Software Engineering I (02161)

Week 8

Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

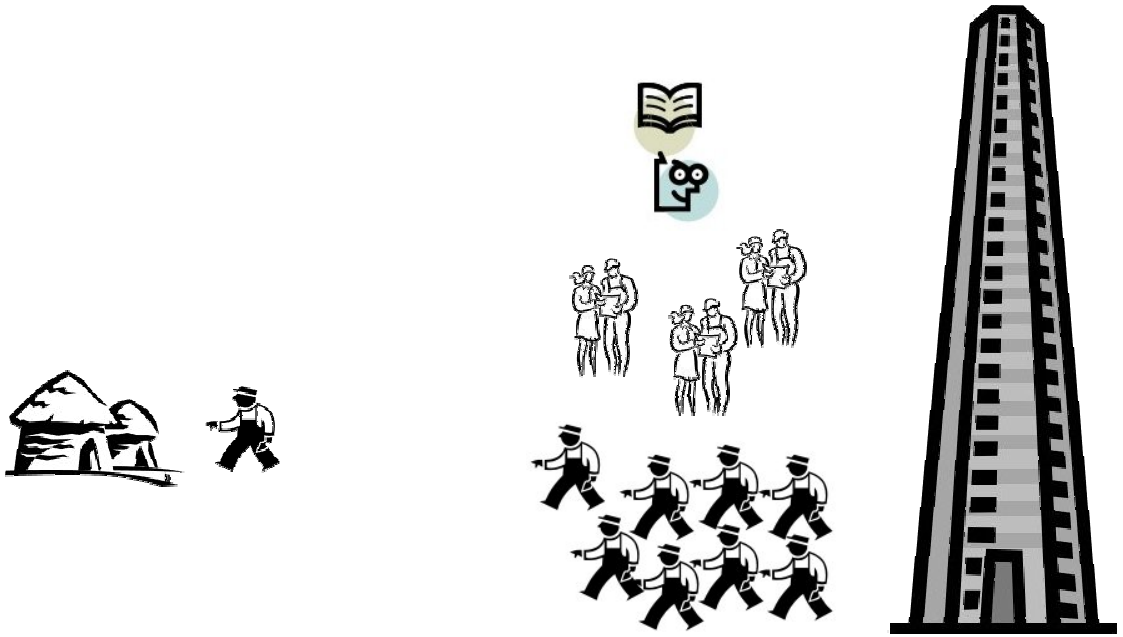
Spring 2016

Last Week

- State machines
- Layered Architecture: GUI
- Layered Architecture: Persistency Layer

Contents

Software Development Challenges



- Challenges of Software Development
  - On **time**
  - In **budget**
  - No **defects**
  - Customer **satisfaction**

Software Development Process

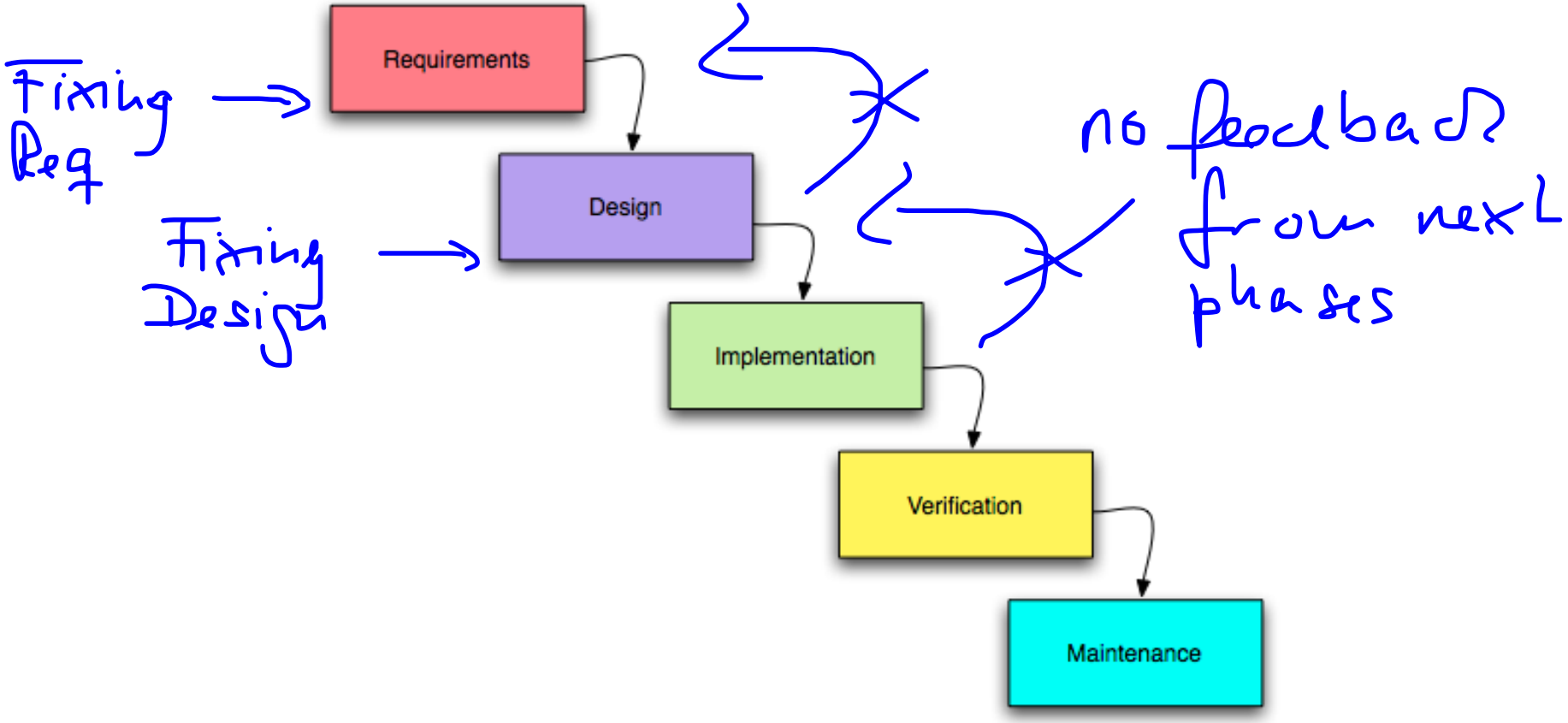
- Activities in Software Development
  - *Understand* and *document* what the customer wants: **Requirements Engineering**
  - *How* to build the software: **Design**
  - *Build* the software: **Implementation**
  - *Validate*: **Testing**, **Verification**, **Evaluation**

→ Set of techniques: Use cases, CRC cards, refactoring, test-driven development, ...

- How to apply the techniques:

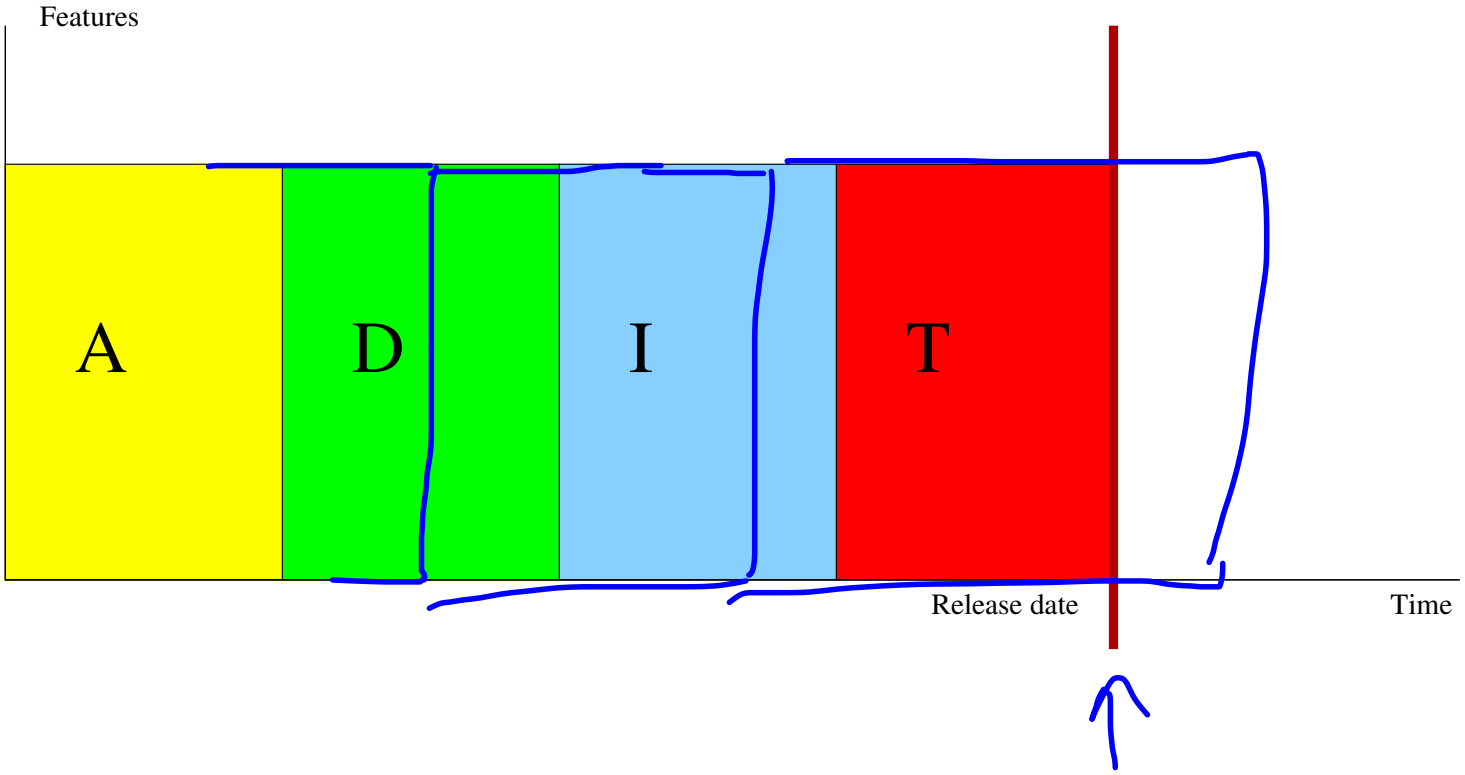
→ Different **software development processes**: Waterfall, Iterative processes, agile, lean, ...

Waterfall process

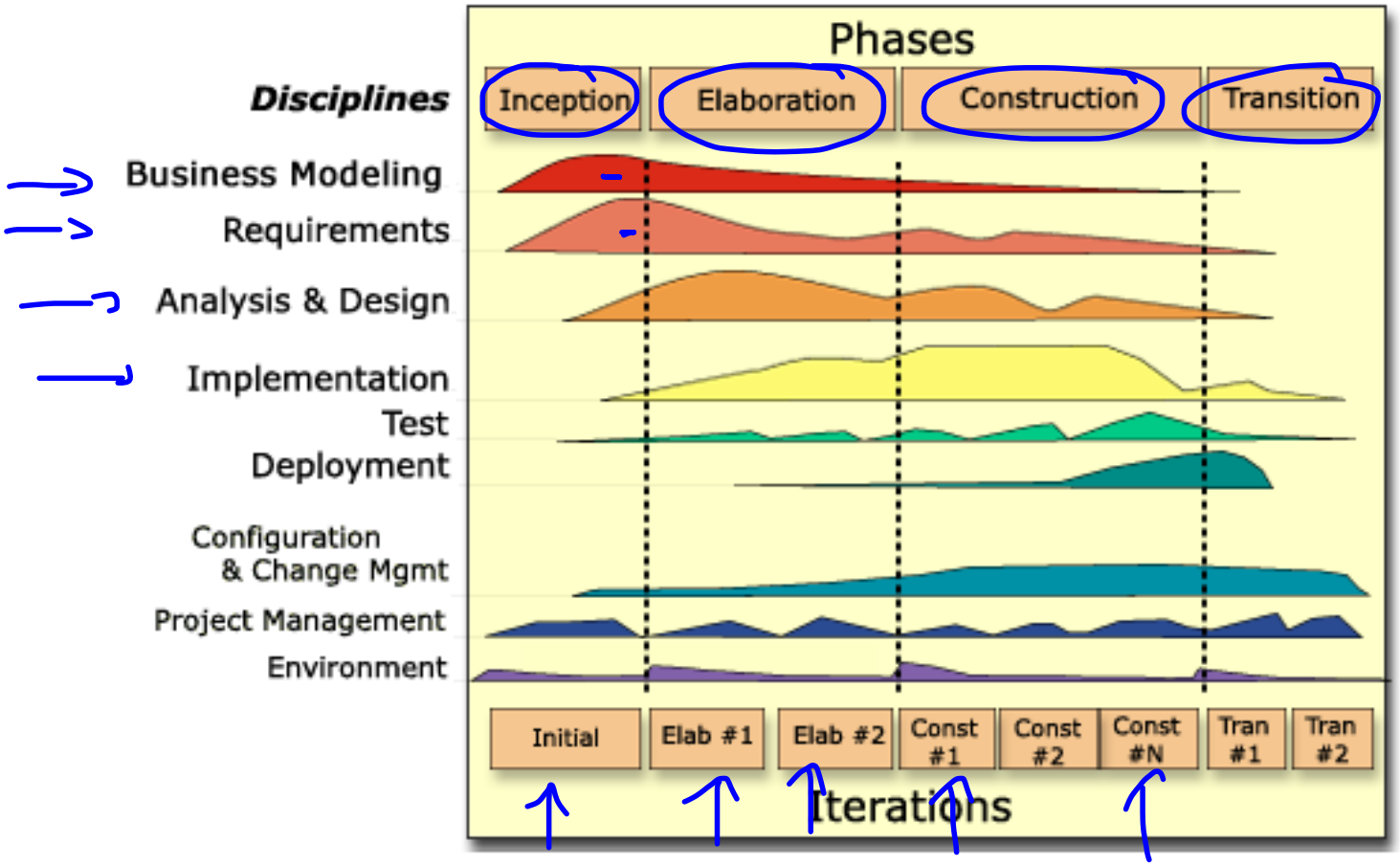


- 1970: Used by Winston W. Royce in a article as a an example of how not to develop software
- 1985: Waterfall was required by the United States Department of Defence from its contractors

Delays in waterfall processes



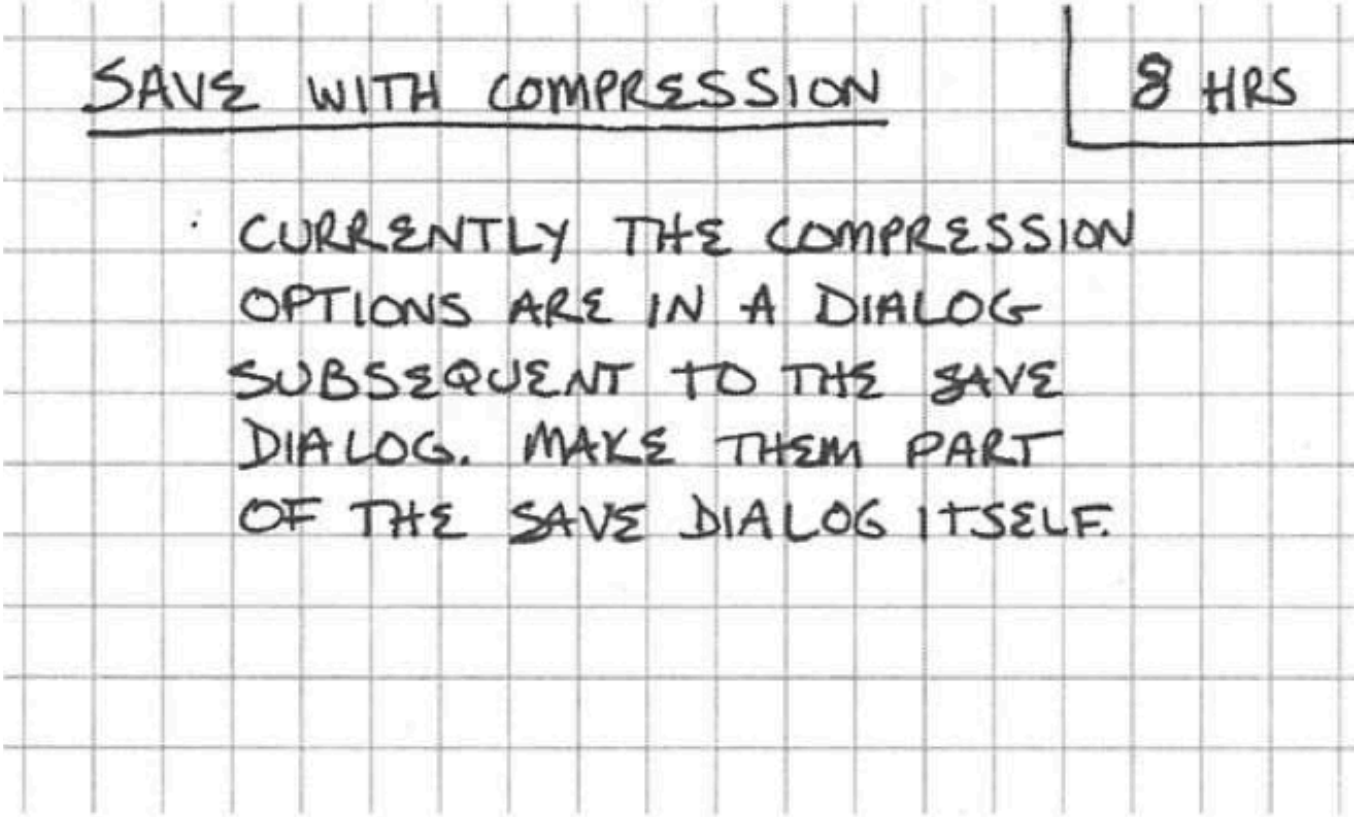
Iterative Processes: E.g. (Rational) Unified Process (1996)



Agile Software Development Methods (1999)

- Examples
  - Extreme Programming (XP) (1999), Scrum (1995–2001), Feature Driven Development (FDD) (1999), Lean Software Development (2003), . . .
  - (Kanban (2010): often seen as a method, but it is a tool to improve processes)
- Common characteristic
  - Driven by very small functionalities with value to the customer: e.g. user stories (XP) / Backlog items (Scrum) / smallest marketable feature (Lean/Kanban) / . . .
  - Short iterations:
    - Each iteration produces a software increment
    - = Small batch sizes = # of user stories you do at once
      Ideal batch size: one (single piece flow)
  - New "extreme" practices like short iterations, pair programming, and test-first.
  - Applies values and principles from Lean Production
  - Based on the Agile Manifesto (2001)

Example of a User story card



Kent Beck, Extreme Programming 2nd ed.

- User story card: A contract between the customer and the devloper to talk about the user story

Manifesto for Agile Software Development (2001)

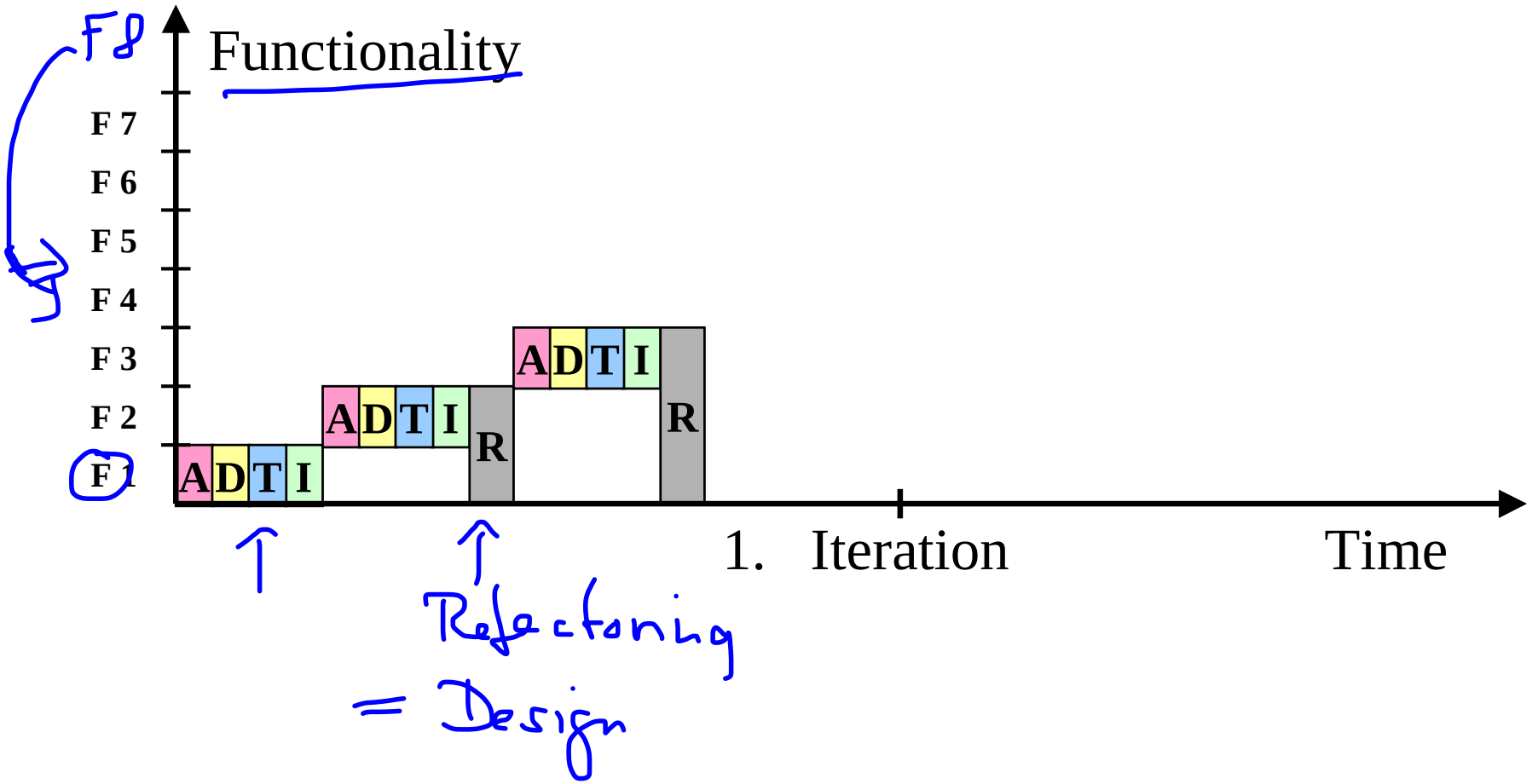
"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- *Customer collaboration over contract negotiation*
- *Responding to change over following a plan*

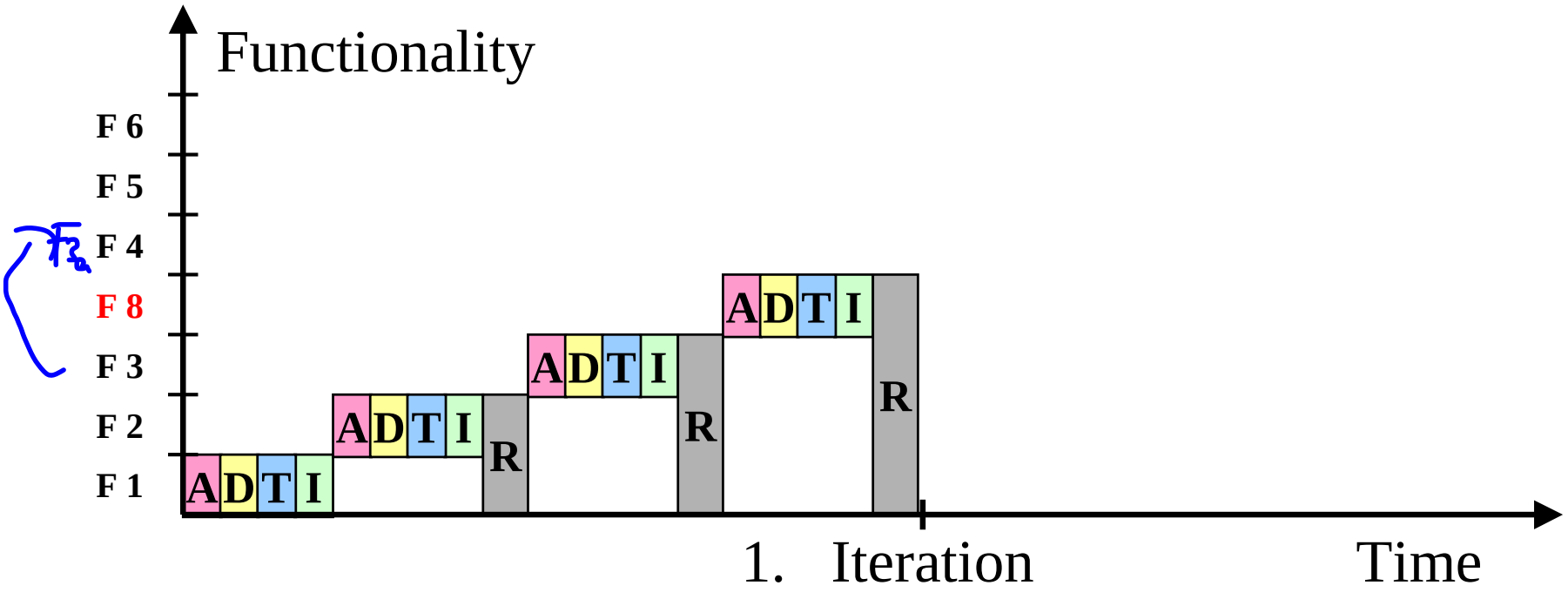
That is, while there is value in the items on the right, we value the items on the left more."

`http://www.agilemanifesto.org`

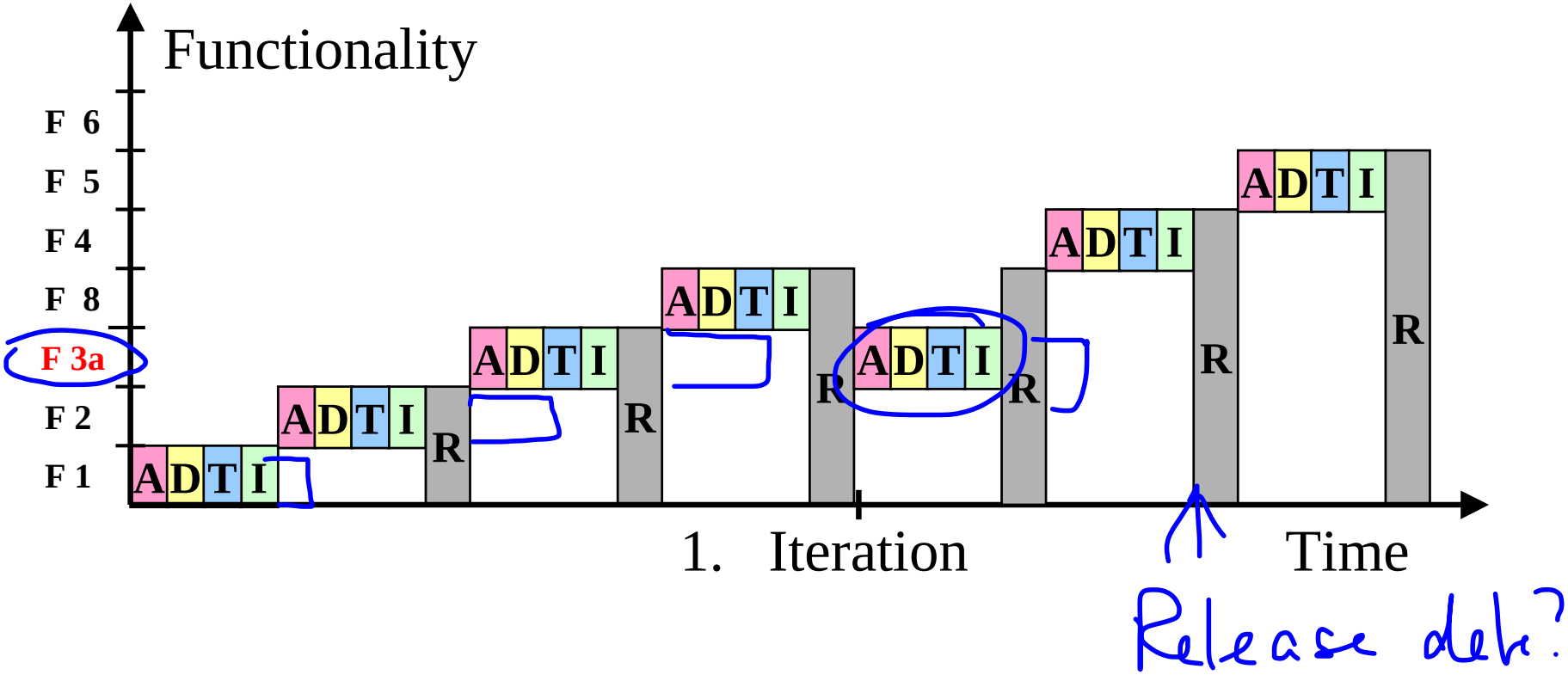
Agile processes and Lean Software Development



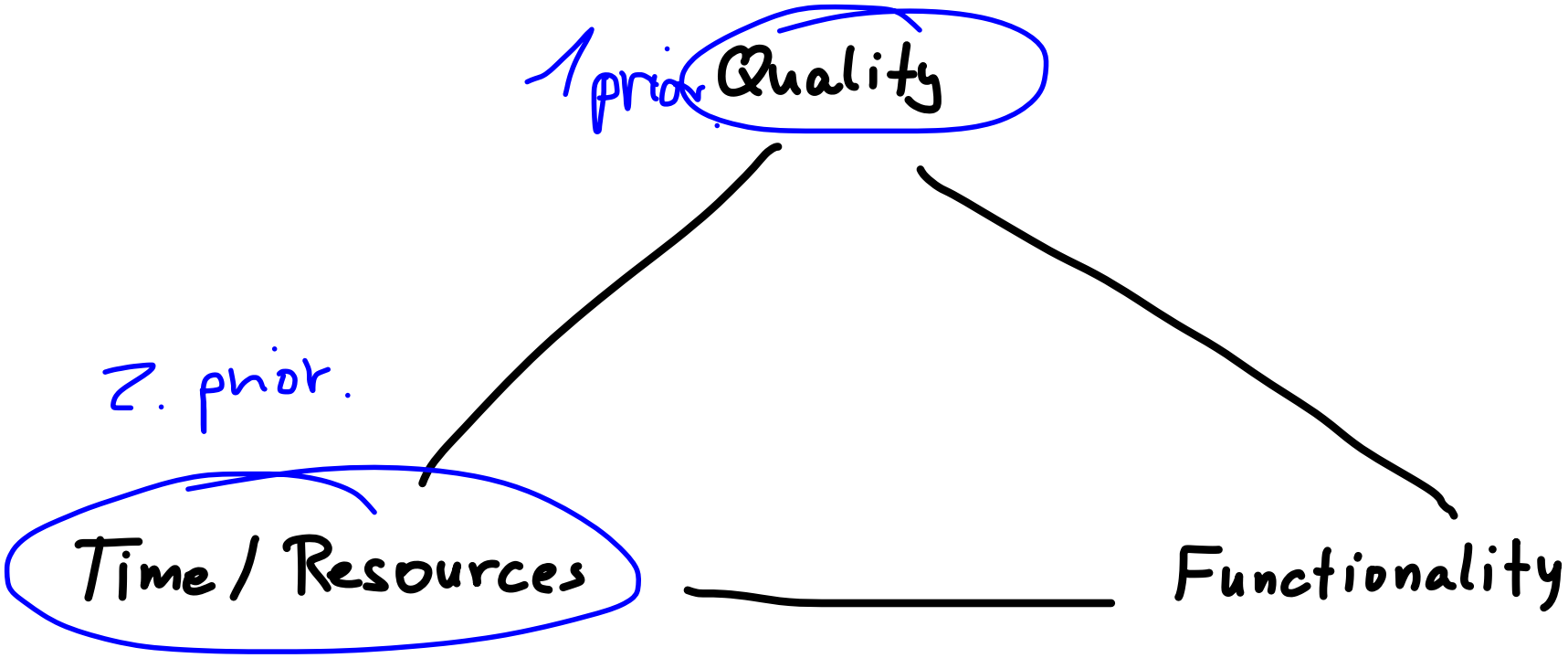
Agile processes and Lean Software Development



Agile processes and Lean Software Development

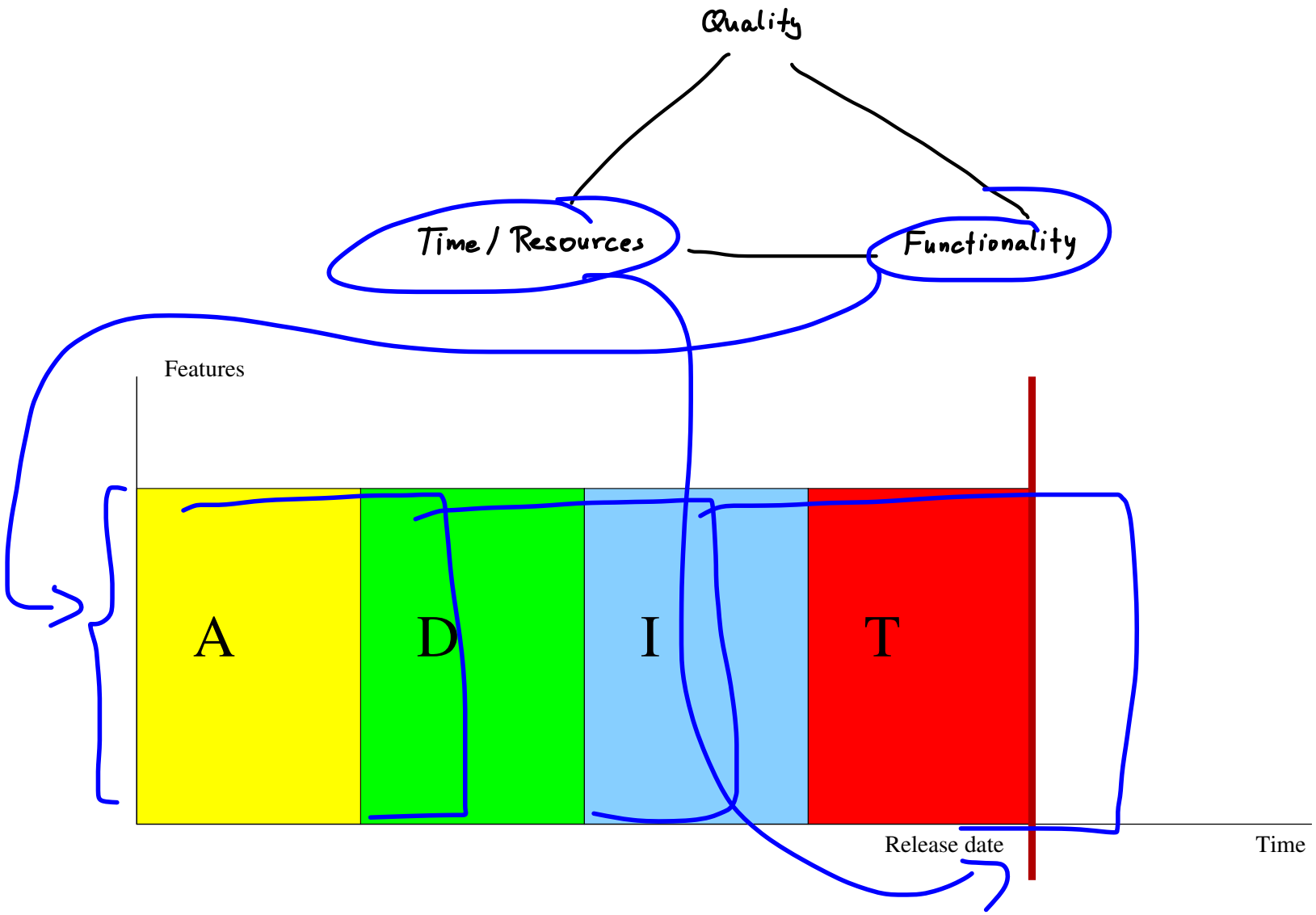


Resource Triangle

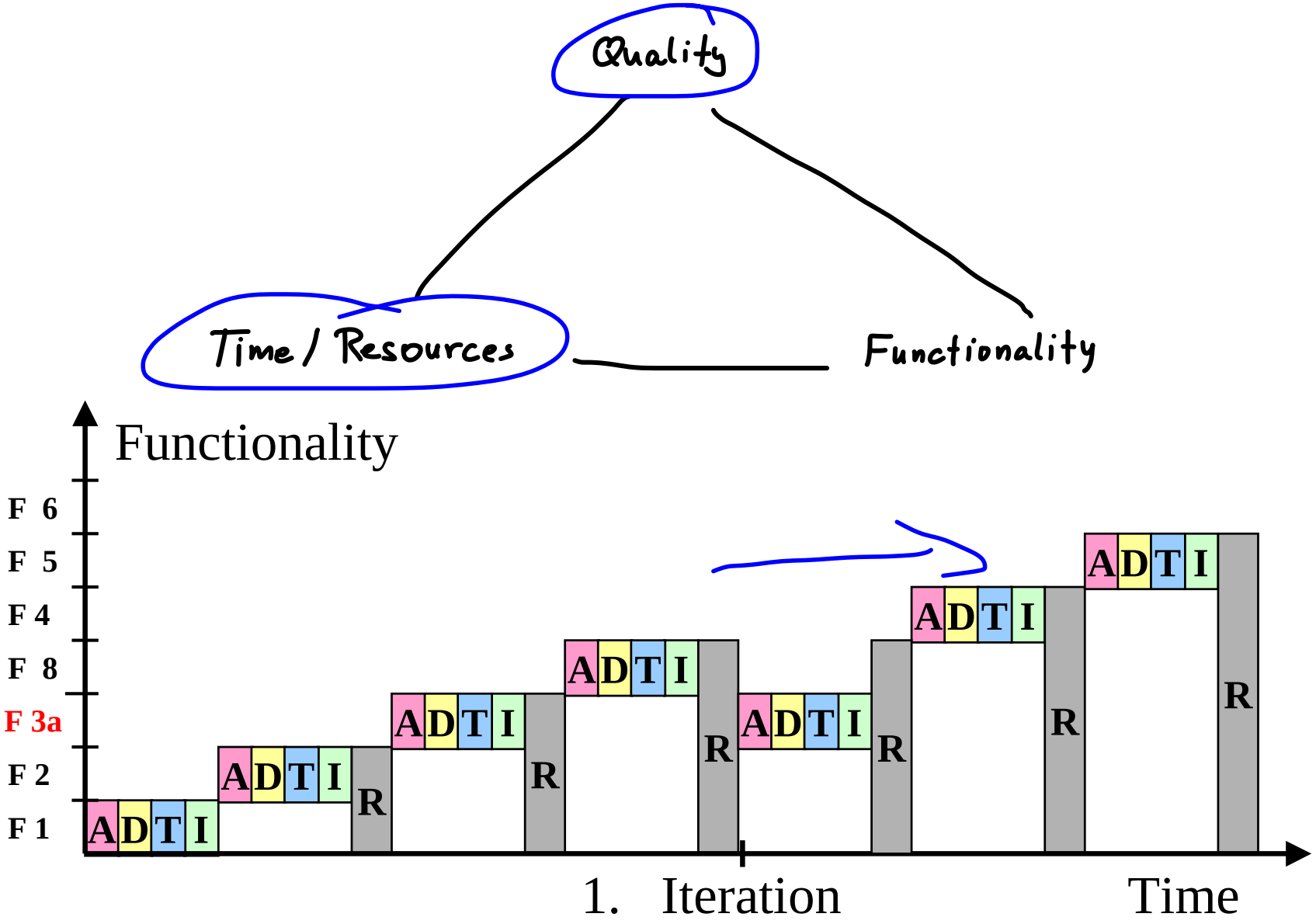


- Can only fix two of them at the same time

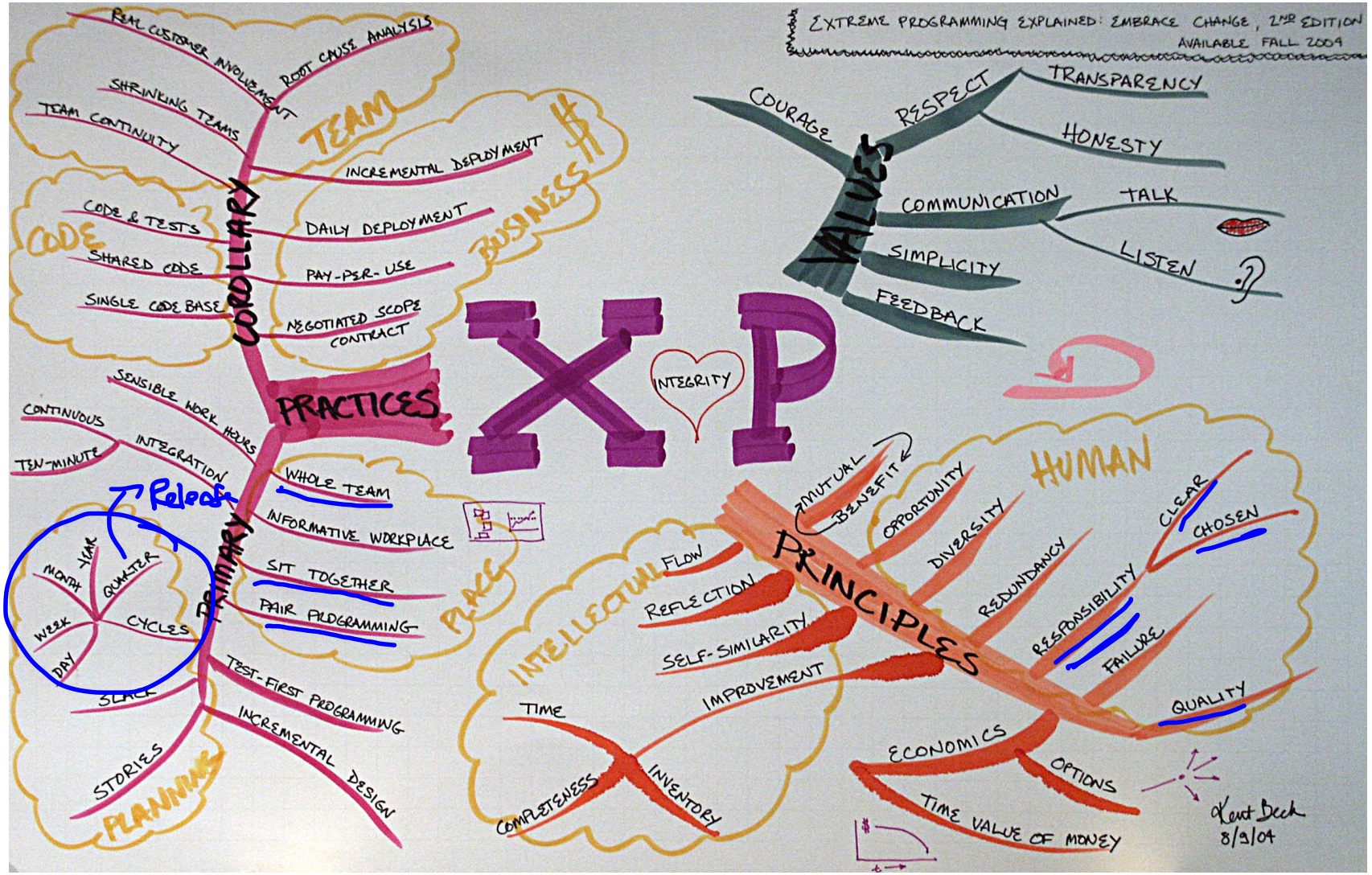
Resource Triangle: Waterfall



Resource Triangle: Agile

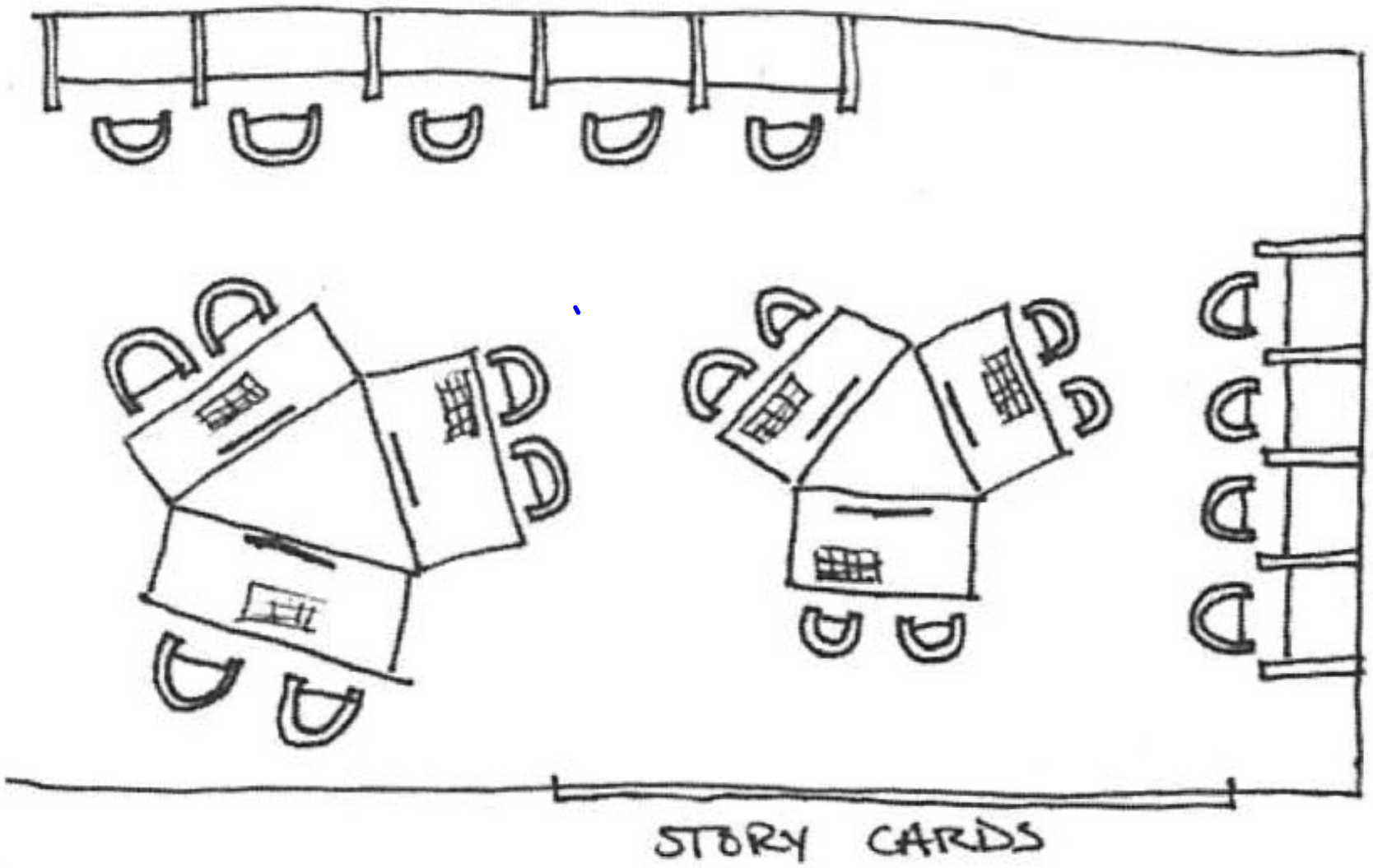


eXtreme Programming (XP)



Kent Beck, Extreme Programming 2nd ed.

Sit-together

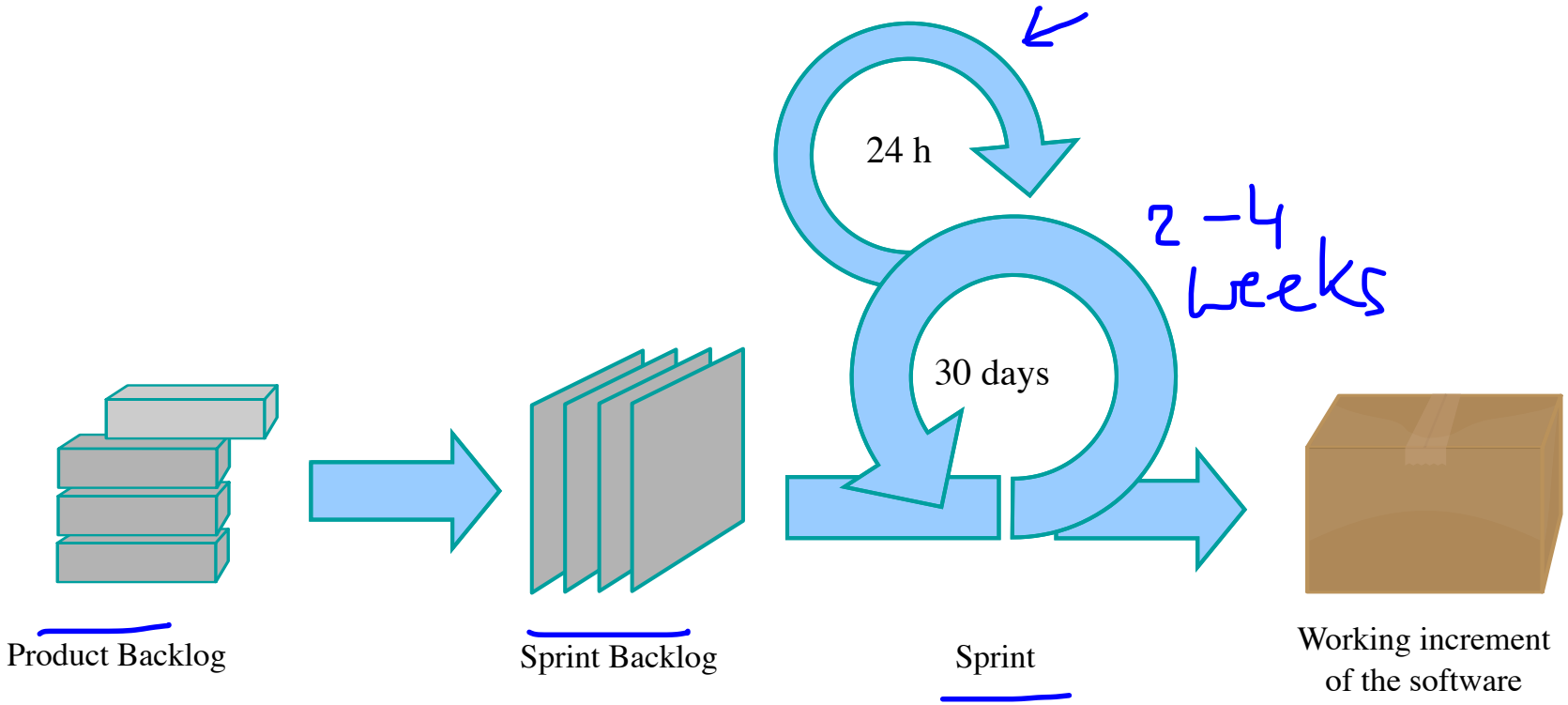


Kent Beck, Extreme Programming 2nd ed.

Visual wall

Kent Beck, Extreme Programming 2nd ed.

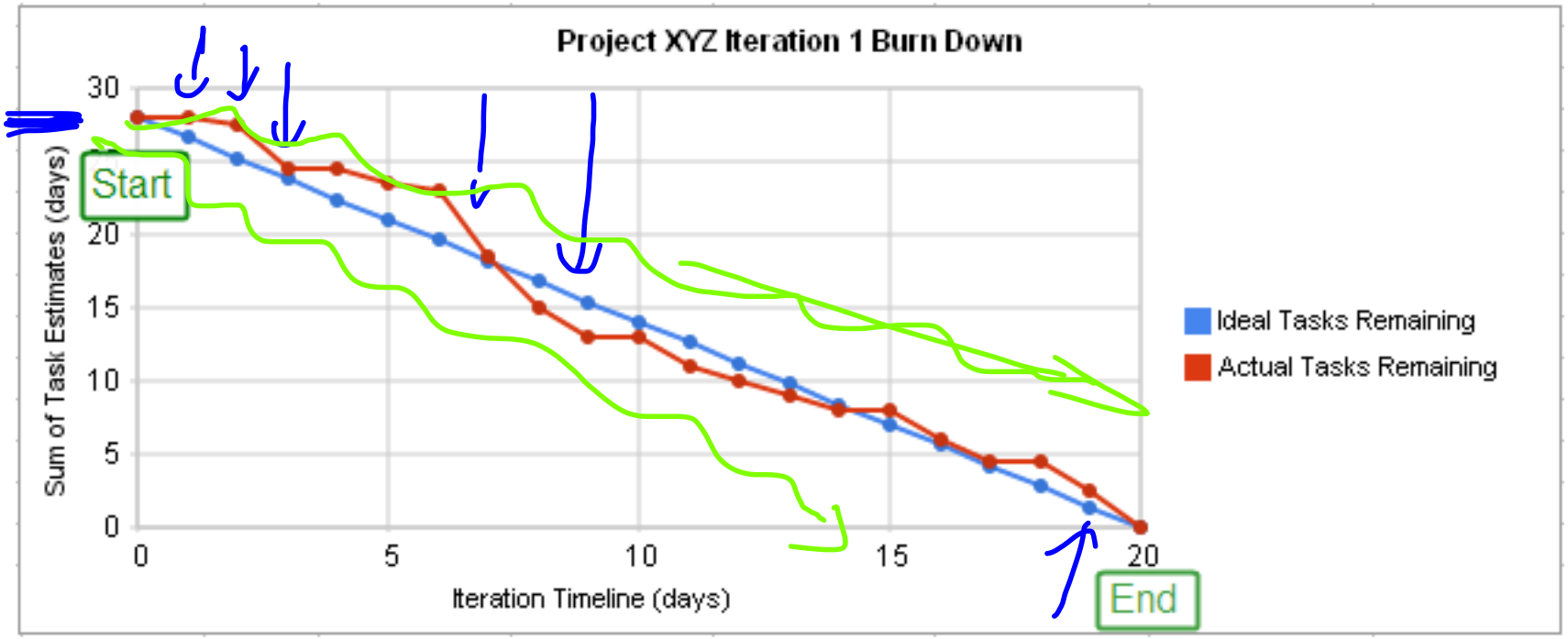
Scrum



Wikipedia

- Robert Martin (Uncle Bob) about "The Land that Scrum Forgot"
  `http://www.youtube.com/watch?v=hG4LH6P8Syk`
  → History about agile methods, the agile manifesto, and Scrum and its relationshop to XP

Burn Down Charts



Wikipedia

Lean Software Development

- Lean Production:
  - Reduce the amount of *waste* in the production process
  - Generate *flow*
- Waste: resources used with does not produce value for the customer
  - time needed to fix bugs → TDD
  - time to change the system because it does not fit the customers requirements
  - time waiting for approval
  - ...

Cycle time

Cycle time

Time it takes to go through the process one time

$$cycle\_time = \frac{number\_of\_features}{feature\_implemantion\_rate}$$

- Example: Waterfall
  - Batch size = number_of_features in an iteration
  - Software: 250 features, feature_implementation_rate = 5 features/week
  - cycle_time = 250 / 5 = 50 weeks
  - Overall time: 50 weeks

  → 1 cycle

Goal: Reducing the cycle time

- Reduce batch size: 1 feature in an iteration
- Software: 250 features, feature_implementation_rate = 5 features/week

$$cycle\_time = \frac{number\_of\_features}{feature\_implemantion\_rate}$$

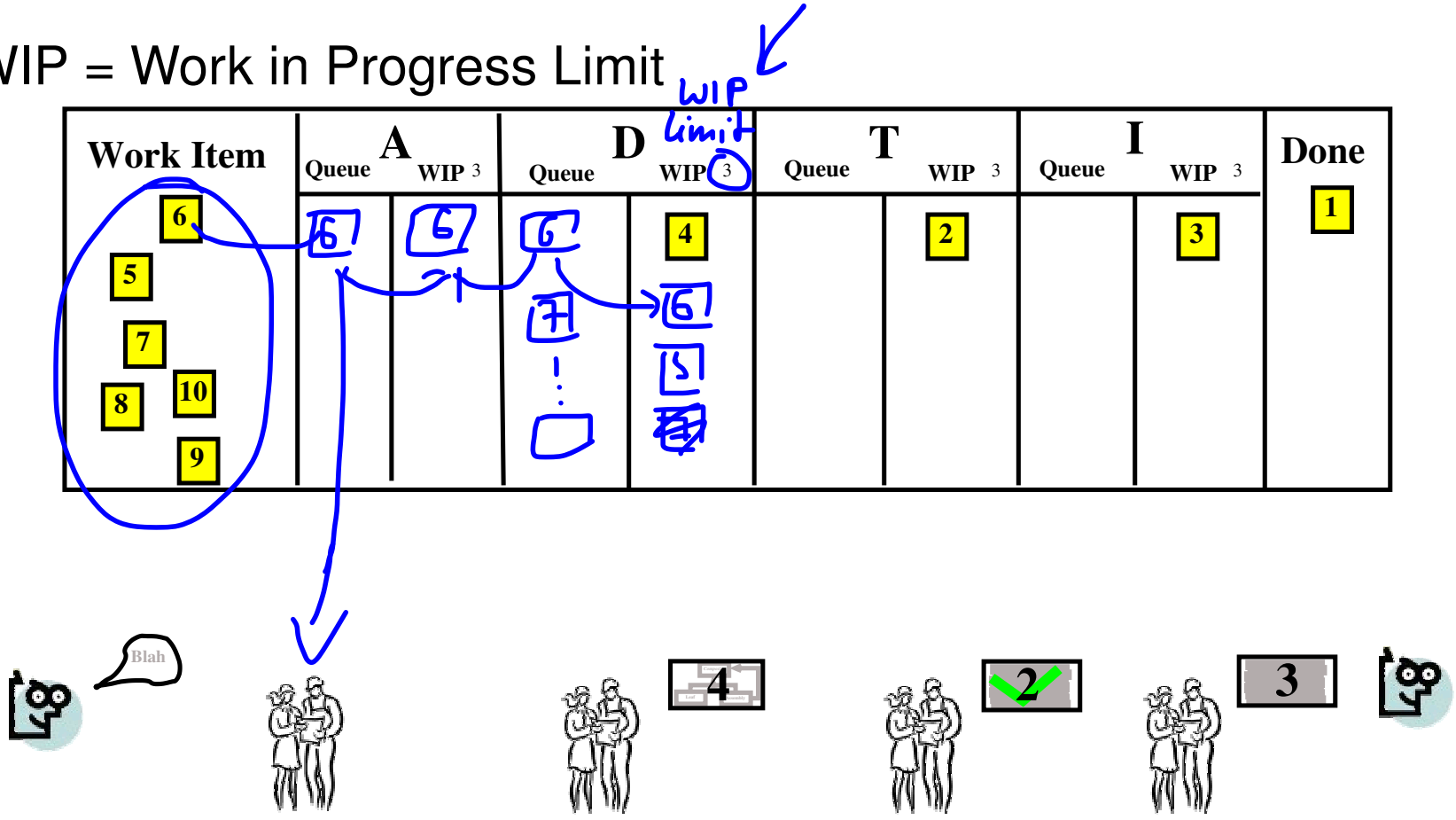
- Agile: cycle_time = 1 / 5 = 8 hours

→ 250 cycles

- Advantages
  - Process adapts to changes in requirements
  - Process improvements and fine tuning

Generating flow using Pull and Kanban

WIP = Work in Progress Limit



Flow through Pull with Kanban

- Process controlling: local rules
- Load balancing: Kanban cards and *Work in Progress (WIP) limits*
- Integration in other processes

Figure from David Anderson `www.agilemanagement.net`

Online Kanban Tool: Trello

- `www.trello.com`: Electronic Kanban board useful for your project
- Example Kanban board `https://trello.com/b/4wddd1zf/kanban-workflow`

Week 8—13

Implementation process

1. Choose a set of user stories to implement
1. Select the user story with the highest priority
   a) Create the acceptance test for the story in JUnit
   b) Implement the user story test-driven, creating additional tests as necessary and **guided** by your design
      - → based on the classes, attributes, and methods of the model
      - → implement **only** the classes, attributes, and methods needed to implement the user story
      - → adapt your design as necessary!!
      - → Criteria: 100% code coverage of the application logic based on the tests you have
   c) Refactor system (= Design)
3. Repeat step 2 with the user story with the next highest priority

Contents

What is version control

Version Control

- Stores and manages versions of documents (e.g. .java files)
- Manages concurrent work on documents
- Manages different software release versions
- Various systems: Concurrent Versions System (CVS), Apache Subversion (SVN), Git, Team Foundation Server (TFS) ...

CVS

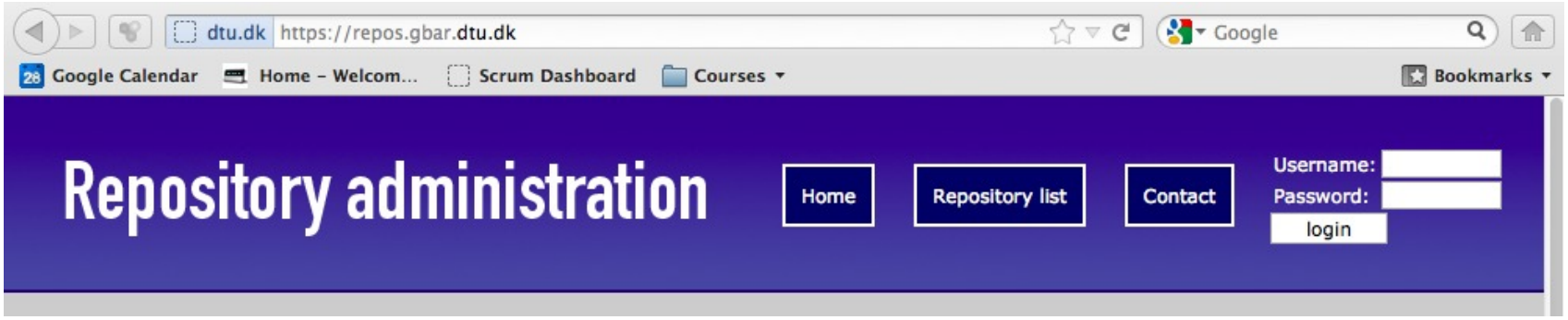
- The presentation focusses on CVS, but the concepts also apply to SVN too and to a lesser extend to Git and TFS
- CVS = Concurrent Versions System
- One central repository
- Command line tools, IDE support
- Files have a tree of versions: branching
- Release: File versions having same *tag*
- Versions: **diffs** (differences) to previous versions

Use cases of CVS

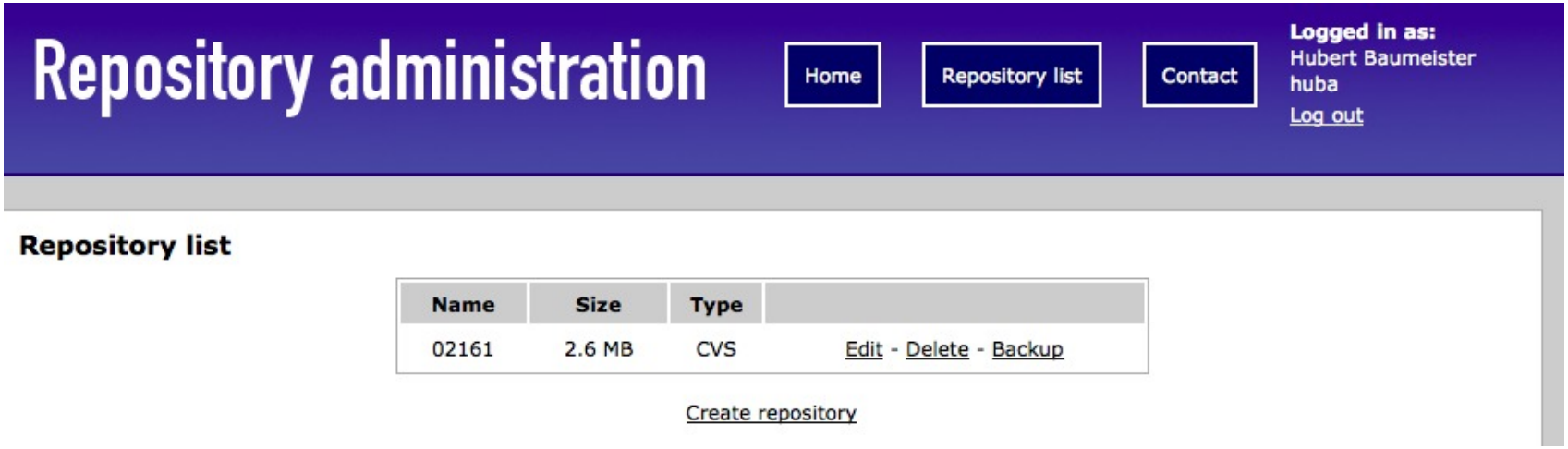
- Creating a repository
- Creating a project
- Checking out a project
- Updating a project
- Committing changes
- Tagging versions
- Branching versions
- Merging branches

Creating a repository

- `http://repos.gbar.dtu.dk`



Creating a repository



The GBar supports CVS, SVN, and Git

Creating a repository

Field:	Value
Rename repository: Alphanumeric characters and underscore.	02161
Options	Anonymous read-only access (disabled): ☐
Checkout	`:pserver:USERNAME@cvs.gbar.dtu.dk:/home/cvs/huba/02161` Please note that you need to add a user to the repository before you check it out!
Current users:	No users yet! [Add new user]
	Update Repository

Creating a repository

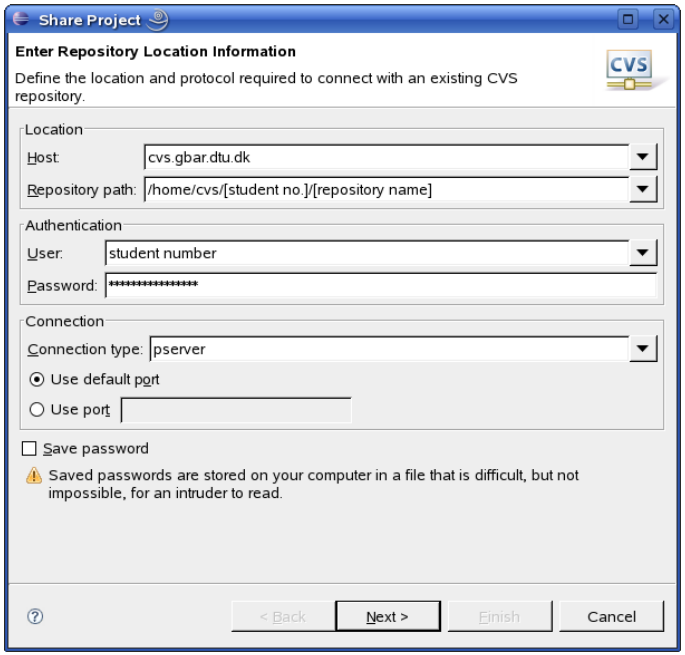
Edit repository

Field:	Value
User name:	
New password:	
Confirm new password:	
	Add user

Back

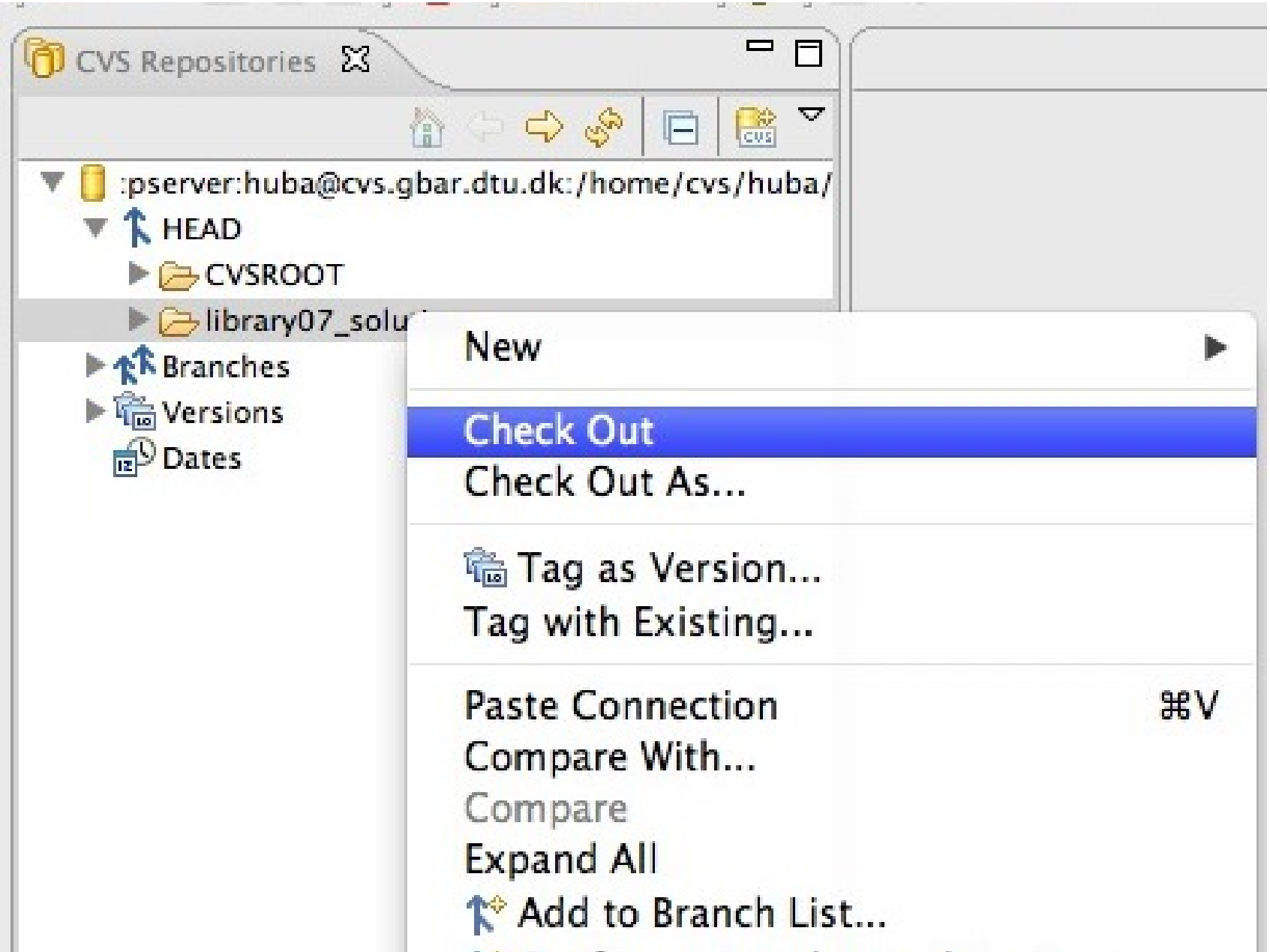
Create a project and *share* it

- Menu: Team→share project and `create a new repository location`

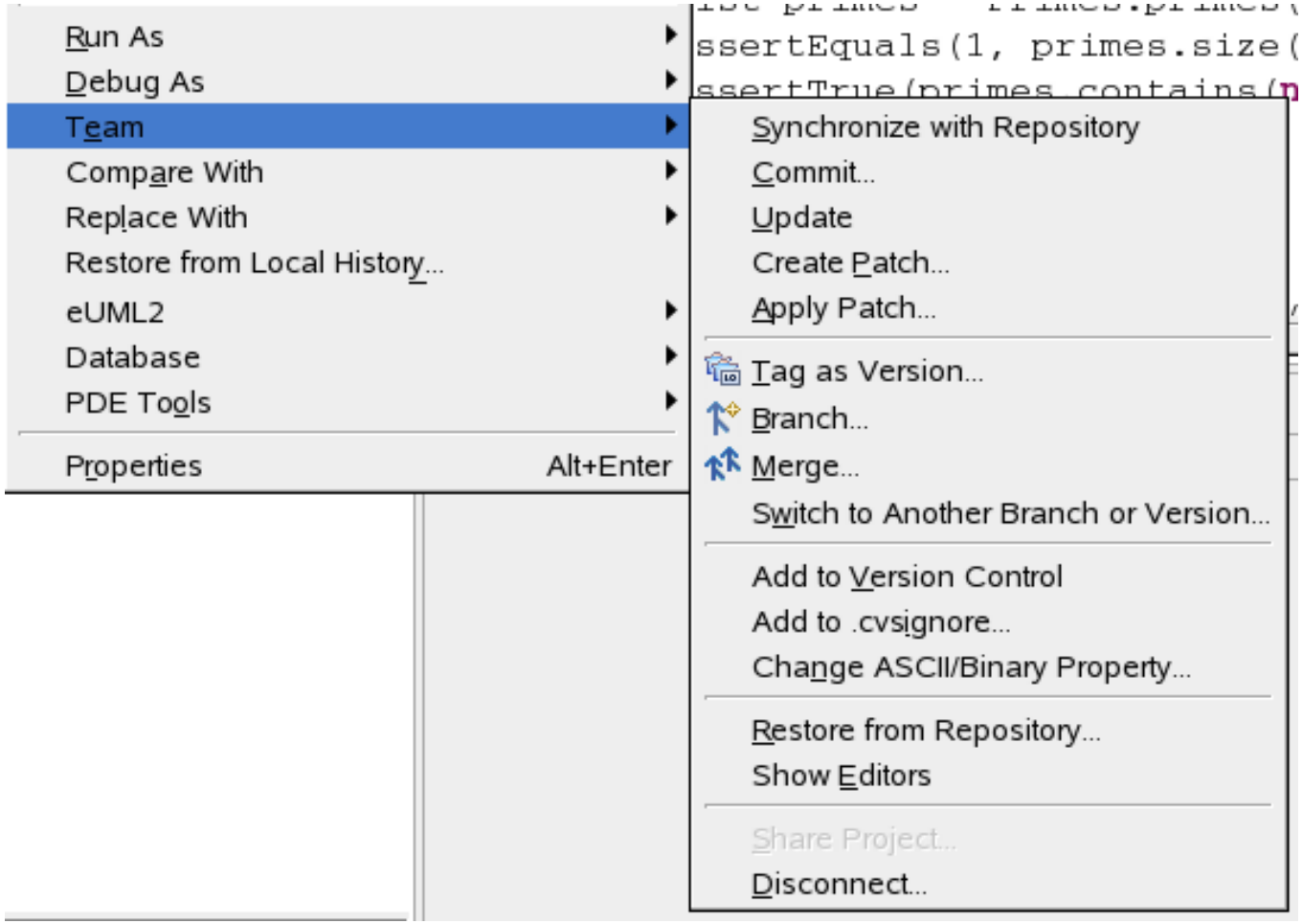


Checking out a project

- CVS Repository Exploring perspective



Package Explorer Team Menu Project



Steps in Developing a Program using CVS

1. Create Repository
2. Create a project and *share* the project
3. For all the programming tasks in an **iteration**
   - 3.1 Run tests; Update project; run tests; fix tests
   - 3.2 Work on the implementation so that all tests run
   - 3.3 Update the repository with your changes
     - 3.3.1 *Update* the project; run tests
     - 3.3.2 *Fix* all compile time errors and all **broken** tests;
     - 3.3.3 *Commit* your changes
4. *Tag* you files for major **project milestones**

Important: Commit only if all tests pass

Committing changes

- Fails if someone else committed the file before
- If commit fails
  1. update, this automatically tries to merge the changes,
  2. compile: fix possible compilation errors
  3. run the tests: fix failing tests
  4. commit again

Update a project

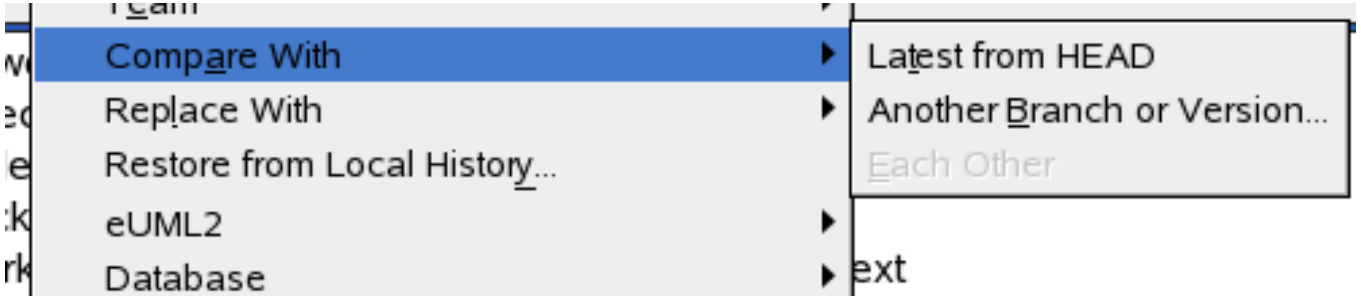
- Gets newest version of the file
- If conflicts
  - → *text* files are **merged**
  - → other files are **overwritten**
    - based on **lines**
    - successful merge: lines are added from both source files
    - unsuccessful merge: the same line is changed in both source files

Unsuccessful merge

- **Same** lines have been changed

```
   public Address() {
      // TODO Auto-generated constructor stub
   }
<<<<<<< Address.java
   public String getStrasse() {  // Local change
=======
   public String getGade() { // Committed change
>>>>>>> 1.2
      return street;
   }
```

Package Explorer Compare With Menu



Compare result: Compare with latest from HEAD

Address.java | Address.java

Java Structure Compare

- Compilation Unit
  - Address
    - getGade()
    - getStrasse()

Java Source Compare

Local File 1.2

```
    }

    public Address() {
        // TODO Auto-generated constructor stub
    }

    public String getStrasse() {  // Local change
        return street;
    }

    public int getPostnumber() {
        return postNumber;
    }
```

Remote File 1.2

```
    public Address() {
        // TODO Auto-generated constructor stub
    }

    public String getGade() { // Committed change
        return street;
    }

    public int getPostnumber() {
        return postNumber;
    }
```

Next Week

- Project planning (traditional and agile)
- Refactoring
- (Design Patterns)