CSE 403
Lecture 11

Static Code Analysis

Reading:

IEEE Xplore, "Using Static Analysis to Find Bugs"

slides created by Marty Stepp

http://www.cs.washington.edu/403/

FindBugs

- **FindBugs**: Java static analysis tool that focuses on bugs and usage errors in code.
  - null pointers
  - useless/dead code
  - unclosed I/O streams
  - infinite loops
  - infinite recursion

- FindBugs has been run on the actual JDK 1.6 source, the Eclipse source, and many errors.
  - What kind of bugs and problems were found?

Checkstyle

- **Checkstyle**: A static analysis tool that focuses on Java coding style and standards.
  - whitespace and indentation
  - variable names
  - Javadoc commenting
  - code complexity
    - number of statements per method
    - levels of nested ifs/loops
    - lines, methods, fields, etc. per class
  - proper usage
    - import statements
    - regular expressions
    - exceptions
    - I/O
    - thread usage, ...

Automated Build Systems

- Fairly essential, used on most large programming projects.
  - Why? Why not Makefiles or shell scripts instead?
  - What are these tools aiming to do?
  - What other tools integrate well with them?
  - What features would you want from an automated build tool?

Ant

- **Ant** ("another neat tool"):
  A Java build management tool.
  - developed by Apache to help build their Tomcat web server
  - expanded into a general tool

- Ant is a commonly used build tool for Java programs giving many more build options than the old "Make" utility.
  - built for Java, so it understands Java concepts like:
    - classpath,
    - javac, .class files,
    - JARs,
    - JUnit, etc.

An Ant Build File

- Similar to Make, but Ant uses `build.xml` instead of Makefile:

```
<project>
    <target name="name">
        tasks
    </target>

    <target name="name">
        tasks
    </target>
</project>
```

- A **task** can be a command such as:

```
  <javac  … />
  <mkdir … />
  <delete … />
```

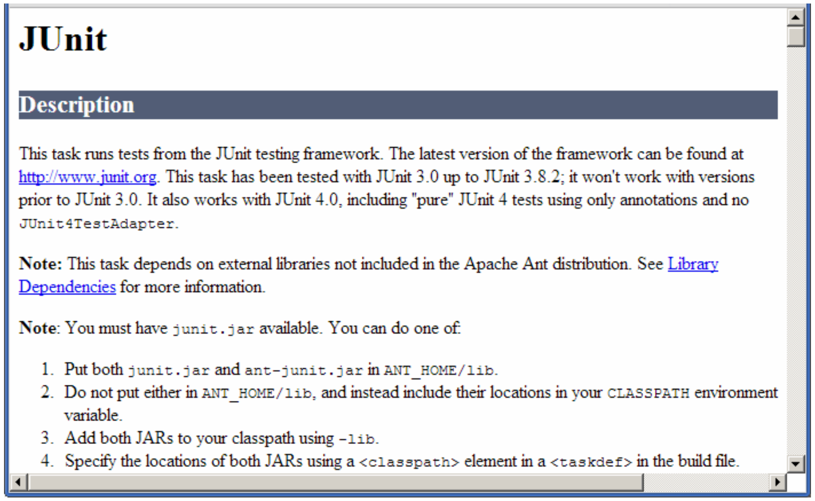
  – More: http://ant.apache.org/manual/tasksoverview.html

Ant build.xml Example

```
<project>
    <target name="clean">
        <delete dir="build"/>
    </target>

    <target name="compile">
        <mkdir dir="build/classes"/>
        <javac srcdir="src"
               destdir="build/classes"/>
    </target>
</project>
```

Ant Task Integration

- To integrate other tools with Ant, download **custom Ant tasks** for those tools.
  - JUnit Ant task
  - Checkstyle Ant task
  - FindBugs Ant task
  - ...



JUnit

Description

This task runs tests from the JUnit testing framework. The latest version of the framework can be found at http://www.junit.org. This task has been tested with JUnit 3.0 up to JUnit 3.8.2; it won't work with versions prior to JUnit 3.0. It also works with JUnit 4.0, including "pure" JUnit 4 tests using only annotations and no `JUnit4TestAdapter`.

Note: This task depends on external libraries not included in the Apache Ant distribution. See Library Dependencies for more information.

Note: You must have `junit.jar` available. You can do one of:

1. Put both `junit.jar` and `ant-junit.jar` in `ANT_HOME/lib`.
2. Do not put either in `ANT_HOME/lib`, and instead include their locations in your `CLASSPATH` environment variable.
3. Add both JARs to your classpath using `-lib`.
4. Specify the locations of both JARs using a `<classpath>` element in a `<taskdef>` in the build file.

  - Search for these,
    and instructions for adding them, on Google

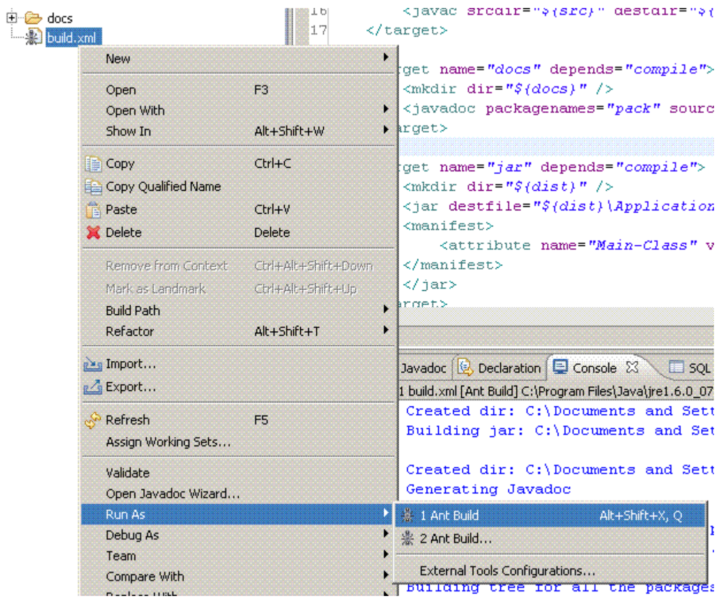
JUnit Ant Task Example

```
<project>
    <property name="src" value="./src" />
    <property name="lib" value="./lib" />
    <property name="classes" value="./classes" />
    <property name="test.class.name" value="com.xyz.MyTestSuite" />
    <path id="test.classpath">
        <pathelement location="${classes}" />
        <pathelement location="/path/to/junit.jar" />
        <fileset dir="${lib}">
            <include name="**/*.jar"/>
        </fileset>
    </path>

    <!-- Define the Ant task for running JUnit: -->
    <target name="test">
        <junit fork="yes" haltonfailure="yes">
            <test name="${test.class.name}" />
            <formatter type="plain" usefile="false" />
            <classpath refid="test.classpath" />
        </junit>
    </target>
```

– on command line: `ant test`

Ant and Eclipse

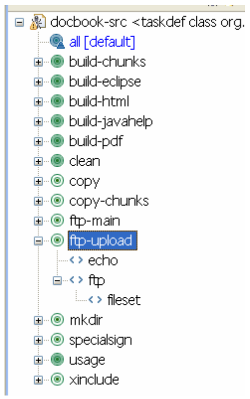
- Ant integrates nicely with Eclipse.
  - You can set up a "Build", "Run", or "Debug" task that uses Ant.
  - Eclipse can create an Ant build file for you from an existing project that builds its code.



```
  <javac srcdir="${src}" destdir="${
</target>

get name="docs" depends="compile">
  <mkdir dir="${docs}" />
  <javadoc packagenames="pack" sourc
rget>

get name="jar" depends="compile">
  <mkdir dir="${dist}" />
  <jar destfile="${dist}\Application
  <manifest>
      <attribute name="Main-Class" v
  </manifest>
  </jar>
rget>
```

```
1 build.xml [Ant Build] C:\Program Files\Java\jre1.6.0_07
 Created dir: C:\Documents and Sett
 Building jar: C:\Documents and Set

 Created dir: C:\Documents and Sett
 Generating Javadoc

 Building tree for all the package
```

  - Eclipse also has an Ant build file editor:



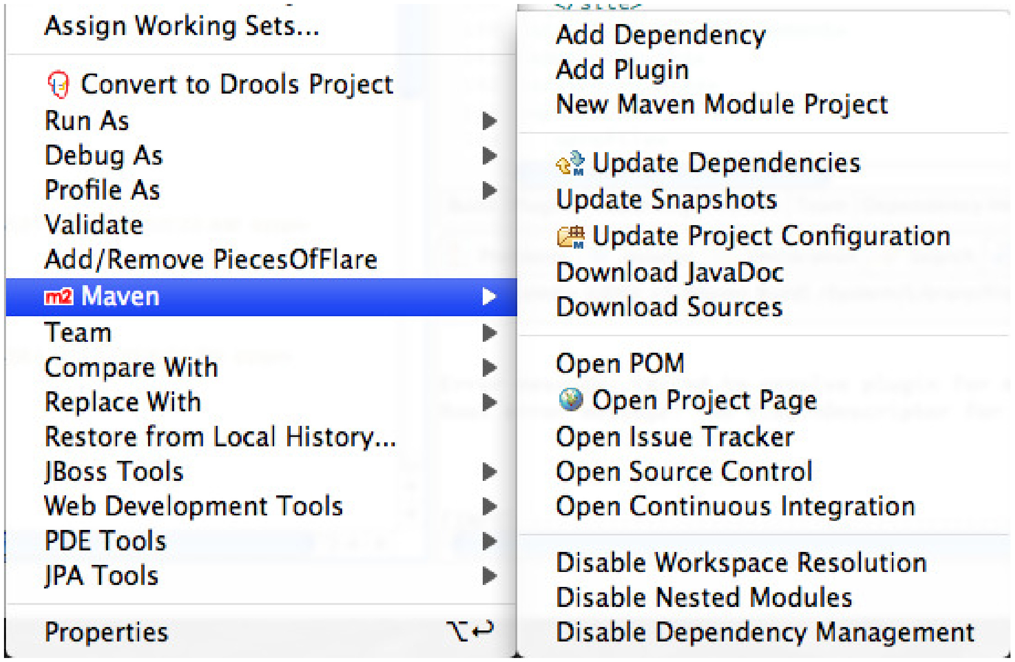
Maven

- **Maven**: A project management, comprehension, and build tool.
  - A successor / replacement for Ant.
  - Made by Apache, makers of Ant.

- Differences from Ant:
  - more powerful; higher level of abstraction
  - great for generating reports and visualizations
  - can run integration tests and continuous integration *(seen later)*
  - can handle deployment of an app or site

Maven and Eclipse

- Since Maven is newer, tool support (e.g. Eclipse integration) was slower to arrive, but it is generally mature now


Assign Working Sets...
Convert to Drools Project
Run As
Debug As
Profile As
Validate
Add/Remove PiecesOfFlare
Maven
Team
Compare With
Replace With
Restore from Local History...
JBoss Tools
Web Development Tools
PDE Tools
JPA Tools
Properties
Add Dependency
Add Plugin
New Maven Module Project
Update Dependencies
Update Snapshots
Update Project Configuration
Download JavaDoc
Download Sources
Open POM
Open Project Page
Open Issue Tracker
Open Source Control
Open Continuous Integration
Disable Workspace Resolution
Disable Nested Modules
Disable Dependency Management