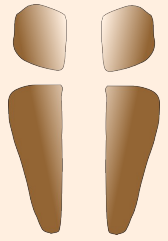


*If you are using Make, you  
are doing it wrong*

**Dr Russel Winder**

It'z Interactive Ltd

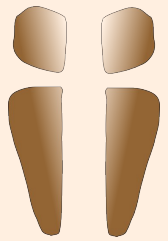
[russel@itzinteractive.com](mailto:russel@itzinteractive.com)



## *Aims and Objectives*

- Applaud Make as a revolutionary tool.
- Show Make is beyond its use by date.
- Show that dynamic languages in general, and Python in particular, are a better platform for build than Make. Internal domain specific languages (DSLs) are the future of build.
- Introduce SCons.

*DSEL – domain specific  
embedded language.*

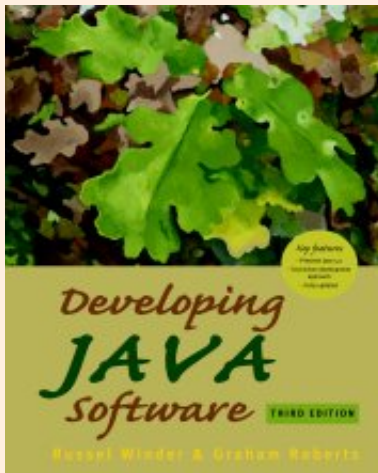
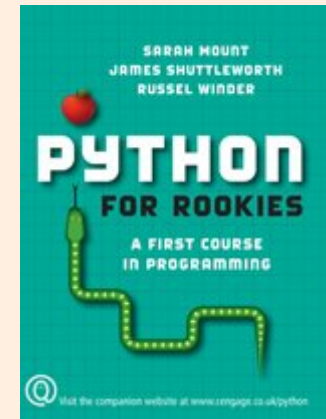


# *Subliminal Advertising*

## ***Python for Rookies***

Sarah Mount, James Shuttleworth and  
Russel Winder

*Thomson Learning*    *Now called Cengage Learning.*



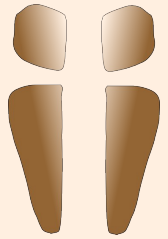
## ***Developing Java Software Third Edition***

Russel Winder and Graham Roberts

*Wiley*

***Buy these books!***





# *The Players in the Build Game*

Make

Ant

Autotools

Autoconf  
Automake  
Libtool

***SCons***

Maven

Rake

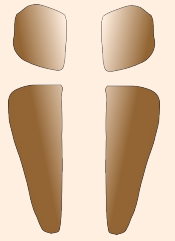
Buildr

CMake

*Rant*

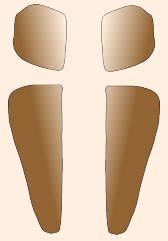
*Gradle*

***Gant***



## *Makes Builds its Case*

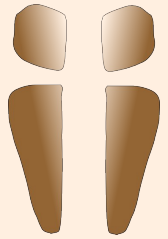
- 1977 – Make released by Dr Stuart I Feldman whilst working at Bell Labs.
- 1978 – Make is clearly a revelation and a revolution.
- 2003 – Make takes its rightful place of honour in the history of computing when ACM awards Stu Feldman the 2003 ACM Software System Award.



## *Make is . . .*

- A tool for ensuring the existence of things given other things that may have been amended.
- A tool for managing actions according to rules.
- An external domain specific language (DSL):
  - Rules specify relationships in one notation.
  - Actions are shell scripts.
- Usable for compiling:
  - C programs.
  - [nt]roff documents.

*DWB – Documenter’s WorkBench*



## A Trivial Make Example

### helloWorld.cpp

```
#include <iostream>
int main ( const int ac , const char *const *const av ) {
    std::cout << "Hello world." << std::endl ;
}
```

```
|> make
g++ -c -o helloWorld.o helloWorld.cpp
g++ helloWorld.o -o helloWorld
|> make clean
rm -f helloWorld helloWorld.o *~
|>
```

### Makefile

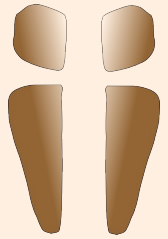
```
CC = g++

EXECUTABLE = helloWorld
OBJECT = ${EXECUTABLE}.o

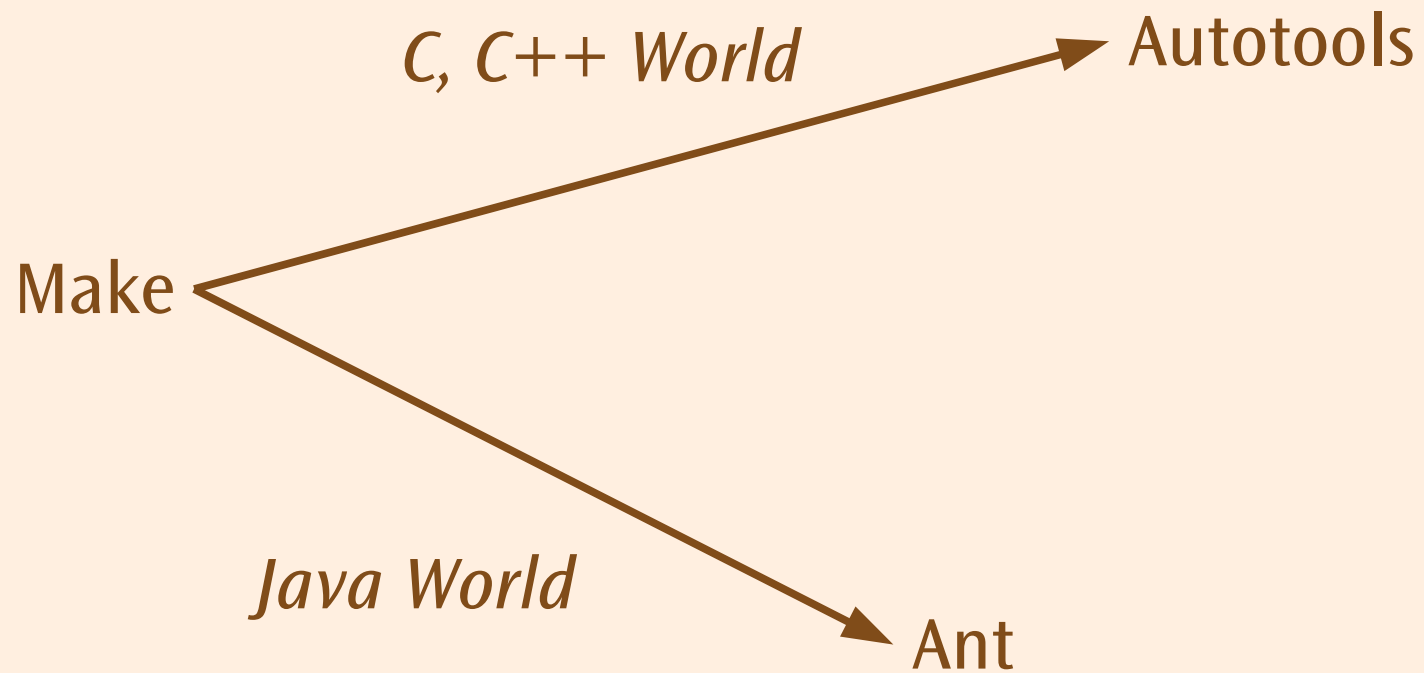
${EXECUTABLE} : ${OBJECT}

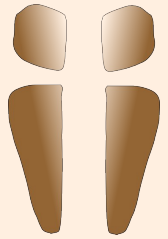
clean :
    rm -f ${EXECUTABLE} ${OBJECT} *~

.PHONY : clean
```



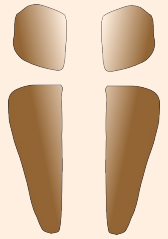
## *Ongoing History*





## *GNU Autotools*

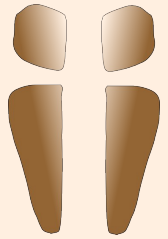
- Autoconf and Automake become the de facto standard build system in the C world. And, indeed, in the C++ and Fortran worlds . . .
  - . . . except for anyone using Windows as the platform in which case Visual Studio is the de facto standard tool.
- Autotools manages platform dependencies, allowing project builds to deal easily with platform differences. Configure then build.



## *Apache Ant*

- Ant was, and is, very much focused on activity in the Java milieu, and (more or less) ignores C, C++, Fortran, L<sup>A</sup>T<sub>E</sub>X, etc.
- Ant encourages platform independence of the build – this is arguable the single biggest reason for its success, Java is platform agnostic and Ant is as well.

*Ant was a revelation in 2000  
when it burst on the world.  
James Duncan Davidson  
probably deserves an award.*



# Ant Says Hello

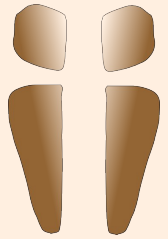
## *build.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<project name="Hello World" default="hello.world">
  <target name="hello.world" description="Output Hello World.">
    <echo message="Hello World."/>
  </target>
</project>
```

```
|> ant -f helloWorld_ant.xml
Buildfile: helloWorld_ant.xml
```

```
hello.world:
  [echo] Hello World.
```

```
BUILD SUCCESSFUL
Total time: 0 seconds
```



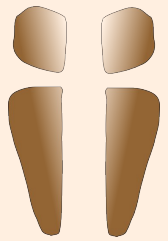
## Ant Makes Hello

### HelloWorld.java

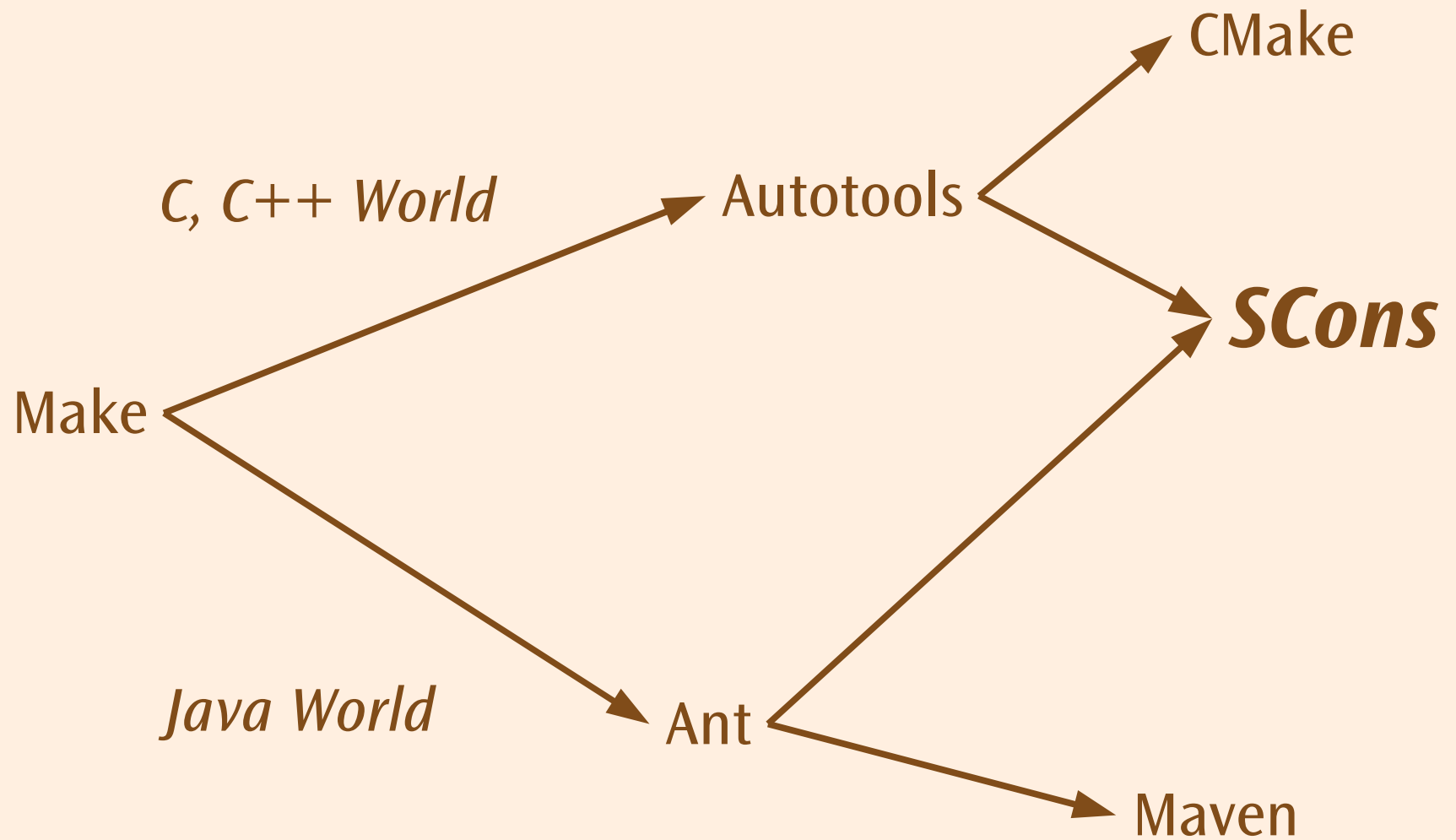
```
public class HelloWorld {  
    public static void main ( final String[] args ) {  
        System.out.println ( "Hello World." );  
    }  
}
```

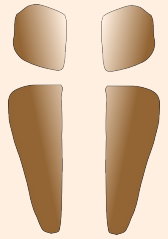
### build.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<project name="Hello World" default="run">  
    <property name="buildDirectory" value="Build"/>  
    <property name="sourceDirectory" value="src/main/java"/>  
    <target name="compile" description="Compile the code.">  
        <mkdir dir="${buildDirectory}"/>  
        <javac srcDir="${sourceDirectory}" destDir="${buildDirectory}" debug="true" source="6" target="6"/>  
    </target>  
    <target name="run" depends="compile" description="Run the application, compiling if necessary.">  
        <java classname="HelloWorld" classpath="${buildDirectory}"/>  
    </target>  
    <target name="clean" description="Clean everything up.">  
        <delete dir="${buildDirectory}" quiet="true"/>  
        <delete quiet="true">  
            <fileset dir="." includes="**/*~"/>  
        </delete>  
    </target>  
</project>
```



# Things Evolve

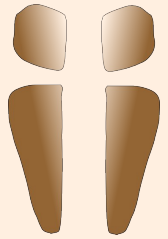




## *Maven*

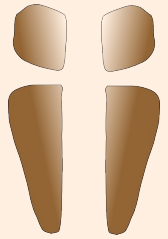
- Most projects fundamentally the same.
- Ant requires all aspects of the project to be specified, and leads to irrelevant variation.
- Maven introduces a project lifecycle model:
  - Standard project hierarchy.
  - Standard set of goals.
  - Remove the need to specify targets, just specify data about the project.
  - Use plugins to create variation.

*Jason van Zyl probably  
deserves an award.*



# Maven

```
<?xml version="1.0" encoding="UTF-8"?>
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd"
>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.concertant.examples</groupId>
  <artifactId>helloWorld</artifactId>
  <packaging>jar</packaging>
  <version>0.0.0</version>
  <name>Hello World</name>
  <build>
    <defaultGoal>package</defaultGoal>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-clean-plugin</artifactId>
        <configuration>
          <filesets>
            <fileset>
              <directory>.</directory>
              <includes>
                <include>*/~</include>
              </includes>
              <followSymlinks>>false</followSymlinks>
            </fileset>
          </filesets>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```



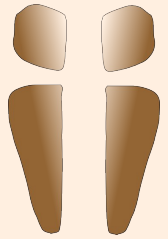
## *SCons*

- SCons build specification is usually called SConstruct.
- SConstruct is a Python script executed in a SCons context:
  - Any and all of Python can be used.
  - The SCons API is a declarative language.

*For the moment, SCons works with any version of Python from 1.5.2 onwards. Support for versions prior to 2.3 is deprecated and will soon be removed.*

*The SCons website is at:  
<http://www.scons.org>*

*SCons comes prepackaged in Debian, Ubuntu, Cygwin and is in MacPorts.*

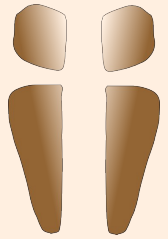


## *The SCons Process*

- SCons works in two phases:
  1. Read and execute the SConstruct file. This creates a directed acyclic graph (DAG) that describes the relationships of all things relating to the build.
  2. Resolve all the build needs determined by analysing the DAG constructed in Phase 1.

*The SCons API is a DSL for constructing DAGs that describe relationships between things in the world.*

*Steven Knight definitely deserves an award.*



# Trivial C++ Build with SCons

## helloWorld.cpp

```
#include <iostream>
int main ( const int ac , const char *const *const av ) {
    std::cout << "Hello world." << std::endl ;
}
```

## scons

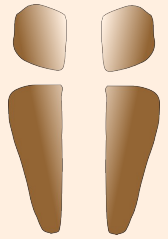
```
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
g++ -o helloWorld.o -c helloWorld.cpp
g++ -o helloWorld helloWorld.o
scons: done building targets.
```

## SConstruct

```
Program ( 'helloWorld.cpp' )
Clean ( '.' , Glob ( '*~' ) )
```

## scons -c

```
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Cleaning targets ...
Removed helloWorld.o
Removed helloWorld
Removed SConstruct~
Removed helloWorld.cpp~
scons: done cleaning targets.
```

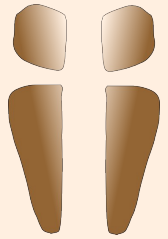


## *SCons and Cleaning*

- In SCons, cleaning is not a target, cleaning is the undoing of a build.

*SCons knows what gets created as part of a build so knows what to clean.*

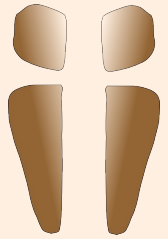
*You can add extra things to the clean activity.*



## *SCons and Targets*

- SCons doesn't have targets as other build frameworks do, SCons describes what files have to exist, and actions taken for a build to be deemed successful.

*You can create the appearance of there being targets in the traditional sense using the Alias function.*



## *Out of Tree Builds – 1*

```
·  
|-- SConstruct  
`-- Source  
    |-- SConscript  
    `-- helloWorld.cpp
```

*Source/SConscript*

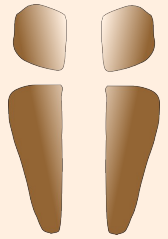
Program ( 'helloWorld.cpp' )

### *SConstruct*

```
buildDirectory = 'Build'
```

```
SConscript ( 'Source/SConscript' , variant_dir = buildDirectory , duplicate = 0 )
```

```
Clean ( '.' , Glob ( '*~' ) + Glob ( '*/*~' ) + [ buildDirectory ] )
```



## Out of Tree Builds – 2

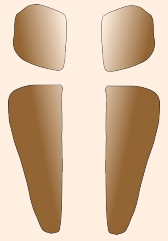
### *scons*

```
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
g++ -o Build/helloWorld.o -c Source/helloWorld.cpp
g++ -o Build/helloWorld Build/helloWorld.o
scons: done building targets.
```

### *scons -c*

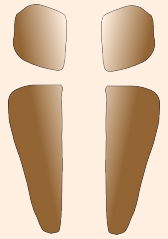
```
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Cleaning targets ...
Removed Build/helloWorld.o
Removed Build/helloWorld
Removed directory Build
scons: done cleaning targets.
```

```
.
|-- Build
| |-- helloWorld
| `-- helloWorld.o
|-- SConstruct
`-- Source
    |-- SConscript
    `-- helloWorld.cpp
```



## *SCons is Pythonic*

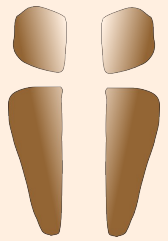
- SCons API relies on:
  - The parameter passing flexibility of Python:
    - Positional parameters.
    - Named parameters.
  - Duck typing for further flexibility of parameters.
    - Especially interchanging scalars and lists.
  - Dynamic execution of code.
  - Graying the boundary between functional call and object initialization.



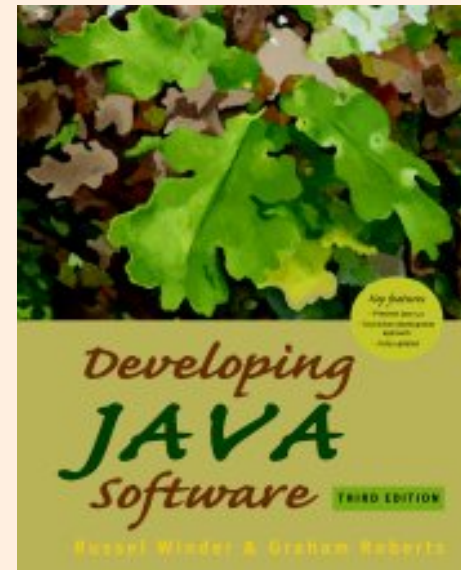
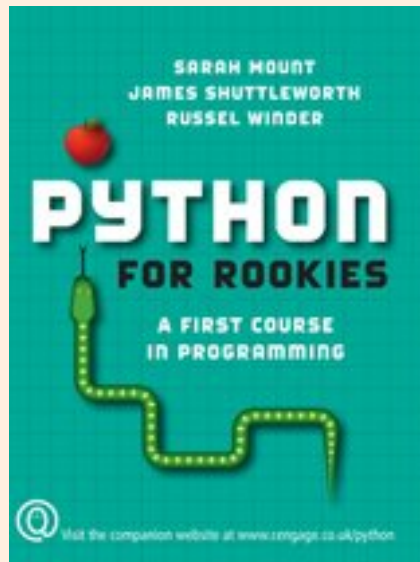
## *Summary*

- Dynamic languages are the future of build frameworks.
- Python is good at being a dynamic language.
- SCons provides a great DSL for describing builds.
- Autotools is the system of past choice, SCons is the system of now and future choice.

*The SCons website is at:  
<http://www.scons.org>*



# *Unabashed Advertising*



*You know you want to buy them!*

