Software Engineering I (02161)

Week 9

Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2015

Last Week

- Software Development Process
- Version Control

Contents

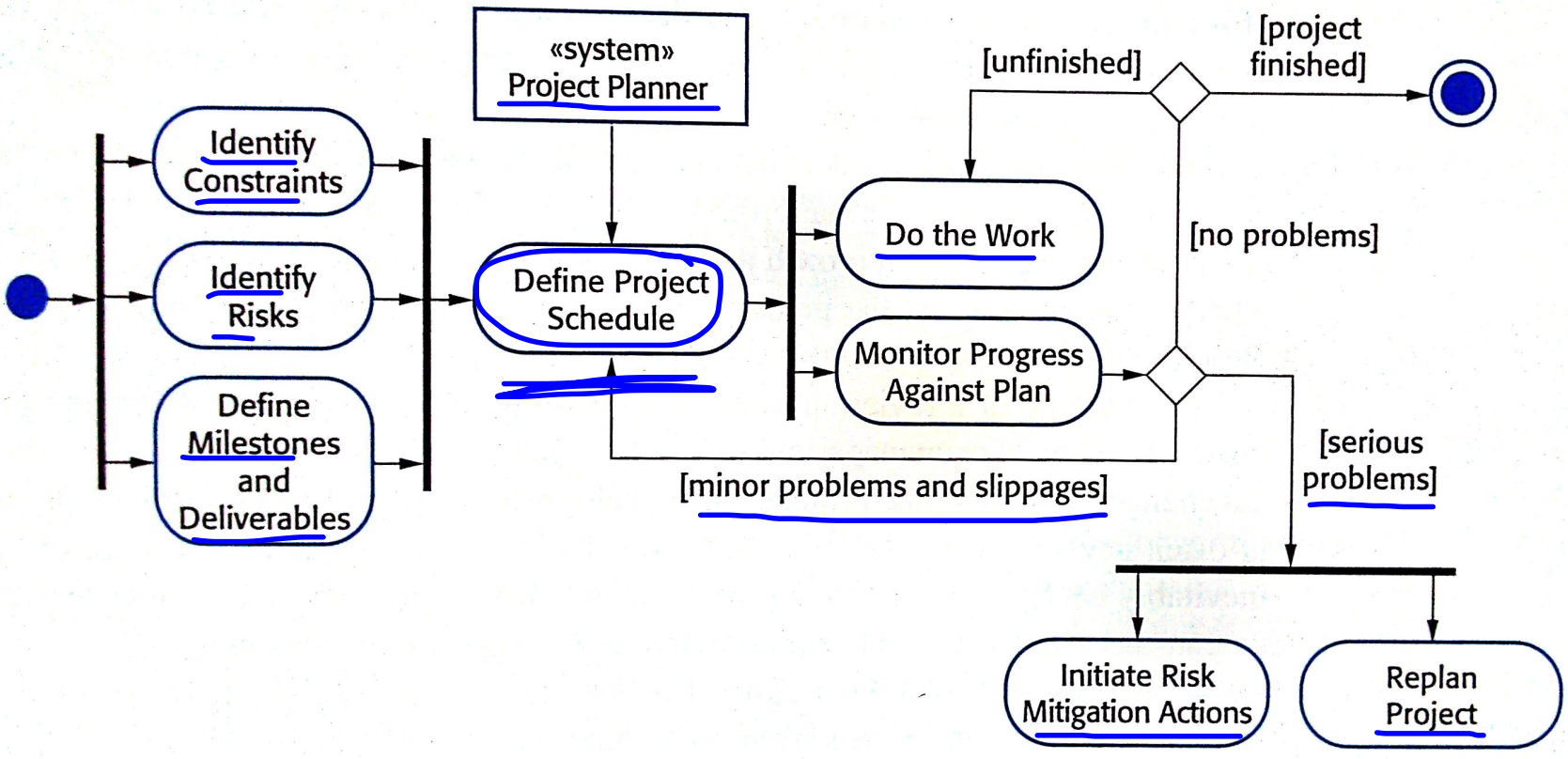
Project Planning

- Project plan
  - Defines:
    - How work is done
  - Estimate
    - Duration of work
    - Needed resources
    - → Price
- Project planning
  - Proposal stage
    - → Price
    - → Time to finish
  - Project start-up
    - → Staffing, ...
  - During the project
    - Progress (tracking)
    - Adapt to changes

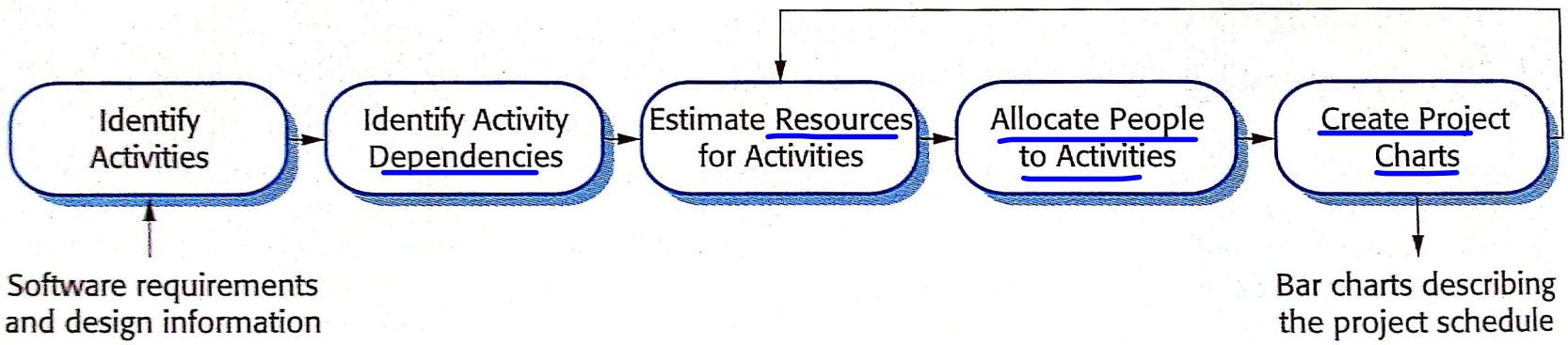
Software pricing factors

- Direct costs
  - Human Resources, consultants, . . .
  - Hardware costs / Software license costs
- Indirect costs / overhead:
  - Running costs: buildings, electricity, . . .
  - 80%— 100% of other costs
- Other factors
  - Competition, . . .

Process planning and executing

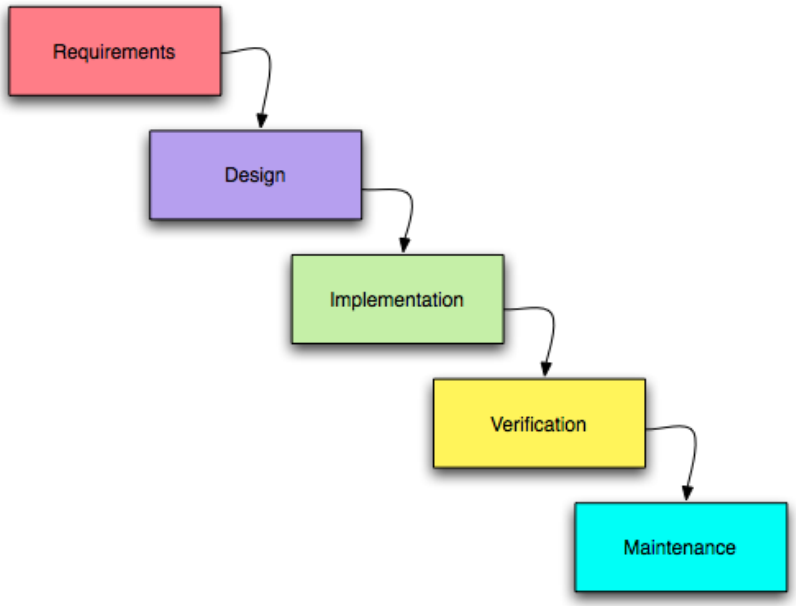


Project scheduling



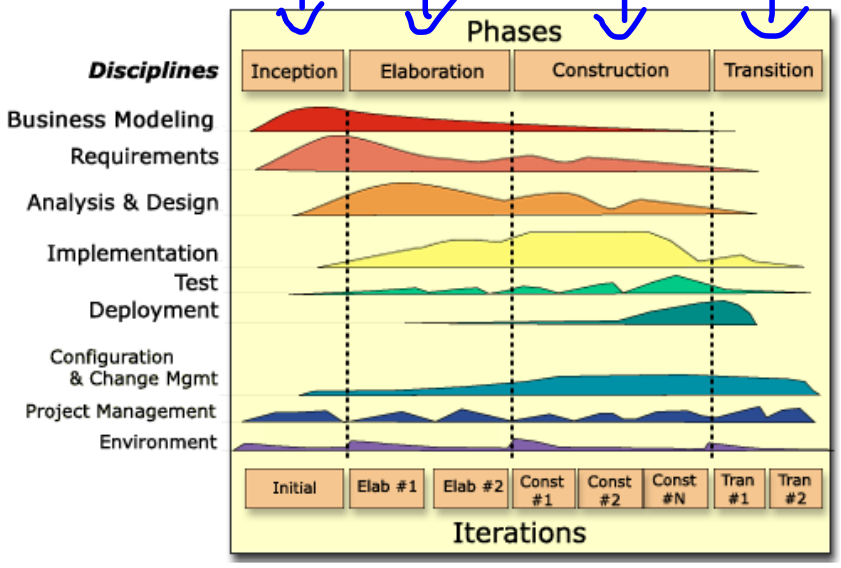
Traditional Processes

- Waterfall



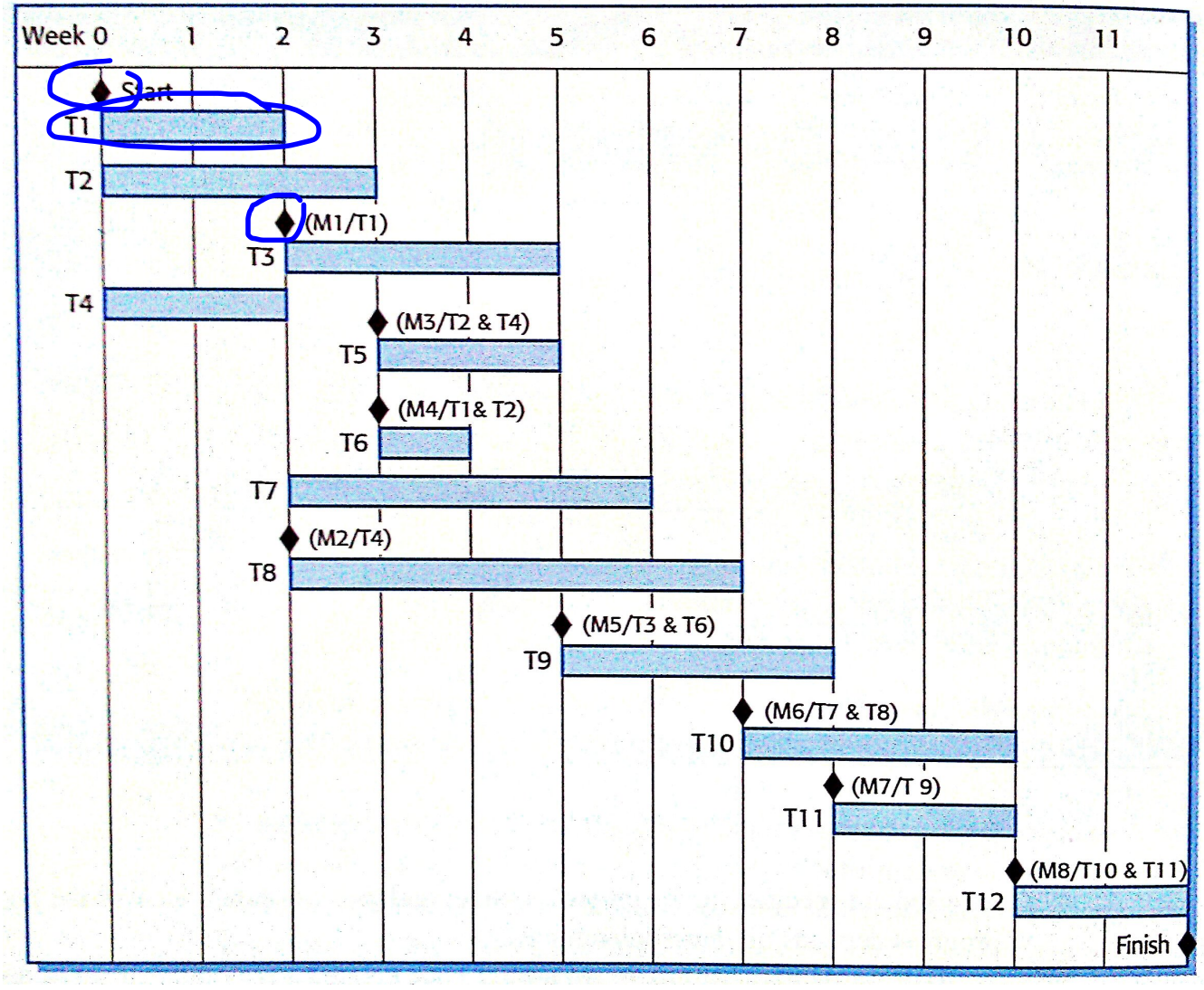
- milestones/deliverables: system specification, design specification, . . .
- Typical tasks: Work focused on system components

- Iterative Development (e.g. RUP)



- Milestones/deliverables: Each phase: go ahead to next phase
- Typical tasks: Work focused on system components

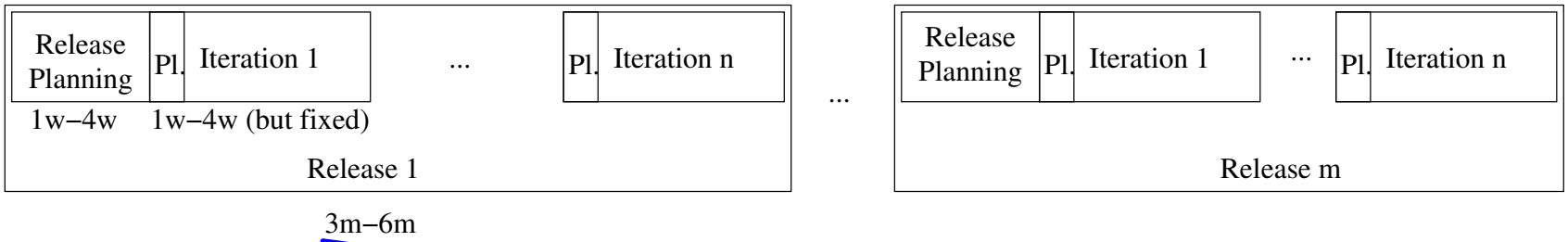
Schedule Representation: Gantt Chart / Bar chart



Planning Agile Projects

- *fixed* general structure

→ quarterly cycle / weekly cycle practices in XP



- *time boxing*
  - fixed: release dates and iterations
  - adjustable: scope
- Planning: Which user story in which iteration / release

Planning game

- Customer defines:
  - user stories
  - priorities
- Developer define:
  - costs, risks
  - suggest user stories
- Customer decides: is the user story worth its costs?
  - $\rightarrow$ split a user story
  - $\rightarrow$ change a user story

Project estimation techniques

- Algorithmic based
  - e.g. COCOMO, COCOMO II, ...
- Experienced based
  - XP: story points
  - Comparision with similar tasks

Algorithmic cost modeling: COCOMO

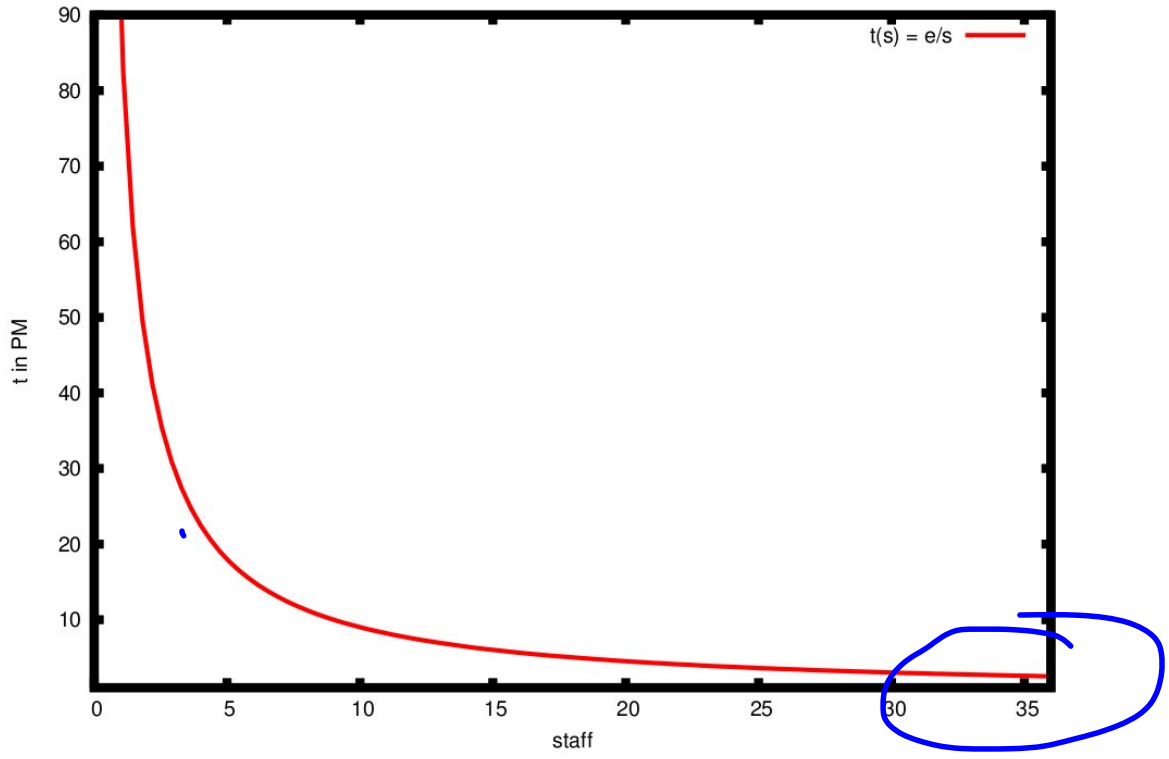
- Constructive Cost Model (COCOMO) by Bary Boehm et al., 1981
  - based on empirical studies
- Start with software size estimation: LOC (lines of code)
  - e.g. function point analysis based on requirements: complexity of functions and data
- Effort: in person months: $PM = a * LOC^b$
  - $2.4 \leq a \leq 3.6$: type of software
  - $1 \leq b \leq 1.5$: cost drivers: platform difficulty, team experience, ...
- Project duration: $TDEV = 3 * PM^{0.33+0.2*(b-1.01)}$
- Staffing: $STAFF = PM / TDEV$
- *"Adding manpower to a late software project makes it later"* Fred Brooks, The Mythical Man-Month, 1975

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
$effort = 90PM$

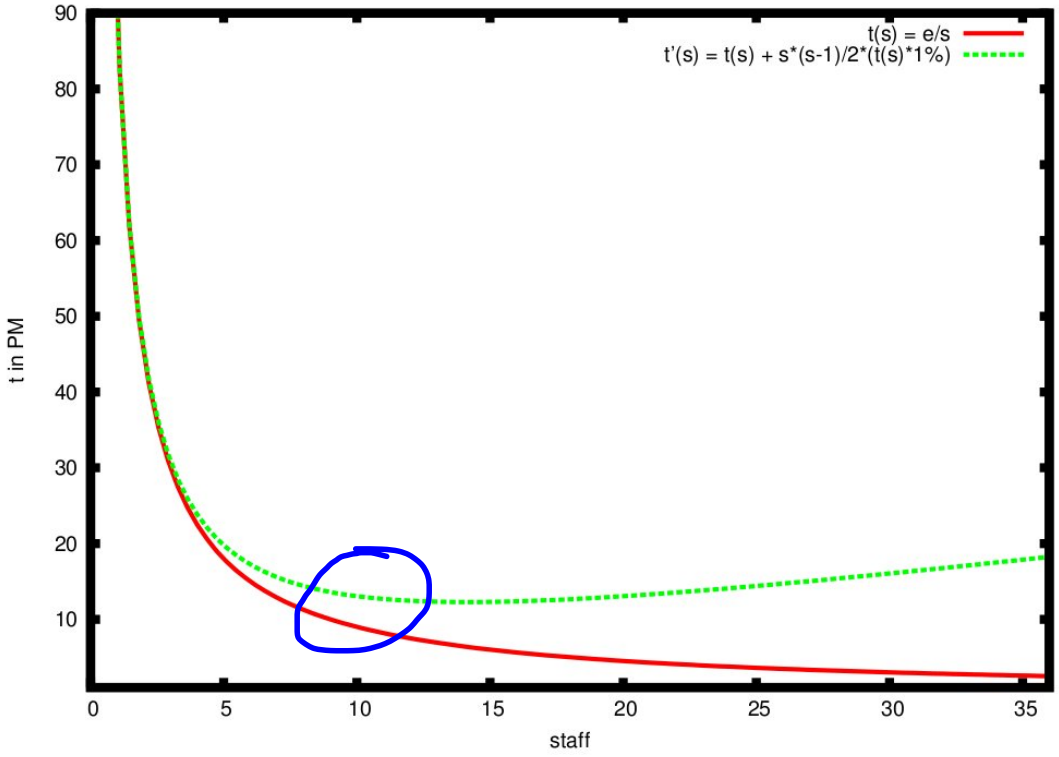
- $t(staff) = effort/staff$
- $TDEV = 15months$

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



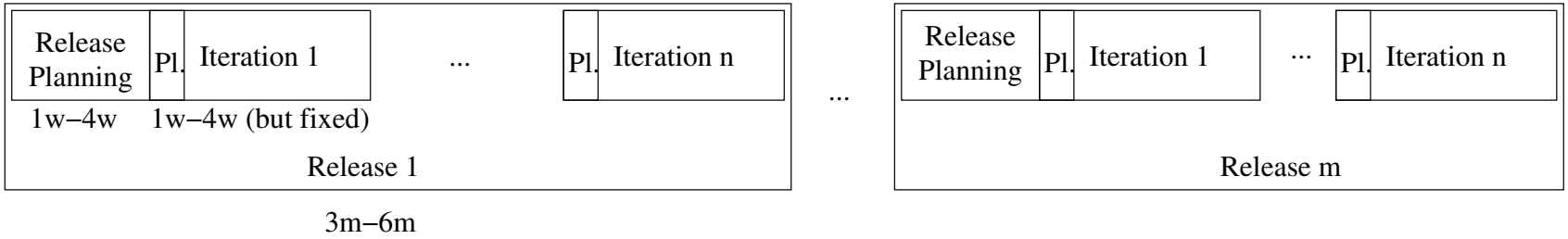
Assume effort $effort = 90PM$

- $t(staff) = effort/staff$
- $TDEV = 15months$
- $t'(statf) = t(staff) + staff(staff - 1)/2 \times 1\% t(staff)$
  Overhead based on 1% of the development time is devoted to talk to *1* other developer (simplified model)
- Plus ramp-up time for the new members

Planning Agile Projects

- *fixed* general structure

→ quarterly cycle / weekly cycle practices in XP



- Releases (quarterly cycle)
  - make (business) sense
  - *user stories* / *themes*
- Iterations with releasees (weekly cycle)
  - *user stories*
- *time boxing*
  - fixed: release dates and iterations
  - adjustable: scope

Scrum/XP: User story estimation (I)

- Estimation
  - Estimate *ideal_time* (e.g. person days / week) to finish a user story
  - real_time = ideal_time * load_factor (e.g. load_factor = 2)
  - Add user stories to an iteration based on real_time and priority
- Monitoring
  - New *load factor*: total_iteration_time / user_story_time finished

→ What can be done in the next iteration

- Yesterdays weather
  - only take load_factor from the last iteration for planning the next iteration
- Important: If in trouble focus on *few* stories and finish them

  → *Don't let deadlines slip* (time boxing)

Scrum/XP: User story estimation (II)

- Estimation
  - Estimate user stories *relative* to other user stories: *story_points*
  - *velocity*: number of story points that can be done in an iteration (initial value is a guess or comes from previous processes)
  - In an iteration: Select up to velocity amount of user stories
- Monitoring
  - *new_velocity*: story points of finished user stories per iteration

$\rightarrow$ What can be done in the next iteration

- user stories with story points up to new_velocity

Lean / Kanban: User story estimation

- No ”iterations”: user stories come in and flow through the system

→ Only a rough estimation of the size of the user stories
  - try to level the size of the user stories
  - Divide larger into smaller ones
- Measure process parameters, e.g., average cycle time
  - E.g. ”After committing to a user story, it takes in average a week to have the user story finished”
- User average_cycle_time and WIP (Work In Progress) Limit to determine the capacity of the process and thus throughput

Contents

Refactoring

- Restructure the program without changing its functionality
- Goal: improved design
- Necessary step in agile processes and test-driven development (TDD)
- Requires: sufficient (automated) tests

Refactoring

- Book: *Refactoring: Improving the Design of Existing Code*, Martin Fowler, 1999
- Set of refactorings
  - e.g. **renameMethod**, **extractMethod**, **encapsulateField**, **encapsulateCollection**, ...

  → complete list `http://www.refactoring.com/catalog/index.html`
- Set of code smells
  - e.g. Duplicate Code, Long Method, Large Class, Long Parameter List, ...

  → `http://c2.com/cgi/wiki?CodeSmell`, or `http://www.codinghorror.com/blog/2006/05/code-smells.html`
  - How to write unmaintainable code `http://thc.org/root/phun/unmaintain.html`
- Decompose large refactorings into several small refactorings
  - Each step: compiles and passes all tests
- IDE's have tool support for some refactorings

Example refactoring: RenameMethod

- Motivation
  - Sometimes a method name does not express precisely what the method is doing
  - This can hinder the understanding of the code; thus give the method a more intention revealing name
- Mechanics
  1) Create a method with the new name
  2) Copy the old body into the new method
  3) In the old body replace the body by a call to the new method; compile and test
  4) Find all the references to the old method and replace it with the new name; compile and test
  5) Remove the old method; compile and test

→ Supported directly in some IDE's

Code smells

If it stinks, change it Refactoring, Martin Fowler, 1999

- Duplicate Code
- Long Method
- Large Class
- Long Parameter List
- Divergent Change
- Shotgun Surgery
- Feature Envy
- Data Clumps
- Primitive Obsession
- Switch Statements
- Parallel Inheritance
- Lazy Class
- Speculative Generalisation
- Temporary Field
- Message Chains
- MiddleMan
- Inappropriate Intimacy
- Alternative Classes With Different Interfaces
- Incomplete Library
- Data Class
- Refused Bequest
- Comments

`http://en.wikipedia.org/wiki/Code_smell`

Code Smell: Data Clumps

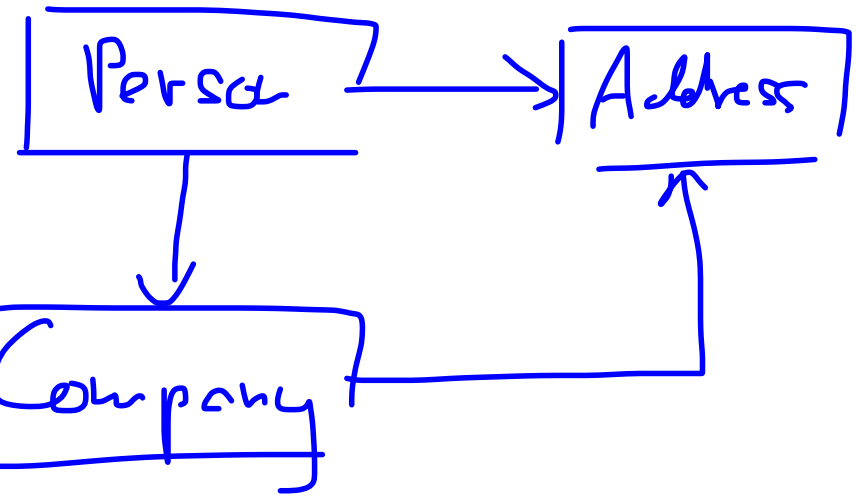
```
public class Person {
   private String name;
   private Calendar birthdate;
   private Company company;
   private String street;
   private String city;
   private String zip;
   ...
}

public class Company {
   private String name;
   private String vat_number;
   private String street;
   private String city;
   private String zip;
   ...
}
```

high cohesion

high cohesion

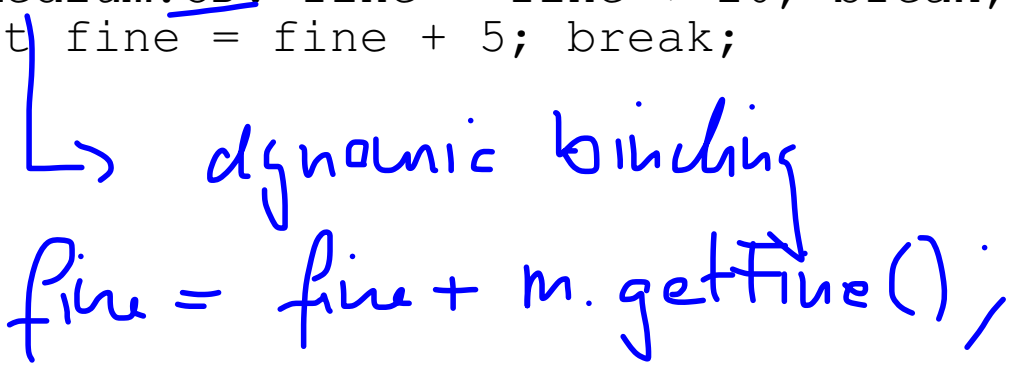
low cohesion

code duplication



Code Smell: Switch Statement

```
public class User {
   public double computeFine() {
      double fine = 0;
      for (Medium m : borrowedMedia) {
         if (m.overdue) {
            switch (m.getType()) {
               case Medium.BOOK : fine = fine + 10; break;
               case Medium.DVD: fine = fine + 30; break;
               case Medium.CD: fine = fine + 20; break;
               default fine = fine + 5; break;
            }
         }
      }
      return fine;
   }
}
```



Better Design

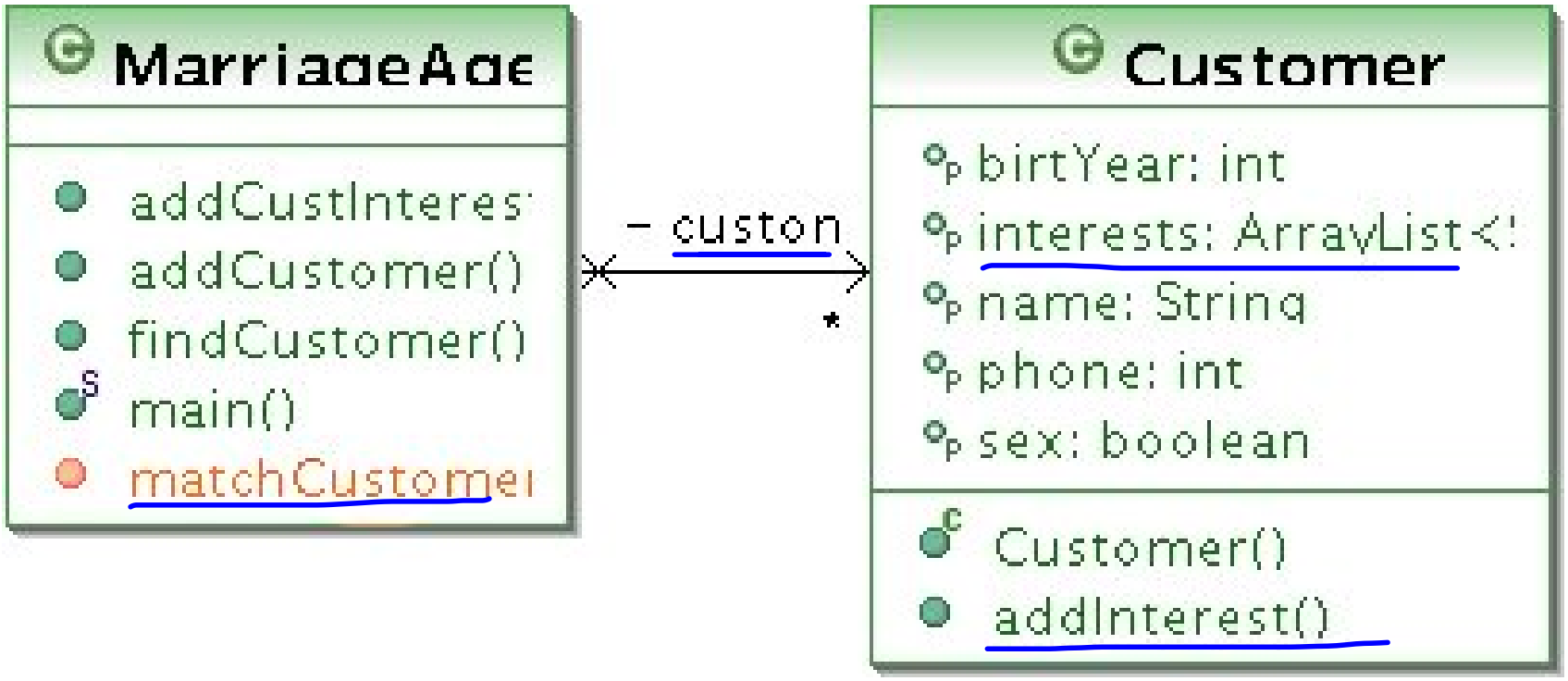
```
public class User {
   public double computeFine() {
      double fine = 0;
      for (Medium m : borrowedMedia) {
         if (m.overdue) { fine = fine + m.getFine();}
      }
      return fine;
   }
}

public class Medium {
   public double getFine() { return 5; }
}

public class Book extends Medium {
   public double getFine() { return 10; }
}

public class DVD extends Medium {
   public double getFine() { return 30; }
}

public class CD extends Medium {
   public double getFine() { return 20; }
}
```

Using "Template Method" Design Pattern

```
public class User {
   public double computeFine() {
      double fine = 0;
      for (Medium m : borrowedMedia) {
         fine =+ m.getFine();
      }
      return fine;
   }
}

abstract public class Medium {
   public double getFine() {
      return isOverdue() ? getFineForOverdueMedium() : 0;
}

public class Medium {
   abstract public double getFineForOverdueMedium();
}

public class Book extends Medium {
   public double getFineForOverdueMedium() { return 10; }
}

public class DVD extends Medium {
   public double getFine() {
      if (isScratched()) return 100;
      return super();
   }
   public double getFineForOverdueMedium() { return 30; }
}
```

Contents

MarriageAgency class diagram



- Refactoring example in detail
  - → `http://www2.imm.dtu.dk/courses/02161/2015/slides/refactoring_example.pdf`
- Framework for running tests as soon the code changes:
  - → Infinitest `http://infinitest.github.io/`

Remark on refactoring

- A refactoring takes a system with green tests to a system with green tests
- Decompose a large refactoring into small refactorings
  - $\rightarrow$ Don't have failing tests (or a broken system) for too long (e.g. days, weeks, ...)
    - Each small refactoring goes from a green test to a green test
    - Ideally, you can interrupt large refactorings to add some functionality and then continue with the refactoring

Next Week

- Principles of Good Design
- Design Patterns