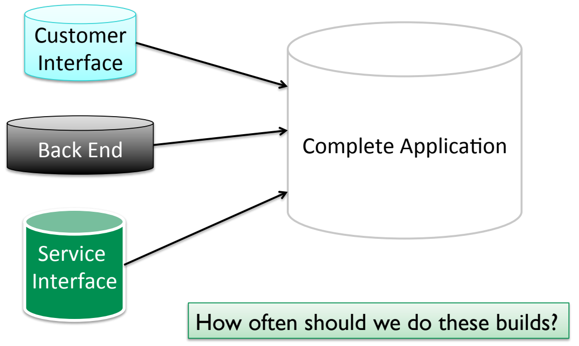
Review

- What are build tools?
  - What do they do?
- What are examples of build tools?
  - How do they work?

Build and Management Tools

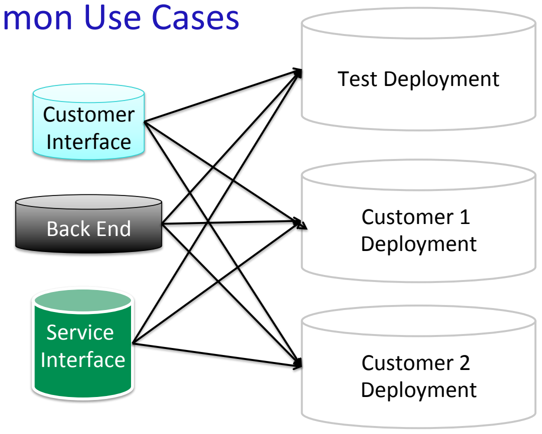
- Maven
- Continuous Integration

Motivating Build Tools: Common Use Cases

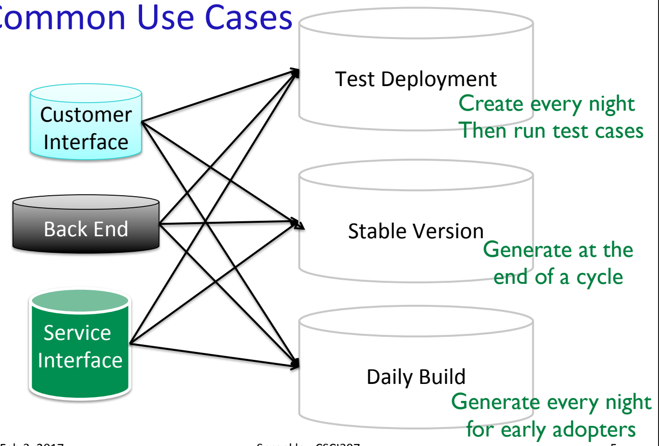


How often should we do these builds?

Motivating Build Tools: Common Use Cases



Motivating Build Tools: Common Use Cases



Comparing Make and Ant

```
simulator: $(OBJECTS)
        $(CC) $(CFLAGS) –o simulator $(OBJECTS)
simulator.o: simulator.c
        $(CC) $(CFLAGS) –c simulator.c

customer.o: customer.c
        $(CC) $(CFLAGS) –c customer.c
…
clean:
        rm $(OBJECTS) simulator
```

```
<target name="compile"
      description="Compile the source code">
   <mkdir dir="build/classes"/>
   <javac srcdir="src"
          destdir="build/classes"
          debug="on">
      <include name="**/*.java"/>
      <classpath refid="build.class.path"/>
   </javac>
</target>
```

Apache Maven Project
http://maven.apache.org/

Apache Maven™

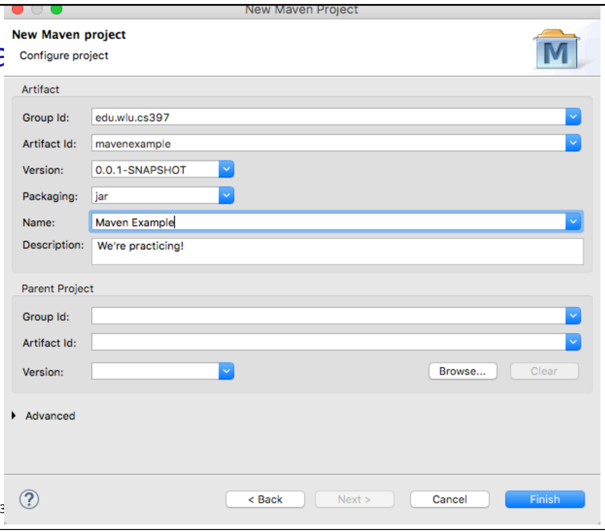
- Maven: Yiddish word meaning *accumulator of knowledge*
- For building and managing any Java-based project
  - Uses a Project object model (POM)
- Goal: download and build a project quickly

http://maven.apache.org/

Maven

- Can be used as standalone tool or within Eclipse (what we'll do)

Ma



Group Id: edu.wlu.cs397
Artifact Id: mavenexample
Version: 0.0.1-SNAPSHOT
Packaging: jar
Name: Maven Example
Description: We're practicing!

Parent Project
Group Id:
Artifact Id:
Version:

Maven Philosophy: Convention Over Configuration

- Maven's location assumptions:
  - source code: ${basedir}/src/main/java
  - Resources: ${basedir}/src/main/resources
  - Tests: ${basedir}/src/test
- Other assumptions:
  - Want to produce a JAR file in ${basedir}/target
  - Compile byte code to ${basedir}/target/classes

How does this convention philosophy help us?

Maven Philosophy: Convention Over Configuration

How does this philosophy help us?

- Ant-based builds *define* locations
  - No built-in idea of where source code or resources are
  - **User** has to supply this information → more work for us!!

Could be for any project:

```
<target name="compile"
      description="Compile the source code">
   <mkdir dir="build/classes"/>
   <javac srcdir="src"
          destdir="build/classes"
          debug="on">
      <include name="**/*.java"/>
      <classpath refid="build.class.path"/>
   </javac>
</target>
```

Maven Philosophy: Convention Over Configuration

- Beyond location conventions…
- **Core plugins** apply a common set of conventions for compiling source code, packaging distributions, generating web sites, and many other processes
  - Example: similar to Ant compile target
- Little effort:
  - Put source in the correct directory
  - Maven handles the rest

Consequences of Convention Over Configuration

- Users may feel forced to use a particular methodology or approach
- Most defaults can be customized
- Can create custom plugins for your requirements

Maven Build Lifecycle

- Defined by a list of *build phases*
- Example build phases
  - `compile` - compile the source code of the project
  - `test` - test the compiled source code using a suitable unit testing framework
  - `package` - take the compiled code and package it in its distributable format, such as a JAR
- When execute a phase, executes life cycle's previous phases first, in order
  - E.g., calling package would execute compile and then test

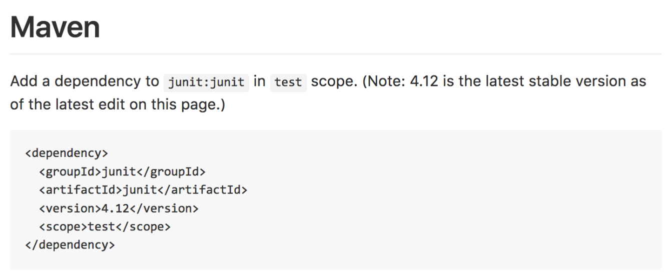
Maven Build Lifecycle

- 3 built-in build lifecycles
  - `default` lifecycle handles project deployment
  - `clean` lifecycle handles project cleaning
  - `site` lifecycle handles the creation of project's site documentation

Creating a Test Class

- Make sure to put your test class in the right place, with an appropriate package name
- Click setUp, add tests
- DON'T ADD JUNIT TO YOUR CLASSPATH

Adding a Dependency



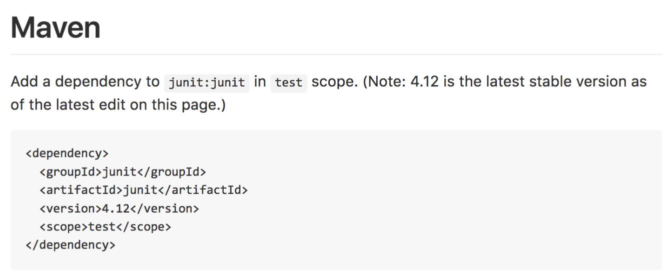
Maven

Add a dependency to `junit:junit` in `test` scope. (Note: 4.12 is the latest stable version as of the latest edit on this page.)

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

- Several different ways this can be done
- After add the first one, can just copy paste the XML code provided

Adding a Dependency



Maven

Add a dependency to `junit:junit` in `test` scope. (Note: 4.12 is the latest stable version as of the latest edit on this page.)

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

- Give Eclipse some time to work it out
- View the Maven Dependencies in Eclipse

Maven Repository

https://mvnrepository.com/

- How to use it
- Typically: looking for a stable release
  - rc = release candidate

Summary: Build Tools

- Automate process of building various "artifacts" from your source code
  - Examples: compile, distribute (jars), documentation, commercial_version, …

Summary: Build Tools

- Automate process of building various "artifacts" from your source code
  - Examples: compile, distribute (jars), documentation, commercial_version, …
- Why is there more than one build tool?
- What are the similarities and differences between make, ant, and maven?

Running Discussion Questions

- Why does the tool exist? What is its purpose?
- What can the tool do?
- What can't the tool do?
  - Because it hasn't been done? Because of current technology limitations? Or some other limitations?
  - If because it hasn't been done, what can do to change that?