

Software Engineering I (02161)

Week 8

Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2013

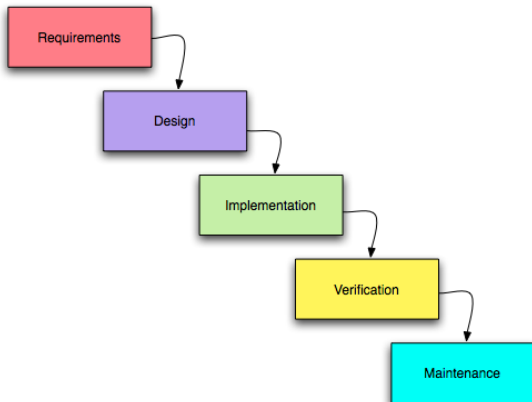
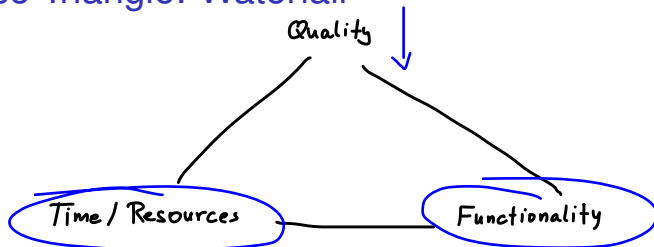
Contents

Software Development Process (cont.)

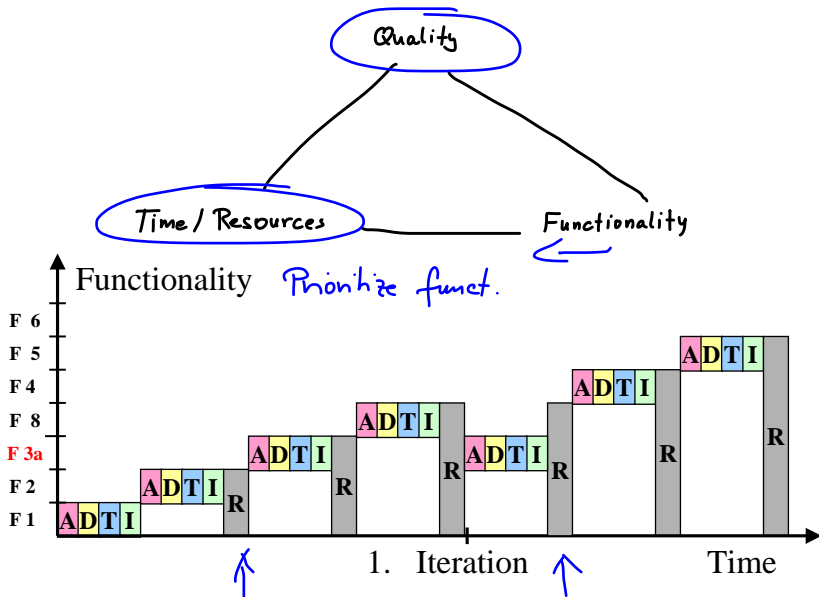
From Requirements to Design: CRC Cards

Version control

Resource Triangle: Waterfall



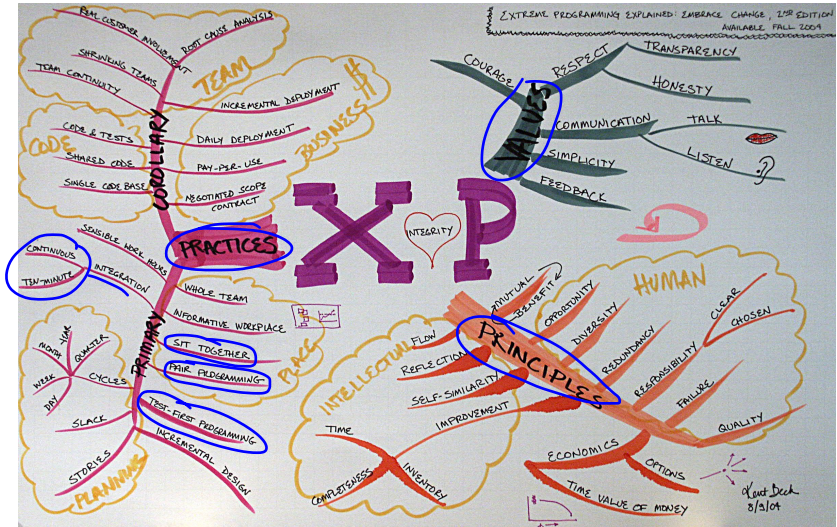
Resource Triangle: Agile



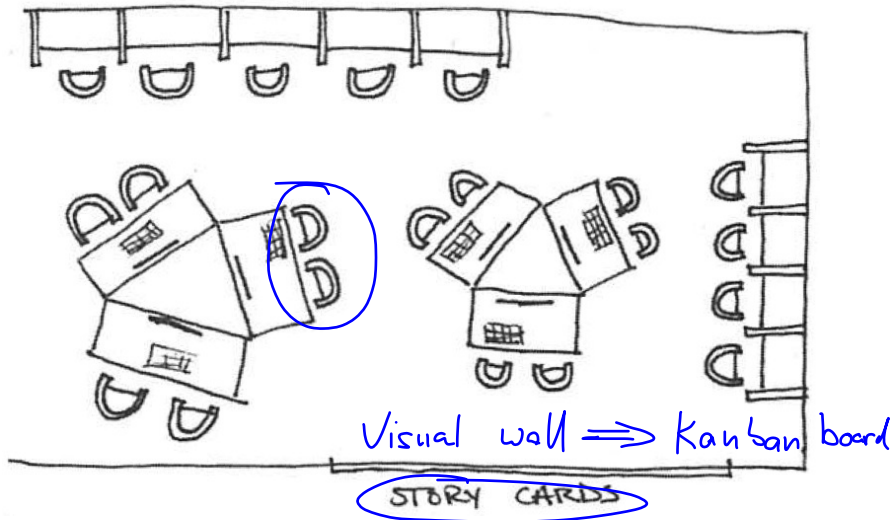
Agile processes

- ▶ Agile software development methods
 - ▶ Extreme Programming
 - ▶ Scrum
 - ▶ Lean Software Development → from (Car) production
 - ▶ Kanban
- ▶ Common characteristic
 - ▶ Short iterations: Each iteration produces a software increment
 - = Small batch sizes
Ideal batch size: one (single piece flow)
 - ▶ Driven by user stories/Backlog items/smallest marketable feature/...

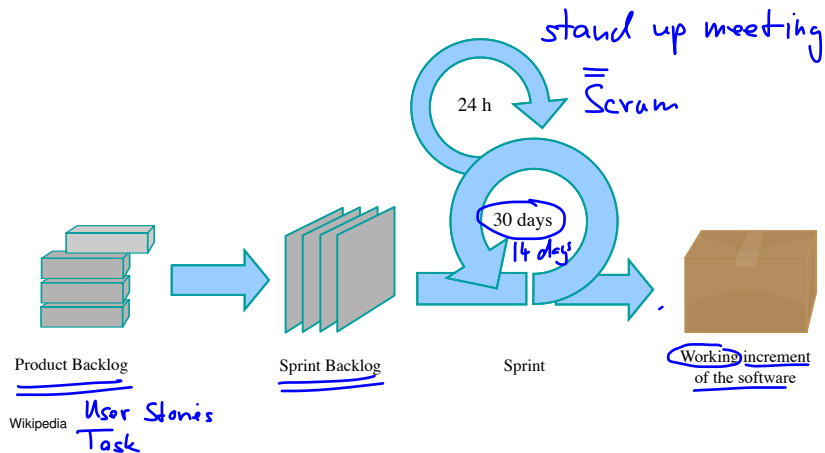
eXtreme Programming (XP)



Sit-together

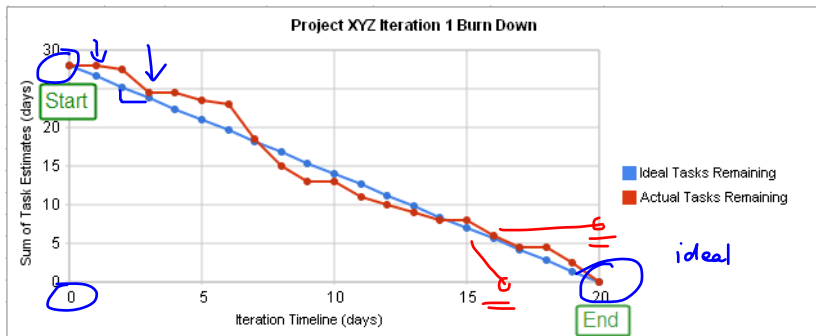


Scrum



Usually practices are from XP

Burn Down Charts



Lean Software Development

- ▶ Lean Production:
 - ▶ Reduce the amount of waste
 - ▶ Generate flow → *streamlined processes*
- ▶ Waste: resources used with does not produce value for the customer
 - ▶ time needed to fix bugs
 - ▶ time to change the system because it does not fit the customers requirements
 - ▶ time waiting for approval → *Cannot be avoided always*
 - ▶ ...

Cycle time

Cycle time

Time it takes to go through the process one time

$$cycle_time = \frac{number_of_features}{feature_implemantion_rate}$$

Batch size = number_of_features in an iteration

Cycle time: Waterfall

- ▶ Software: 250 features, ~~50 weeks~~,
feature_implementation_rate = 5 features/week

$$cycle_time = \frac{number_of_features}{feature_implemantion_rate}$$

- ▶ Waterfall: cycle_time = 250 / 5 = 50 weeks
- 1 cycle
- ▶ Question: How to reduce the cycle time?
 - Get feedback from the process

Reducing the cycle time

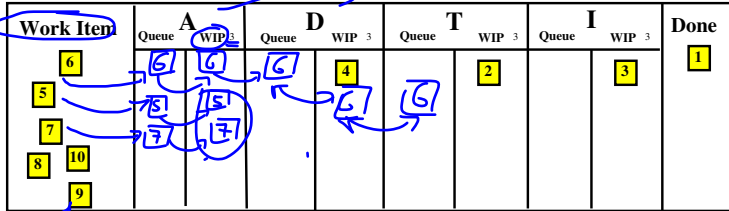
- ▶ Software: 250 features, 50 weeks,
feature_implementation_rate = 5 features/week

$$cycle_time = \frac{number_of_features}{\underline{feature_implemantion_rate}} \uparrow$$

- ▶ Agile: cycle_time = 1 / 5 = 8 hours
- 250 cycles
- Process improvement: incease in features / week

Generating flow using Pull and Kanban

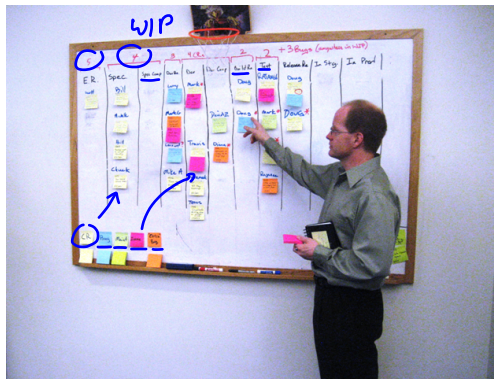
WIP = Work in Progress Limit



User story



Flow through Pull with Kanban



- ▶ Process controlling: local rules
- ▶ Load balancing: Kanban cards and *Work in Progress (WIP) limits*
- ▶ Integration in other processes: e.g. Scrum + Kanban = Scrumban

• Visual rep. of the process: Discover problems

Online Tool

- ▶ www.targetprocess.com: Electronic Kanban board useful for your project

Contents

Software Development Process (cont.)

From Requirements to Design: CRC Cards

Version control

From Requirements to Design

Design process

- 1 Glossary/architecture: possible classes, attributes, and operations
- 2 Take one use case scenario / user story
 - a) Devise a test for the scenario
 - b) Realize that scenario by adding new classes, attributes, associations, and operations so that you design can execute that scenario
 - c) implement
- 3 Repeat step 2 with the other use case scenarios / user stories

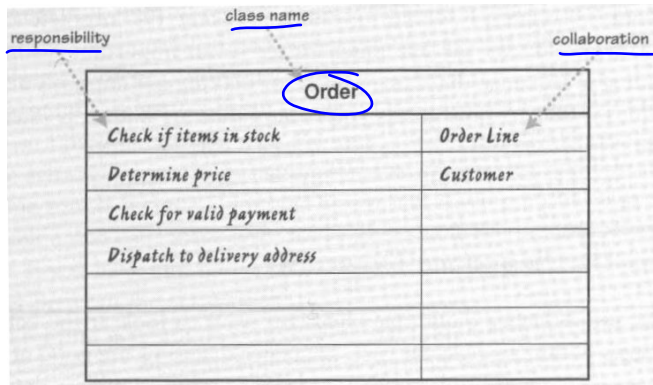
UML Class diagram



Introduction CRC Cards

- ▶ Class Responsibility Collaboration
- ▶ Developed in the 80's *by Kent Beck and Ward Cunningham*
- ▶ Used to
 - ▶ Analyse a problem domain
 - ▶ Discover object-oriented design
 - ▶ Teach object-oriented design
- ▶ Object-oriented design:
 - ▶ Objects have state and behaviour
 - ▶ Objects delegate responsibilities
 - ▶ "Think objects"

CRC Card Template



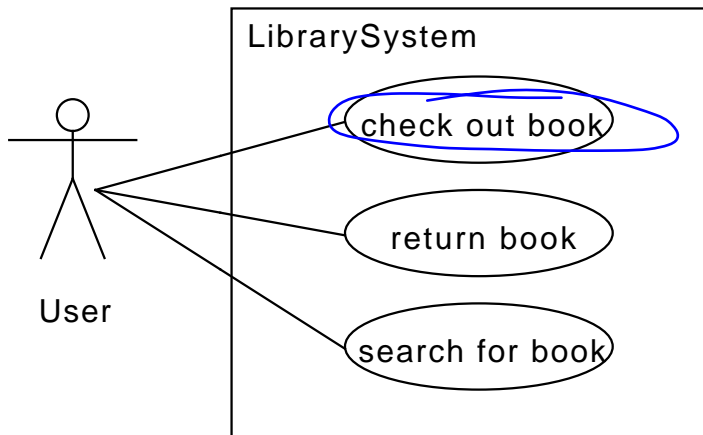
A larger example

- <http://c2.com/doc/crc/draw.html>

Process

- ▶ Basic: Simulate the execution of use case scenarios / user stories
- ▶ Steps
 1. Brainstorm classes/objects/components
 2. Assign classes/objects/components to persons (group up to 6 people)
 4. Execute the scenarios one by one
 - a) add new classes/objects/components as needed
 - b) add new responsibilities
 - c) delegate to other classes / persons

Library Example: Use Case Diagram



Library Example: Detailed Use Case *Check Out Book*

- ▶ **Name:** Check Out Book
- ▶ **Description:** The user checks out a book from the library
- ▶ **Actor:** User
- ▶ **Main scenario:**
 - 1 A user presents a book for check-out at the check-out counter
 - 2 The system registers the loan
- ▶ **Alternative scenarios:**
 - ▶ The user already has 5 books borrowed
 - 2a The system denies the loan
 - ▶ The user has one overdue book
 - 2b The system denies the loan

Example II

- ▶ Set of initial CRC cards: Librarian, Borrower, Book
- ▶ Use case **Check out book** main scenario (user story)
 - ▶ "What happens when Barbara Stewart, who has no accrued fines and one outstanding book, not overdue, checks out a book entitled Effective C++ Strategies+?"

Library Example: CRC cards

LIBRARIAN

CHECK OUT BOOK



Library Example: CRC cards

LIBRARIAN

CHECK OUT BOOK

BORROWER

Library Example: CRC cards

BORROWER

can borrow



Library Example: CRC cards

BORROWER

CAN BORROW

KNOW SET OF BOOKS

Library Example: CRC cards

BORROWER

CAN BORROW

KNOW SET OF BOOKS

BOOK

Library Example: CRC cards

Book

KNOW IF OVERDUE

1

Library Example: CRC cards

Book

KNOW IF OVER DUE

KNOW DUE DATE

1.

Library Example: CRC cards

Book

KNOW IF OVER DUE

KNOW DUE DATE

DATE

Library Example: CRC cards

DATE

COMPARE DATES



Library Example: CRC cards

DATE

COMPARE DATES

DATE

Library Example: CRC cards

LIBRARIAN

CHECK OUT BOOK

BORROWER

Library Example: CRC cards

<u>Book</u>
KNOW IF OVER DUE
KNOW DUE DATE
CHECK OUT

DATE

Library Example: CRC cards

<u>Book</u>	DATE
KNOW IF OVER DUE	
KNOW DUE DATE	
CHECK OUT	
CALCULATE DUE DATE	

Library Example: CRC cards

<u>Book</u>	DATE
KNOW IF OVER DUE	
KNOW DUE DATE	
CHECK OUT	
CALCULATE DUE DATE	
KNOW BORROWER	

Library Example: CRC cards

Book

KNOW IF OVER DUE

KNOW DUE DATE

CHECK OUT

CALCULATE DUE DATE

KNOW BORROWER

DATE

BORROWER

Library Example: All CRC cards



Process: Next Steps

- ▶ Review the result

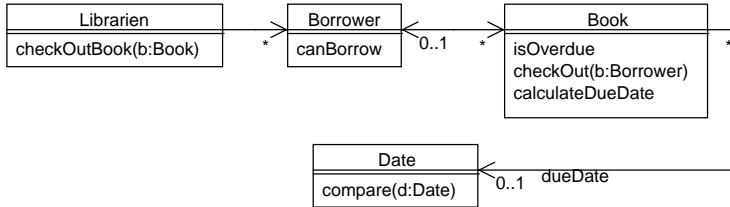
- ▶ Group cards
- ▶ Check cards
- ▶ Refactor

- ▶ Transfer the result

- ▶ Implement the design test-driven
- ▶ UML model

→ Class diagram + Sequence diagram

Example: Class Diagram (so far)



Process: Agile software development

- ▶ Take a user story (according to plan)
- ▶ Create an automatic acceptance test
- ▶ Design the behaviour of the user story using CRC cards
- ▶ Implement the ~~the~~ design test-driven

Contents

Software Development Process (cont.)

From Requirements to Design: CRC Cards

Version control

What is version control

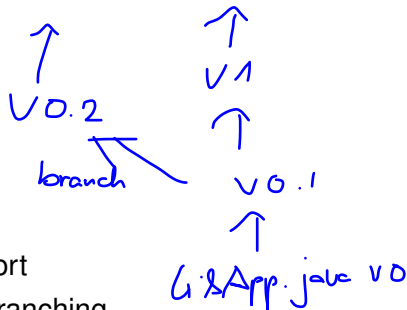
Version Control

- ▶ Stores and mangages versions documents (e.g. .java files)
- ▶ Manages concurrent work on documents
- ▶ Manages different software release versions
- ▶ Various systems: CVS, svn, git, ...

Very similar
supported by DTU

CVS

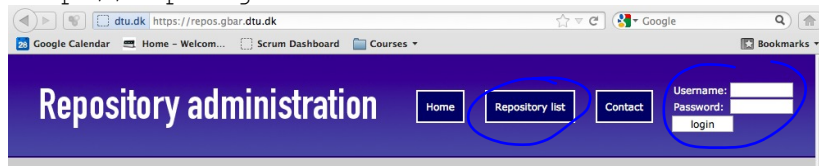
- ▶ Concurrent Versions System
- ▶ One central repository
- ▶ Command line tools, IDE support
- ▶ Files have a tree of versions: branching
- ▶ Release: File versions having same tag
- ▶ Versions: diffs (differences) to previous versions



Unix: diff & patch
Commands

Creating a repository

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



Creating a repository

Repository administration

[Home](#)[Repository list](#)[Contact](#)

Logged in as:
Hubert Baumeister
huba
[Log out](#)

Repository list

Name	Size	Type	
02161	2.6 MB	CVS	Edit - Delete - Backup

[Create repository](#)

Creating a repository

Field:	Value
Rename repository: <small>Alphanumeric characters and underscore.</small>	<input type="text" value="02161"/>
Options	Anonymous read-only access (disabled): <input type="checkbox"/>
Checkout	<div><div><code>:pserver:USERNAME@cvs.gbar.dtu.dk:/home/cvs/huba/02161</code></div><div>Please note that you need to add a user to the repository before you check it out!</div></div>
Current users:	<div><div>No users yet! [Add new user]</div><div><i>needs to have users</i></div></div>
<input type="button" value="Update Repository"/>	

Creating a repository

Edit repository ,

Field:	Value
User name:	<input type="text"/>
New password:	<input type="password"/>
Confirm new password:	<input type="password"/>
<input type="button" value="Add user"/>	

[Back](#)

Create a project and *share* it

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

paste info

Share Project

Enter Repository Location Information

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

Location

Host: cvshbar.dtu.dk

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

Authentication

User: student number

Password: [masked]

Connection

Connection type: pserver

☒ Use default port

☐ Use port: []

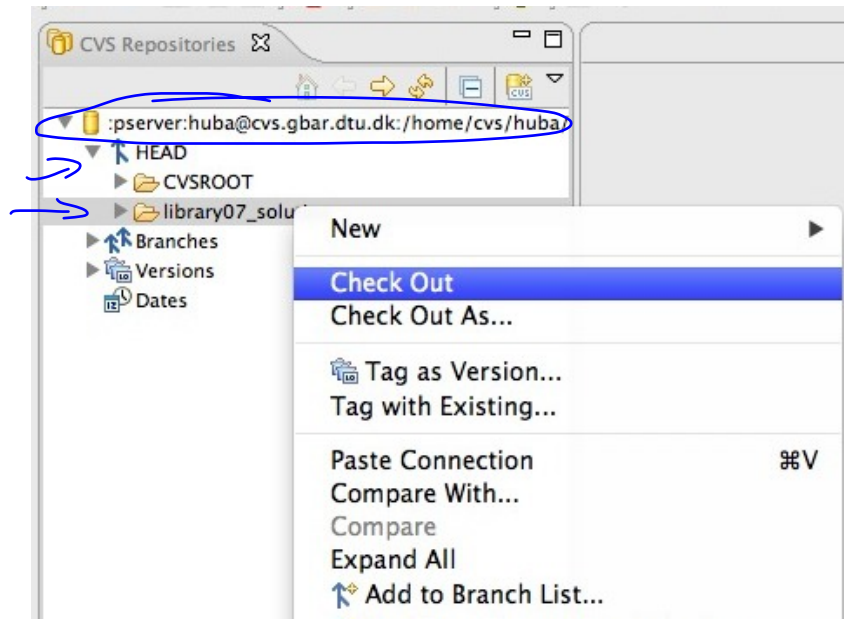
☐ Save password

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

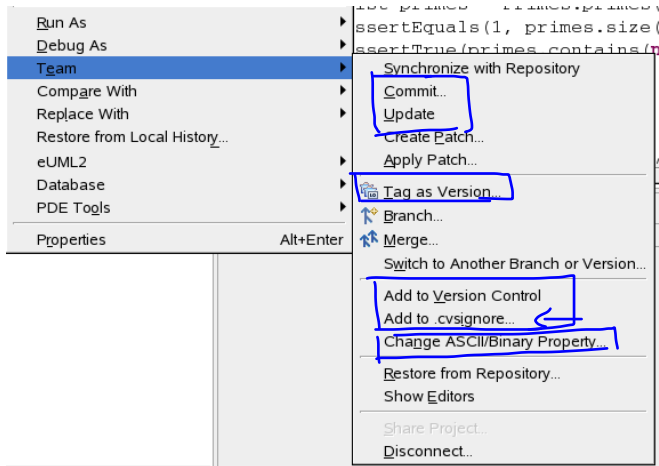
< Back Next > Finish Cancel

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 Commit 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 fail → update, merge, commit

Update a project

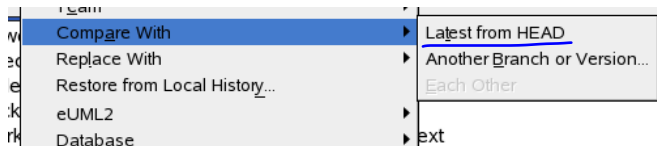
- ▶ Gets newest version of the file
- ▶ If conflicts
 - *text* files are **merged**
 - other files are **overwritten**
 - ▶ based on **lines**
 - ▶ successful merge
 - ▶ unsuccessful merge

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 getPostnummer() {  
    return postNumber;  
}
```