

- The user can change files in his/her local working copy at any time (no locks).

- The working copy and the repository are synchronized with the commands `update` und `commit` (for all files or selected set of files or a single file).
Problem

- What happens, when a user executes an update and there are changes in the local working copy already?

→ **Merge** of the changes in the repository (since the last update) and the changes in the working copy.
Merge: Scenario 1

Changes in repository
(by a commit of a different user)

Changes in working copy
Merge: Scenario 1

file.txt

Changes in repository

update

file.txt

Updated working copy
Merge: Scenario 2

Changes in repository

Changes in working copy
Merge: Scenario 2

Changes in repository

Changes in working copy

Conflict

update

file.txt

file.txt
Conflicts

- **Conflicts** will be indicated in the files in the working copy:

  <<<<<<

  **Change 1**

  ------

  **Change 2**

  >>>>>>>

- **Conflicts** must be resolved manually by the user in his working copy.
Remarks: Binary Files

- Merges make sense in text files only!
- Binary files should not be merged.
- Which files are binary must be made explicit to the version management system.
- MS Word documents are binary for most version management systems

SVN makes some good guesses!
Problems

- What happens, when the user executes a commit without getting all changes from the repository first?

→ Impossible!!

→ Before committing, a user needs to execute an update (and if necessary resolve the conflict).

→ By this strategy conflicts only occur in working copies!

→ There is always a user responsible for resolving it.
Comparison

Pessimistic approach:
+ no conflicts
-- no concurrent work on the same object
(for long files or „forgotten“ check-ins this is very problematic)

Optimistic Approach:
- conflicts (fortunately very rare)
++ concurrent work possible
(the last one needs to clean up the mess)
Comparison

For typical software projects (big teams working concurrently), optimistic consistency mechanisms have turned out to be more appropriate:

- concurrent work
- in combination with responsibilities, conflicts are rare
Typical scenario

Note: On the following slides, the command line commands are stated! In eclipse, this (and much more) can be done by the "Team menu"
Typical scenario
Typical scenario

> commit
up-to-date
check failed

In Git: Repositories are local → “commit” would be a commit + a push in Git
Typical scenario

> commit
up-to-date
check failed

> update
M file.txt

Version management system

> commit

In Git: “update” would be “fetch + pull”
Typical scenario

> commit
up-to-date check failed
> update
M file.txt
> commit
• every **commit** automatically creates a new version of the file with a new version number
• we can retrieve earlier versions of a file and go back to earlier versions
• we can compare different versions of a file
• we can define different configurations (branches) of the same project
• we can define releases (a set of related versions of files)
Configurations and releases

Configuration: today (head) release

V1.1
  └── V1

V1.2
  └── V1

V1.3
  └── V1

V1.4
  └── V1

Main.java

Appl.java

index.html

main.html
More Concepts

- every commit can **(and should)** be accompanied by a brief comment what changes were made (and why)
- Users can automatically be notified by changes
- Change history shows who made which changes at which time (for a single file or a complete project).
More Concepts

- you can “tag” some versions (in SVN this is done by “SVN copying” a file to a specific “tags” directory

- …
Guideline

Which documents should be in a repository?

All documents and files that belong to the software and the development process which cannot be automatically reproduced from other files or documents (by all other users).

Remember: Software is more than the code. Also documents should be in the repository!
- A decentralized version of a repositories
- With focus on
  - Commit objects:
    a set of files with changes with respect to parent commit objects
  - Heads:
    a reference to a commit object
- Local repository:
  - Commit, checkout, merge/pull, branch
- Remote repositories:
  - Remote repository reference (origin)
  - Clone, tracking, fetch, pull, push

Commit is local!
Git: Links

Git Concepts
- https://www.sbf5.com/~cduan/technical/git/

Practical use:
- GitHub: https://guides.github.com/activities/hello-world
IV. Working together
IV.5. Collaborative Development Environments (CDE)

DTU Compute
Department of Applied Mathematics and Computer Science
Streamline communication among different stakeholders in a development process – often web interface on top of a version management system

Example: Redmine, gforge, …
Concepts

- Users
- Roles (responsibilities, skills)
- Tasks (development, bug fix, …)
- Task tracking (live cycle of a task)
- Communication (smoothly integrated)
  - notifications
  - more or less structured discussions (wiki, news, email, …)

Note: This year, we do not use a explicit CDE (Redmine). But, GitHub provides some minimal CDE functionality.