CSC3045 & CSC3052

Source Code Control

School of Electronics, Electrical Engineering & Computer Science

Queen's University, Belfast

Dr Darryl Stewart

What is Source Code Control?

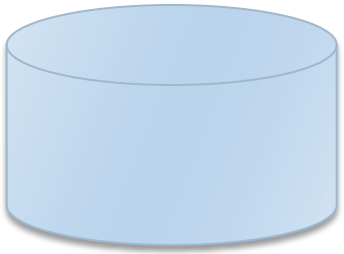
- Allows multiple people to work on the code for a project at the same time on their own machine
- The changes made by each person can then be quickly integrated together
- If a bug is introduced into a project which was not present in a previous version then the older working version can be retrieved
- It can also be used for documents etc. (not just code)
- Changes to the documents are usually identified by incrementing a number or letter code, termed the "revision number", "revision level", or simply "**revision**" and associated with the person making the change.
- Also known as Revision Control, Version Control, Code Management

How does it work?

- Three main components:

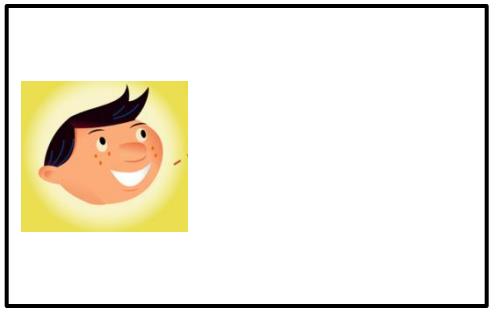
Repository

- Central or "master" copy of project



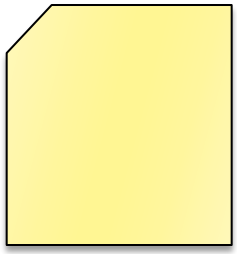
Client

- A person with a computer or terminal that wants to work on the project

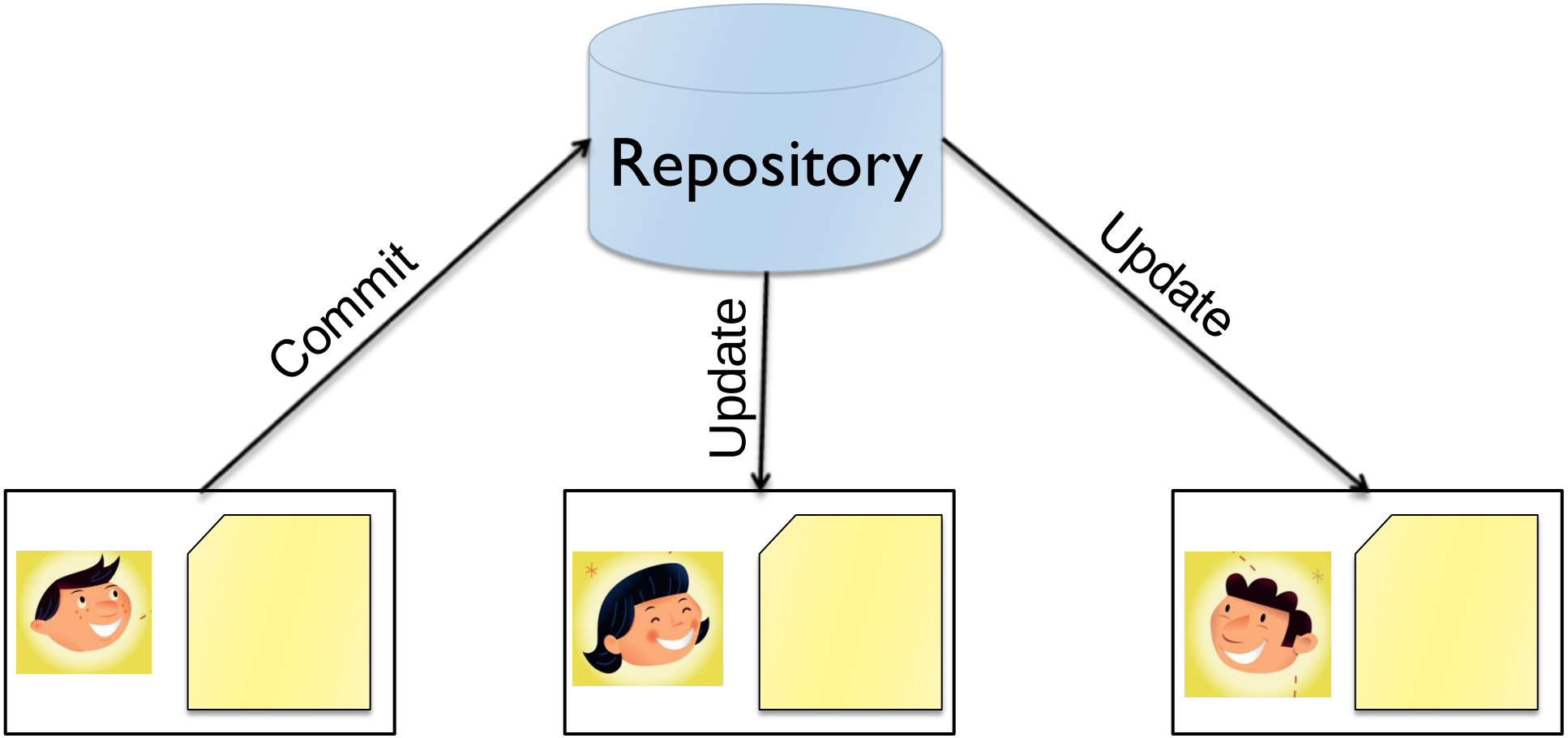


Working Copy (or sandbox)

- A copy of the repository's contents, local to the client



How does it work?



- Developers can Commit changes to the repository
- Developers can Update to get changes others committed to the repository

Benefits

- Centralized store for project artefacts (code, documents, artwork etc.)
- Historical record of changes over time
- Retrieval of older versions
- Parallel team development
- Code synchronization
- Multiple version management
- Changes are associated with individuals

Without Source control…

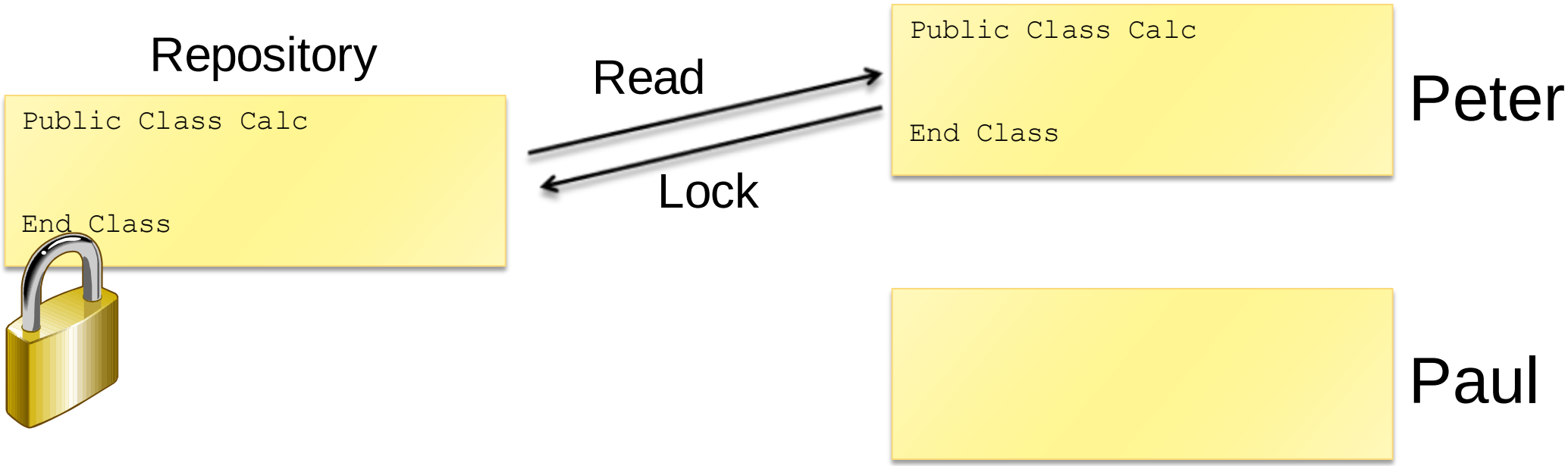
- Frequent backups are needed
  - Requiring lots of wasted storage space
  - Requiring lots of time to make or revert to
- Easy to forget why certain changes were made and when they were made
- Cannot easily restore system to older working status

Approaches to versioning (**Lock – Modify – Unlock**)

Lock – Modify – Unlock

- When a developer checks out a file it is locked so that no other developer can check it out.
- Only one developer can work on a file at one time.

Peter locks the file and then reads it



Approaches to versioning (**Lock – Modify – Unlock**)

Paul tries to lock the file but is not allowed

Repository

```
Public Class Calc


End Class
```

No Lock!

```
Public Class Calc


End Class
```

Peter

Paul

Approaches to versioning (**Lock – Modify – Unlock**)

Peter commits the new file and then unlocks

Repository

```
Public Class Calc

  Public Function Add (...)
         Add = n1 + n2
  End Function

End Class
```

Commit

Unlock

```
Public Class Calc

  Public Function Add (...)
         Add = n1 + n2
  End Function

End Class
```

Peter

Paul

Approaches to versioning (**Lock – Modify – Unlock**)

Paul can now lock and read to edit the file

Repository

```
Public Class Calc

  Public Function Add (...)
         Add = n1 + n2
  End Function

End Class
```

Read

Lock

```
Public Class Calc

  Public Function Add (...)
         Add = n1 + n2
  End Function

End Class
```

Peter

```
Public Class Calc

  Public Function Add (...)
         Add = n1 + n2
  End Function

End Class
```

Paul

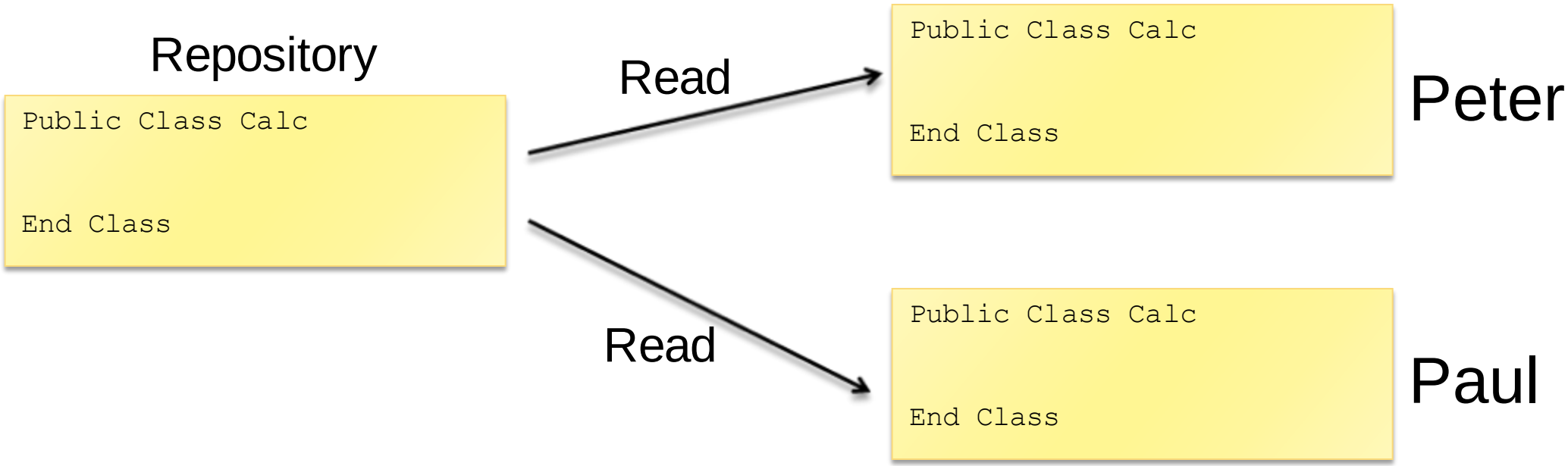
<u>Problem with this approach</u>

- Only one developer can work on something at a time
- Peter could forget to unlock

Approaches to versioning (**Copy – Modify – Merge**)

Copy – Modify - Merge

- Many developers can check out the same file
- Conflicts are merged



Approaches to versioning (**Copy – Modify – Merge**)

They both edit their own working copy

Repository

```
Public Class Calc
                    1

End Class
```

```
Public Class Calc
                    1*

Public Function Add (...)
        Add = n1 + n2
  End Function

End Class
```

Peter

```
Public Class Calc
                    1*

  Public Function Mult(...)
        Mult = n1 * n2
  End Function

End Class
```

Paul

Approaches to versioning (**Copy – Modify – Merge**)

Paul commits his changes first

Repository

```
Public Class Calc
  Public Function Mult(...)
          Mult = n1 * n2
  End Function

End Class
```

2

Commit

```
Public Class Calc
  Public Function Add (...)
          Add = n1 + n2
  End Function

End Class
```

1*

Peter

```
Public Class Calc
  Public Function Mult(...)
          Mult = n1 * n2
  End Function

End Class
```

2

Paul

Approaches to versioning (**Copy – Modify – Merge**)

Peter then tries to commit his changes but gets an **out of date error**

Repository

```
Public Class Calc

  Public Function Mult(...)
          Mult = n1 * n2
  End Function

End Class
```

2

Can't Commit

```
Public Class Calc

Public Function Add (...)
          Add = n1 + n2
  End Function

End Class
```

1*

Peter

```
Public Class Calc

  Public Function Mult(...)
          Mult = n1 * n2
  End Function

End Class
```

2

Paul

Approaches to versioning (**Copy – Modify – Merge**)

Peter creates a new merged version

Repository

```
Public Class Calc

  Public Function Mult(...)
          Mult = n1 * n2
  End Function

End Class
```

2

Read (Update)

```
Public Class Calc

  Public Function Mult(...)
          Mult = n1 * n2
  End Function

  Public Function Add (...)
         Add = n1 + n2
  End Function

End Class
```

2*

Peter

```
Public Class Calc

  Public Function Mult(...)
          Mult = n1 * n2
  End Function

End Class
```

2

Paul

Approaches to versioning (**Copy – Modify – Merge**)

Peter then commits the merged version

Repository

```
Public Class Calc                    3

  Public Function Mult(...)
          Mult = n1 * n2
  End Function

  Public Function Add (...)
         Add = n1 + n2
  End Function

End Class
```

Commit

```
Public Class Calc                    3

  Public Function Mult(...)
          Mult = n1 * n2
  End Function

  Public Function Add (...)
         Add = n1 + n2
  End Function

End Class
```

Peter

```
Public Class Calc                    2

  Public Function Mult(...)
          Mult = n1 * n2
  End Function

End Class
```

Paul

Approaches to versioning (**Copy – Modify – Merge**)

Paul does a read and so everyone has all changes

Repository

```
Public Class Calc                    3

  Public Function Mult(...)
          Mult = n1 * n2
  End Function

  Public Function Add (...)
         Add = n1 + n2
  End Function

End Class
```

Read

```
Public Class Calc                    3

  Public Function Mult(...)
          Mult = n1 * n2
  End Function

  Public Function Add (...)
         Add = n1 + n2
  End Function

End Class
```

Peter

```
Public Class Calc                    3

  Public Function Mult(...)
          Mult = n1 * n2
  End Function

  Public Function Add (...)
         Add = n1 + n2
  End Function

End Class
```

Paul

<u>Benefits of this approach</u>

- Developers can work on the same files in parallel
- Conflicts are flagged up and can be merged

Subversion allows this approach

Subversion (SVN)

- Open source and freely available software
- Very widely used
- Allows Copy – Modify – Merge approach
- A successor to the formerly widely used Concurrent Versions System (CVS)
- Works on Apache web server or can be Standalone
- Scriptable and Fast
- Many plug-ins a

- Download subversion from :

  http://subversion.tigris.org/

- Download the SVN book

SVN

- Once it is installed, there are various commands that can be called to control SVN:

General client command line

```
svn [command ] [arguments ]
```

Read or update your working copy

```
svncheckout / svnco
svnupdate / svnup
```

Make changes

```
svnadd
svndelete / svndel
svncopy / svncp
svnmove / svnmv
```

Commit your changes

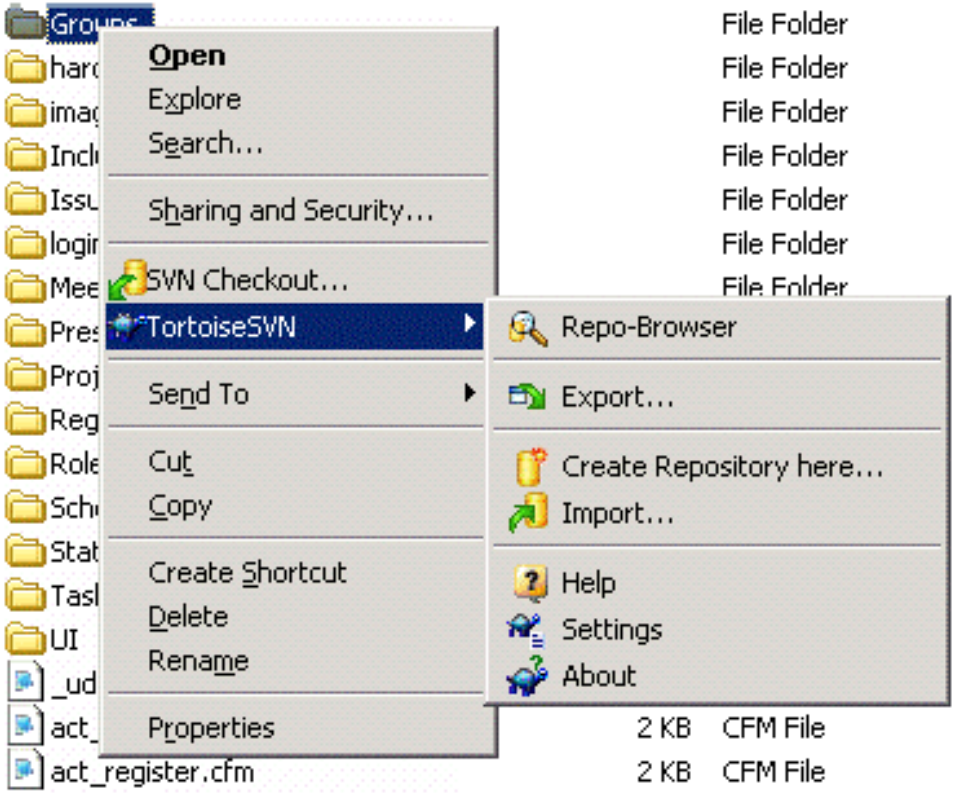
```
svncommit / svnci
```

TortoiseSVN

- Free Subversion Client
- Download from :

  http://tortoisesvn.tigris.org/

- Provides a Windows Explorer Shell Extension



Benefits

- Easier to manage repositories
- Browse repositories through repo-browser
- Easier to update/commit

Working with Subversion and TortoiseSVN

1. Create a new folder to store local copies (called a **sandbox**)
2. Link the Sandbox to the Repository
   - *RightClick* on the Sandbox folder, use *SVN Checkout*
   - Use the URL for the repository
     - Could begin with "file:///"or "svn://" or http://
     - Yours will be sent to you along with usernames and passwords
3. Add any files to the Sandbox folder that you want to include in the repository
4. Commit these files to the repository
   - *RightClick* on files and select *Add* in the SVN menu
   - *RightClick* on the Sandbox, use *SVN Commit… - you may need to check some boxes*
5. The files can then be modified in the sandbox and recommitted

Working with Subversion and TortoiseSVN

6. Other developers can now create new sandboxes on their own machines and do an *SVN Checkout* from the repository into their sandbox
7. They can edit their versions and do commits
8. If there are conflicts then these will be flagged when you try to do an ***SVN Update*** on your sandbox
9. You must then use:
   - ***Check for Modifications***
   - *RightClick* on the conflicting file, use ***Edit Conflicts***
   - *Use the editor to merge the modifications into an acceptable new version*
   - *Save the new version and then…*
   - *RightClick on the Sandbox, use* ***Tortoise SVN****, use* ***Resolved…***
10. This new version should then be committed using *SVN Commit*

Working with Subversion and TortoiseSVN

- The **Trunk** folder holds the stable fully tested and integrated versions of the software
- The **Branches** folder is used to hold working development versions of the software
- Once functionality has been tested and completed in a branch then the branch should be merged back into the trunk.
  - It is your responsibility to ensure that the merged version is fully working before committing to the Trunk
  - DO NOT BREAK THE TRUNK BUILD!!!
- When committing files – only commit source files – NOT compiler generated files – use the **"Unversion and add to ignore list"** option to remove non source files from the repository

Working with Subversion and TortoiseSVN

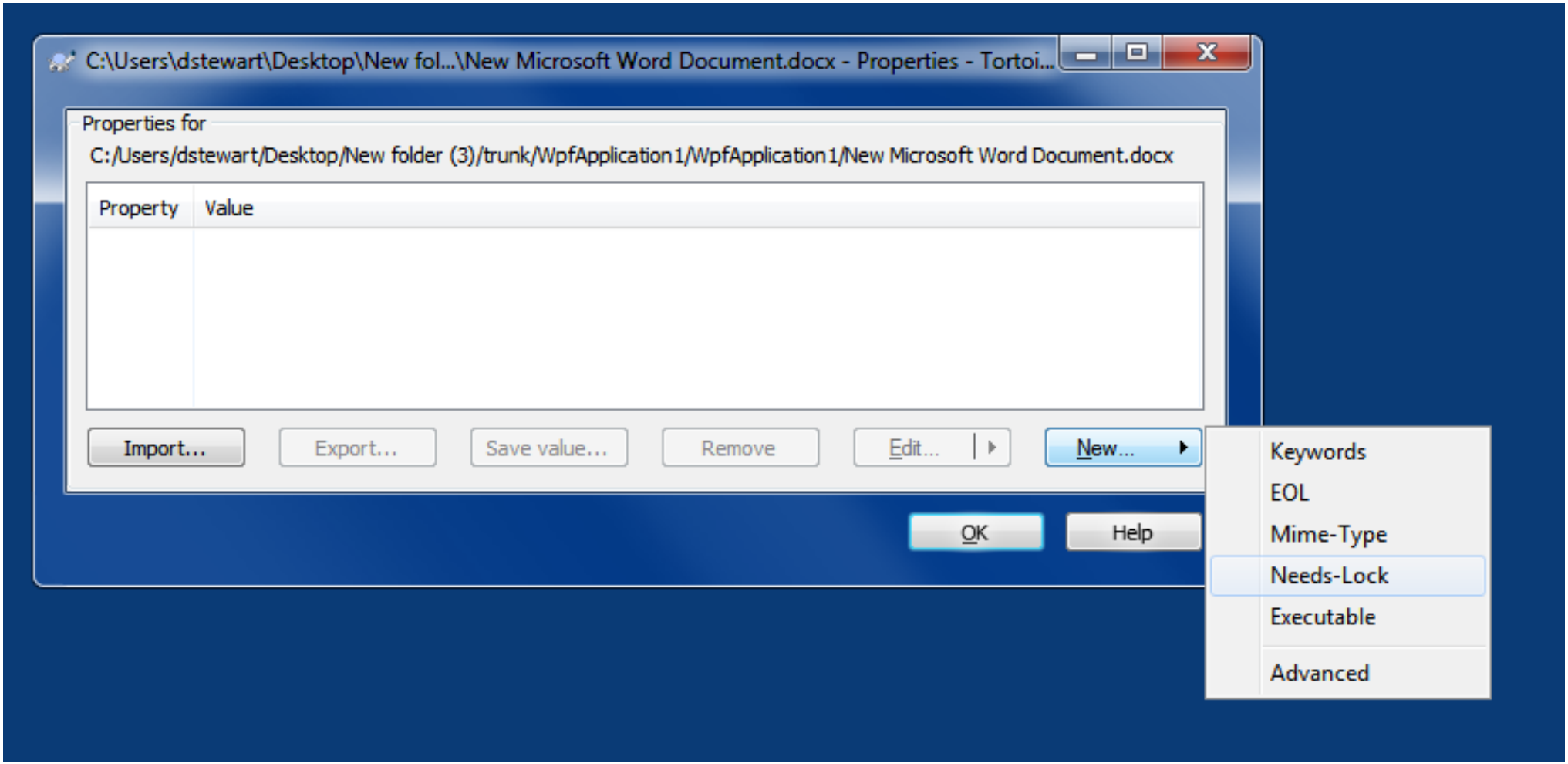
- Add comments for each commit, e.g.

  "John: Added function for X"

  "John & Sarah: Added registration page"

- **<u>Always</u>** do an **UPDATE** before you try a COMMIT

- For non-text files you should add the "Needs-Lock" property so that only one person can edit it at a time
- Database files/word documents/pictures/other binary files…
- To Edit the file you need to use **"Get Lock"**
- Don't forget to **commit the changes** so that the Lock gets opened again



AnhkSVN addon for Visual Studio

- SVN options or committing and updating etc are all included within Visual Studio via the AnhkSVN addon
- Use the file menu to find SVN options
- Automatically picks up repository details for files already under version control

StatSVN

- StatSVN is one of the tools that I use to generate reports on the activity of teams and individual developers in teams

Take home messages

- Agile Development relies upon teams of developers working on the same code base
- This might mean they work on the same files at the same time
- A Source Code Control System is needed to manage the versions of code
- It can be a *Lock-Modify-Unlock* system or *Copy-Modify-Merge* system
- Copy-Modify-Merge has many advantages
- **Subversion** is a very popular **Copy-Modify-Merge** solution – which you will all be using

- Your team has its own SVN repository
- Making concurrent modifications to DB files can be problematic
- **Add a <u>comment</u> for every commit to the repository**
- **I will be monitoring activity on the repository (not just numbers of commits)**