Software Engineering I (02161)

Week 8

Assoc. Prof. Hubert Baumeister

Informatics and Mathematical Modelling
Technical University of Denmark

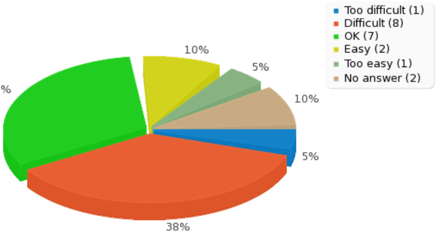
Spring 2011

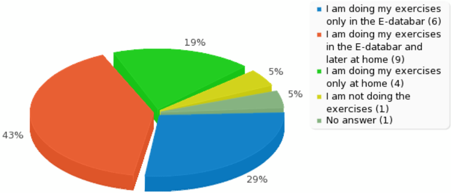
Recap

- Layered Architecture
  - Persistent Layer
- Architecture
  - Defines the structure the system
  - Architectural patterns: Model View Controller (MVC), Layered Architecture, Client Server, Repository, Pipe and Filter
- Good Design Principles:
  - Don’t repeat yourself (DRY), Keep it short and simple (KISS), (Low cohesion / high coupling)
- Design Patterns (I):
  - (Object-oriented) solutions to common design problems
  - Template Method, Observer Pattern, (State Pattern), Composite Pattern

Result of the survey

- 21 have answered (from 113) = 18%
  - 13 Bachelor of Software Technology, 5 Bachelor of IT & Communication, 2 Other, 1 No answer
- Average time for the exercises 3.44h





Topics

- Project planning
  - How to split up a programming assignment
- The use of version control, ie SVN, CVS or similar
- Design Patterns
- Project management
- Refactoring - how do we apply the patterns and principles of good design to turn not so good code into better code.
- Extreme Programming
- Use cases
- (More on Software architecture)
- (Model-View-ViewModel)
- (Entity Systems)

Project plan

- Defines how the work will be done
  - Break down the work into activities and assign these to project team members
  - Estimate how long each activity will take
- Created at the start of a project but should be adapted in course of the project
- Helps asses progress on the project

Project planning happens at

Project planning happens at:

a) Proposal stage
   - Defines the resources (and thus the price) for a project

b) During the project startup phase
   - Contains project monitoring mechanism (also required in the proposal state; e.g. with EU projects)

c) Periodically throughout the project
   - As more information is available, the plan can be more detailed

Software pricing

Factors that influence the price

1) Effort:
    - How much time will it take to create the project (usually measured in person months or person years (i.e. how many months it will take one person to do the task)
2) Travel
3) Hardware costs / Software license costs
4) Overhead
    - Costs related to running costs: (i.e. buildings, secrataries, electricity, time one is not working ...)
    - Can be 80% — 180% of the other costs
5) Competition
    - e.g. reduce the price to win a competition to, e.g., enter a market
6) Other business factors (e.g. need to employ people until the next big project)

Classical project planning

- e.g. for waterfall, iterative development
- based on engineering project management

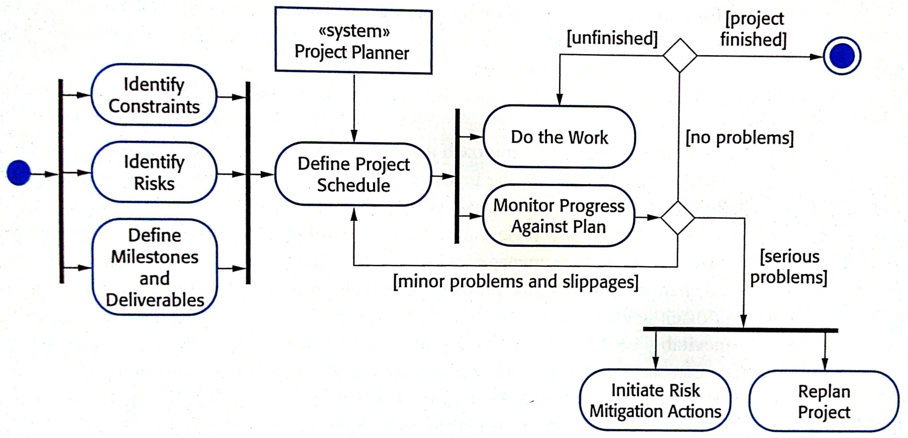
→ try to plan as much upfront:

  - what activities/tasks to do
  - who is doing which activity
  - milestones/deliverables

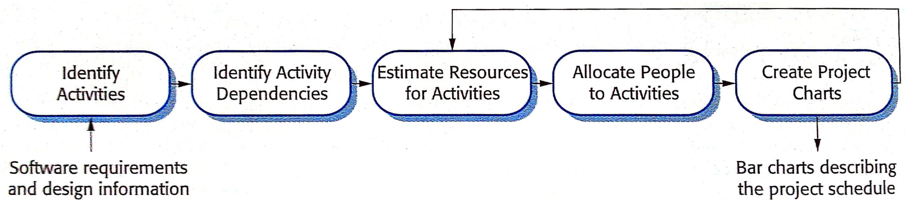
Structure of a project plan

- Introduction
- Project organization
- Risk analysis
- Hardware and software resource requirements
- Work breakdown
- Project schedule
- Monitoring and reporting mechanism

Process planning and executing



Project scheduling


Identify Activities
Identify Activity Dependencies
Estimate Resources for Activities
Allocate People to Activities
Create Project Charts
Software requirements and design information
Bar charts describing the project schedule

- Work breakdown in task
  - Task duration 1 week – 10 weeks

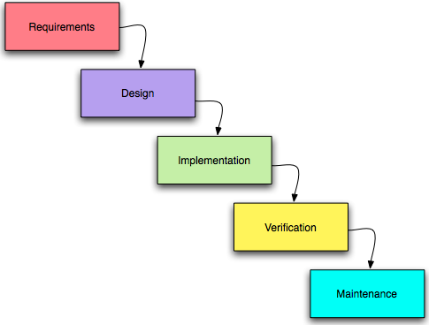
→ Take into account that something can go wrong

→ Contigency estimates may add 30% – 50%

- Work breakdown depends on the software development process
  - e.g. Waterfall: Analysis, Design, Implementation, Test
  - e.g. Iterative: Inception phase, elaboration phase, construction phase, maintanance phase

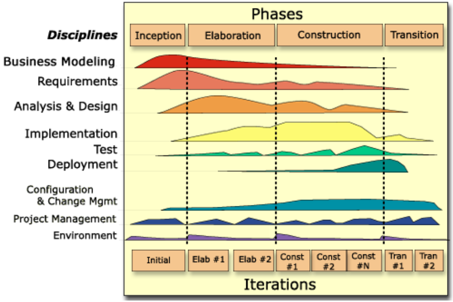
Processes

- Waterfall



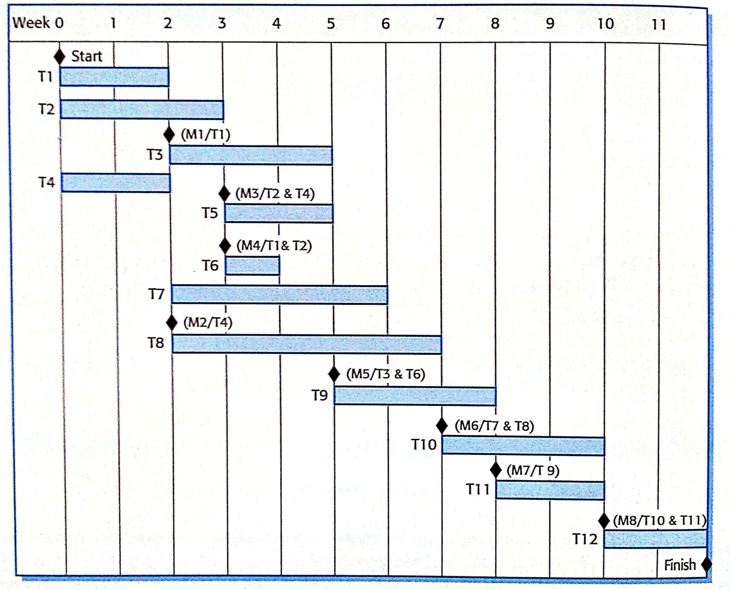
- Typical milstones/deliverables: system specification, design specification, . . .
- Typical tasks: based on the architecture components / functionality

- Iterative Development (e.g. RUP)



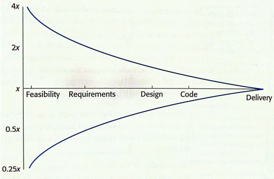
- Milestones/deliverables: At the end of each phase; decides if one wants to proceed to the next phase
- Typical taks: based on the architecture components / functionality

Schedule Representation: Gantt Chart / Bar chart



Project estimation techniques

- To determine the costs and resources for a project, the effort (e.g. in Person Months (PM) or Person Years (PY)) needs to be estimated as well as the time the project will take, and the number of staff



- Experienced based
  - Classical: estimates are based on the experience of the manager

  → e.g. XP bases its estimation on points or ideal person days estimated by the developer
- Algorithmic based
  - e.g. COCOMO, Intermediate COCOMO, COCOMO II

Algorithmic cost modeling: COCOMO

Constructive Cost Model (COCOMO) by Bary Boehm et al., 1981

- Based on empirical investigation of classical projects (later adapted to more modern processes)
- Effort: in person months: $PM = a * size^b$
  - based on expected size of the program (in 1000 lines of source code instructions) (original COCOMO model)
  - Constants $2.4 \leq a \leq 3.6$ (depend on organization and type of software to be developed and on cost drivers , e.g., dependebility req., platform difficulty, team experience, . . .), $1 \leq b \leq 1.5$
  - More complex model is COCOMO II which includes different effort computations depending on the phase of the model

Project time and staffing

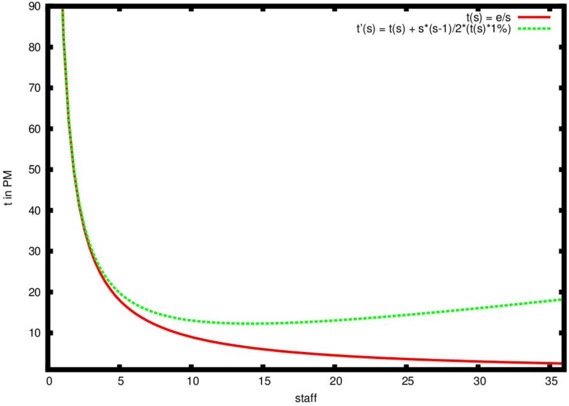
- Number of staff: $STAFF = PM/TDEV$
- Project duration: $TDEV = 3 * PM^{0.33+0.2*(B-1.01)}$
  - More staff does not necessarily mean that the project can be finished earlier
  - Basic idea: more staff needs additional communication: $\sum_{i=1}^{s-1} i = s(s-1)/2$

Brooks's Law

Brooks's Law

"...adding manpower to a late software project makes it later."

Fred Brooks: The Mythical Man-Month: Essays on Software Engineering, 1975



- Assume effort $e = 90PM$
- Lines of code ($22,000SLOC$) ($a = 2.4$, $b = 1.17$)
- How does the development time depend on the number of developers?

a) $t(s) = eff/s$

b) $t'(s) = t(s) + s(s-1)/2 \times 1\% t(s)$ Overhead based on 1% of the development time is devoted to talk to 1 other developer (simplified model)

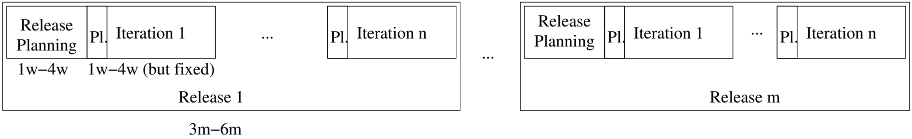
$TDEV = 3 * eff^{0.33+0.2*(b-1.01)} = 15m$ ($b = 1.17$)

Summary COCOMO

- While COCOMO has been refined to take into account new methodologies and technologies, still
  - High dependency on methods, organization, project type, programming languages, . . . to define the constants
  - What is the size of a project? Depending on the programming language, the number of source code instructions is very different
- Works after one has defined the requirements and done a first system design
  - With waterfall, finding the system requirements is expensive
  - Not clear how to use with agile software development methods because they refine the requirements during the development

Planning Agile Projects

- Aglie projects (here XP projects) are based on fixed general structure of the project: releases with iterations



- Releases have to make (business) sense as a whole; i.e. can be used by the customer or be sold
  - Defined by a set of user stories
  - Ideal: Short releases: 3 – 6 months
- Releases are structured in iterations (1 – 4 weeks but fixed)
  - Contains a set of user stories to implemented

  → time boxing: release dates and end of iteration dates are fixed; what can change is planned functionality (e.g. the number of user stories)
- Two planning phases: Release planning and iteration planning

Planning game

- Release planning:
  1) Define user stories that make up an iteration
  2) Assign user stories to iterations
  - Two clear distinguished roles
  a) Customer defines which user story goes into which iteration based on business value, risk, and costs (i.e. development effort)
  b) Developer owns the estimates for a user story (the user cannot tell the developer when the story should be done)
    - Ignore dependencies between user stories

    → estimates in ideal man days/weeks or story points (gut feeling based on experience)
  - After each iteration, the plan and progress are evaluated and possibly adjusted
- Iteration planning (at the start of each iteration):
  - Define tasks for user stories of the iteration
  - Programmers commit to tasks

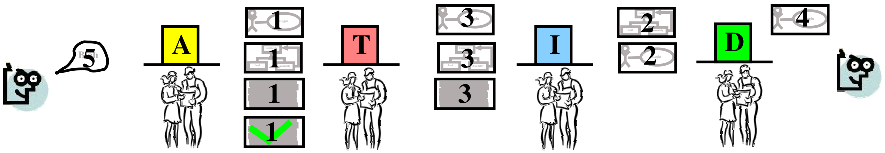
Project estimation and monitoring

- Two possibilities
  1) Estimate the ideal time (e.g. person days/weeks) needed to implement the feature. To get the real time multiply this with a load factor (e.g. 2)
  2) Estimate the relative difficulty among user stories: e.g. user story 1 is more difficutl than user story 2 and assign (arbitrary) points to the
- Progress is monitored by how many user stories are completed
  1) Defines the load factor (*lf*) (e.g. $lf = total\_iteration\_time / user\_story\_time\_finished$)
  2) Defines velocity: Number of points per iteration
  - If in trouble: Focus on few stories that can be finished instead of having many unfinished stories

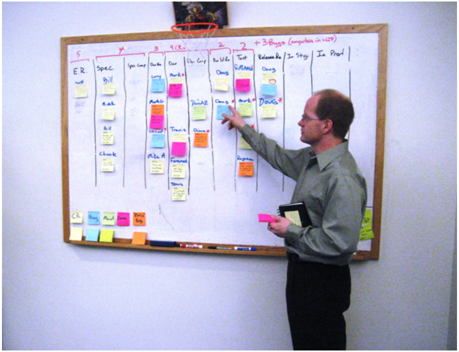
  → Don't let deadlines slip (time boxing)
- *Yesterdays weather*: For the planning of the iteration use the load factor / velocity of the only the previous iteration

Kanban for Project Monitoring

Work Item	A		D		I		T		Done
	Queue	WIP	Queue	WIP	Queue	WIP	Queue	WIP	
		5		4		2		3	1



Kanban for Project Monitoring



- Process controlling through local rules
- Problems in the process (like blockages or empty activities) can be easily seen on the kanban board and fixed (by process adaptation or help from others)

What is version control?

Version Control

”Revision control (also known as version control, source control or (source) code management (SCM)) is the management of multiple revisions of the same unit of information” Wikipedia

- Stores versions of a file (e.g. a source file)
- Allows to retrieve old versions
- Allows to compare different versions
- Allows to merge different versions (e.g. make one file from two different versions of a file)

→ Is used in projects to

- for concurrent development of software

  → each programmer works on his version of the file: The results need to be merged

CVS

CVS

Concurrent Versions System

- Originally a set of command line tools
  → But there exist ”nicer” interfaces: e.g. Eclipse
- A set of files and each file has a tree of ”versions”
  - In principle each file is treated separately from each other
  → use tagging to indicate that a set of files belong together to, e.g. form a version/release of a software package
  → branching allows to have parallel versions
- Implemented by storing the differences between the file versions (and not whole files)
- CVS stores its file in a central repository

What are the use cases of version control / CVS?

- Creating a CVS repository
- Creating a project within a CVS repository
- Checking out a project from a CVS repository
- Updating a file from a CVS repository
  - Comparing with previous versions
  - Automatically merging changes (note: only files with the ASCII attribute can be merged automatically)
- Committing changes

  → fails if someone has changed the repository file

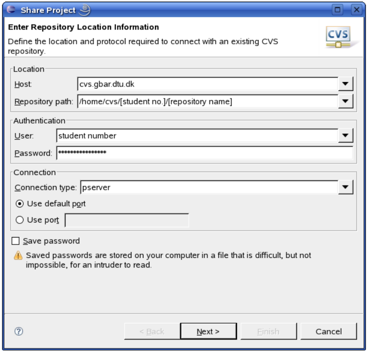
  → requires to to an update, fixing all the conflicts, and then committing again
- Tagging versions
- Branching a version
- Merging a branch

Creating a repository

1. Go to `http://cvs.gbar.dtu.dk`
2. Login using students number and password.
3. Select ”create new repository”
4. Choose a name, eg. 02161
5. Click on the newly generated repository and add the other student numbers from the group with the button ”Add CVS user from DTU”

Creating a project within a CVS repository

- From within Eclipse, select a project in the package explorer and then choose Team→share project and `create a new repository location`
- Fill out the form



Enter Repository Location Information

Define the location and protocol required to connect with an existing CVS repository.

Location

Host: cvs.gbar.dtu.dk

Repository path: /home/cvs/[student no.]/[repository name]

Authentication

User: student number

Password: ****************

Connection

Connection type: pserver

● Use default port

○ Use port

□ Save password

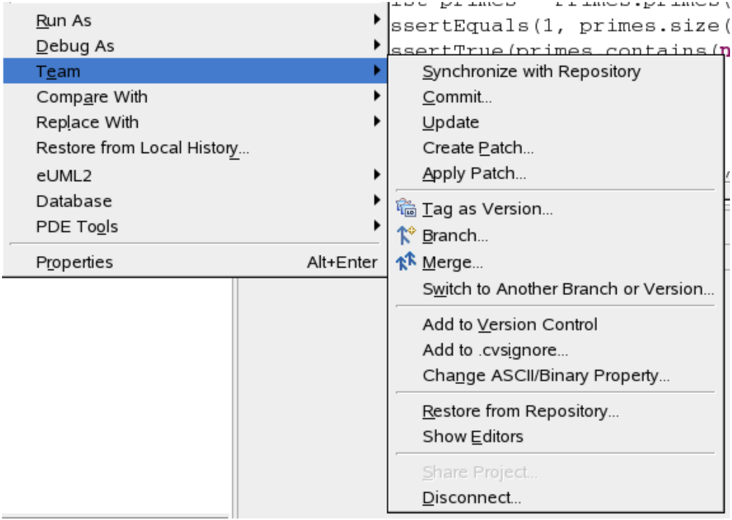
⚠ Saved passwords are stored on your computer in a file that is difficult, but not impossible, for an intruder to read.

- Click next, mark ”Use project name as module name”, click next and finish

Checking out a project from a CVS repository

- Open the ”CVS Repository Exploring” perspective (Window→open perspective→other
- If not present, create a new repository location selecting new→repository location in the right button menu
- Open the repository location and then HEAD to get to the projects for that location (use Branches and Versions to get to project branches and project versions)
- Right click and then check out the project. You can use as project name a new name or the name of the project in the CVS repository

Package Explorer Team Menu Project



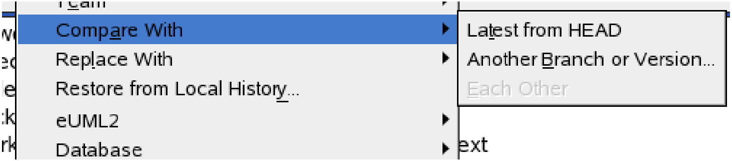
```
ssertEquals(1, primes.size(
ssertTrue(primes.contains(n
```

Update a project from a CVS repository

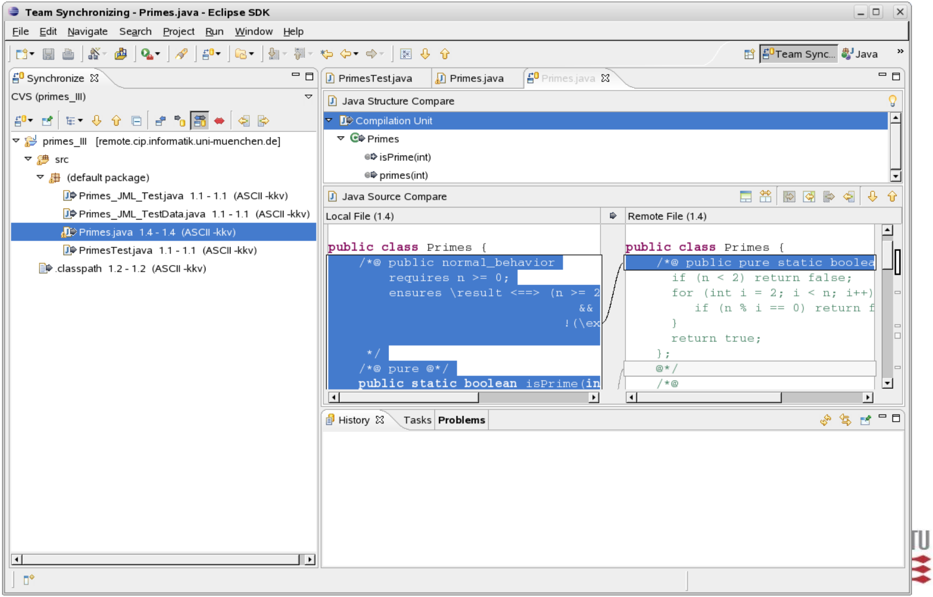
Copies all the changes which are in the repository to the current version of the local files

- If the local files have not been modified after the last update / check out, the local files are overwritten
- If the local files are modified, then they are merged
  - → Merging happens only for files marked with the ASCII property; Other files will be overwritten and the local files will be copied to a different name
    - Use the team menu to change the ASCII/Binary property
  - → Merging might fail. Then the local file will contain both versions, the repository and the local version
    - → Use the compare with menu to check for conflicts

Package Explorer Compare With Menu



Compare result: Compare with latest from HEAD



- primes_III [remote.cip.informatik.uni-muenchen.de]
  - src
    - (default package)
      - Primes_JML_Test.java 1.1 - 1.1 (ASCII -kkv)
      - Primes_JML_TestData.java 1.1 - 1.1 (ASCII -kkv)
      - Primes.java 1.4 - 1.4 (ASCII -kkv)
      - PrimesTest.java 1.1 - 1.1 (ASCII -kkv)
  - .classpath 1.2 - 1.2 (ASCII -kkv)

- Compilation Unit
  - Primes
    - isPrime(int)
    - primes(int)

Local File (1.4)

```
public class Primes {
    /*@ public normal_behavior
        requires n >= 0;
        ensures \result <==> (n >= 2
                                  &&
                                 !(\ex

      */
    /*@ pure @*/
    public static boolean isPrime(in
```

Remote File (1.4)

```
public class Primes {
    /*@ public pure static boolea
      if (n < 2) return false;
      for (int i = 2; i < n; i++)
         if (n % i == 0) return f
      }
      return true;
    };
    @*/
    /*@
```

Committing changes to a CVS repository

- Use commit from the team menu
- You are required to give a comment
- Commit fails if someone else committed changes after your last update
  - → Resolve this by updating, repairing any conflicts, and then committing again
  - A good idea is to do an update before each commit

Steps in Developing a Program using CVS

1 Create Repository

2 Create a project within a repository

3 For all the programming tasks in an iteration

3.1 Update the files / directory you will be working on

3.2 Work on the implementation so that all tests run

3.3 Commit your changes

3.3.1 Update the project

3.3.2 Fix all compile time errors and all broken tests;
If fixing took longer, repeat from step 3.3.1

3.3.3 Commit your changes

4 Tag you files for major project milestones

Introduction to the project

- What is the problem?
  - Project planning and time recording system
- What is the task?
  - Create a
    - Project plan
    - Requirement specification
    - Programdesign
    - Implementation
    - Tests
- Deliver a
  - report describing the requirement specification, design, and implementation (as a paper copy and PDF uploaded to CampusNet)
  - an Eclipse/NetBeans project containing the source code, the tests, and the running program (uploaded to CampusNet as a ZIP file)

Organisational issues

- Groups with 2, 3, or 4 students
- Report can be written in Danish or English
- Program written in Java and tests use JUnit
- On Monday, May 9 there will be a short (10min) demonstration of the program in the E-databar
  - → At least the tests need to be demonstrated
- Report and Eclipse/NetBeans project is to be delivered and uploaded during the demonstrations on May 9
- Each section, diagram, etc. should name the author who made the section, diagram, etc.

Organisational issues

- **You can talk with other groups (or previous students that have taken the course) on the assignment, but it is not allowed to copy from others parts of the report or the program.**
  - *Any text copy without naming the sources is viewed as cheating*
- There will be a CampusNet group created for each project group
- Latest Friday 26.3 18:00 each project group has to have put the project plan on the CampusNet

Exercises

- Exercises will continue after the lectures
  - for technical help and questions regarding the project description
- Lectures continue until before Easter
- Exercise after Easter: 13:00-15:00
- In case of questions with the project description send email to `hub@imm.dtu.dk`

Planning your project

- Questions to be answered by the planning process
  - How many person hours does a project need
  - How much time does a project need
  - What are the additional resources: e.g. hardware, software, person with certain qualifications (e.g. graphic designer, . . .)
  - When to do what
- Base for the planning process
  - Overview over the functional requirements: Use cases more or less detailed described
  - Overview over the intended architecture: e.g. Web application, stand-alone application etc.
- In your case: resources are fixed; adjust the functionality of the system

  → When to do what

Techniques for planning your project 1

- Step 1 Determine a set of scenarios (e.g. based on Use Case scenarios) that your system should be able to do
  - Do a brain storming on the requirements (use cases)
    - What are the scenarios? (success, failure, . . .)
    - Is the set of use cases complete?
  - Include *user stories* for writing the report
    - E.g. Drawing class diagram
    - Documenting use cases
    - Sequence diagrams
    - Writing introduction
    - ...
- Step 2 Do a brain storming on the intended architecture of the system (usually, the customer has some requirements here: e.g. implemented as a Web application . . .
  - Only a rough idea is needed

Techniques for planning your project 2

- Step 3 Estimate the Use Case Scenarios
  - How long, in ideal man hours, do you think you need for implementing the use case scenario?
  - Multiply this with a load factor of 2 to get the real man hours
  - This estimation includes
    - Design
    - Define the detailed scenarios
    - Implementation
    - Testing
    - . . .

Techniques for planning your project 3

- Step 4: Count how many resources you have:
  - E.g. 5 weeks * 8 h = 40 person hours per person times
  - 2—4 persons corresponnds to 80—160 person hours per team
- Step 5 Order the use case scenarios by their value to the customer (In real life this is something the customer needs to do!!!)
  → Add the time for the scenarios until the time reaches the available time
- The result is an initial plan
  → The plan needs to be updated as the project proceeds

Techniques for planning your project: Remarks

- The planning should include the writing of the report!
- Plan need not be perfect!
  - Don't spent too much time
  - Experience with the problem and its implementation changes the plan
  - Plan needs to be updated every iteration