Build Automation

Kurt Christensen

Kurt Christensen

Computer programmer (17 years) and
software development coach (9 years)

github.com/projectileboy

Available for purchase at:
kurt.j.christensen@gmail.com
Twitter: projectileboy

What are we trying to *do*?

Build software! But how? Where does this fit into the overall topic of continuous delivery?

- Write code
  - Tools - IDEs, version control
- Test code
  - Testing tools and frameworks
- Build code
  - **Build automation** and continuous integration
- Deploy code
  - Environments

What are we trying to *do*?

- Compile, test, package, deploy...
  - In other words, manage "the build lifecycle"
  - And more: generate documentation, generate keys, rebuild databases, obfuscate and minify, etc.
- Manage dependencies
  - Between our own projects and sub-projects
  - Between our project and third-party libraries
- Understand our project and all of its parts

NOTE: Thoughtful use of version control systems is a separate but related issue!

What are our options?

- Shell scripts
- Makefiles
- Ant
- Ivy
- Maven
- Gradle
- Rake, Buildr and others

Shell Scripts

Simple scripts which execute various command-line programs within a shell

Everything we're trying to accomplish *can* be done with shell scripts

"The assembly language of build automation"

Shell Scripts

An example: gradlew.bat for Gradle

https://github.com/gradle/gradle/blob/master/gradlew.bat

Shell Scripts

So what's the problem?

Well... it's the assembly language
of build automation...

Platform-dependent

BUT... shell scripts are still very often useful wrappers for all of these other build tools!

make

make provides something like a
domain-specific shell scripting
language for building software

make still works fine, and is everywhere
(it's how most GNU projects get built)

make

An example: a makefile for make

http://ftp.gnu.org/gnu/make/

make

So what's the problem?

Nothing, if you're building a C/C++ application for Unix

Otherwise, not so helpful…

(For example: http://www.cs.swarthmore.edu/~newhall/unixhelp/javamakefiles.html)

Ant

Platform-independent tool which executes *targets* defined in XML build files

We define targets, within which we specify various *tasks* to be executed (e.g., javac) - most tasks we would ever want are bundled in Ant; many more available from third-parties

(Example: JBoss-specific deployment tasks)

Targets can depend on other targets

Ant

Relatively easy to define our own tasks

(Example: generating test data SQL from an Excel spreadsheet)

Macro facility helps eliminate duplicate code

Enables platform-independent methods for defining properties (e.g., classpath) (also easy to read in external property files

We can have parent and child build scripts

Ant

An example: Ant's build.xml

http://svn.apache.org/viewvc/ant/core/trunk/build.xml

Sidebar: MSBuild, NAnt

NAnt is the .NET build tool inspired by Ant

MSBuild is the official Microsoft build automation tool, similar to NAnt

(We won't explore in depth; they're both very similar to Ant in both spirit and syntax)

Ant

So what's the problem?

- Extremely verbose XML-based language
- You tend to write the same Ant scripts over and over and over again
- Certain things aren't handled well, or at handled at all
  - Dependency management
  - Imperative scripting language without decent imperative coding constructs (e.g., for loops)

Ivy

Increasingly, applications are less about writing new code and more about stitching together functionality from existing libraries

Ivy adds *dependency management* to Ant

Ivy

An example: Ivy's ivy.xml, and build.xml

http://svn.apache.org/viewvc/ant/ivy/core/trunk/ivy.xml

http://svn.apache.org/viewvc/ant/ivy/core/trunk/build.xml

Ivy

So what's the problem?

You're still left with an enormous Ant script!

Maven

Declarative rather than imperative - Project Object Model (POM) file defines project structure and dependencies

Maven *goals* are executed within the context of a *lifecycle*

For Java project which follow certain conventions, Maven eliminates much of the boilerplate Ant code

Maven

Goals are defined within *plugins*

Most plugins you care about are bundled with Maven, although you can write your own

As with Ant, we can have parents and children

Maven

Provides dependency management

Artifacts (i.e., external libraries) are stored in and retrieved from *repositories*

Repository is by default a well-known global repository, but can be an internal corporate repository

Maven

An example: Maven's pom.xml

http://svn.apache.org/viewvc/maven/maven-3/trunk/pom.xml

Maven

So what's the problem?

Very unforgiving of those who choose to structure things differently

Example: Deploying a non-trivial set of web applications to ATG

Gradle

Ant + Ivy with the power of Groovy,
an actual programming language!

Build domain-specific language handles
most tasks, but vanilla Groovy code
can also be used

Provides a certain amount of convention
over configuration, similar to Maven

Gradle

An example: Gradle's build.gradle

https://github.com/gradle/gradle/blob/master/build.gradle

Gradle

So what's the problem?

Requires Groovy knowledge

Small community

Hasn’t achieved critical mass – minimal tool support, harder to find help, etc.

..and much, much more!

Rake - Make, in Ruby

Buildr - similar to Gradle,
but based on Ruby instead of Groovy

See: http://en.wikipedia.org/wiki/List_of_build_automation_software

Limited Life Experience

+

Overgeneralization

=

Advice

- *Paul Buchheit, creator of Gmail*

Recommendations

Vanilla Java app? Maven is the standard

…but if you're using Ant and it's working for you, leave it alone! (But consider Ivy)

If your build does non-trivial unusual things, Maven will fight you

If you're green-fielding, or doing something unusual, investigate unusual options

(e.g., Gradle)

Questions?

Comments?

Insults?

Twitter: @projectileboy

kurt.j.christensen@gmail.com