Software Development Review

Lecture 5a

Agenda

• Building Java projects (Ant)
• Software version management (SVN)
• Software testing
Motivation

• Write a very simple Java program:
  HelloWorld.java
• Compile the program
  javac HelloWorld.java
• Execute the program
  java HelloWorld
• Fine, if we have a single program and no other library dependencies

A software “project”

• Complex code project
  – Consists of several source files
  – Compile-time and run-time dependencies
• Source files and test files
• Not just a single build instruction but allow different configuration parameters
  – Configure security, DBMS etc.
• Should run on multiple platforms
What we would like to have

• A single program/tool that can manage complex projects
• Link external libraries
• Execute external commands (copy etc.)
• Compile the project with a single command
• Separate source and class files
• Call test programs with a single command
• Run the application

Existing approaches

• **Makefile** for C/C++
  – autotools for building libraries
  – Rather OS dependent
• **Ant** for Java
  – Can also be used for other programming languages
• Several “homegrown” software packages
• Things in common:
  – Compilation, linking, execution is automated
Ant

- Platform independent
- Based on XML rather than script like interface
  - Basically consists of
    - Targets
    - Tasks
    - Properties

```xml
<project>
  <target name="clean">
    <delete dir="build"/>
  </target>

  <target name="compile">
    <mkdir dir="build/classes"/>
    <javac srcdir="src" destdir="build/classes"/>
  </target>

  <target name="jar">
    <mkdir dir="build/jar"/>
    <jar destfile="build/jar/HelloWorld.jar"
        basedir="build/classes"/>
  </target>

  <target name="run">
    <java jar="build/jar/HelloWorld.jar" fork="true"/>
  </target>

</project>
```

Very simplistic example
Target

- A “target” defines the “action” to be done
  - Compile, create jars, run, download code
- Targets can depend on each other

  `<target name="A"/>`
  `<target name="B" depends="A"/>`

- Targets can are user defined

Task

- A “task” is a piece of code to be executed.

  `<TaskName attribute1="value1"
    attribute2="value2" ... />`

- Several, OS-independent built-in tasks such as
  - Copy, chmod, echo, jar, java, zip

  `<copy todir="../new/dir">
    <fileset dir="src_dir"/>
  </copy>`
Property

- A “property” is basically a variable that can have a certain value
- Properties can be set via the property task
  
  ```xml
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  ```
- Usage:
  ```bash
  javac srcdir="${src}" destdir="${build}"
  ```
Agenda

- Building Java projects (Ant)
- Software version management (SVN)
- Software testing

Motivation

- Software is usually developed by a team
  - Developers are often distributed
  - Do not want to exchange single files
- Different versions of the software need to be supported
- Provide access to the source code

Subversion Access

This project’s SourceForge.net Subversion repository can be checked out through SVN with the following instruction set:

svn co https://phpmyadmin.svn.sourceforge.net/svnroot/phpmyadmin
phpmyadmin
Basic desirable features

• Single repository where the code is available (*master copy*)
• Developers can send their code parts (commit code to repository)
• Get latest code (checkout code)
• Check for differences to latest version
• Roll back to previous version
• Detect and repair conflicts
• Create software branches

Two main tools

• CVS (Concurrent Version System)
  – Very popular
  – Version on single files
• SVN (Subversion)
  – Very similar to CVS
  – Seems to replace CVS
  – Version on entire repository
  – Main focus here
Keywords

• Head
• Branch
• Tag
• Version
• Check out

Simple Example

• Example code repository:
  – http://example.com/OurProject
• Check out the code
  
  **svn checkout** http://example.com/OurProject

  A  OurProject/index.html
  A  OurProject/src/Company.java
  A  OurProject/src/Database.java
  A  OurProject/src/build.xml
  A  OurProject/README.txt

  Checked out revision 27.

  Additionally, .svn gets created
Edit code and commit

```
svn status
M src/Database.java
```

- The file has been changed, indicated by M (modified)
- Need to commit the code

```
svn commit src/Database.java -m "Fixed JDBC bug"
Sending src/Database.java
Transmitting file data.
Committed revision 28.
```

Add new files to repository

```
svn add src/Trader.java src/Client.java
A src/Trader.java
A src/Client.java
```

- Then, need to commit again:

```
svn commit -m "add code for trading place"
Adding src/Trader.java
Adding src/Client.java
Transmitting file data ..
Committed revision 29.
```
Other important commands

- **diff**: retrieve differences between local files and repository
- **update**: retrieve changes from repository. *Use this command before you commit!*
- **delete, rename**
- **revert**: undo all changes in a dir

Version numbers

- Typically, (open source) software is versioned like follows:
  - Major.Minor.bugfix
    - Major: major changes - no compatibility to previous versions; interface changes
    - Minor: minor changes/features that typically do not change the interface
- How does this relate to SVN?
  - Create **tags** for versions
  - **Branches** for bug fixes for a certain version
A running example (1)

• Create a repository for a project called “OurProject”

```
svnadmin create /tmp/OurProject
```

```
ls /tmp/OurProject
conf dav db format hooks locks README.txt
```

A running example (2)

• Edit your source code

```
/tmp/myDir/branches/
/tmp/myDir/tags
/tmp/myDir/trunk
/tmp/myDir/trunk/OurProject.java
```

• Import it to the code repository

```
svn import /tmp/myDir file:///tmp/OurProject -m
“add code to repository”
```
A running example (3)

• Developer 2 checks out the code to his working directory:

```bash
svn co file:///tmp/OurProject
A OurProject/trunk
A OurProject/trunk/OurProject.java
A OurProject/branches
A OurProject/tags
Checked out revision 1.
```

A running example (4)

• Developer 2 changes a file and commits the changes:

```bash
csvn commit -m "small change"
Sending    trunk/OurProject.java
Transmitting file data .
Committed revision 2.
```
Remarks

- Use SVN also for your project
- Every user can have her own, local development tree
- Note:
  - If SVN runs on your local machine, the data is not backed up!
  - In practice, a code repository is well backed up and accessible via HTTPs or SSH

Agenda

- Building Java projects (Ant)
- Software version management (SVN)
- Software testing
Motivation

• Testing software is essential
• Theory and practise would fill another lecture
• However, try to adhere to some basic rules

Basic Rules

• Write a test class for each class
  – Eventually use JUnit
• Test all possible input parameters and see how the behaviour changes
• Tests should not be done by the person that wrote the original class!
• Testing should be done to find errors!
  – Don’t test for correct results only!
Conclusions

• Use building tool for software projects
• Use a version management system (SVN, CVS, etc.)
• Test all your classes like you document all your classes