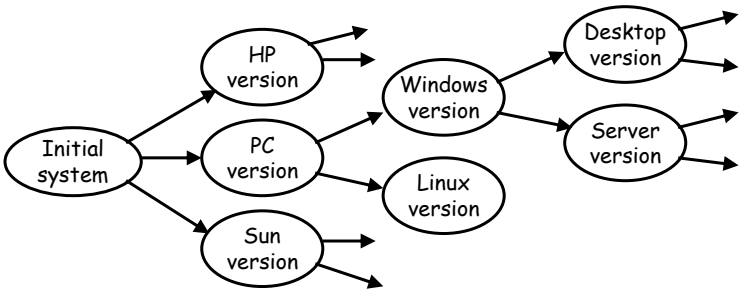
Configuration Management

- A large software system may comprise hundreds of evolving components, with many clients and differing requirements
- We will consider:
  - Configuration management
  - Build control
  - Tools: Three approaches: make, Ant and Maven
- This is a general introduction to concepts
- In what follows, "components" will often mean code, but could also cover plans, specifications, tests, documentation, etc

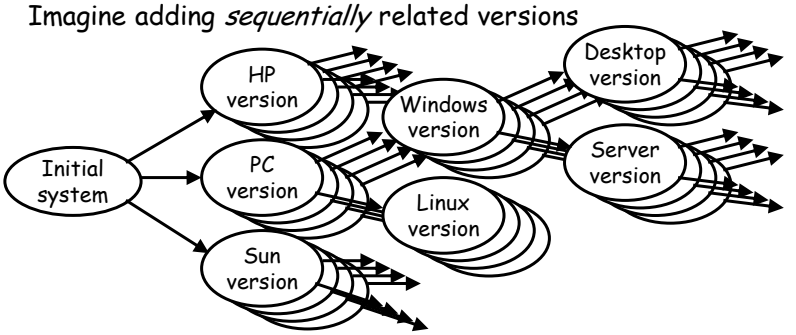
Configuration management

- Consider a large software system developed as a commercial product
  - Potentially for many clients
  - Each with different situations/requirements
- Clients and/or innovation drive *the evolution process*
  - Development of new or improved versions of modules
- So, there may be *different released versions/configurations* for many reasons:
  - For different computers (hardware)
  - For different operating systems
  - With client specific functions

- There may be *parallel* versions of components
  - E.g: Same function but for different OS
- An illustration of *parallel versions* from Somerville:



- There may be *sequentially related versions*
  - E.g: Earlier, and later, corrected or upgraded versions
- Imagine adding *sequentially* related versions



- Some clients may choose to remain with older, known, stable releases of the software
  - For stability/continuity, or contractual/financial reasons
  - So, we cannot/should not discard old versions of components (within any contracted support period)

- Each released configuration, for a specific client, will have a *particular combination* of *particular versions* of components
  - Many may be in common with some, though perhaps not all, other released configurations
  - Some may be particular to the released configuration
  - Some may not even be the most up-to-date versions of the particular components

The configuration database

- We now see the overall configuration management problem:
  - We need a substantial *configuration database* to record all the relevant information:
  - Components, configurations, clients, …
- It should be useable:
  - to assess the impact of changes
  - to generate reports
  - to control system builds for releases
  - to answer questions about clients and their specific release configurations
  - to monitor progress on requested changes
  - …
- Clearly Computer Aided Software Engineering (CASE) tools are valuable here
- See *http://en.wikipedia.org/wiki/Configuration_management_database*

Build control tools

- See Wikipedia: *http://en.wikipedia.org/wiki/Build_automation*
- Originally, developers built OS command scripts to compile, link and deploy systems comprising many source code modules (and other files)
  - A manual process would not be reasonable…
  - … nor easily repeatable
- The *"make"* scripting language offered a better alternative
  - A classic Unix tool
  - Its primary focus was on automating the calls to the compilers and linkers
  - This was the beginning of Build Automation
- With increasing complexity of the build processes more advanced tools have been designed:
  - See the very long list on Wikipedia: *http://en.wikipedia.org/wiki/List_of_build_automation_software*

Build control tools

- We will look briefly at three approaches:
  - make, Apache Ant and Maven
- There are also proprietary tools such as
  - ElectricCommander
    (see *http://www.electric-cloud.com/* and customers link)
  - Visual Build
    (see *http://www.kinook.com/VisBuildPro/* and customers link)
  - MSBuild – Microsoft
    (see *https://msdn.microsoft.com/en-us/library/dd393574.aspx*)
    - standalone, and required by Visual Studio

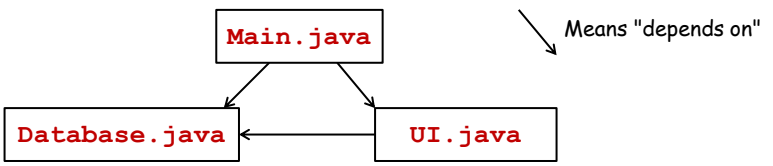
make

- make is a utility for automatically building and deploying executable programs
  - Controlled by files called makefiles
  - These contain *rules* specifying how to derive the target program from source files – "declarative"
  - Dates from 1970s, but still widely used, especially in Unix
  - Many on-line descriptions
- Each rule has the form:

```
target : source1 source2 ...
        command
```

  - It means: To get `target` up to date:
  - First make sure all the `source`s are up to date
  - Then if `target` is older than any up to date `source`s, execute `command`
- Many sophisticated features: variables, macros, conditionals, ...

Example makefile



- To build a runnable `Main`, we need an up to date `Main.class`
- Could have a makefile like this:

```
Main : Main.class

Main.class : Main.java Database.class UI.class
  javac -d bin Main.java

UI.class : UI.java Database.class
  javac -d bin UI.java

Database.class : Database.java
  javac -d bin Database.java
```

- Note: javac can do *this* example
  - But does *not* do full dependency/change analysis

Apache Ant - "Another Neat Tool"

- See *http://ant.apache.org/*
- Dates from 2000, widely used
- Can be used stand-alone (command line), or as IDE plug-in
  - A standard part of Eclipse
- Oriented to Java
- Ant uses an XML "build file" to describe the build process
- It is procedural rather than declarative
- `build.xml` contains "targets" (rules)
  - The targets are named goals, not files
  - Each mentions other targets that it depends on – so should be executed first
  - Each then contains its own building "tasks"

```
<target name="..." depends="...">
  <delete .../>
  <javac .../>
</target>
```


- Ant is relatively "low level"
  - The build file author has full, detailed control
  - So *very flexible*
- Ant has a large catalogue of sophisticated tasks available
  - Some mirror command line operations (delete, copy, ...)
  - Others automate complex steps: compiling, documentation generation, jar construction, JUnit test management
  - (The `javac` task is a more powerful wrapper for the standard Java `javac`)
- The language has specialized features tailored to system building:
  - Concise specifications of sets of files
    - with inclusion and exclusion patterns
  - Concise specifications of "paths" – collections of directories/folders
    - again with inclusion and exclusion patterns

Ant example

Assume the same three class Java project as for make:

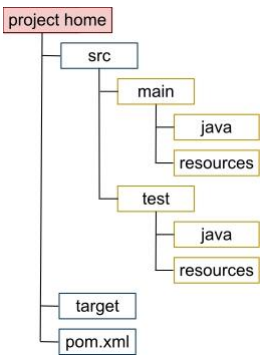
```
<project name="Main" default="compile" >
    <target name="clean" description="Remove junk">
        <delete>
            <fileset dir="bin" includes="*.class"/>
        </delete>
    </target>
    <target name="compile" depends="clean" descr... >
        <mkdir dir="bin"/>
        <javac srcdir="." destdir="bin"/>
    </target>
    <target name="run" depends="compile" descr...>
        <java classname="Main"
              classpath="bin" fork="true"/>
    </target>
</project>
```


Apache Maven

- See *http://maven.apache.org*
- From the Apache web site:
  "Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information."
- "Maven is a Yiddish word meaning *accumulator of knowledge*"
  - See note later about "convention"
- Dates from about 2002, with Maven 1.0 released in 2004
- Originally for Java, but plugins for other languages
- Can be used stand-alone (command line), or as IDE plug-in
  - Integrated into Eclipse, IntelliJ IDEA, NetBeans
- Popularity? Hard to assess…

- Maven is *higher level* than Ant:
- Philosophy: "Convention over configuration"
- Projects with a *standard structure*:
  - Detailed configuration is not required
  - A default *Project Object Model* is assumed
  - Maven already "knows" how to compile, test, build…
  - Developer is "shielded from the details"
- Projects with non-standard structure/features:
  - The POM must be customized and described
  - Maven projects must be understood in great detail – "what happens… is not immediately obvious just from examining the Maven project file" – HARD!
  - Has led to criticism: e.g. Neal Ford: "Why Everyone (Eventually) Hates (or Leaves) Maven" **http://nealford.com/memeagora/2013/01/22/why_everyone_eventually_hates_maven.html**

- The Maven default POM:



[By 用心阁 (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0) or GFDL (http://www.gnu.org/copyleft/fdl.html)], via Wikimedia Commons]

- **pom.xml** describes the project configuration
  - Auto-generated for standard projects
  - Customize for non-standard: "is huge and can be daunting in its complexity" (from the Apache web site)
  - Reference: **https://maven.apache.org/pom.html**

- A "simple" **pom.xml**: (from maven.apache.org)

```
<project xmlns=... ...>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```


- A slightly extended **pom.xml** from https://spring.io :

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/200
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/m
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.springframework</groupId>
    <artifactId>gs-maven</artifactId>
    <packaging>jar</packaging>
    <version>0.1.0</version>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>2.1</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer
                                    implementation="org.apache.maven.plugins.shade.
                                    <mainClass>hello.HelloWorld</mainClass>
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

End of lecture