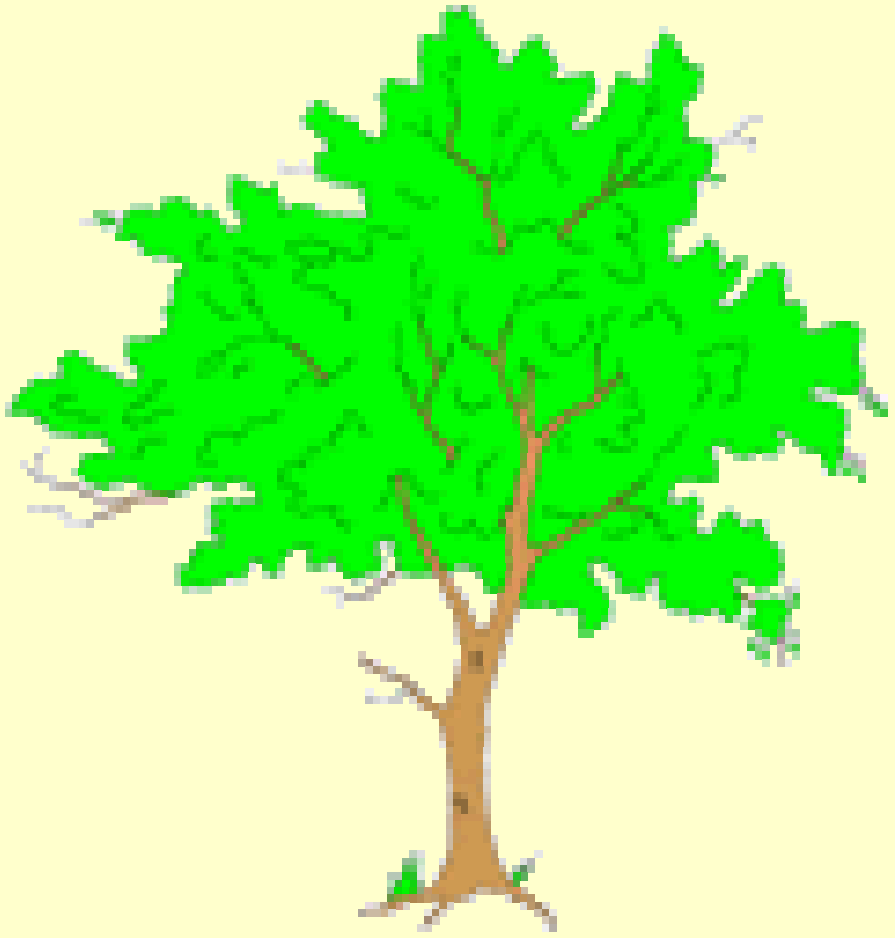
Leave branching to trees

Stewart Brodie

ACCU Conference

Friday 16th April 2004



Introduction

- Good
  - Controlling source code is good
  - Knowing the source of your project is good
  - Keeping simple change histories is good
- Bad
  - Complicated change histories ...
  - ... caused by excessive branching

HEAD-cases

- Uses the latest revision of each source file in any given build
- No idea which sources files are going in any build
- No hope of reproducing any reported faults
- This is not really source control

Source History for Idiotic Teams?

The One Big Project Model

- One step up from the HEAD-cases
- Entire repository is tagged
- Consistent, known set of sources
- Only suitable for small teams and projects
  - One person
  - One version
  - One customer?

And then we discovered branches

- At some point in our lives as software developers, we discover that our source control system permits multiple lines of development: branches
- This new toy will solve all our problems
- Everything becomes a branch.
- Why?
- Because it can? Because it gives me impressive looking revision numbers for my files?

Multiple branches: one per customer

- One branch per customer
- Copying of changes between branches
  - Changeset dependency problems
- Difficult to track which branch has which changes
- Easier than branching individual files
- Avoids work for one customer “infecting” other customers' builds

Weeping willow (and you will be)

Changes on branches

- Favoured by systems which work with changesets
- Lots of merging
- Spaghetti revision history ...
- ... but impressive looking graphics!
- Who can understand the history of a component?
- Complexity of revision history is a barrier to understanding the history - especially for new joiners
- Not entirely bad way of working

A monster is born

- Branches can take on a life of their own
- Lots of branching implies lots of merging!
- Merging technology is not perfect
- One famous Linux inventor said:

> “It was actually a case of 2,300 times, different people had done updates in parallel, and they had to be merged. And I think about fifty of those were manual. Everything else was automatic.”

An alternative approach

- Split the software up into **meaningful** subcomponents each with an independent version number
- Emphasises:
  - Strong interface design
  - Replaceable components
  - Customisable components
- Many benefits, but not without its problems too

Advantages

- Strong design
  - Interfaces become most important
  - Impact of interface change must be thought through
  - Minimise version dependencies
  - Simplifies development of unit tests
- Manageable level of software control
- Easy to assign ownership of sections of the code
- Promotes **RE-USE**??

Disadvantages

- Management of several version numbers
- Control of which versions of which components go into which builds
- Horror and fear because this is moving into the area of ....
- ... Configuration Management!

Fear of Configuration Management

- Why are people afraid of CM?
- CM doesn't have to be a complex task
- CM doesn't have to be a management function
- CM does imply that the build master is in charge, and not the individual engineers

Configuration Management overview

- What combination of components goes into a build?
- Independent control of which components go in a build ...
- .. AND which versions of those components go in a build.
- Not just the latest version of each component

Simple CM file

```
# Component file for project XYZ
#
# $Id: components,v 1.7 2004/04/10 12:34:56 stewart Exp $
#

buildsystem/generic          generic-1_45
buildsystem/linux            linuxbuildsystem-0_4

Linux/Kernel                 Kernel-2_6_3
glibc                        glibc-2_3_1
i386-linux-gcc               gcc-3_3_3
busybox                      busybox-0_61pre2

src/apps/myapp               myapp-3_45

src/libs/libpng              libpng-1_2_6
src/libs/libjpeg             jpeg-6a
src/libs/openssl             openssl-0_9_7
```

Single definition of project

- This file defines everything that goes into the project
- This file should be under source control!
- Once it is under source control, any build need only be identified by the revision number of the CM definition file ...
- ... and which build environment you used

Aside on build environments

- Define the core environment
- Different sets of initial environment variables can control any customisation of builds
- Try to localise all customisations in build environment startup scripts
- Avoid customising components on a per-customer basis (use feature enable/disable options based on the customised build environment).

Using the CM file

- Tag CM file instead of all sources in build
- Process build sources with scripts
- Checkout is a simple loop of VCS commands
- Compare CM files with scripts which understand the format (provides nicer log messages and simple build difference lists)

Branches are not entirely evil

- There are cases where branches are the right thing to do.
- BUT, they are not needed as often as some people seem to think.
- Going back to fix bugs in old releases
- Major changes – but if it' s completely new, why are you changing an existing component?

Use branches for short-term changes?

- Why do you make changes to software?
- Changes are usually new features or bug fixes
- Surely you intend to use these in future versions?
- So develop them on the trunk
- CM can ensure that experimental or incomplete versions of components do not go in builds.
- Stable versions can be branched if absolutely necessary – but only when necessary.

Branching Summary

- Branches should not be treated as a new toy
- Branches decrease comprehension
- Branches are not completely evil - just mostly
- Branches look good on real trees – they can be a sign of confused direction of development on source trees

Configuration Management Summary

- Configuration Management need not be complex
- Configuration Management is not a just a management activity
- Configuration Management need not be expensive
- Even the simple CM system described here helps increase understanding of the project due to:
  - Better componentisation
  - Central single point of control of what is in a build
  - Enforcing more thought about interfaces (possibly!)

Leave branching to trees

Stewart Brodie

ACCU Conference

Friday 16th April 2004