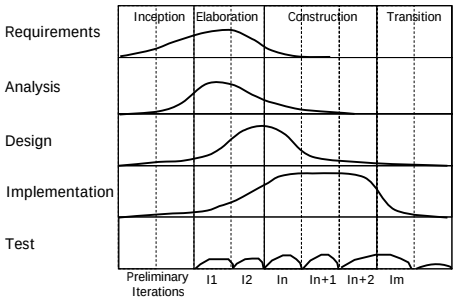
UCL

Build Management

Wolfgang Emmerich
Professor of Distributed Computing
University College London
http://sse.cs.ucl.ac.uk

UCL

Context



2

UCL

Learning Objectives

- To understand the rationales for using build management tools
- To know the principles of using build management tools
- To appreciate the need for continuous builds
- To be able to set up an external build process for a Java project

3

UCL

Motivation

- Often development artifacts are derived from others, e.g.
  - HTML API documentation from Java source
  - Java bytecode from Java source
  - DLL libraries from object code
  - Executable from object code and DLL libraries
  - Integration test report from test cases and executable
- Derivation can be performed by IDE
- May be more desirable to have this done outside the IDE if
  - Result of the build is too large to be manageable by one IDE
  - Derivation takes very long (e.g. deriving the integration test report from from a large number of test cases and the executable code)
  - Derivation needs to be repeated for different target platforms (e.g. package software for different operating systems)
  - Derivation should be continuous without human intervention, e.g. whenever new version is checked into trunk configuration

4

UCL

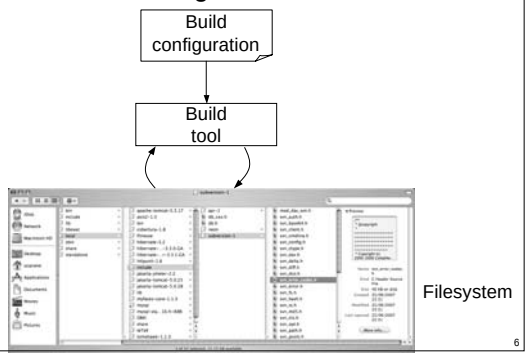
Requirements for Build Management Tools

- *Build management tools* fully automate the derivation of possibly complex artifacts from source artifacts.
- Requirements
  - Define a configuration language to specify concisely how the derivation is to be done
    - Individual steps
    - Identification of dependencies between artifacts
  - Interpreter for the language that executes the build
  - Build incrementally so that only those artifacts that have been affected by a change are derived again
  - Support clean-up by getting rid of all intermediary derived artifacts
  - Integration with
    - Program Editor
    - Configuration management system (to support continuous builds)

5

UCL

General build management tool architecture



6

UCL

Build configuration language

- Rule or template-based language
- Interpreted language (build tool contains interpreter)
- Allow us to define
  - artifact types (based on patterns of file names, e.g. extensions)
  - derivation tasks to express how input artifacts are transformed into an output artifact and where are they going to be stored
  - Dependencies
  - Build options

7

UCL

Overview of Build Management Tools

All these tools are open source and freely available

- Make, gmake, nmake - the oldest build management tools. Today used mainly in OS development and under Windows
- Ant - built by Apache Software Foundation. Very popular with Java development projects
- Maven - More powerful project management and reporting features than ant
- CruiseControl - continuous build management. Use in conjunction with ant or maven

8

UCL

Example Language: Ant build files

- XML language
- Core concepts:
  - Define properties (e.g. directory names)
  - Paths (e.g. class paths)
  - Targets with dependencies
  - Tasks that are carried out for each target
- Projects can extend the ant build language and define explicit tasks
- Implementations of these need to be provided in a jar file that is dynamically loaded by the ant processor
- Well integrated into Eclipse IDE
  - IDE can launch ant builds
  - IDE can generate ant build files so that builds can be performed outside IDE

9

UCL

Example Ant build file

```
<project basedir="." default="build" name="SimpleMetricsPlugin">
 <property name="ECLIPSE" value="/Applications/eclipse-ee"/>
 <path id="Plug-in Dependencies.libraryclasspath">
   <pathelement location="${ECLIPSE}/plugins/org.eclipse.jar"/>
   <pathelement location="${ECLIPSE}/plugins/org.eclipse.swt.jar"/> …
 </path>
 <path id="SimpleMetricsPlugin.classpath">
   <pathelement location="bin"/>
   <path refid="Plug-in Dependencies.libraryclasspath"/>
 </path>
 <target name="clean">
   <delete dir="bin"/>
 </target>
 <target name="init">
   <mkdir dir="bin"/>
   <copy includeemptydirs="false" todir="bin">
     <fileset dir="src" excludes="**/*.launch, **/*.java"/>
   </copy>
 </target>
```

10

UCL

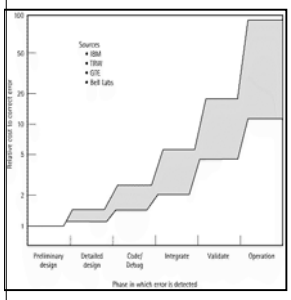
Example ant build file (cont'd)

```
 <target depends="build-project" name="build">
 <target depends="init" name="build-project">
   <echo message="${ant.project.name}: ${ant.file}"/>
   <javac destdir="bin">
     <src path="src"/>
     <classpath refid="SimpleMetricsPlugin.classpath"/>
   </javac>
 </target>

 <!-- and so on -->

</project>
```

11

UCL

Continuous integration



- Agile development projects test a lot and often in order to detect defects early
- Unit tests are executed locally in the workspace of the developer, but integration tests need to be executed against a fully integrated configuration
- Aim of *continuous integration* is to build a fully integrated executable whenever a new item is checked to the trunk so that this can be tested automatically.

12

UCL

Continuous integration tools

- Continuous integration tools are plugged into SCM tools
- Configure SCM tool to listen to commits of particular configurations (typically the trunk)
- Whenever commit is executed an (incremental) build of that configuration is launched using a build management tool
- Once the build is complete continuous integration tools trigger integration tests

13

UCL

Automated integration testing

- A *Smoke test suite* consists of a small set of representative integration tests that can determine whether the build worked and was deployed into the test environment
- If smoke test fails the changes that caused the failure are discarded so that an unbroken build is available
- Once smoke test succeeds a *regression test suite* can be started to identify newly introduced defects
- An *integration test suite tests* that newly delivered features are working in the integrated configuration
- We will discuss this in detail in a separate lecture

14

UCL

Key Points

- Build management tools support the derivation of complex artifacts from multiple inputs
- Need to be usable outside the IDE to be called by continuous integration servers

15

UCL

References

- S. Feldman. Make - a program for maintaining computer programs. Software Practice & Experience 9(4):255-265. 1979
- J. Tilly and E. Burke. Ant: The definitive guide. O'Reilly Media. 2002
- M. Fowler. Continuous Integration. Thoughtworks. 2006. www.martinfowler.com/articles/continuousIntegration.html
- B. Boehm: Software Engineering Economics. Prentice Hall. 1981

16