Maven in the wild

An introduction to Maven

Maven gone wild!!

An introduction to Maven

Presentation Summary

- An overview of Maven
- What Maven provides?
- Maven's principles
- Maven's benefits
- Maven's features
- Related Maven technologies
- Questions?

Mergere

An overview of Maven

- What is Maven?
  - A project management framework: a set of standards, repository format and software
  - Model and tools which provide inherent utility
  - Bring standards to builds and project management, not a “we know better” framework
  - A lens for quality
  - A way to build relationships between many projects
  - Encourage a culture of reciprocity

Mergere

What does Maven provide?

- A comprehensive model for software projects
- Tools that interact with Maven's declarative model
  - Coherence
  - Reusability
  - Agility
  - Maintainability

Mergere

Maven's principles

- Modeled after Christopher Alexander's idea of patterns
- To provide a lingua franca or shared language for project management

Mergere

Maven's principles

- Convention over configuration
  - Standard directory layout
    - Finding all project content
  - One primary output per project
    - Client/Server/Utility code example
    - Separation of concerns (SoC)
  - Standard naming convention
    - foo-1.0.jar vs foo.jar

Mergere

Maven's principles

- Declarative execution
  - Maven Project Object Model (POM)
  - Maven's Super POM
  - Maven's build life cycle

Mergere

A typical POM

- With this you can compile, test, package, install and create a minimal site! How?

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.myproject</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>My App</name>
</project>
```

Mergere

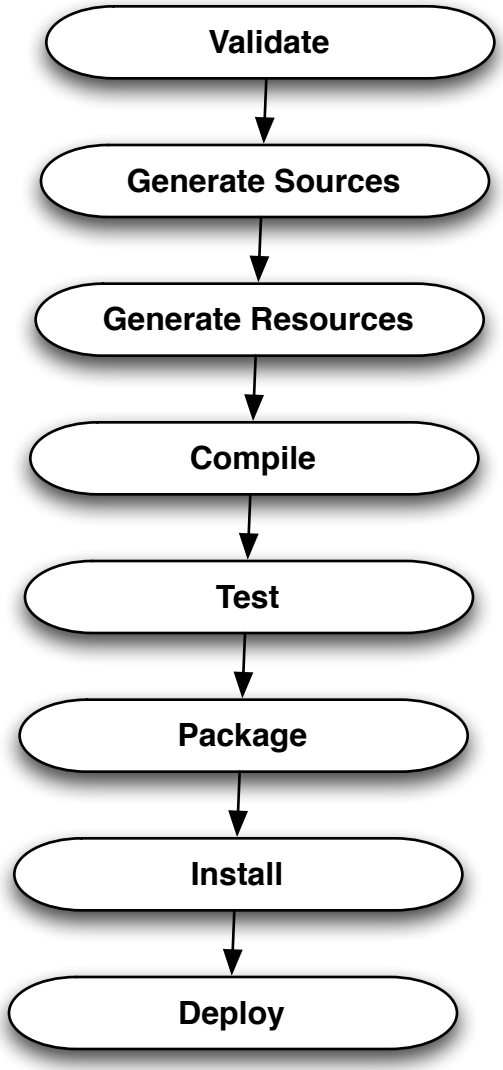
Super POM

```
<project>
  <modelVersion>4.0.0</modelVersion>
  ...
  <build>
    <directory>target</directory>
    <outputDirectory>target/classes</outputDirectory>
    <finalName>${artifactId}-${version}</finalName>
    <testOutputDirectory>target/test-classes</testOutputDirectory>
    <sourceDirectory>src/main/java</sourceDirectory>
    <scriptSourceDirectory>src/main/scripts</scriptSourceDirectory>
    <testSourceDirectory>src/test/java</testSourceDirectory>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
      </resource>
    </resources>
    <testResources>
      <testResource>
        <directory>src/test/resources</directory>
      </testResource>
    </testResources>
  </build>
</project>
```

Mergere

Maven's build life cycle

- Builds in Maven *follow a pattern*
- *Plug-ins slot into the life cycle to add functionality*
- Ensures developers moving between projects *do not need to learn new processes*



Mergere

Maven's principles

- Reuse of build logic
  - Maven is a plug-in execution framework coordinated by the build life cycle
  - All build logic is encapsulated in plug-ins with a separation of concerns in mind (SoC)
  - Plug-ins can be applied to any Maven Project

Mergere

Maven's principles

- Coherent organization of dependencies
  - Repositories
  - Artifacts
  - Dependency analysis

Mergere

Repositories

- Where all the artifacts are stored
  - *Local repository* refers to a cache of artifacts used on local machine
  - *Remote repository* refers to a source of artifacts, available over file:, http:, etc.
- Central repository contains popular *redistributable artifacts* (over 8000 at last count)
  - http://repo1.maven.org/maven2/

Mergere

Artifacts

- The result of a build in Maven is called an *artifact: a JAR, a WAR, an EAR*
- *A single artifact can be referenced by any number of projects*

Dependency analysis

- Declaratively specifying dependencies allows you to perform exacting analysis. This is hard to do with an arbitrary directory full of JARs
- When working with multiple projects that use the same dependency, determining a suitable version is critical

Mergere

Maven's benefits

- Removes a lot of the burden of build and project maintenance
- Easy for users to embrace best practices
- Draw upon the active Maven open-source community for solutions
- Take your development process to the next level

Mergere

Related Maven technologies

- Maven Wagon
- Maven SCM
- Continuum
- Maven Repository Manager
- IDE integration
- In the near future:
  - Distributed Continuum (GBuild)
  - Dashboard
  - Project Infrastructure Bootstrapper
  - Maven Issue & Maven Wiki

Mergere

Maven 2.0 Features

- Fast, small – embeddable
- Enhanced dependency support
- Build life cycle
- Enhanced plug-in support
- Multi-module project support
- Site and documentation enhancements
- Release management
- *Archetypes* - project templates
- Build Profiles

Mergere

Transitive Dependencies

- Always enabled in Maven 2.0
- Features:
  - Don't need to declare dependencies of dependencies yourself
  - Dependency mediation
  - Intelligent scoping
  - Fine grained control over versioning and exclusions

Mergere

Snapshot Handling

- Deploying to a shared repository gives a version with a *time stamp and build #*
- Don’t need to update dependency version to get updated builds
- Updates daily, on-demand, or at a particular interval

Mergere

On-demand Features

- Plug-ins can be *requested on-demand* from the command line
- In Maven 1, this required manual installation
- For example, `mvn idea:idea` will generate an IDEA project file without modifications to your project

Plugin Version Discovery

- Can opt not to declare a plugin version in your project
- Will regularly check for a new release, and download it if desired
- Users can opt to get prompted for new releases of plugins
- Release tool will record the active version for reproducible builds

Mergere

Plugin Languages

- Java, Beanshell
- Java is the most common
- Beanshell is new, useful for *rapid prototyping*
- Can support others with a small amount of work if there is demand
  - For example: Jython, Groovy, JRuby

Mergere

Multiple Modules

- Maven 2 natively deals with multi-module builds
- A module refers to another project in the build tree
- Goals are performed on all found modules by default
- Modules can in turn have modules

```
<modules>
  <module>wagon-provider-api</module>
  <module>wagon-providers</module>
</module>
```

Mergere

Site and Documentation

- Accepts *several input formats*
  - Almost Plain Text (Wiki like)
  - Xdoc (Maven 1.0 compatible)
  - FAQ (Maven 1.0 compatible)
  - Docbook
- Presently outputs XHTML, Xdoc, Docbook, Latex and RTF, and PDF

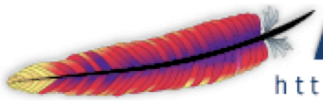
Mergere

Example APT Document

```
 ------
 Generating a Site
 ------
 Maven Documentation Team
 ------
 13 May 2005
 ------

Building a Site

* Creating Content

  The first step to creating your site is to create some content. In
  Maven 2.0, the site content is separated by format, as there are several
  available.

-------------------
+- src/
   +- site/
      +- apt/
      |  +- index.apt
      +- site.xml
--------------------

  The Xdoc format is the same as
  {{{http://maven.apache.org/using/site.html} used in Maven 1.0}}.
  However, <<<navigation.xml>>> has been replaced by the site descriptor
  (see below).
```

Mergere

Example APT Document



Apache Maven Project

http://maven.apache.org/

Maven

Last Published: Tue May 31 09:32:59 EST 2005 — Apache | Maven 1.0 | Maven 2

Maven 2.0
- Introduction
- Download
- Release Notes
- General Information
- For Maven 1.0 Users
- Road Map

User's Guide
- Getting Started
- Configuration
- Dependency Mechanism
- Developing Plugins
- Developing Plugins with Marmalade
- Creating a Site

Reference
- Project Descriptor
- Settings Descriptor
- Available Plugins
- Mojo API
- Ant Tasks

Developers
- Documentation Needed

Building a Site

Creating Content

The first step to creating your site is to create some content. In Maven 2.0, the site content is separated by format, as there are several available.

```
+- src/
   +- site/
      +- apt/
      |  +- index.apt
      +- site.xml
```

The Xdoc format is the same as used in Maven 1.0. However, `navigation.xml` has been replaced by the site descriptor (see below).

Mergere

Release Assistance

- Resolves information in the project *to make the release reproducible*
- Updates the version information, commits and tags a release
- Does a clean checkout and builds the release

Mergere

Project Archetypes

- Currently archetypes for:
  - JAR/WAR
  - Site
  - Java Plugins
  - Can easily create your own archetypes
- Uses Velocity
- Downloaded from the repository so they are easily shared

Mergere

Build Profiles

- Change the build depending on the *environment*
  - Dependencies, repositories, plugins and configuration
- *Trigger* by operating system, JDK, extistence of software, and so on, as well as command line parameter
- Per user or per project
- Used to *set up standard environments*:
  - Development, Test, QA and Production

Mergere

Support for Other Languages

- Being implemented as plugins
- Currently we have seen work on a C# compiler, and plan to support C/C++ environments on Unix and Windows

Mergere

Questions?

- Thanks for listening!

Mergere