Building Lessons

~~Paul Grenyer~~
Alan Griffiths

I LONDON
accu 2007
Paul Grenyer

Genesis of Session

- ~~Paul~~
- ~~Who? - Experienced software engineer~~
- ~~What? - Set up/fixed build systems as part of several roles~~
- Alan
- Who? - Process improvement, C++, etc.
- What? - contract to fix a build system
- Sounds like the basis of a session!

What do we want from this?

- What I want:
  - Present some experiences
    - How general is my experience?
    - What lessons have I missed?
- What you want...
  - Hear an experience report
    - Compare with your experiences
    - Share your suggestions

From my ACCU2002 presentation "reworking the organisation"

- **No reproducible build process**

"...I spent two days experimenting with different version of the JDK, the EJB container and so fourth before I could get the [small] part of the system I was working on to compile. (I'd been on the project for three weeks before I'd worked out how to build and deploy it all.)...

"There was no unit testing or smoke testing regime – and it wasn't even easy to determine if the result of a check-in would compile. The build broke on a regular basis – developers were reluctant to synchronise their work with source control for fear of losing days sorting out the build...

"I never did understand how the change control system was expected to work – but more on that in a moment..."

Paul's examples: 1st Permie Job

- No build system beyond the IDE
- No source control system
- No tests
- No continuous integration

Paul's examples: 2nd Permie Job

- Green field
- Introduced source control
- Introduced unit tests
- Only used IDE build system
- No continuous integration (but the need was identified)

Paul's examples: 3rd Permie Job

- Had everything already set up
- My team was very good at generating scripts for variants

Paul's examples: 1st Contract

- System was in two halves and release build system needed multiple manual steps
- Issues with maintaining IDE and NAnt build system
- Tasked with introducing Continuous Integration
- Few unit tests
- Coverage and code checking tools almost impossible to use on existing codebase

Paul's examples: 2nd Contract

- Brought into to sort out build system
- All projects in separate SVN modules
  - (bad!), so brought together
  - Made checking out very irritating
- Continuous integration difficult to set up
- Needed a system for creating mock databases for integration testing
- Needed automatic deployment
- Needed better unit test coverage and mocking

Paul's examples: 4th Permie Job

- Green field project
- Set up complete build environment
- Using Ant, JUnit, CruiseControl, code checking and coverage tools

Alan's second example

- Contractor employed to “do the build”
- Built the system every day
- Took all day doing it
- When I took over:
  - Check out and build subsystem 1
  - Copy some artefacts; edit some header files
  - Check out and build subsystem 2 – fix errors
  - Copy some artefacts; edit some header files
  - Check out and build subsystem 3 – fix errors
  - Copy build artefacts to a share

Terminology

- Revision Control System
- Build System
- Automated Tests
- Unit Tests
- Continuous Integration
- Code Checking
- Test Coverage
- Mocking

Common problems

- Unclear which source version matches production binaries
- Build process requires multiple manual steps
- Testing requires multiple manual steps
- Automated testing neglected altogether
- Test results open to interpretation
- Test coverage measurements neglected
- Automatic code checking neglected

Does this sound familiar?

- Who's encountered problems like this?
  - (Show of hands)
- Who's dealt with problems like this?
  - (Show of hands)
- Any related issues
  - (flip chart)

The job

- Instead of developing the application – fixing this build process
  - On the face of it the build system seems better than usual – so what can need fixing?
- So in this contract the “Users” are developers
  - We all have some insight into this user domain

The main example

- Unclear which source matches production?
  - NO - release candidate build automates branch and tag
- Build requires multiple manual steps?
  - NO - Single command to build, test and package
- Testing requires multiple manual steps?
  - NO - Unit and integration tests automated
- Test results open to interpretation?
  - NO - Test failure aborts the build

Technical stuff

- Windows only
- C++, Python, XLA, C#
- Ant (+ Ant contrib) build script
  - Each component had its own build (devenv, py2exe, msbuild, ...)
- Overnight build on “build box”

The “half day test”

- The “half day test” of a project's maturity
  - “Can you arrive, take a standard developer workstation and complete the setup and processes needed to build and test the system within half a day?”
- A good start
  - after security, intros, finding a desk, etc. I had the system building and passing tests by the end of the first day.

So what was the problem?

- First stop – talk to the people in charge
  - I quickly found the first six stakeholders (most of them managers)
  - This later expanded to eleven stakeholders (including four developers)
- Document the result of these discussions
  - Get agreement on the problem
  - Get agreement on the first steps

Approaching a Solution

- Used Wiki to document initial context and specify the problems
- Propose list of solutions and resulting contexts
- Prioritise based on benefits, cost and risk
- Lots of meetings to involve stakeholders in context/problem/solution/result

Starting Context

- Multiple teams (quants, infrastructure, applications) contributing code to the “system”
- Multiple components split by programming language - not by function or code ownership
- Plans to add additional components to the system (but “very hard” with existing build system)
- Plans to restructure components of the system (but “very hard” with existing build system)
- Existing build server needs to be retired (out of warranty), new server needs to be set up

Starting Implementation

- An Ant script (spread over multiple files)
  - encompasses a range of usage for, development, release and patch release activities. The script controls a lot of build steps:
    - Updating working copies
    - Compiling the code
    - Running unit and integration tests
    - Packaging
    - Logging build output and publishing results
  - The dependencies between components are implicit both in the structure of the script and are spread across the included files

Build Scripting
What are the options?

- make
- Ant
- SCons
- Bjam
- Maven
- Suggestions...

Some issues to consider

- Build steps not separable
  - e.g. always does “svn update”s
- Every checkin included in overnight build
  - And each overnight build is a release candidate
- Bad checkins not detected until overnight build fails
- Hard to maintain build script – preventing restructuring codebase
- Each component in a separate Subversion repository (with its part of the build script)

More issues to consider

- This leads to problems:
  - inconvenient for those working on individual components as build steps like update, build, test, package are interdependent
  - Adding new components involves multiple, poorly understood, changes to the global script

Possible next steps

Document use cases for the build script (with a view to redevelopment)

Split components upon functional lines

Reduce the scope of build failures to individual components

Enable developers to work against precompiled binaries

Adding additional automated testing

Rework script to:

- simplify adding additional components
- remove implicit dependencies

Separating out deliverables to reduce the scope of releases

Move to a single Subversion repository

Implement Continuous Integration

First Step
Continuous Integration

- Very easy to put Continuous Integration in place (one day)
- Not so easy to get agreement from management that this matters (three weeks!!)
  - Concern that developers would “lean” on the build machine – and “not test checkins properly”
  - The benefit of knowing the state of the build is hard to anticipate – until faced with evidence that it is broken most of the time
- Very popular with developers (and, after they'd seen it in practice, the managers)

Continuous Integration What are the options?

- CruiseControl
  - http://cruisecontrol.sourceforge.net/
- Build-o-matic
  - http://build-o-matic.sourceforge.net/
- CruiseControl.net
  - http://cruisecontrol.sourceforge.net/
- Suggestions...

Second Step
Rework Build System

- Reduce coupling so that:
  - Easy to add new components
  - Components can be build in isolation or combination
  - Build tasks (clean, update, build, test, package) can be run in isolation or combination
- Difficult to validate “Legacy Code”
  - Original script “one big lump” with no tests
  - Full build cycle time consuming

Second Step
Reworking a build system

- Irony: changing the build system led me to...
  - ...run lots of manual tests
  - ...whose results needed careful interpretation
- Ant properties present similar problems to global variables
- Hard to isolate build steps to test
- Build results not repeatable as binaries contain timestamps
- Any suggestions?

Third Step
New Release Build Server

- Add a “release build” process to CruiseControl server
  - Ran in parallel to legacy build server
  - Again lots of manual checking
- Final switchover to the new build scripts
- Switch off legacy build server

Fourth Step
Patch Builds

- a manual process on the old build machine
- Involved working with old versions of parts of the build script (some before I started)
- Lots of scenarios for releasing subsets of the components
- Required to work on old build machine for validation

Fifth Step
Dependency Management

- The earlier changes controlled the build order explicitly – by looping through a master list of components
- A better approach is to specify dependencies infer the build order

Dependency Management What are the options?

- A tool that does this is Apache Ivy
  - It also provides a repository for build products
  - And integrates well with Ant
- Maven
  - Seemed too Java-centric
- Vesta
  - Language independent but based on NFS, no Ant integration

A bit about Ivy

- Uses explicit dependencies
  - Determines build order
  - Versioned storage & retrieval of artefacts

```
<ivy-module version="1.0">
   <info organisation="Octopull" module="branch23"/>
   <publications>
      <artifact name="branch23" type="obj"/>
      <artifact name="branch23" ext="h" type="include"/>
   </publications>

   <dependencies>
      <dependency name="root2" rev="latest.integration" />
      <dependency name="root3" rev="latest.integration" />
   </dependencies>
</ivy-module>
```

The Setup

Repository	location	usage	comment
local	part of each work area	storing locally build artefacts	there could be several work areas on a PC
integration	shared on build server	storing recent (last few days) integration versions	
release	network fileshare	storing release candidates, releases and patched releases	replaces previous fileshare as archive

What happened?

- CI delivered early
- Discovered failed earlier attempt
  - Build system rewritten in Python
  - But never adopted
- Refactored build system
  - made adding components easy...
  - ...and gave separate targets for clean, update, build, unit-test, integration-test and package
- Dependency management
  - Facilitates splitting components

The “TODO” list again...

- Document use cases for the build script (with a view to redevelopment)
- Split components upon functional lines
- Reduce the scope of build failures to individual components
- Enable developers to work against released binaries
- Adding additional testing
- Rework script to:
  - ~~simplify adding additional components~~
  - ~~remove implicit dependencies~~
- Separating out deliverables to reduce the scope of releases
- Move to a single Subversion repository
- ~~Implement Continuous Integration~~

Halfway?

- Build still monolithic
  - Have to build all components (not just those being worked on) – but as artefacts are published to an Ivy repository this is now easy to change
- Dependencies managed
  - was implicit in structure of build script, became a shared list giving build order, now deduced from explicit dependencies for each component
- Components need splitting and restructuring
  - But the facilities are now available and documented

The End

mailto:alan@octopull.demon.co.uk