Subversion

Open source version control

EHSDI

eBuzima

Why version control?

- If more than one developer is working on a project – how can they all make changes to the source code...
  - How can one developer be sure they have the latest versions of another developers files?
  - How can a developer be sure when they overwrite a file that they aren't losing someone else's changes?
- Version control solves these problems by managing versions of source code files – i.e. every change gets a *revision number*

Why version control?

- Version control also gives us a history of a software project
  - If we accidently delete some code, we can look for an older version that still has the code
  - We can see who made what changes
  - Serves as a backup

History

- **Source Code Control System (SCCS)**: the first version control software developed by Bell Labs in 1972 for UNIX systems
- **Revision Control System (RCS)**: developed in the 80s as a free and evolved alternative
- **Concurrent Version System (CVS)**: based on RCS, it added better project management and branching
- **Subversion (SVN)**: started in 2000 to fix bugs and add features to CVS

Terminology

- **Repository**: a storage location for projects that SVN will manage
- **Checkout**: to download a copy of a project from a repository
- **Commit**: to upload files to a repository after making changes
- **Update**: to download the latest versions of files from a repository when your local copies are out of date

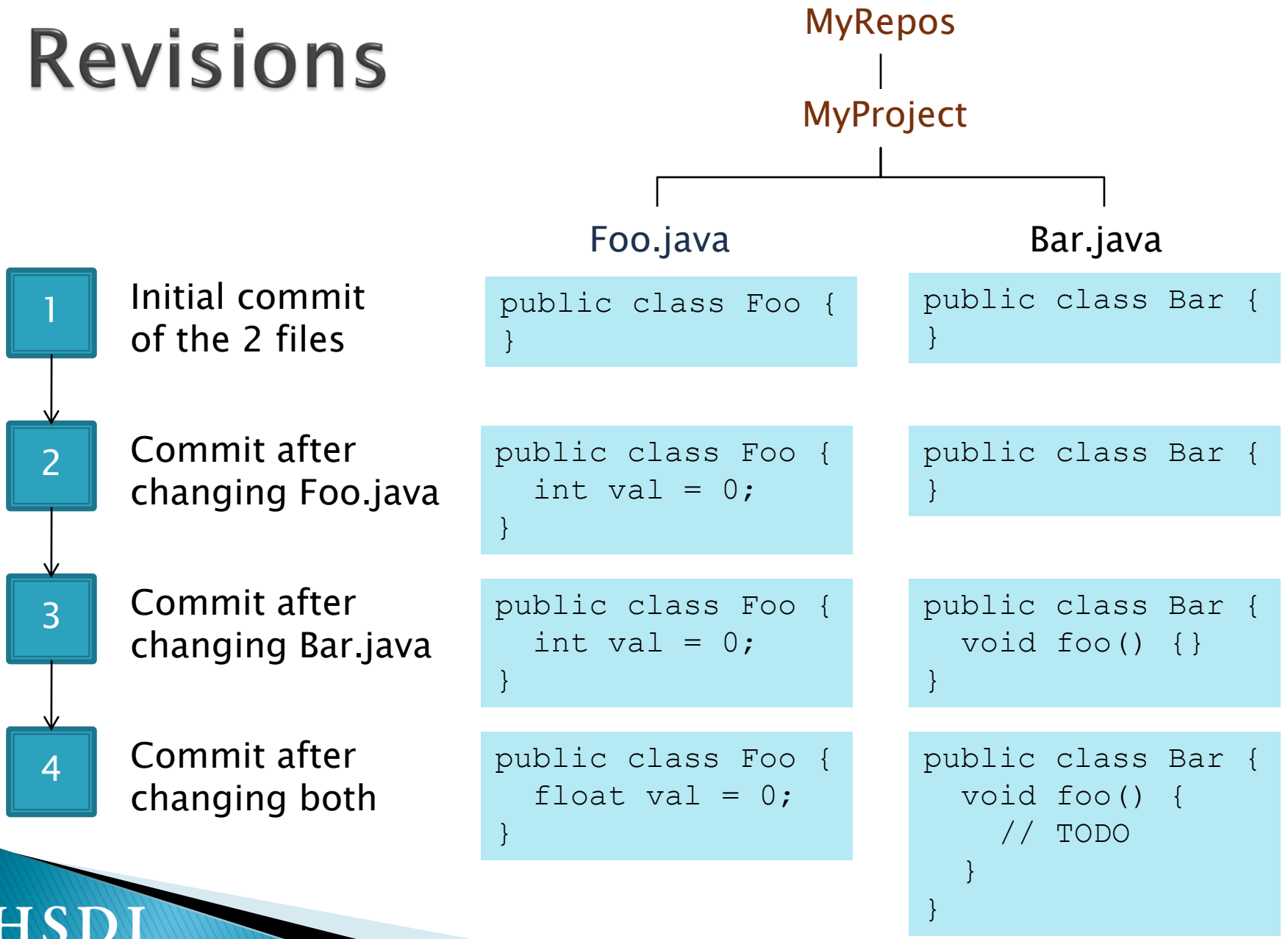
Repositories and projects

- SVN stores projects in a repository
- However a project is just a folder in a repository, like all the other folders, e.g.

```
MyRepos\
        MyProject\
                Foo.java
                Bar.java
        SomeOtherProjects\
                AnotherProject\
                        Main.java
```

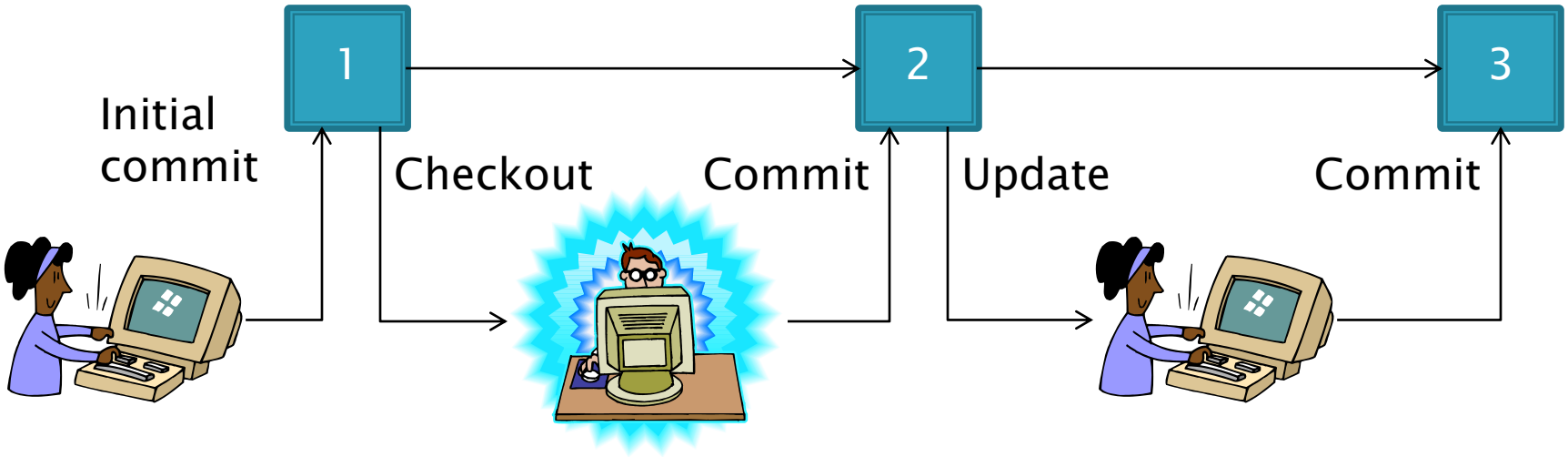
- For example: http://svn.openmrs.org/

Revisions


MyRepos
MyProject
Foo.java
Bar.java
1
Initial commit of the 2 files
public class Foo {
}
public class Bar {
}
2
Commit after changing Foo.java
public class Foo {
int val = 0;
}
public class Bar {
}
3
Commit after changing Bar.java
public class Foo {
int val = 0;
}
public class Bar {
void foo() {}
}
4
Commit after changing both
public class Foo {
float val = 0;
}
public class Bar {
void foo() {
// TODO
}
}

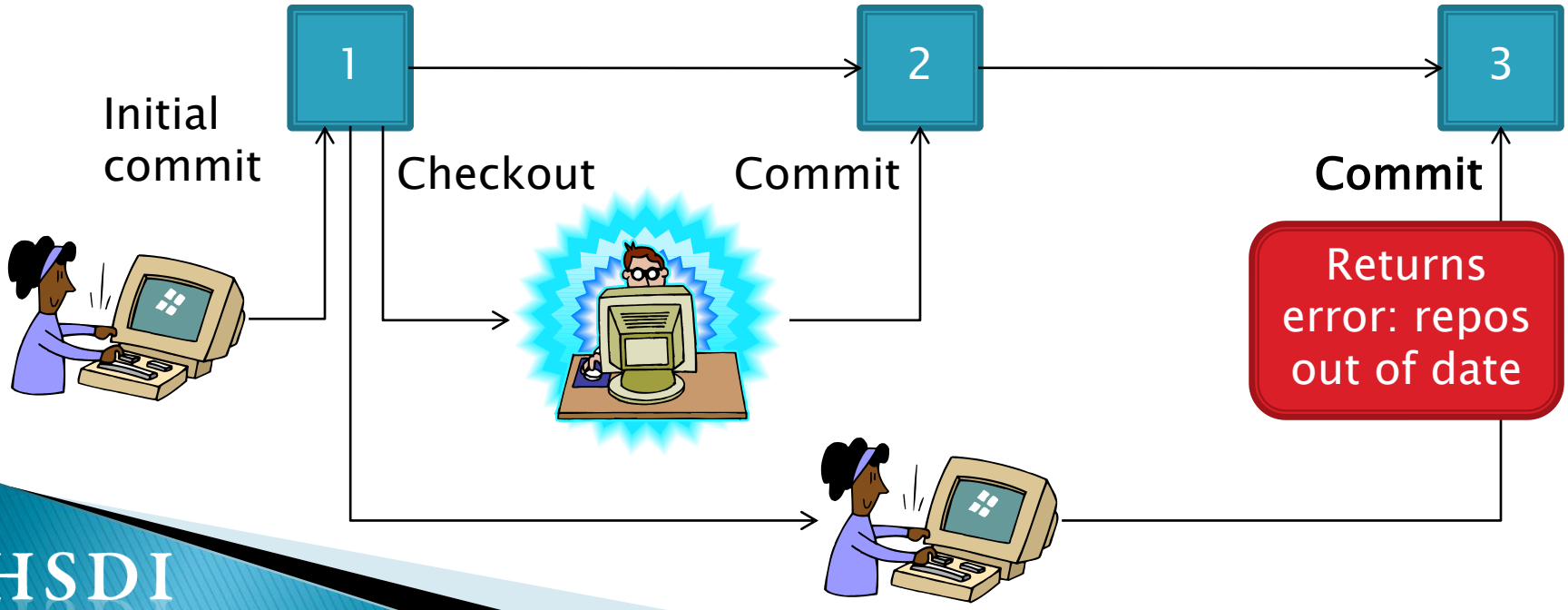
Committing and Updating

- If more than one developer is working on a project then they should update their local copy before making changes



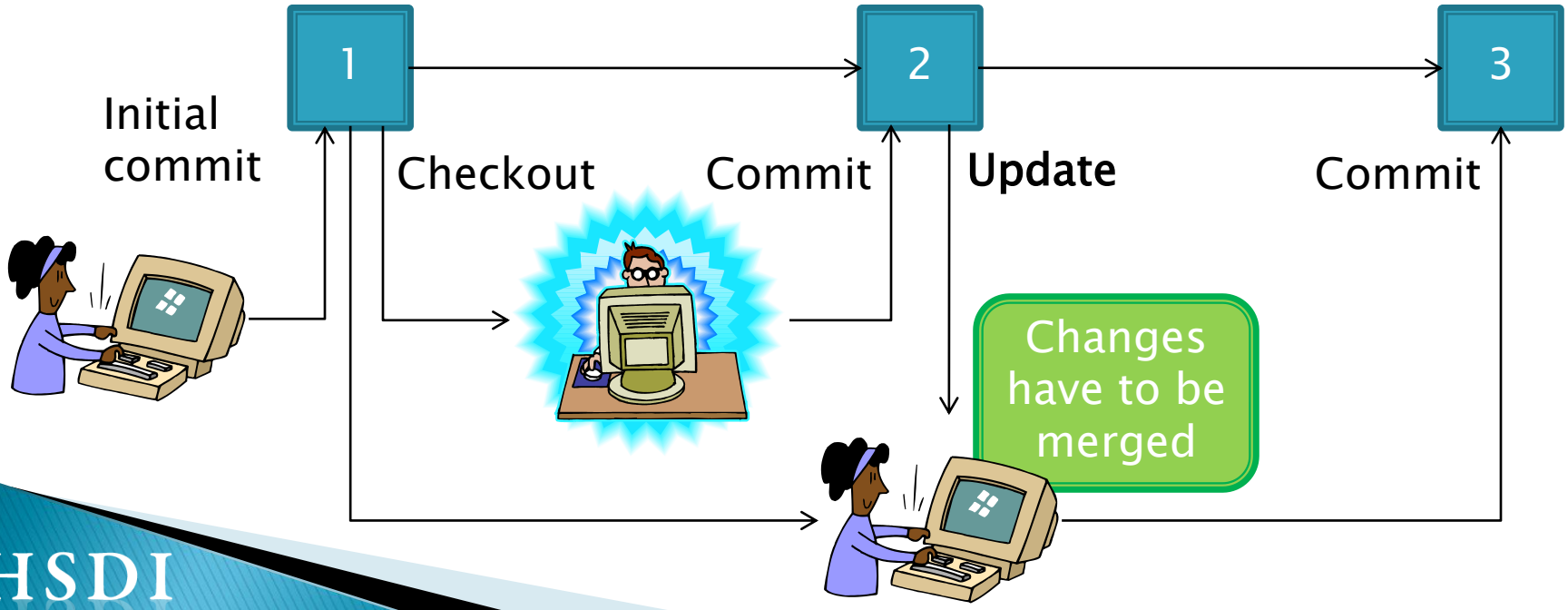
Update

- It's possible for more than one developer to have files checked out at the same time
- What if they both make changes to the same file?



Update

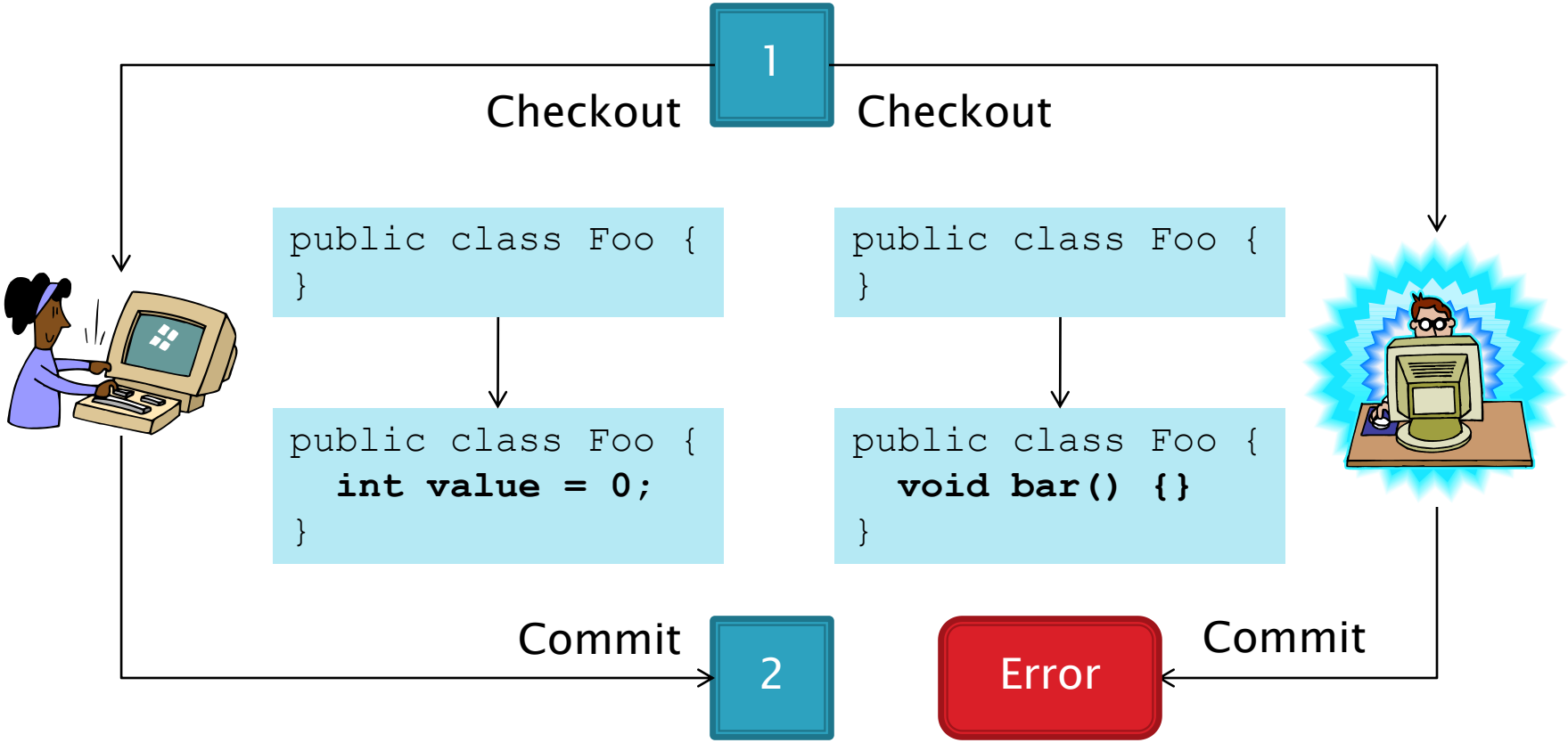
- SVN returns an error because the repository is at revision 2, but the client is checking in with revision 1
- Client must update before they can commit



Merging changes

- Updating won't lose the changes you've made locally
- The changes in the latest revision are merged with the changes in your local copy
- This merging process is done automatically and produces a file which needs fixed by the developer

Example


1
Checkout
Checkout
public class Foo {
}
public class Foo {
}
public class Foo {
int value = 0;
}
public class Foo {
void bar() {}
}
Commit
2
Error
Commit

```
public class Foo {
   int value = 0;
}
```

```
public class Foo {
   void bar() {}
}
```

Commit

2

Update

`svnmerge` tool creates a version of the file with both sets of changes

```
public class Foo {
<<<<<<< .mine
   void bar() {}
=======
   int value = 0;
>>>>>>> .r4
}
```

svnmerge

developer then manually merges the changes into a new working version

```
public class Foo {
   void bar() {}
   int value = 0;
}
```

3

Commit

Commit comments

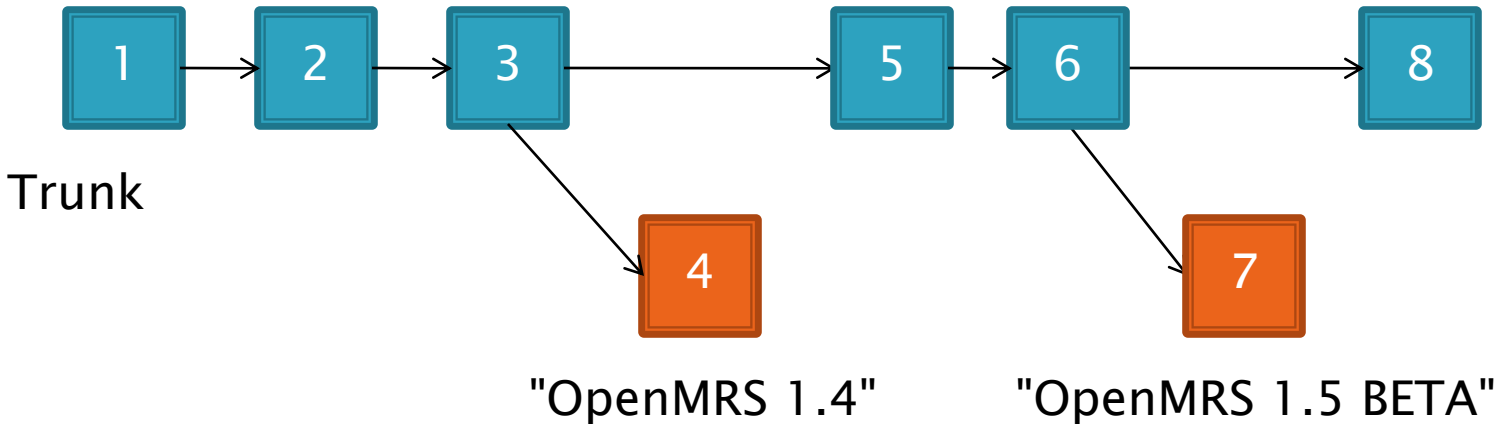
- Every commit has a comment which should tell other developers very briefly what has changed
- Because a commit can only have one comment for all the files being committed, it is often necessary to commit files individually or in smaller groups

Project structure

- Projects within a repository are usually organized into trunk, branches and tags
- Trunk is the main code base of the project

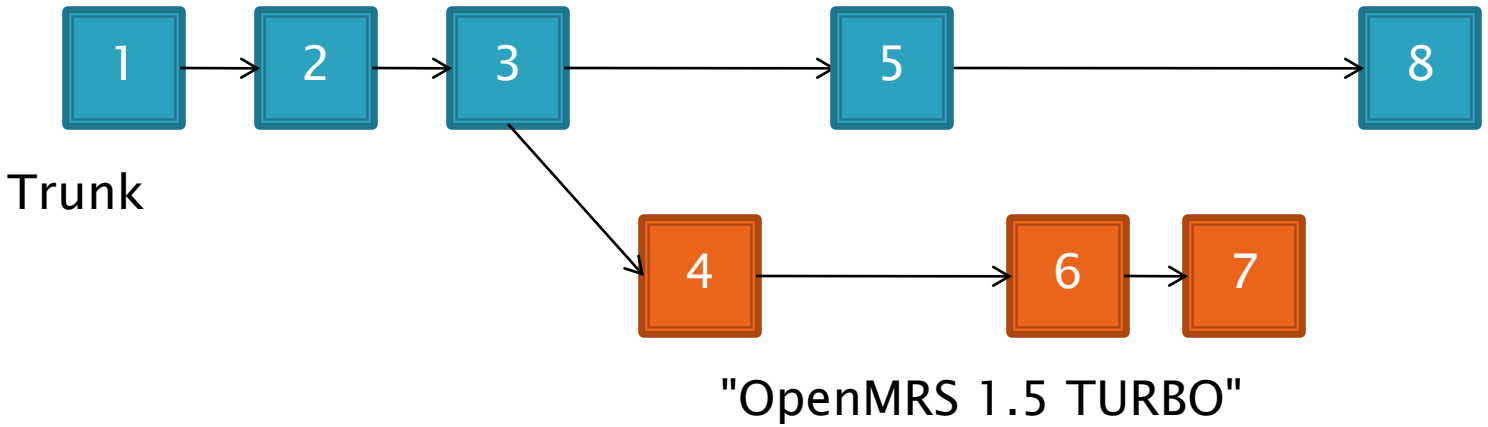
Tags

- A tag is a snapshot of a project at a specific revision
- Allows us to give a meaningful name to a revision, e.g. "OpenMRS 1.4"



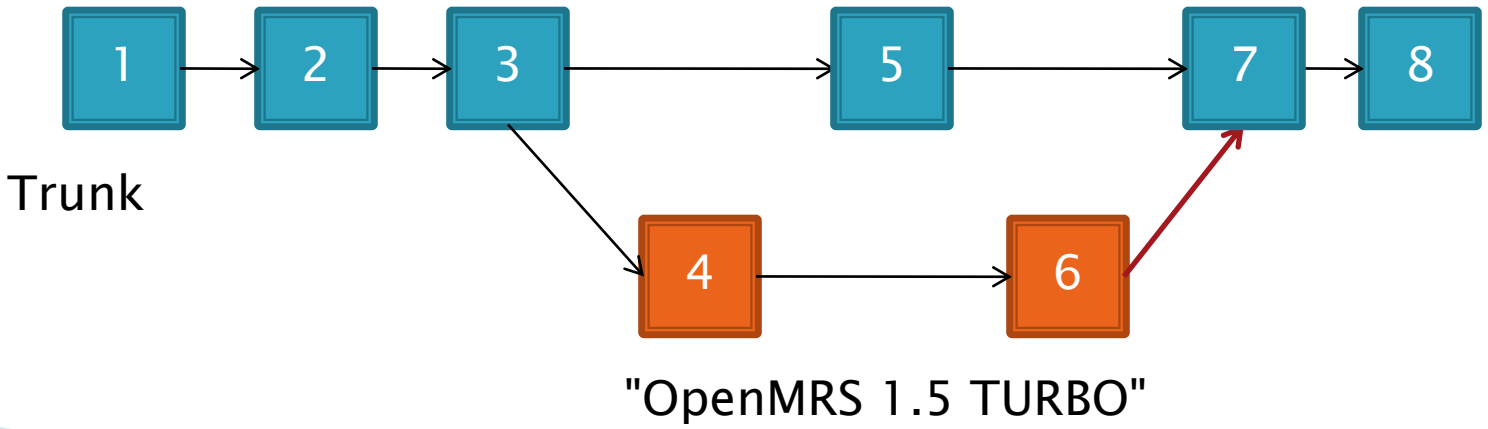
Branches

- Unlike a tag, a branch is a copy which will be modified
- Allows changes to be made in parallel to the originating branch, which is usually the trunk



Merging

- Branches can be merged
- Often features added to a branch will be merged into the trunk
- A branch that won't be merged back into the trunk is called a *fork*



Repository access

- SVN supports several different access methods:
- **Filesystem**
  - Local e.g. `file:///path/to/repos`
  - Remote e.g. `file://host/path/to/repos`
- **HTTP – using WebDAV with Apache 2**
  - HTTP e.g. `http://host/url/to/repos`
  - HTTPS e.g. `https://host/url/to/repos`
- **SVN protocol**
  - Unencrypted e.g. `svn://host/url/to/repos`
  - With SSH e.g. `svn+ssh://host/url/to/repos`

Creating a respository

1. Create a directory to hold the repository
2. Use `svnadmin` to initialize it

For example: creating a repository in a Linux home directory...

```
> cd ~
> mkdir svn
> svnadmin create svn
```

Repository can now be accessed using the file protocol...

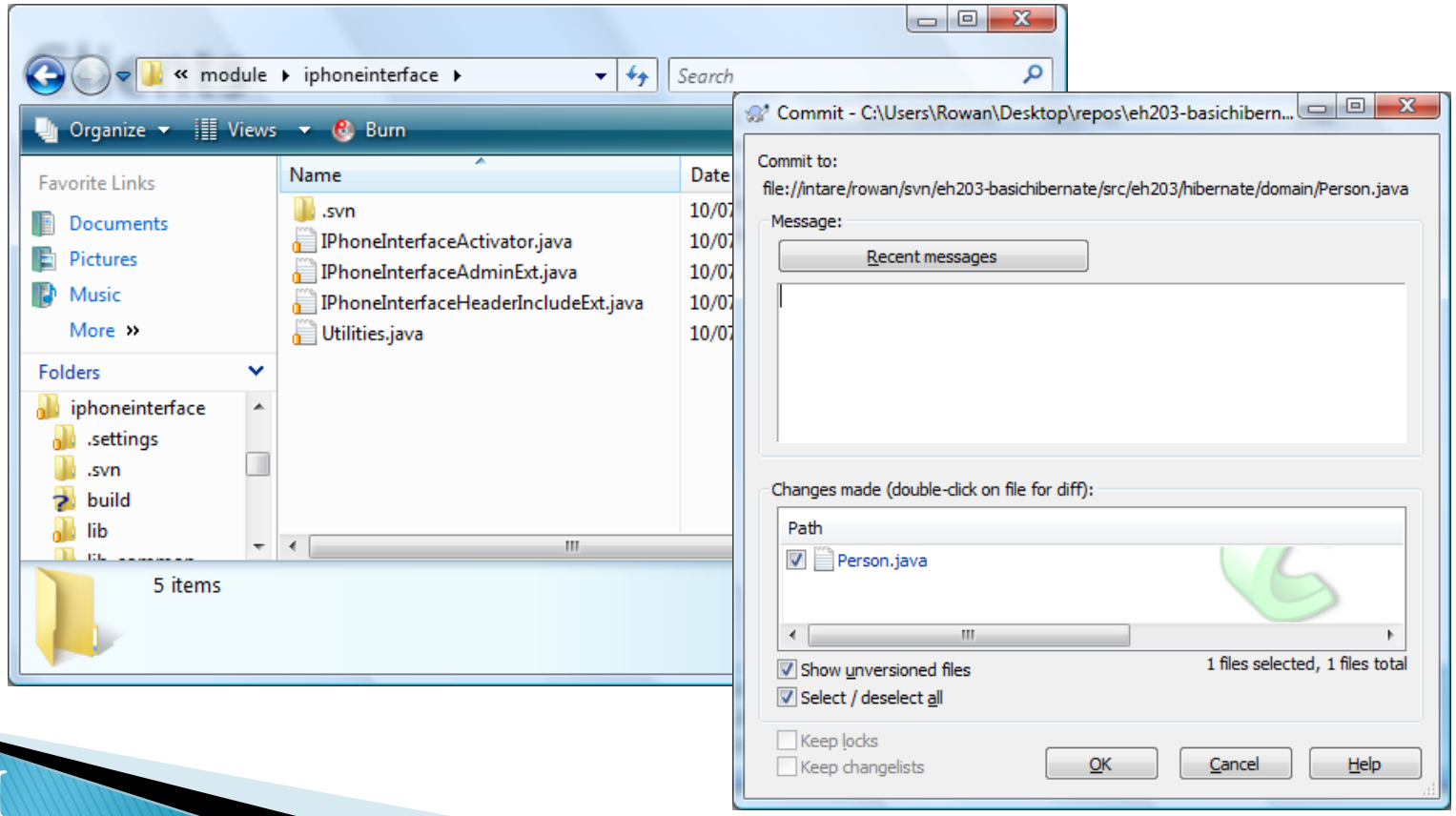
file://<host>/<path>/svn

Clients

- SVN has its own set of command line tools for client access, e.g.
  - `svn co file://intare/svn/myproject`
- However several graphical clients exist which are easier to use
  - TortoiseSVN (Windows only)
  - Subclipse (plugin for Eclipse)
  - SmartSVN (written in Java)
  - RapidSVN

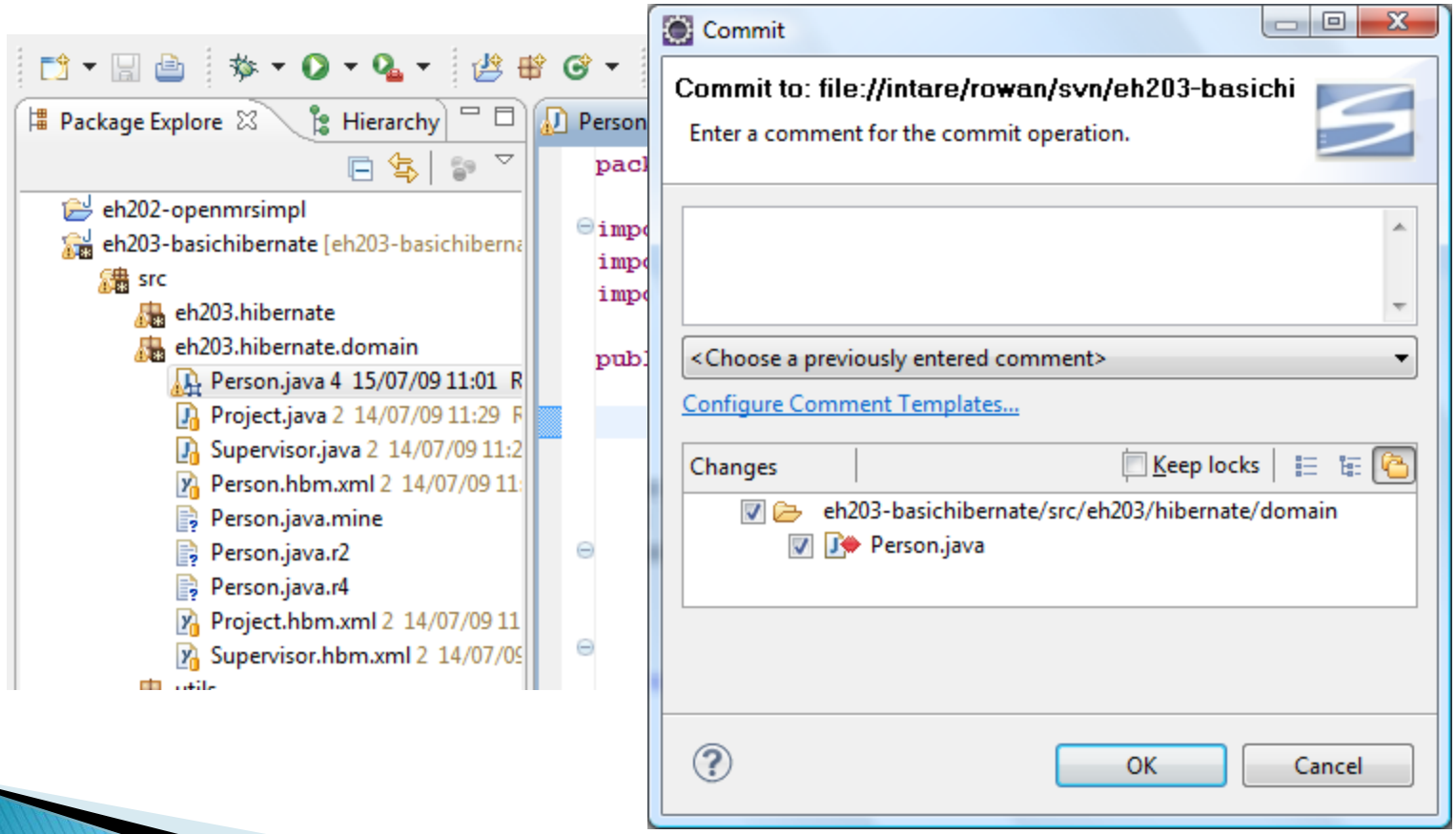
Clients: TortoiseSVN

- This is a popular client for Windows because it integrates with Explorer



Clients: Subclipse

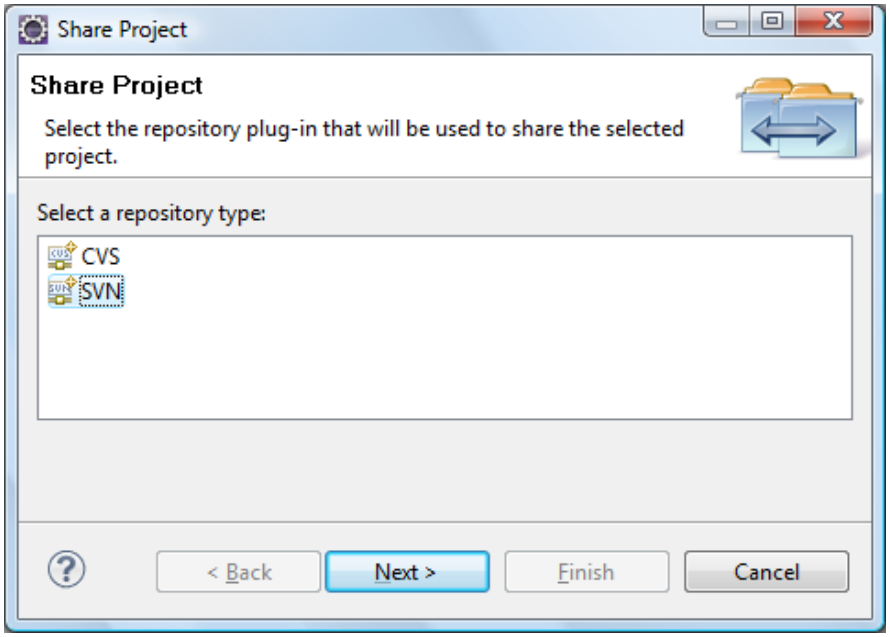
- This is a plugin for Eclipse



EHSDI
eBuzima

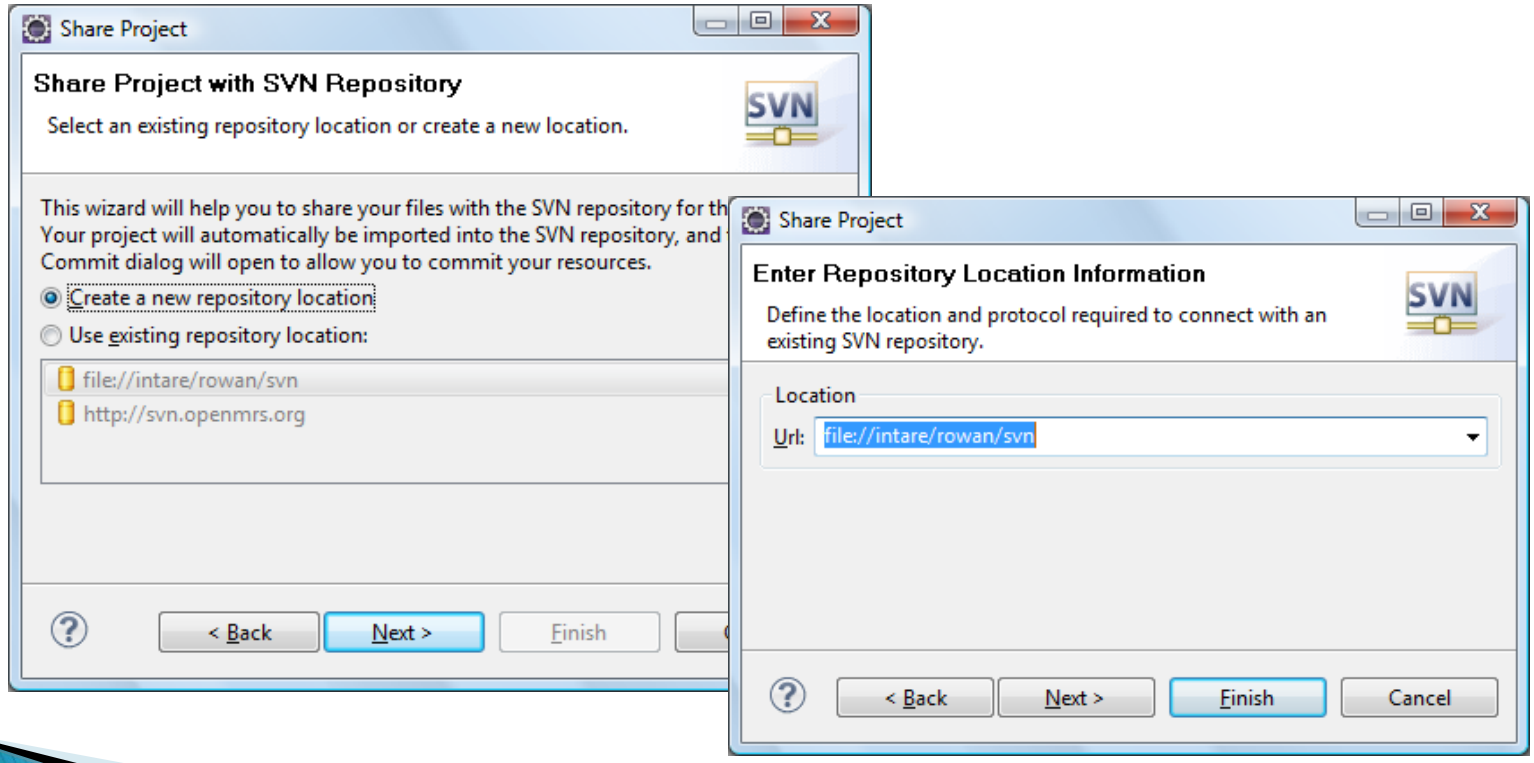
Subclipse: exporting to SVN

- Right-click on the project to be shared, and select *Team → Share Project*
- Select *SVN* as the repository type



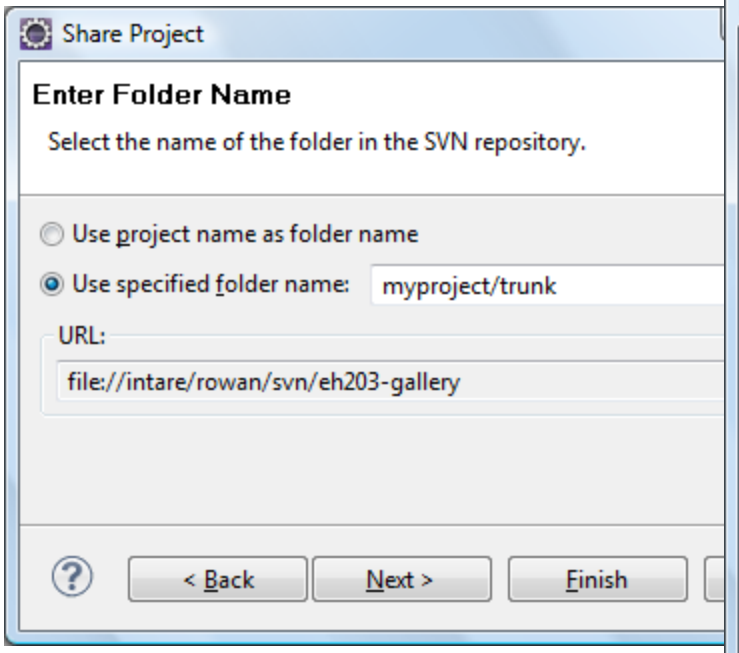
Subclipse: exporting to SVN

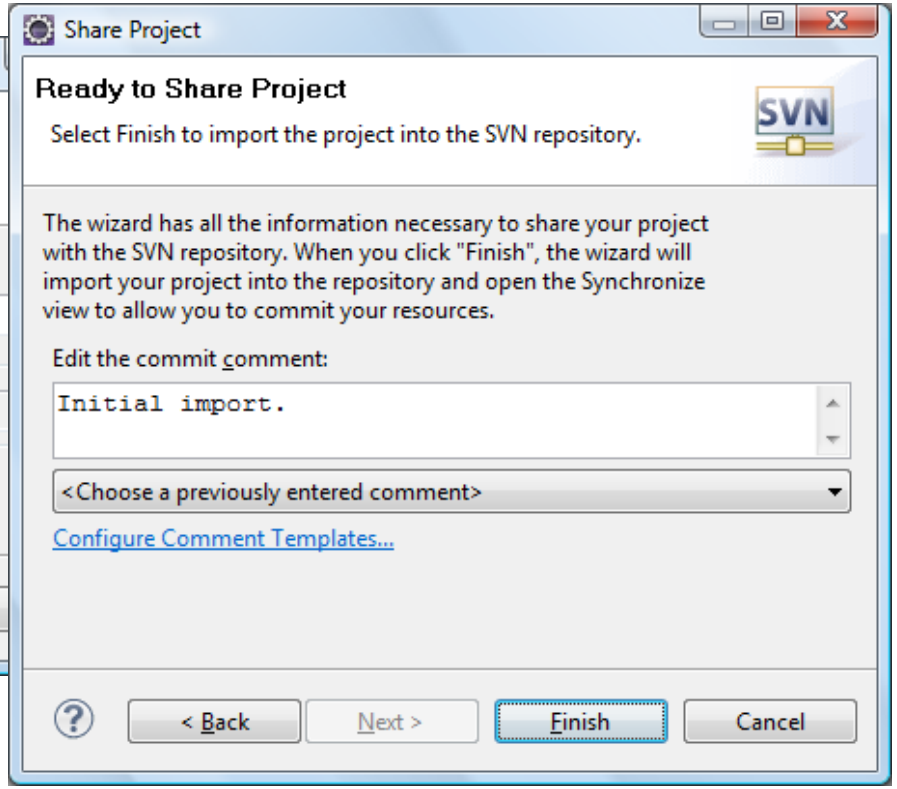
- If you do not have an existing repository configured, create a new one...



Subclipse: exporting to SVN

- Specify the path for the project, and a commit comment. Then finish!

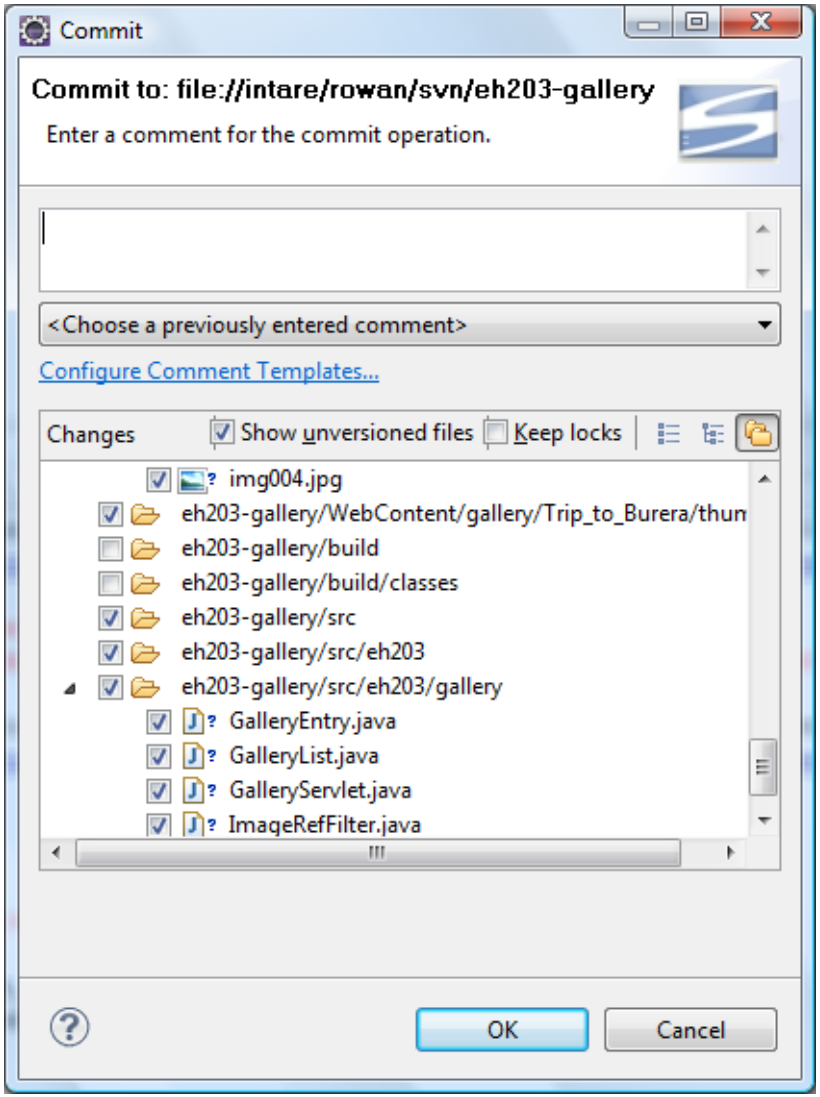




EHSDI
eBuzima

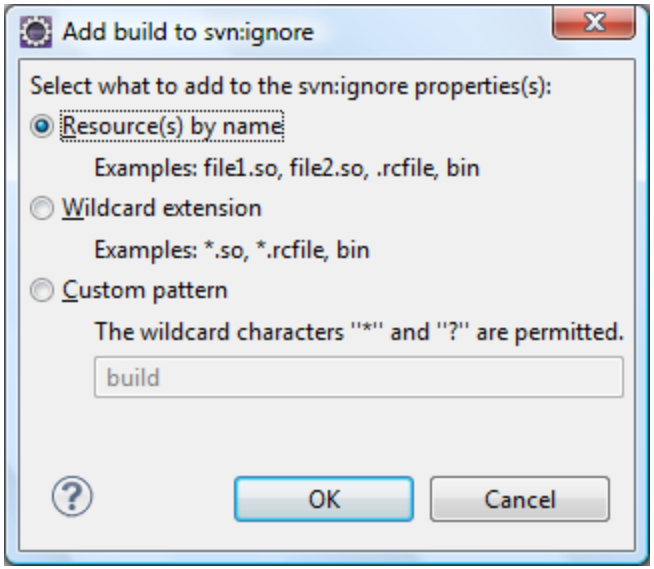
Subclipse: exporting to SVN

- The initial import only creates the project folder in the repository
- Go to *Team → Commit...*
- Uncheck the files such as the *build* directory that should not be added to the repository



Subclipse: exporting to SVN

- To prevent Subclipse from trying to commit files such as the build directory in future:
  - Right-click on them and select *Team → Add to svn:ignore*
- This will require another commit



Subclipse: icon key

Normal – not changed since last commit

Modified – changed since last commit

Added – added since last commit

Deleted – deleted since last commit

Non-versioned – not under version control

Ignored – has been added to svn:ignore

References

- Websites
  - http://subversion.tigris.org/
  - http://en.wikipedia.org/wiki/Subversion_(software)