Branches in Subversion, Debugging, scmbug

Software Engineering and Scientific Computing

Exercises Third Day

Hanna Remmel
Institute of Computer Science
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
valtokari@informatik.uni-heidelberg.de

software
engineering
heidelberg

RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

Discussion

- How was the second exercise?
  - CppUnit
  - Doxygen

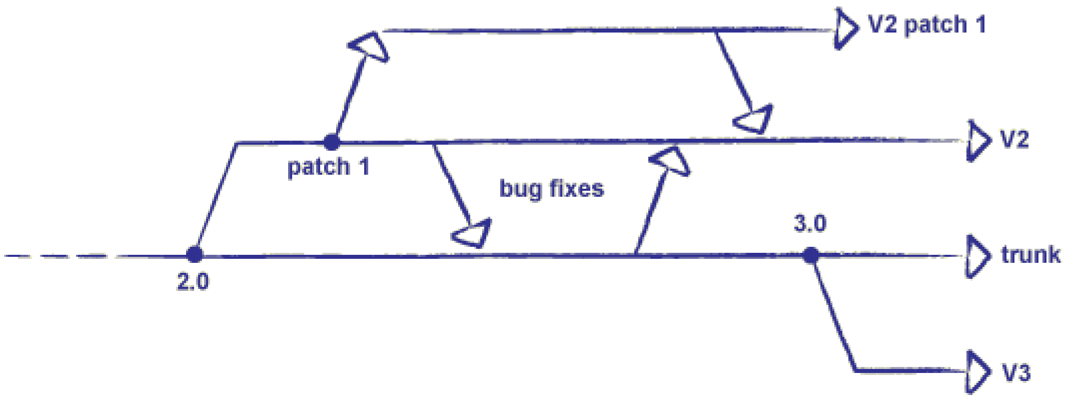
Content

- Branching Subversion
- Debugging
- Scmbug

[http://software-carpentry.org/]

Branching, Merging, and Tagging

- Sometimes you want to work on several different versions of software at once
  - Example: need to do bug fixes on Version 3 while making incompatible changes toward Version 4
  - Or want two sets of developers to be able to write and test large changes independently, then put things back together
- All modern version control systems allow you to branch a repository
  - Create a "parallel universe" which is initially the same as the original, but which evolves independently
  - Can later merge changes from one branch to another
- Also common to create tags
  - Symbolic labels that identify particular revisions, such as "Release_2.0"
  - Makes it easy to go back to an important revision later



Managing Branches

- Much better than just copying all the source files
  - The version control system remembers where the branch came from, and can trace its history back
  - Example: fix a bug on one branch, merge the changes into other branches that have the same bug
- Warning: many people become over-excited about branching when they first start to use it
  - Keeping track of what's going on where can be a considerable management overhead
  - On a small project, very rare to need more than two active branches

Subversion Command Reference

Name	Purpose
`svn add`	Add files and/or directories to version control.
`svn checkout`	Get a fresh working copy of a repository.
`svn commit`	Send changes from working copy to repository (inverse of update).
`svn delete`	Delete files and/or directories from version control.
`svn diff`	Shows changes for directories/files in a unified diff format.
`svn help`	Get help (in general, or for a particular command).
`svn log`	Show history of recent changes.
`svn merge`	Merge two different versions of a file into one.
`svn mkdir`	Create a new directory and put it under version control.
`svn rename`	Rename a file or directory, keeping track of history.
`svn revert`	Undo changes to working copy (i.e., resynchronize with repository).
`svn status`	Show the status of files and directories in the working copy.
`svn update`	Bring changes from repository into working copy (inverse of commit).

Symbolic Debuggers

- A debugger is a program that runs another program on your behalf
  - Sometimes called a *symbolic* debugger because it shows you the source code you wrote, rather than raw machine code
- While the target program (or debuggee) is running, the debugger can:
  - Pause, resume, or restart the target
  - Display or change values
  - Watch for calls to particular functions, changes to particular variables, etc.
- Do *not* need to modify the source of the target program!
  - Depending on your language, you may need to compile it with different flags
- And yes, the debugger modifies the target's layout in memory, and execution speed...
  - ...but a lot less than print statements...
  - ...with a lot less effort from you

Debugger Features

- Interactive debuggers typically show: The source code
  - The call stack
  - The values of variables that are currently in scope
    - I.e., global variables, parameters to the current function call, and local variables in that function
  - A panel displaying what your program has printed to standard output and/or standard error

Kinds of Debuggers

- There may be several ways to get into the debugger
  - Launch the debugger, load the target program, and start work
  - Run the debugger with the target program as a command-line argument
  - Switch into debugging mode in the middle of an interactive session
- Sometimes also do post mortem debugging
  - When a program fails badly, it creates a core dump
    - Copies all of its internal state to a file on disk
  - Load that dump into the debugger, and see where the program was when it terminated
    - Not as good as watching it run...
    - ...but sometimes the best you can do

Integrated Development Environments

- Debuggers are usually part of integrated development environments (IDEs) Tools like this are available for every modern language
  - [Microsoft Visual Studio] on Windows
  - [Eclipse] for Java (and now C++)
- Also usually contain a class browser that presents an outline of the project's modules, classes, functions, variables, etc.
- More about debugging on

  http://software-carpentry.org/debugging.html

scmbug

- Glue between Subversion and Bugzilla
- The reason for all these nasty errors commiting when
  - No issue number is given
  - Issue is not assigned to you
  - Issue is not in the rights status
- Also the reason for
  - The output of changed files in Bugzilla comments

[Bill Watterson]

Conclusion

- Dare to do some steps in Software Engineering
  - You can only judge their value, if you tried some out
- Talk to other people about it
  - You can learn a lot from your colleagues (in other groups)

References

- Software caprentry (http://software-carpentry.org)

Hanna Valtokari
Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
valtokari@informatik.uni-heidelberg.de

software
engineering
heidelberg

RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG