

02161: Software Engineering I

Week 9: Version Control, Software Development Process, and Project Introduction

Hubert Baumeister

Informatics and Mathematical Modelling
Technical University of Denmark

Spring 2008



Contents

- 1 Version control
- 2 Software Development process
- 3 Introduction to the project

Contents

1 Version control

- Introduction
- Use Cases
- How to use CVS with Eclipse

2 Software Development process

3 Introduction to the project

What is version control?

Version Control

"Revision control (also known as version control (system) (CVS), 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. to different versions from two different programmers)
- 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**
 - experiment with the system (one can go always back to an earlier version)

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
 - Merging changes (**note:** only files with the ASCII attribute can be merged)
- 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

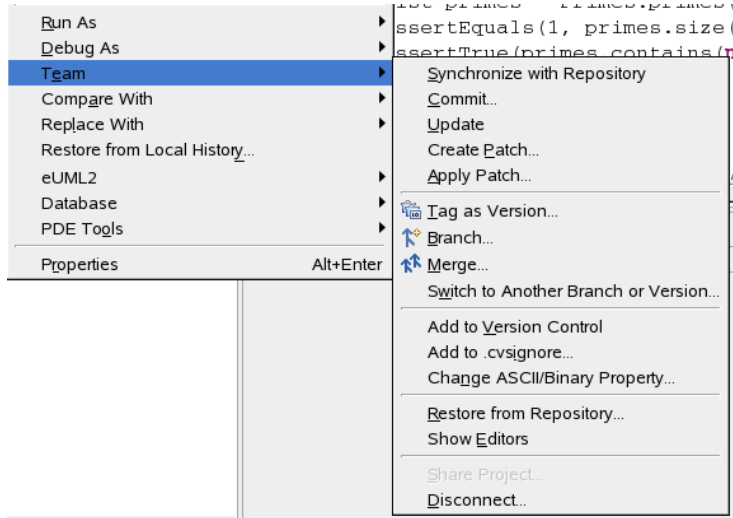
The screenshot shows the 'Share Project' dialog box in Eclipse. The title bar says 'Share Project'. The main heading is 'Enter Repository Location Information'. Below this, it says 'Define the location and protocol required to connect with an existing CVS repository.' There are four sections: 'Location' with 'Host' (cvsgbar.dtu.dk) and 'Repository path' (/home/cvs/[student no.]/[repository name]); 'Authentication' with 'User' (student number) and 'Password' (masked); 'Connection' with 'Connection type' (pserver) and radio buttons for 'Use default port' (selected) and 'Use port' (with a text field); and a 'Save password' checkbox. A warning icon and text state: 'Saved passwords are stored on your computer in a file that is difficult, but not impossible, for an intruder to read.' At the bottom are buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

- 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

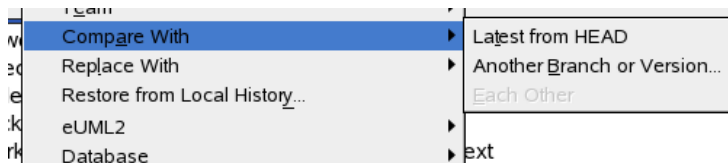


Update a project from a CVS repository

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

The screenshot shows the Eclipse IDE interface with the following components:

- Team Synchronizing - Primes.java - Eclipse SDK** (Title Bar)
- File Edit Navigate Search Project Run Window Help** (Menu Bar)
- Toolbars** (Icons for file operations and team synchronization)
- Synchronize** (Left Panel):
 - CVS (primes_III)
 - primes_III [remote.cip.informatik.uni-muenchen.de]
 - src
 - (default package)
 - Primes_JML_Testjava 1.1 - 1.1 (ASCII-kkv)
 - Primes_JML_TestData.java 1.1 - 1.1 (ASCII-kkv)
 - Primes.java 1.4 - 1.4 (ASCII-kkv)** (Selected)
 - PrimesTestjava 1.1 - 1.1 (ASCII-kkv)
 - classpath 1.2 - 1.2 (ASCII-kkv)
- PrimesTestjava Primes.java Primes.java** (Tabs)
- Java Structure Compare** (Right Panel):
 - Compilation Unit
 - Primes
 - isPrime(int)
 - primes(int)
- Java Source Compare** (Right Panel):

Local File (1.4)	Remote File (1.4)
<pre> public class Primes { /*@ public normal_behavior requires n >= 0; ensures \result <==> (n >= 2 && !(\ex */ /*@ pure @*/ public static boolean isPrime(int </pre>	<pre> public class Primes { /*@ public pure static boolean if (n < 2) return false; for (int i = 2; i < n; i++) if (n % i == 0) return f } return true; }; @*/ /*@ </pre>
- History Tasks Problems** (Bottom Panel)

Committing changes to a CVS repository

- Use commit from the team menu
- You are required to give a comment
- Commit fails if some 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

Contents

- 1 Version control
- 2 **Software Development process**
 - Introduction
 - Project Planning
 - Project Plan Example
 - Executing a plan
- 3 Introduction to the project

Software Development Process

- Basic steps when creating software
 - Plan the project
 - Understand the problem
 - Build the solution
 - Test the solution
 - Maintain the solution
- In a lot of cases understanding the problems requires building the solution
 - Only in the interplay between trying to understand the problem and trying to find a solution and testing it, one begins to really understand the project
 - Try to get **feedback** as early as possible
 - Show the customer your models, UI mock ups, implementations, tests as early as possible and as often as possible
 - But also yourself, you need to get feedback from building the various models, implementations, and tests
 - "Whenever you write a model, code, test, . . . , you learn more about the problem and its solution"



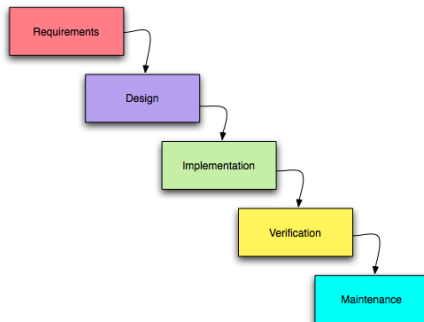
Software Development Process

- Each of the steps has its associated set of **techniques**
 - Understand the problem:
 - Build use cases
 - Interview the customer
 - create a domain language (glossary, class diagram containing the domain terminology and its relationship, scenarios, ...)
 - ...
 - Build the system
 - Use of class diagrams, sequence diagrams, state machines
 - Use of patterns
 - ...

Software Development Process

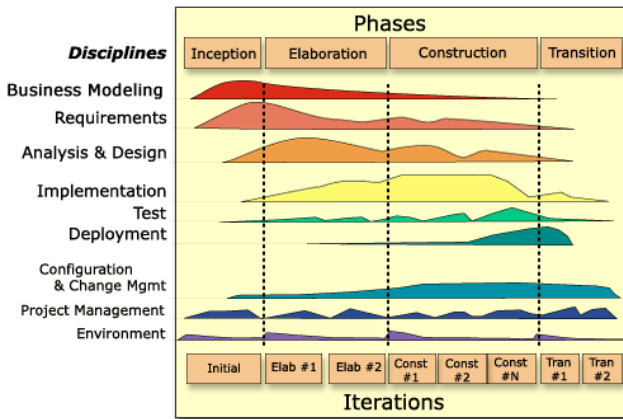
- However, the techniques can be applied in different orders
 - Different software development processes
 - e.g. Waterfall and Iterative processes (e.g. Rational Unified Process (RUP) or Extreme Programming (XP))

Waterfall model



- An activity has to terminate before the next activity begins
 - No feedback possible from the later activities
 - Takes too long time for the system to be build which does not allow the customer to give feedback

Iterative Processes: E.g. Rational Unified Process



- Inception, Elaboration, Construction, Transition corresponds to Plan the project, understand the problem, build the solution, test the solution, maintain the solution
 - All activities occur throughout the project

Techniques for planning your project 1

- **Step 1** Determine a set of scenarios (aka User Stories or Use Case scenarios) that your system should be able to do
 - Usually comes from the problem description
 - Do a brain storming on the requirements (use cases)
 - What are the scenarios? (success, failure, ...)
 - Is the set of use cases complete?
- **Use case**
 - A collection of scenarios with a common goal
 - Usually the common goal represents a functionality of the system
 - Each functionality can have different scenarios: success, failure, ...
- **User Story**
 - A story about the use of the system told from the viewpoint of the user
 - This is more like an end-to-end story
 - Can include different use case scenarios
 - A user story can be a **use case scenario**


Techniques for planning your project 2

- **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
 - How many interacting programs?
 - Where do the programs run?
 - How do they communicate?
- **Step 3** Estimate the User Stories
 - How long, in ideal man hours, do you think you need for implementing the user story?
 - Multiply this with a load factor of 2 to get the real man hours
 - This estimation includes
 - Drawing all the diagrams
 - Documenting the use case scenarios involved
 - Designing (class diagrams, sequence diagrams)
 - Implementing
 - Testing
 - Updating the report
 - ...

Techniques for planning your project 3

- **Step 4** Determine how long an iteration should take: e.g. 1 week
 - This gives you the amount of hours you have available in a week
- **Step 5** Assign user stories to iterations
 - Basically based on **customer value**
 - if two user stories have the same value to the customer, prefer the one, from which you learn more, e.g. about the resulting architecture
 - Choose as many user stories as fit into the iteration
 - Sum up the real man hours for a user story; the sum should be less than the hours a week assigned to an iteration
 - The use stories are the **milestones** of the iteration

Techniques for planning your project: Remarks

- The planning should include the writing of the report!
 - You should reserve time in the estimation for writing the report part relevant to that user story
 - Create own **task** which get their estimated man hours and schedule them in an iteration
- **Plan needs not be perfect!**
 - Don't spent **to much** time
 - Don't get stuck in the planning paralysis
 - **Experience** with the problem and its implementation **changes** the plan
 - Plan needs to be **updated** every iteration
 - Rough sketch of a plan suffices
- Based on
 - Extreme Programming Explained by Kent Beck, Addison Wesley 2004
 - Planning Extreme Programming by Kent Beck and Martin Fowler,  Addison Wesley 2000

Example Plan: Adventure Game

Adventure Game

- Goal: Implementing a command line based adventure game
- A player moves through a collection of rooms and levels. Within each room, the player can look, take up object, and put down objects.

Example Plan: Step 1: User cases ...

- Use cases
 - register
 - change room
 - start game
 - advance to next level
 - look room
 - look inventory
 - handle object

Example Plan: Step 1: ... and User stories

- User stories
 - Player registers successful for the game
 - Player registers, but name is already used
 - Player starts game
 - Player finishes game
 - Player advances to next level
 - Player looks into a room
 - Player moves to adjacent room
 - Player wants to move to adjacent room, but is not allowed to
 - Player takes up object
 - Player lays down object
 - ...

Example Plan Step 2: Basic Architecture

- One program with a command line interface
- The program prints out descriptions and the use enters commands

Example Plan Step 3: Estimate User Stories

- User stories

- Player registers successful for the game
 - 2h
 - 4 h
- Player registers, but name is already used
 - 2h
 - 4 h
- Player starts game
 - 2h
 - 4 h
- Player finishes game
 - 2h
 - 4 h
- Player advances to next level
 - 2h
 - 4 h
- Player looks into a room
 - 2h
 - 4 h
- ...

Example Plan Step 4: Iteration Lengths

- Resources
 - 2 people
 - 10 hours a week
- Iteration lengths: 1 week
 - Available resources in an iteration 20h
- The first release should be delivered after 4 weeks
 - thus it contains 4 iterations

Example Plan Step 5: Assigning user stories to iterations

- Iteration 1
 - Planning
 - Creating the base structure for the report (1h → 2h)
 - Player starts game
 - Player looks into a room
 - Player moves to adjacent room
- Iteration 2
 - Player advances to next level
 - Player finishes game
 - Player takes up object
 - Player lays down object
 - Player registers successful for the game
 - ...

Example Plan Step 5: Assigning user stories to iterations

- Iteration 3

- Player registers, but name is already used
- Writing the introduction to the report (2h → 4h)
- ...

- Iteration 4

- Player wants to move to adjacent room, but is not allowed to
- Finishing the report (4h → 8h)
- ...

How to run the project: For each iteration

- Update the plan
 - Check if there are open user stories from the last iteration
 - Incorporate feedback from the user
 - Any new user stories?
 - Did the customer change user stories or the priority of user stories?
- Find out how to implement the user stories for this iteration
 - Create tasks and distribute tasks
 - Brainstorm on how to implement the user stories
 - E.g. use CRC cards
 - Update model **and report**
 - use case descriptions
 - use case diagram
 - glossary
 - class diagram
 - sequence diagram
- Implement and test the user stories
- If there is the danger that not all user stories can be finished, concentrate on those **that can be finished** instead of leaving all user stories unfinished

Contents

- 1 Version control
- 2 **Software Development process**
 - Introduction
 - Project Planning
 - Project Plan Example
 - Executing a plan
- 3 Introduction to the project

Introduction to the project

- What is the problem?
 - Project planning and time recording system
- What is the task?
 - Create a
 - Requirement specification
 - Programdesign
 - Implementation
 - Tests
- Deliver a
 - report describing the requirement specification, design, and implementation
 - CD containing the source code, the tests, and the running program

Organisational issues

- Groups with 2, 3, or 4 students
- Report should be written in Danish or English
- Program should be written in Java and tests should use JUnit
- On May 13 there there will be a short (10min) demonstration of the program in the E-databar
 - At least the tests need to be demonstrated
- Report and CD is to be delivered during the demonstrations on May 13
- Each section, diagram, etc. should name the author who made the section, diagram, etc.

Organisational issues

- You can talk with other groups 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
- Latest **Friday 18.4 18:00** must each project group have created a group on CompusNet
 - The project groups members should be members of the group
 - You have to invite the teaching assistants David and Thomas (one of them will deny the invitation) and me (Hubert) to the group
 - By that date must be the project plan put on the CampusNet