Joyful Metamorphosis: Converting an Enterprise Ant Build to Maven

Jason van Zyl

Founder
Sonatype | Apache Maven
http://www.sonatype.com | http://maven.apache.org

TS-7496

Make Your Team Happy With Maven!

Regain control of your development infrastructure

Understand the concerns related to converting massive Ant builds, the benefits of conversion, and how to convert an Ant build in a practical way.

Presentation Agenda

Understanding What Maven Is

Why Maven Is Important

The Ant Beast to Tame

Issues Converting Wayward Ant Projects

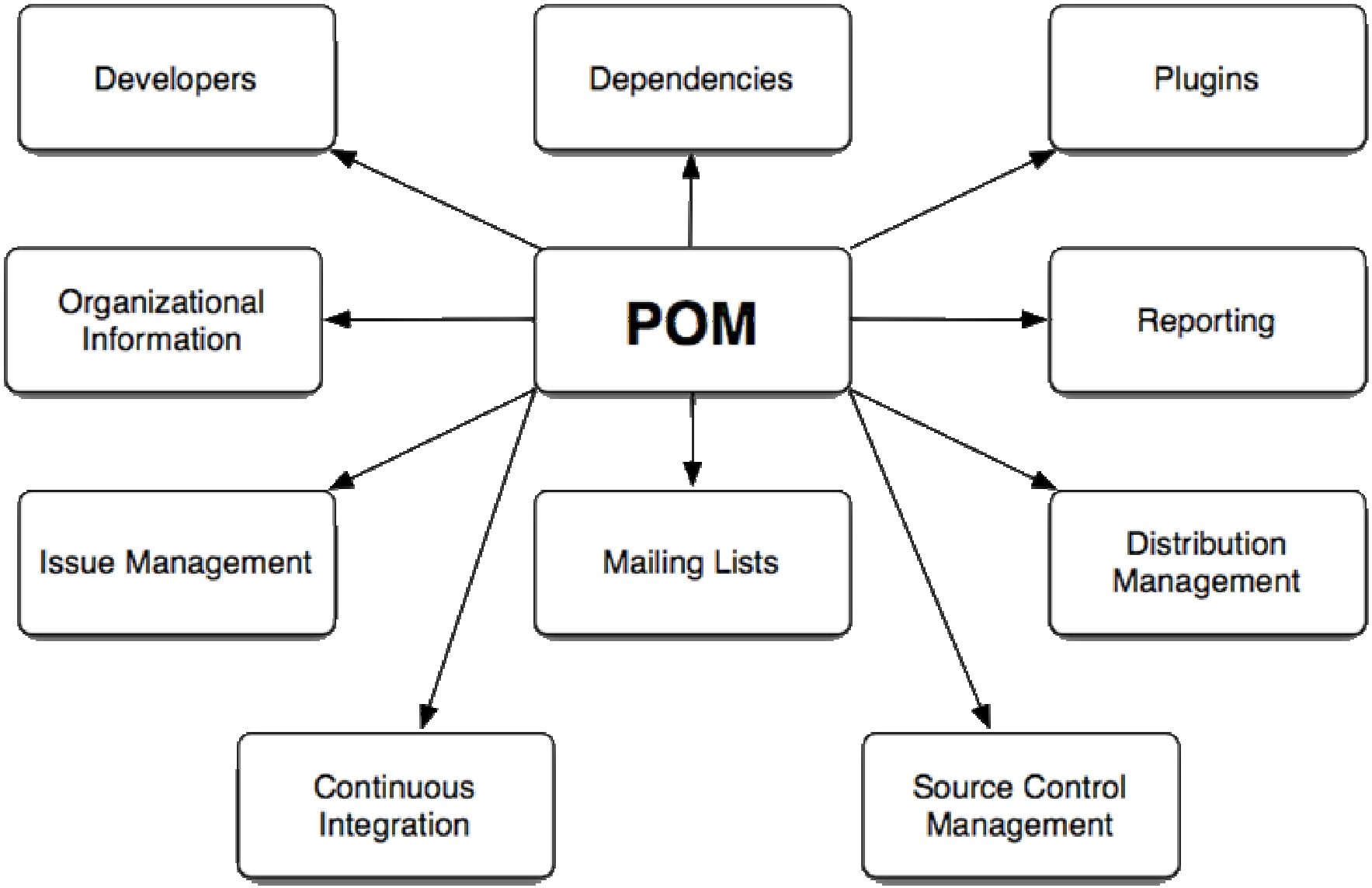
Results and Conclusions

What Is Maven?

- Build tool
  - Similar to Make, or Ant, but fundamentally different in that the larger context of all groups developing with Maven is taken into consideration
- Dependency management tool
  - Similar to Ivy, but not bolted on as an afterthought
- Site management tool

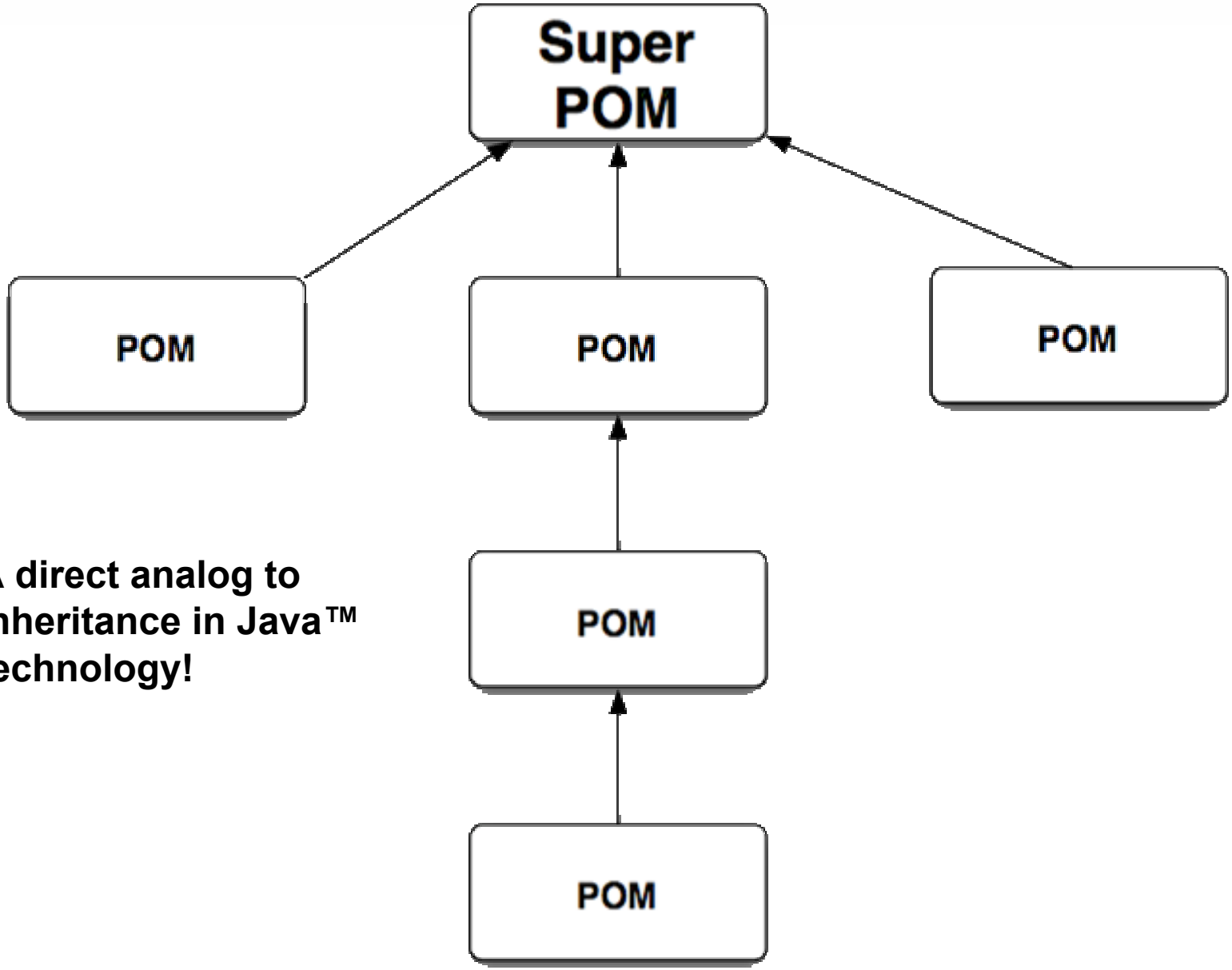
What Maven Really Is...

- Model for software projects
- Patterns for software development and development infrastructures
- Ultimately the basis for a new form of team collaboration


Developers
Dependencies
Plugins
Organizational Information
POM
Reporting
Issue Management
Mailing Lists
Distribution Management
Continuous Integration
Source Control Management

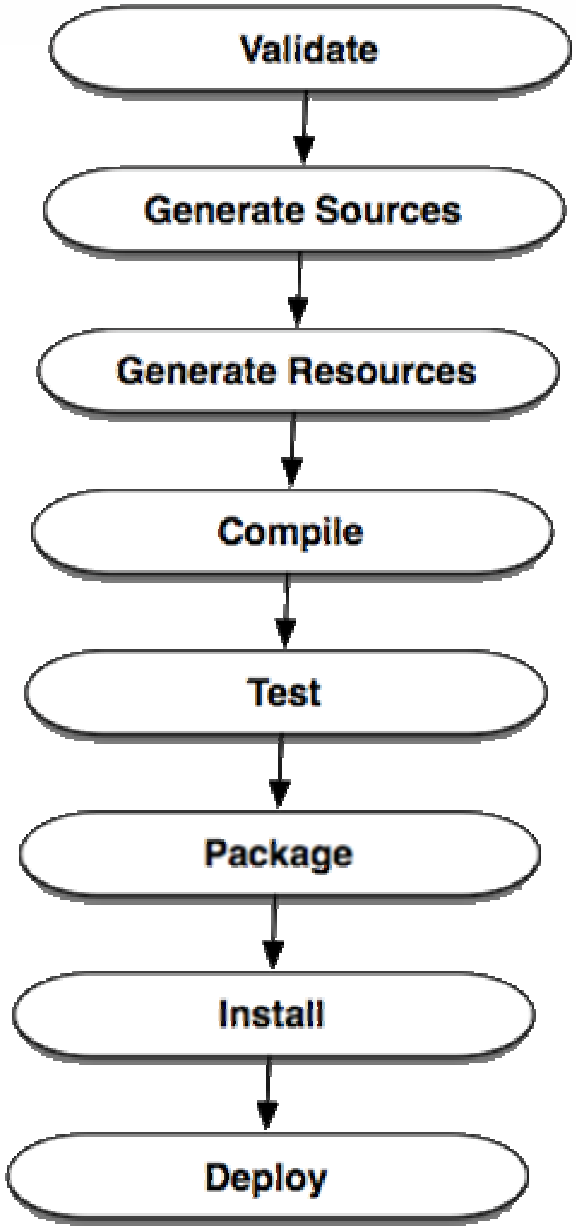
Simple Project Object Model

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.iona.superapp</groupId>
  <artifactId>superapp</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</name>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```



A direct analog to inheritance in Java™ technology!

Doesn't matter what you build, the lifecyle is the same: Java Archive (JAR), WAR, EAR.



What's different are the tools you plug into each phase.

A Consistent Model and Pattern Makes...

- Automation easier
  - Releasing
  - Continuous Integration
  - Mylar and IDE Bootstrapping
- Tooling easier
  - Dependency Metadata
  - Plugin Metadata
  - Standard Lifecycle
- Real dependency analysis possible
  - Try integrating 15 projects that each have a directory of JAR files

Development Infrastructures

- Development Infrastructures must provide for:
  - Building
  - Testing
  - Continuous Integration
  - Releasing
  - Artifact Management
  - Provisioning
  - Documenting

Presentation Agenda

Understanding What Maven Is

Why Maven Is Important

The Ant Beast to Tame

Issues Converting Wayward Ant Projects

Results and Conclusions

Infrastructure, Portland, and Maven

- Intentional infrastructure
  - You have to invest in your infrastructure to yield returns
  - It must be carefully planned, infrastructures don't just happen
- Long-term sustainability
  - The key is keeping people involved
  - Sustainability by virtue vs. sustainability by force
- Healthy growth in mind
  - Some constraints are necessary
- Promotion of community
  - Create opportunities for people to interact

Questions About Infrastructure

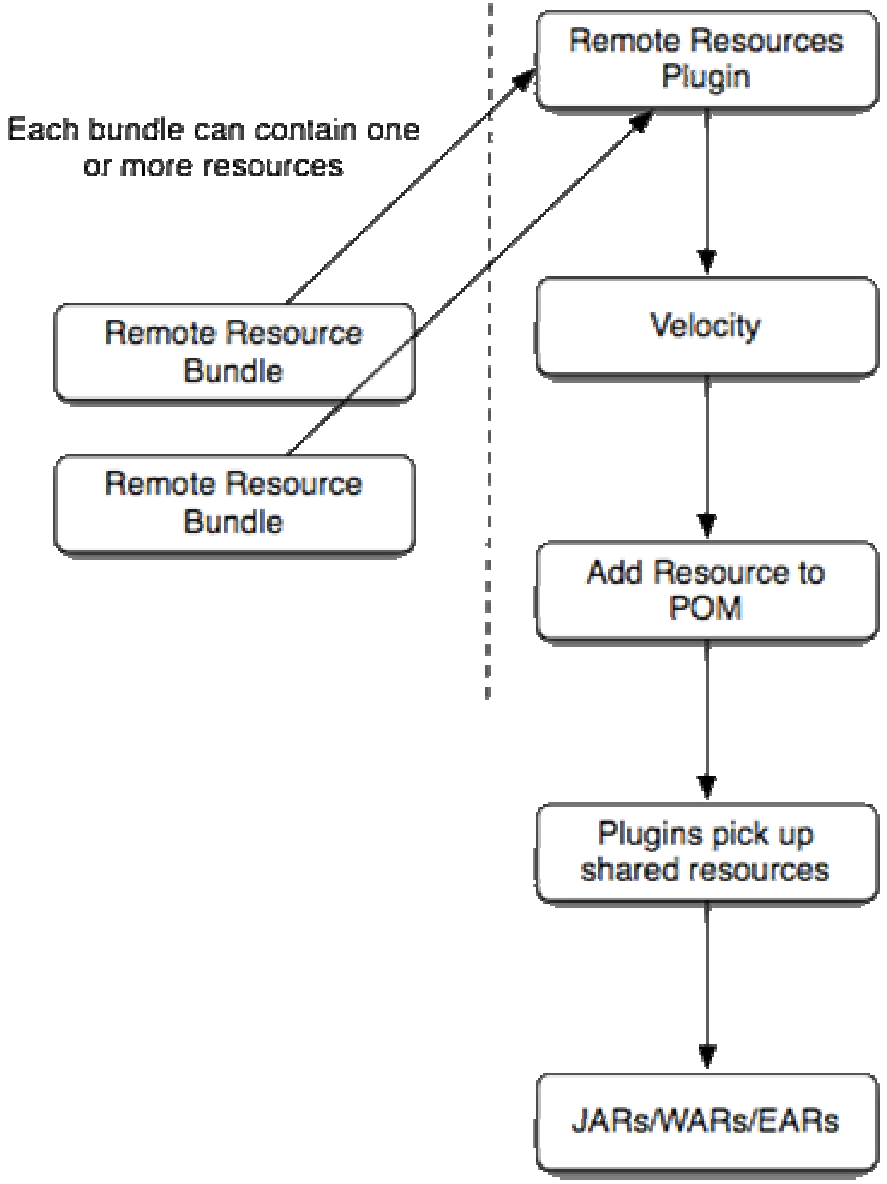
- Do you think your infrastructure is wildly different then anyone else's?
- Is your infrastructure a competitive advantage?
- Would you like help with the most difficult infrastructure problems your organization might face?

Inherent Utility

- The infrastructure should provide a lot to make complicated tasks easier
- When complicated tasks are made easy, there is more time for the bigger picture
- What's in the bigger picture?

Maven and Social Capital

- Term coined by Jane Jacobs (The Death and Life of Great American Cities)
- People's ability to work together in groups
- Creation of real communities using the idea of patterns and conscious planning
- A concrete example of the Maven community working towards improvement; Apache release process for projects using Maven



The remote resources plugin takes the bundles, puts them in a ClassLoader and hands it off to Velocity.

Each resource from each bundle is processed through Velocity. The POM is pushed into the Velocity context.

All resources once processed are placed in ${basedir}/target/maven-shared-archive-resources and a Resource entry is added to the POM which corresponds to that directory.

Any archiving plugin that is aware of the shared archive resources Resource entry in the POM can pick them up and insert them into the archive it creates.

All archives created by projects using Maven now have their all their legal requirements for packaging satisfied transparently

Maven's Objectives

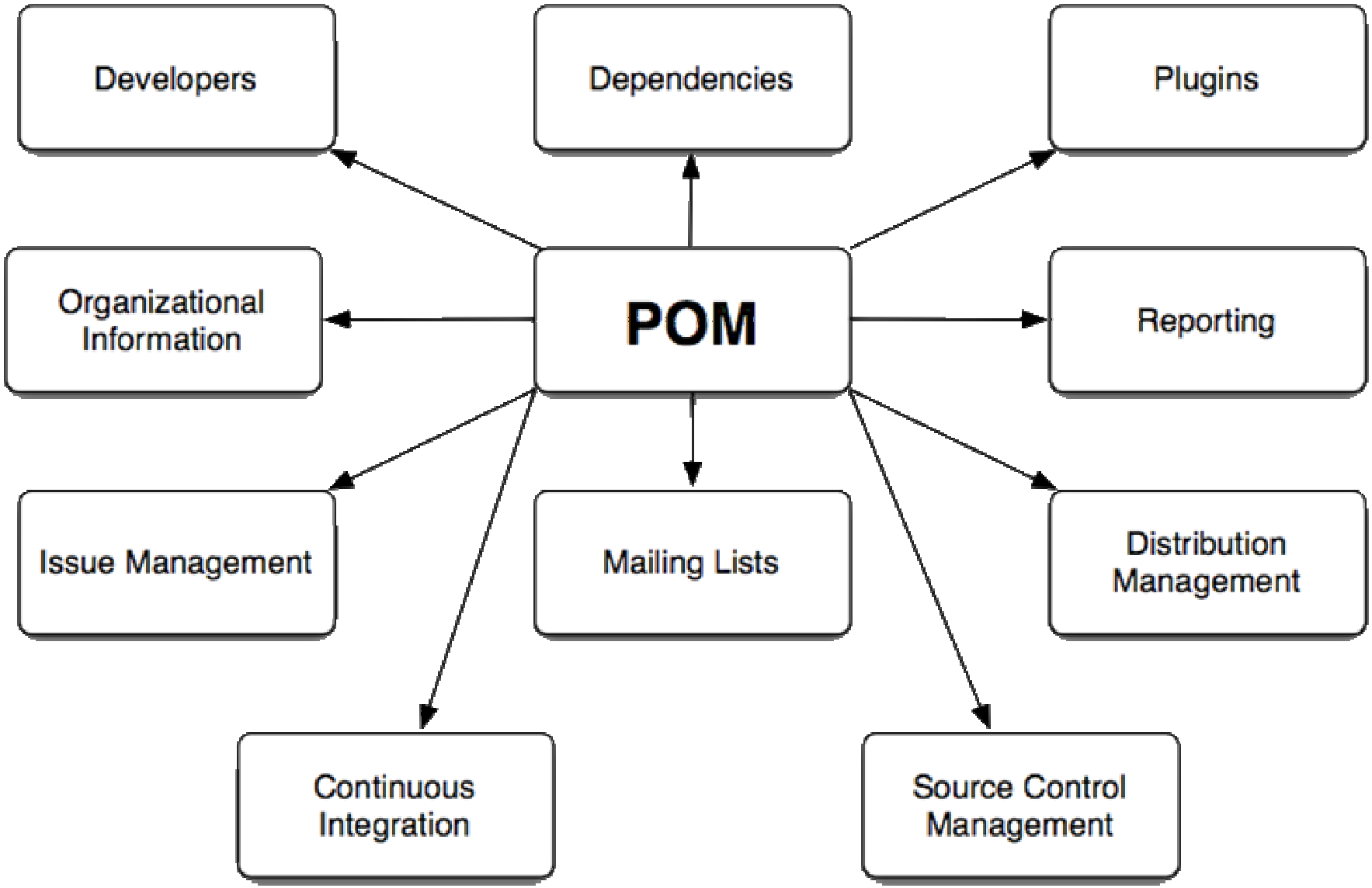
- Standards and Best Practices for build infrastructures
  - We use standard APIs and best practices when developing applications; Maven pushes this practice down to the level of infrastructure
- Provide a shared language for build infrastructure management
  - Patterns for build infrastructures in the spirit of "A Pattern Language" by Christopher Alexander
- Create healthy and robust build infrastructures that hold up to high degrees of flux

Maven's Principles

- Model-driven development
- Convention over configuration
- Reuse and encapsulation of build logic
- Coherent organization of dependencies

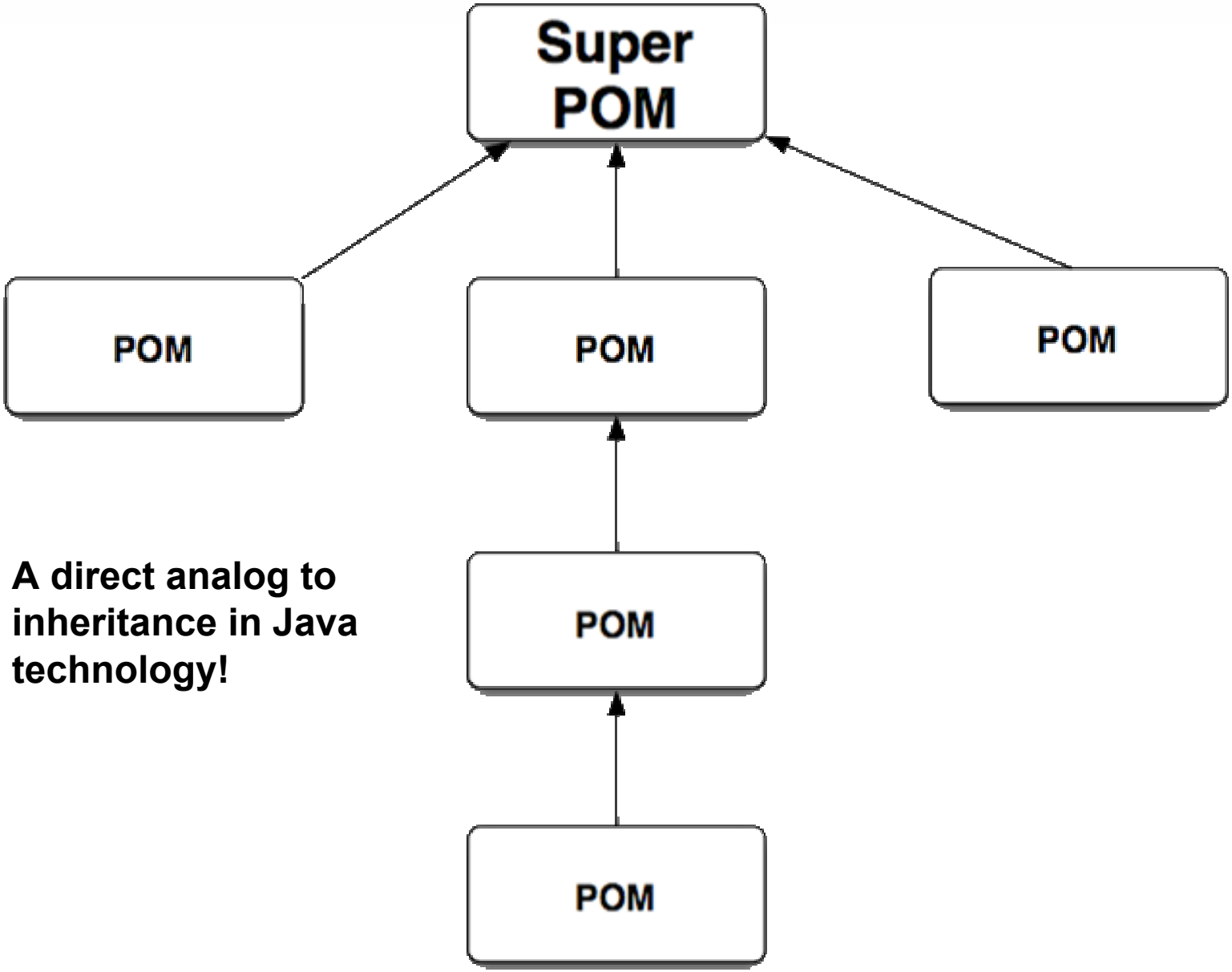
Model-Driven Development

- Maven's project object model (POM)
- Maven's build life cycle


Developers
Dependencies
Plugins
Organizational Information
POM
Reporting
Issue Management
Mailing Lists
Distribution Management
Continuous Integration
Source Control Management

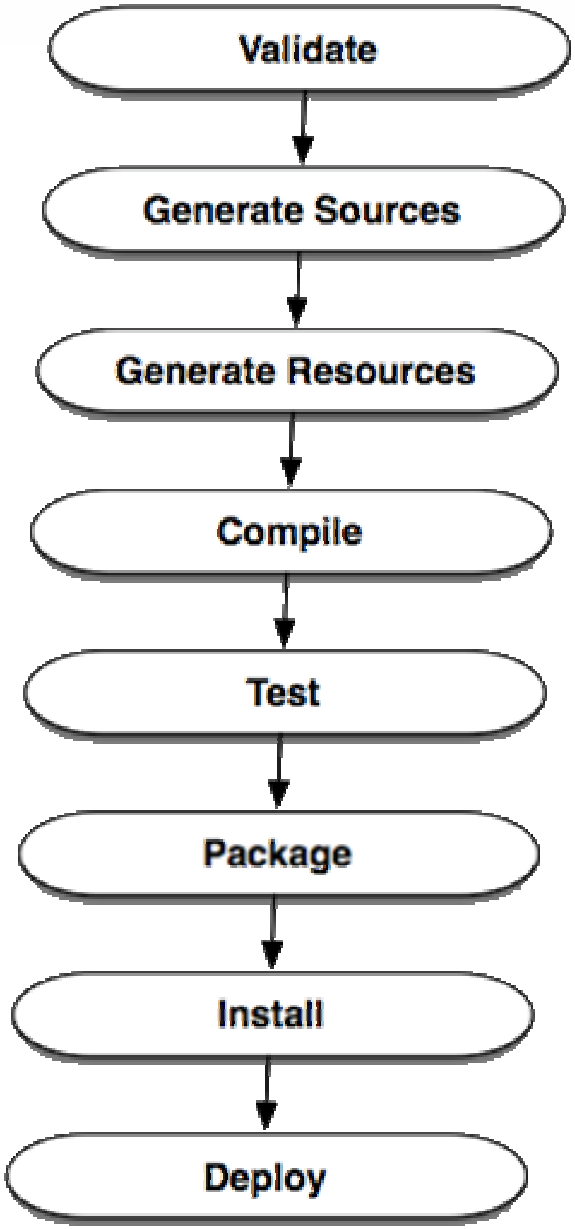
Simple Project Object Model

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.iona.superapp</groupId>
  <artifactId>superapp</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</name>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```



A direct analog to inheritance in Java technology!

Doesn't matter what you build, the lifecyle is the same: JAR file, WAR, EAR.



What's different are the tools you plug into each phase.

Convention Over Configuration

- Standard directory layout
- One primary artifact per build
- Standard naming conventions

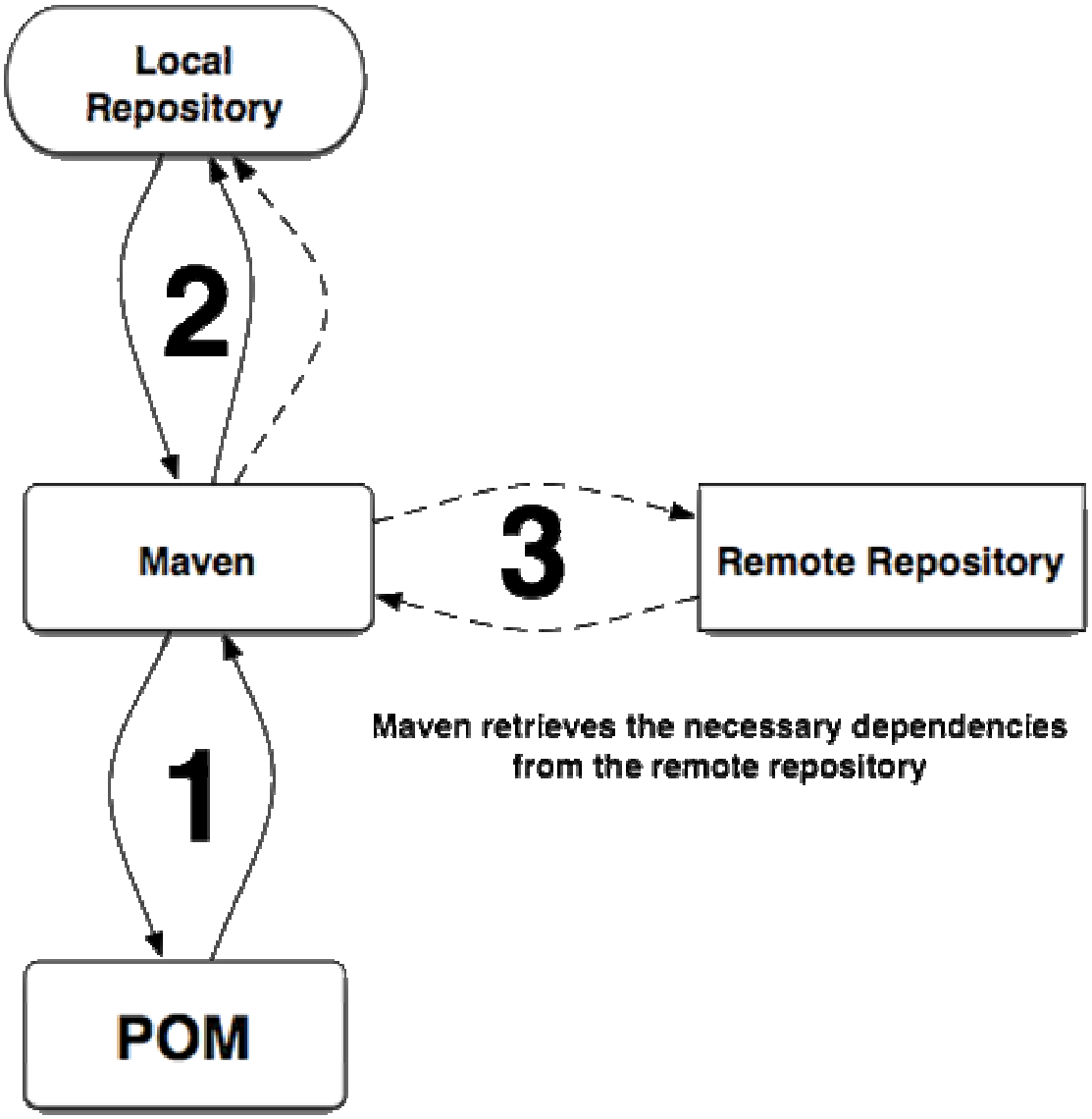
Encapsulation and Reuse of Build Logic

- All build logic is encapsulated in plugins
- Maven is a plugin execution framework
- Plugins can be applied to all Maven projects

Coherent organization of Dependencies

- Say what you need, not where or how to get it
  - Dependencies in Maven are requested in a declarative fashion
- Artifacts and Repositories
  - Remote repositories for the satisfaction of dependencies
  - Local repository is the developers personal analog to a set of remote repositories

Maven checks the local repository for the required dependencies



Maven retrieves the necessary dependencies from the remote repository

Maven gathers dependency information from the POM

What Maven Provides

- Coherence—An orderly, logical, and aesthetically consistent relation of parts
- Reusability—Reuse not only of software components but the best practices of an entire industry
- Agility—Maven allows easier integration and project straddling
- Maintainability—Maven projects are more maintainable as there are far fewer surprises

Benefits of Maven

- Relieves the burden of project and build maintenance
- Easy for new users to embrace Best Practices
- Focus on adding value to your applications
- Draw upon the community for solutions

Presentation Agenda

Understanding What Maven Is

Why Maven Is Important

The Ant Beast to Tame

Issues Converting Wayward Ant Projects

Results and Conclusions

Maven Poll (User's Group)

- IBM
- RedHat
- Standard Bank
- UMB Bank
- UBS Investment Bank
- Accenture
- Fujitsu Services
- IONA
- Cisco
- Disney
- E*Trade
- Bank of America
- Capgemini France
- Bekk
- FJA-US
- Kantega
- JBoss
- Apache
- and more...

The Project

- The presentation system for a large financial services company
- Java platform
  - 27,242 files
  - 5,823,350 lines
- Ant
  - 129 files
  - 23,412 lines
- Three primary deliverables
- Two-week rollout cycle

Problems to Solve

- Removal of the Inner Platform Effect
  - The build created with Ant had become so complex that it was almost impossible for anyone besides its creators to modify or improve it
- Adoptions of Development Infrastructure Standards
  - Maven has become the de-facto model for development infrastructures with the POM, standard build lifecycle, plugin model, and dependency management
- Ease the Cost of Maintenance
  - The size of the build was getting out of control and making improvements were costly and teaching new people was pretty much impossible

Conversion Issues

- Directory structures
  - Converting initially with the conversion vs. leaving the original directory structure
- Multi-artifact builds
  - Using Maven's model of one artifact per build vs. adjusting Maven to work with your model
- Checked-in dependencies
  - Using Maven's native repository mechanism vs. using an existing checked in store of dependencies
- Tooling
  - Using Maven plugins where possible vs. using your Ant scripts as Ant-based Maven plugins

Presentation Agenda

Understanding What Maven Is

Why Maven Is Important

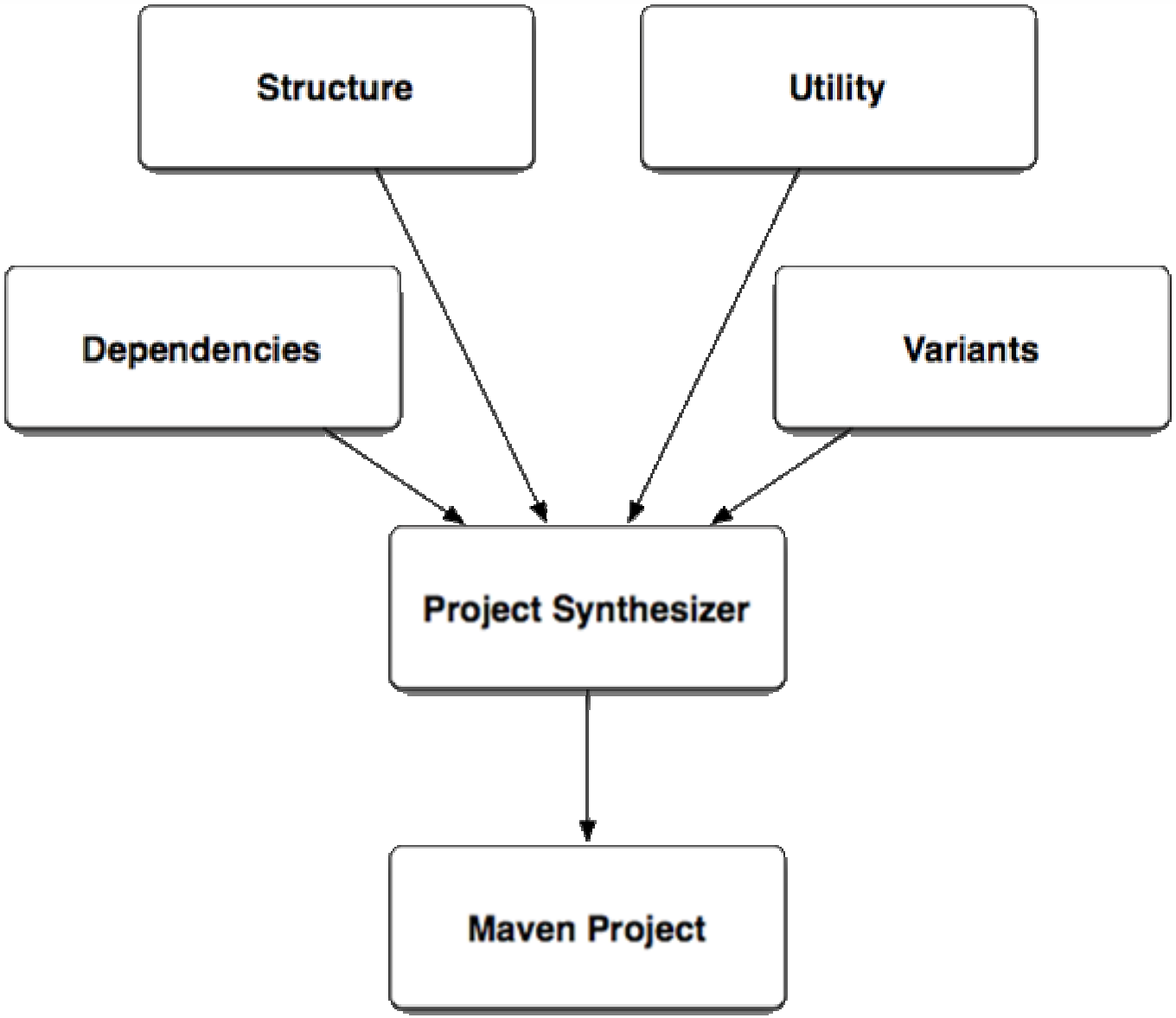
The Ant Beast to Tame

Issues Converting Wayward Ant Projects

Results and Conclusions

Conversion Tools

- Nexus
  - A repository management application that indexes Maven repositories and provides a query mechanism to let us search for artifacts using their MD5 checksum
- Metamorphosis
  - An adaptable conversion tool that allows the creation of POMs with artifact information and populates local and remote Maven repositories; This takes care of 85% of conversion work and eliminates almost all manual lookups of dependencies
- One-off scripts
  - Sometimes you just need to whip off some Bash or Ruby to take care of some of the edge cases


Structure
Utility
Dependencies
Variants
Project Synthesizer
Maven Project

Maven Ant Tasks

- Retrieve remote dependencies
- Create file sets from remote dependency sets and manipulate using familiar Ant idiom
- Generally integrate with Maven's infrastructure
  - Use POMs
  - Install Artifacts
  - Deploy Artifacts

Maven Ant Tasks :: Setup

```
<project name="maven-ant-tasks-example" default="build">
  <target name="init">
    <path id="maven.classpath">
      <pathelement
location="${basedir}/lib/maven-ant-tasks-2.0.6.jar" />
    </path>

    <typedef resource="maven/antlib.xml" uri="urn:maven">
      <classpath refid="maven.classpath"/>
    </typedef>

    <!-- Setup properties -->
    <property name="src.dir" value="${basedir}/src/main/java"/>
  </target>
  ...
</project>
```

Maven Ant Tasks :: Setup

```
<target name="setup" depends="init">
  <artifact:localRepository
id="local.repository"
      location="${basedir}/target/local-repo" layout="default"/>

  <artifact:dependencies
    pathId="dependency.classpath" filesetId="dependency.fileset">

   <dependency
     groupId="commons-logging"
     artifactId="commons-logging" version="1.1"/>
  </artifact:dependencies>

  <artifact:pom file="pom.xml" id="maven.project"/>

 <artifact:dependencies
pomRefId="maven.project" filesetId="pom.fileset"/>
 </target>
```

Maven Ant Tasks :: Building

```
  <target name="build" depends="setup">
    <mkdir dir="${out.dir}"/>
    <javac
classpathref="dependency.classpath"
srcdir="${src.dir}" destdir="${out.dir}"/>

<jar basedir="${out.dir}" destfile="${jar}"/>

    <copy todir="${assembly.dir}">
      <fileset refid="dependency.fileset"/>
      <fileset refid="pom.fileset"/>
      <mapper type="flatten"/>
    </copy>

    <zip basedir="${assembly.dir}" destfile="${assembly}"/>
  </target>
```

Maven Ant Tasks :: Deploying

```
<target name="deploy" depends="build">

  <artifact:install file="${jar}">
    <pom refid="maven.project"/>
  </artifact:install>

  <artifact:deploy file="${jar}">
    <remoteRepository url="scp://server/directory"/>
    <attach file="${assembly}" classifier="full" type="zip"/>
    <pom refid="maven.project"/>
  </artifact:deploy>
</target>
```

AntRun Plugin

- Use familiar Ant idiom to perform procedural tasks when you have something custom, or for which there exists no Maven plugin
- Script Ant from within a Maven POM
- Delegate to a standard build.xml file

AntRun Plugin :: Setup

```
<plugin>
 <artifactId>maven-antrun-plugin</artifactId>
 <executions>
  <execution>
   <phase>package</phase>
   <configuration>
    <tasks>
     <property
          name="generatedSources
          value="${generatedSources}"/>
     <ant antfile="${basedir}/build.xml"
          inheritAll="true" inheritRefs="true">
      <target name="do"/>
     </ant>
    </tasks>
   </configuration>
  </execution>
 </executions>
</plugin>
```

AntRun Plugin :: Delegating to build.xml

```
<project name="superapp">
  <target name="do">

   <!--
   Property from the POM that was
   injected into the build.xml file -->
    <echo message="generated sources = ${generatedSources}"/>

    <!-- Generate some sources using an Ant talks -->
    <antlr
     target="grammar.g"
     outputdirectory="${generatedSources"/>

  </target>
</project>
```

Ant Plugin

- The best way to utilize your Ant knowledge in Maven
- Encapsulate large blocks of Ant logic in a Maven plugin for widespread re-use
- To a Maven user an Ant-based plugin is indistinguishable from a Java technology-based plugin

Ant Plugin ::Setup

```
<plugin>
 <artifactId>maven-plugin-plugin</artifactId>
 <dependencies>
  <dependency>
   <groupId>org.apache.maven</groupId>
   <artifactId>maven-plugin-tools-ant</artifactId>
   <version>2.0.4</version>
  </dependency>
  <dependency>
   <groupId>ant-contrib</groupId>
   <artifactId>ant-contrib</artifactId>
   <version>1.0b2</version>
  </dependency>
 </dependencies>
 <configuration>
  <goalPrefix>ap</goalPrefix>
 </configuration>
</plugin>
```

Ant Plugin :: Plugin Metadata

```
<pluginMetadata>
  <mojos>
    <mojo>
      <!-- Name of the Maven goal -->
      <goal>do</goal>
      <!-- Name of the Ant target to call -->
      <call>do</call>
      <description>Do some useful things with Ant</description>
    </mojo>
  </mojos>
</pluginMetadata>
```

Ant Plugin :: The Implementation

```
<project name="antbased" basedir="." default="printout">
  <target name="init">
    <taskdef resource="net/sf/antcontrib/antlib.xml">
      <classpath refid="maven.plugin.classpath"/>
    </taskdef>
    <property name="antbased.print.sysprops" value="true"/>
  </target>

  <target name="do" depends="init">
    <if>
      <isset property="antbased.print.sysprops"/>
      <then>
        <echo>java.version = ${java.version}</echo>
      </then>
    </if>
  </target>
</project>
```

Presentation Agenda

Understanding What Maven Is

Why Maven Is Important

The Ant Beast to Tame

Issues Converting Wayward Ant Projects

Results and Conclusions

The Result

- The Maven build resulted in:
  - 97 POM files
  - 2,910 lines
- We eliminated 32 build files and 20,502 lines of build logic
- We used standard Maven plugins to remove most of the builds logic and converted their Axis tooling written in Ant and wrapped it as an Ant-based Maven plugin

More Information on Maven

- Apache Maven Project
  - http://maven.apache.org
  - http://maven.apache.org/ant-tasks.html
  - http://maven.apache.org/plugins/maven-antrun-plugin/
- Maven: The Definitive Guide
  - http://www.sonatype.com/book

Q&A

Jason van Zyl

Joyful Metamorphosis: Converting an Enterprise Ant Build to Maven

Jason van Zyl

Founder
Sonatype | Apache Maven
http://www.sonatype.com | http://maven.apache.org

TS-7496