Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Software Engineering and Design
CS2 – Intro and Team Setup

Prof. Dr. Jürgen Vogel (juergen.vogel@bfh.ch)
Prof. Urs Künzler (urs.kuenzler@bfh.ch)

Team Setup

Case Study 2 & Team Setup

CS2: complete SE project

- in teams, you will work on the graded SE project CS2
  - the project is divided into 9 Tasks
  - each task is a typical SE task (analysis, design, implementation)
- counts 25% of overall grade
  - each task will be graded
    - based on documents, software and/or presentation
  - totaling to ~ 140 points

Team Setup

Team Setup

- N teams
  - each with a unique color codename
  - 4-5 students per team
  - self-register for a team of your choice via Moodle
- NOW

Team Rules

Team rules

- your teammates depend on your contribution
- in the coming weeks, you will meet your team regularly during and outside the class hours (depending on project tasks)
- if you cannot attend a meeting, you are required to
  - (1) tell your team members
  - (2) find out what you have missed
  - (3) deliver your contribution on time
- underperforming team members will be downgraded!

CS2 Introduction

A regional health authority wishes to procure an patient management system (PMS) to manage the care of patients suffering from mental health problems. The overall goals of the system are

1. To provide medical staff (doctors and health visitors) with timely information to facilitate the treatment of patients.
2. To support patients and their relatives in coping with the desease.

Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems. The health authority has a number of clinics that patients may attend. To make it easier for patients to attend, these clinics are not just run in hospitals. They may also be held in local medical practices or community centres. Patients need not always attend the same clinic and some clinics may support 'drop in' as well as pre-arranged appointments.

The nature of mental health problems may be that patients are often disorganised so may miss appointments, deliberately or accidentally lose prescriptions and medication, forget instructions or make unreasonable demands on medical staff. In a minority of cases, they may be a danger to themselves or to other people. They may regularly change address and may be homeless on a long-term or short-term basis. Where patients are dangerous, they may need to be 'sectioned' – confined to a secure hospital for treatment and observation.

Users of the system include clinical staff (doctors, nurses, health visitors), receptionists who make appointments and medical records staff. Reports are generated for hospital management by medical records staff. Management have no direct access to the system.

The system is affected by two pieces of legislation

1. Data Protection Act that governs the confidentiality of personal information
2. Mental Health Act that governs the compulsory detention of patients deemed to be a danger to themselves or others.

The system is NOT a complete medical records system where all information about a patients' medical treatment is maintained. It is solely intended to support mental health care so if a patient is suffering from some other unrelated condition (such as high blood pressure) this would not be formally recorded in the system.

Imagine your project team signed the contract for implementing the MHC-PMS.

Good luck ☺

Project Scope (1)

- implementing the entire MHC-PMS would be too much
- each team will therefore only address a small part of it, in the form of dedicated and independent applications specifically…
  - …for one of the following target user groups
    1. **doctor** in a clinic
    2. **health visitor** for ambulant patients
    3. (ambulant) **patient**
    4. close **relative** of a patient
  - …for one of the following diseases
    1. addiction
    2. depression
    3. social anxiety disorder
    4. burnout
- technology stack: Java-based Web application

Project Scope (2)

Pick a unique user group – disease combination

- user group
  1. **doctor** in a clinic
  2. **health visitor** for ambulant patients
  3. (ambulant) **patient**
  4. close **relative** of a patient
- disease
  1. addiction
  2. depression
  3. social anxiety disorder
  4. burnout

Team	User Group	Disease
red		
green		
blue		
yellow		
orange		
black		
white		

Next Steps

- In the RE phase, we will explore use cases and user requirements by the means of Design Thinking
- each team shall investigate the problem space from a user perspective and conduct a short interview with a person of your choice who has relevant experience as a doctor, nurse, patient, ...
- **Timeline**
  - Today: prepare the settings for an interview or field study
    - when, who and by whom, where
    - start preparing questions
  - 17.03. – 30.03. work on Task 1: Design Thinking
  - 30.03., 16:15: presentation of results

Note: If you do not have access to someone with experience in mental health care, it is also ok to investigate the general health domain.

Repository Setup

Project Files

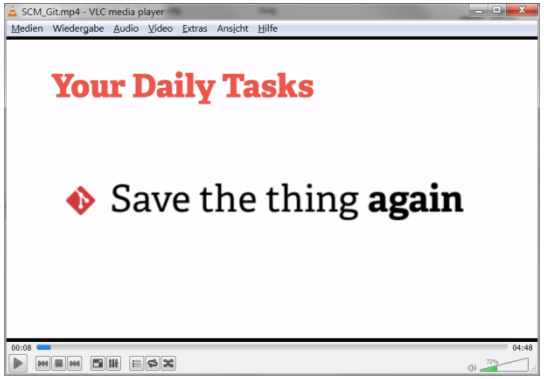
- Project files (documents, presentations, code, …)
  - managed via sharing platform
- Each team is required to keep a small project diary with
  - meetings: date/time, attendance, minutes
  - todo's: what, who, deadline, status (open, work in progress, done)
  - use a plain text document: diary.txt

IDE Setup: Java and Eclipse

- install the latest version of Java SDK (Java SE or EE)
  - Open JDK: https://jdk.java.net/
- install the latest version of Eclipse IDE for Java EE Developers
  - https://eclipse.org/

SCM / Version Control

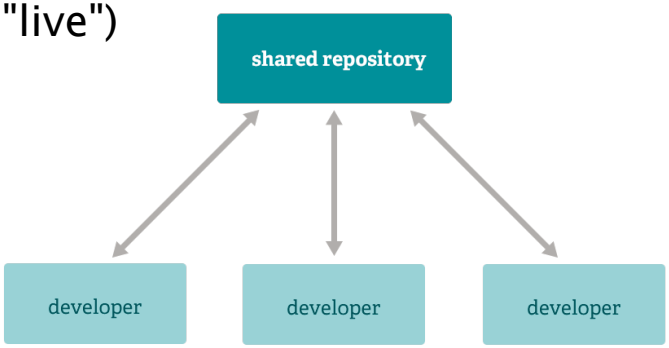
- software development is team work
  - cooperation between developers (architects, programmers, UI designers, …), project managers, and users
- source code (class, data, media, configuration, … files) nowadays is mostly managed in specialized and centralized repositories: version control system
  - also known as "Source Code Management" SCM
  - implements client-server pattern
- many different version control systems exist
  - e.g., CVS, Subversion, Git, Rational Team Concert, …
  - see http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
- all files in the version control repository get some additional meta data
  - author (of the original file and any update)
  - version (incremented with every update)
  - version comment (to be provided by the author for each new version)



Video
https://git-scm.com/video/what-is-version-control

Basic SCM Workflow

- files are not edited directly in the repository (i.e., "live") but need to be copied to a local directory first
- creating a new file version, an editor therefore must follow a specific workflow
  1) check out a certain version of the file(s)
     - in most cases the latest version
  2) modify files as necessary
  3) check if file has been updated by some other editors in the meantime
     - parallel work may happen anytime
     - if yes, check whether there are conflicting changes
       - e.g., modifying the same paragraph
     - if yes, resolve conflict (i.e., decide on final text)
       - often requires some communication with the editor
       - also known as "merging"
  4) once done, check in new version of file(s)
     - together with some meaningful comment
     - also known as "commit"



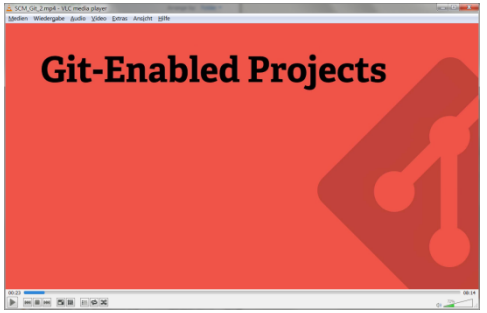
Version History

- this workflow may generate many different versions for the same file
  - also known as "history"
- editors may
  - access old versions if required (i.e., old versions are backups)
  - compare different versions of the same file to
    - review the changes of colleagues and how they affect "your" code
    - identify the cause for newly introduced bugs,
    - analyze design decisions,
    - etc.
  - work on different versions in parallel
    - also known as "branches"
    - e.g., work on a new feature in the "task x.y" branch, try out an idea in an "experimental" branch and fix a bug in the "main" branch
- based on a set of files with specific version numbers, an overall version of the entire project can be defined
  - e.g., a release
  - also known as "snapshot", "tag", "label", or "baseline"

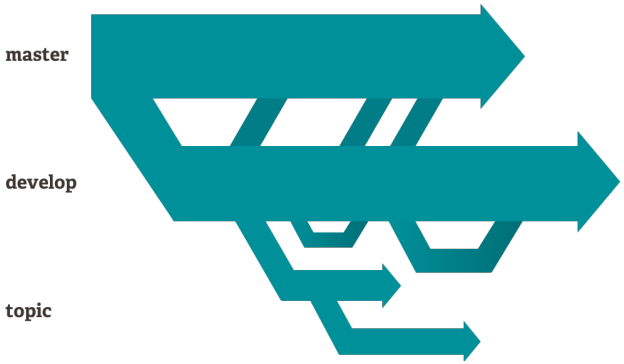
Git

Git (http://git-scm.com/)

- is a replicated SCM with 3 copies of your files
  1) Eclipse workspace
  2) local Git repository
  3) server Git repository
- has originally been developed for the open source development of the Linux kernel (by Linus Torvalds)
- core feature is parallel development in branches
  - e.g., create a branch per feature and later on decide which feature should go into the next release
  - i.e., ideas may be developed in private and are published once ready (or abandoned quietly)
- supports a 2-step commit workflow
  1. preparation ("staging"): notify Git about changed files
  2. commit: add the change to the repository



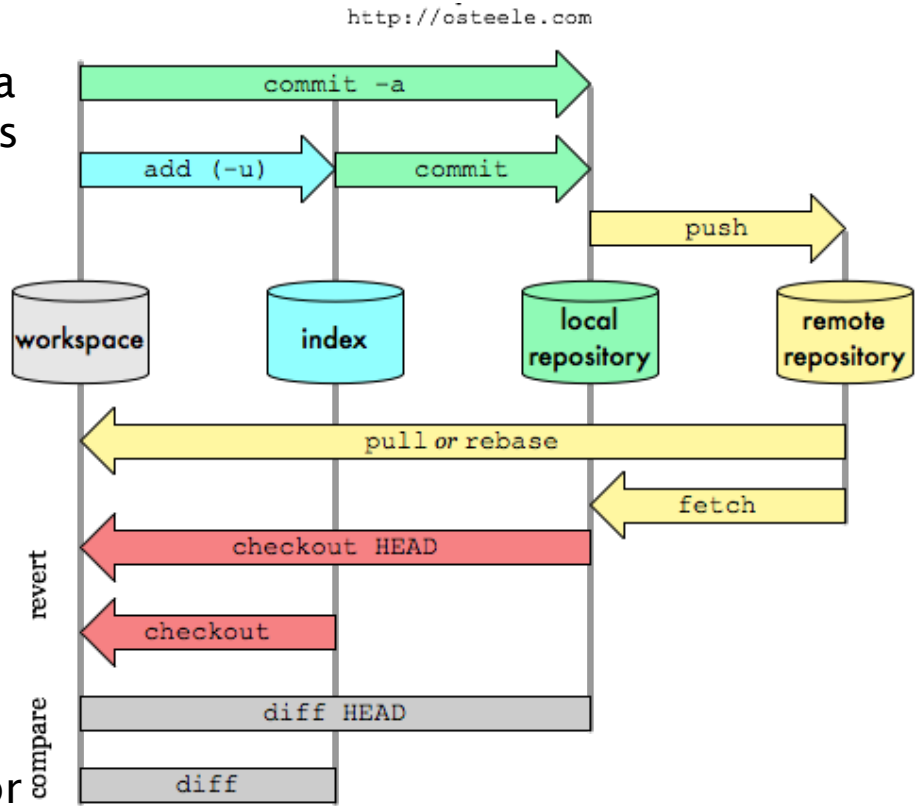
Video
https://git-scm.com/video/what-is-git



Git Commands Overview

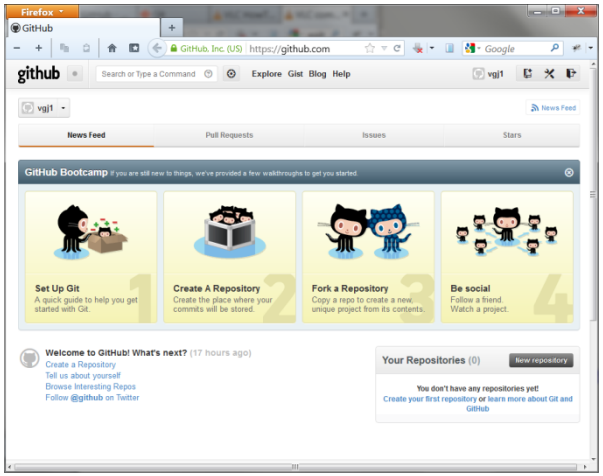
Git commands for syncing resources (in a certain branch) between these 3 locations

- (additional index used by Git to monitor changes to the local workspace)
- `add` (stage): marks the current state of a workspace resource for the next commit
- `commit`: updates the local repository with all staged resources from the workspace
- `push`: updates the resources in the remote repository with their branch version of the local repository (only for non-conflicting changes or merged resources)



- `fetch`: updates the resources in the local repository with their branch version in the remote repository
- `merge`: merges all changes (starting from the common root) from the named version of the local repository into the current branch of the workspace and marks any conflicting changes directly in the concerned files
- `pull`: executes a fetch and merge from the remote repository

Git Setup: Server



GitHub (https://github.com/)

- hosted Git Server
- usage is free for open source projects

Task: GitHub Setup

- each team member creates an account
  - use your bfh login name (e.g., vgj1)
- for each team, create a single repository
  - project: ch.bfh.bti7081.s2020.{team color}
  - enter description
  - check "public" and "initialize with readme"
- invite your team members and vgj1/UrsKuenzler to your repository
  - go to Settings -> Collaborators -> Add...
- send mail to vgj1/klu1 with list of team members and their git login name
- remember HTTPS URI to your repository ("subversion checkout URL")
  - e.g., https://github.com/vgj1/ch.bfh.bti7081.s2020.purple

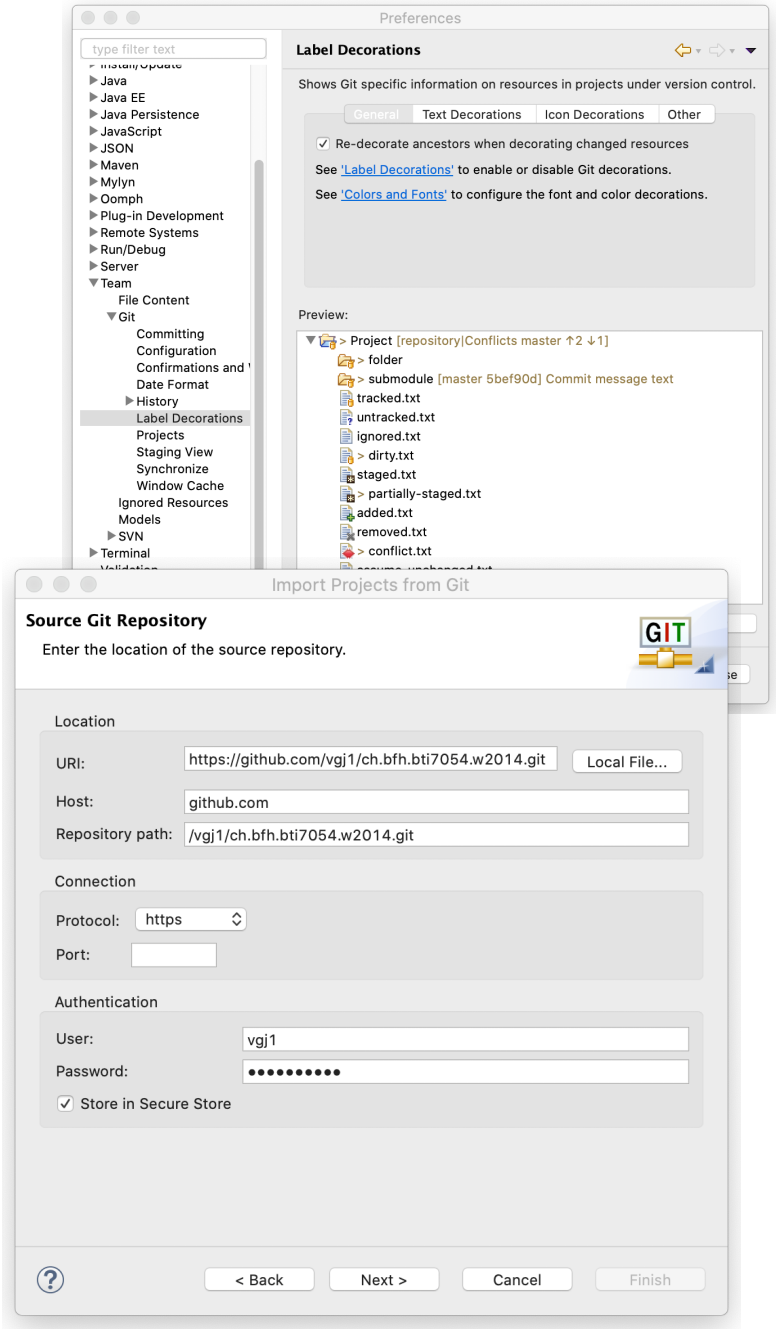
Git Setup: Eclipse Client

Eclipse offers a fully integrated Git client

- general configuration via Preferences -> Team -> Git
  - get familiar with the "label decorations" in the Eclipse project explorer
- project configuration/management in Git Perspective (Window → Perspective → …)

Task: Import Project

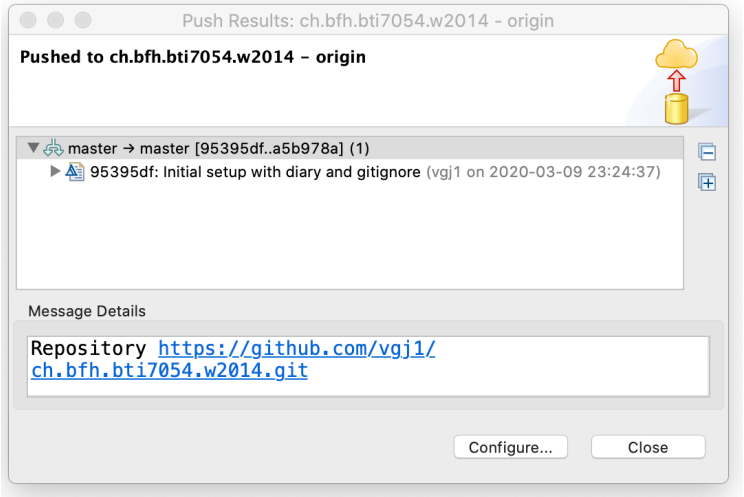
- File → Import… → Git → Projects from Git
- Clone URI → Next
- enter: URI of your project and your credentials
- select master branch → Next
- set local destination folder → Next
- select "Import as general project" → Next
- select "Finish"



Commit Project Diary

Task: Do on **one** team member's machine

- create a new folder `doc`
- create your project diary file in `doc`: `\doc\diary.txt`
- right click on project -> Team -> Commit…
- in window "Unstaged Changes"
  - select `diary.txt`
    - right click → Add to Index
  - select `.project`
    - right click → Ignore
  - select `.gitignore`
    - right click → Add to Index
- enter a meaningful comment in window "Commit Message"
- click "Commit and Push…"
- verify in GitHub Web-client



Pull Project Diary

Task: Do on the others' machines

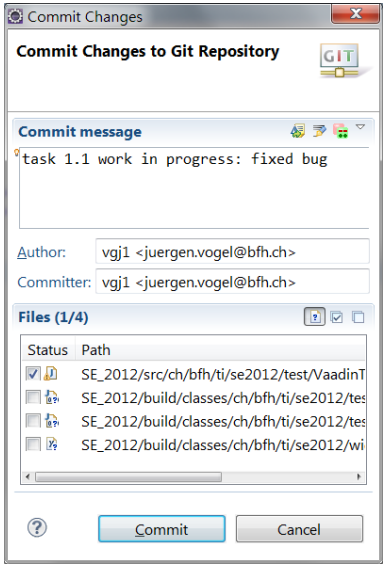
- right click on project -> Team -> Pull
- you should now see the new folder and file

Task: Handling Merge Conflicts

Task: Trigger and handle merge conflicts

- modify `diary.txt` in parallel in your team

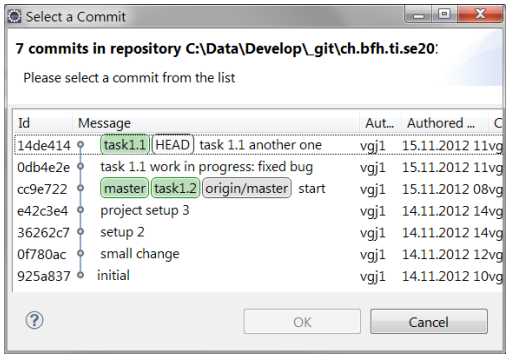
Git Workflow in Detail (1)



1. **working** (developing) and committing changes **locally**
   - now we are ready to work (=update local files)
   - intermediate changes (=not yet to be delivered to the rest of the team) may be committed anytime to the local Git repository
   - to commit, right click on project and select Team -> Commit
     - select which file updates belong to this commit
       - NEVER commit any files from the build directory!
     - always put a meaningful commit, e.g., "added new tasks to diary.html"
     - deselect the "push" option to keep your changes local
2. **reviewing the history**
   - the history gives an overview of all changes
   - right click on project, select Team -> Show in History
     - you can see all commits, comments, the author, the concerned files
   - right click on a certain file, select Team -> Show in History opens file view

Git Workflow in Detail (2)

3. **reverting local changes**
   - local changes can be reverted to an earlier version (=undo)
   - for single files (or entire project): right click on file (or project), select Replace with -> Commit (or Branch) and select the correct version



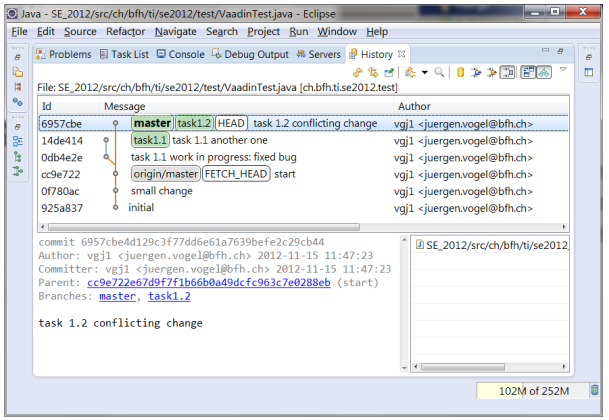
Git Workflow in Detail (3)

4. **publishing local changes**
   - if our local branch is ready to be delivered to the team, commit it to GitHub
     - before TEST whether everything works
       - do not deliver untested code to your team members: nothing is worse than checking out buggy code and having to figure out what is wrong
   - BUT: each team member may update the shared global repository at any time and there may be conflicting changes, e.g., changes to the same method

a) **update** local copy of the master branch
   - switch to "Git Repository Exploring" perspective
   - right click on the Git repository, select "Fetch from upstream" and click ok

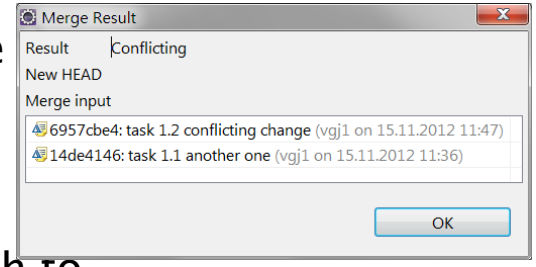
b) **merge** and resolve conflicts
   - switch to the master branch (right click on project, Team -> Switch)
   - right click on project, Team -> Merge, select branch to merge with option "commit", click "Merge"
   - if there is no conflict, you just get a status report
   - else you need to resolve the conflict (next slide)

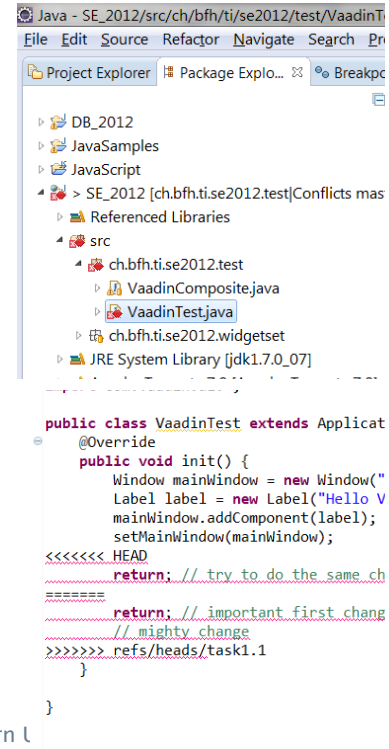
c) **commit** all changes via Team -> Push to Upstream



Git Workflow in Detail (4)

5. **resolving conflicts**
   - a conflict means: another developer has changed code in the same area as you
   - switch to the master branch (right click on project, Team -> Switch)
   - right click on project, Team -> Merge, select the branch to merge with option "commit", click "Merge"
     - if there is a conflict, you get a conflict report
   - during merge, Git has modified the source code of all files with conflicts
     - files with conflicts are marked in project explorer
     - inside the file, a special notation shows both versions
   - now you need to manually fix all these areas
     - open Merge Tool for some support (see next slide)
     - talk to your teammate to figure out the correct code
   - when everything is working (test!), commit all changes
     - right click on file, Team -> Add to Index to stage the file
     - commit / push





Git Workflow in Detail (5)

5. **resolving conflicts – merge tool**
   - right click on project, Team -> Merge Tool, select "workspace version"

- clicking on a file with a conflict opens the comparison view
  - here you can view your local version and the remote version side by side
  - and copy changes between them
  - or directly edit the code
- modify your code until all conflicts are resolved
- right click on file, Team -> Add to Index to stage the modified resource
- right click on project, Team -> Commit to push your changes to GitHub