Java Power Tools
Getting it all together

John Ferguson Smart
Principle Consultant
Wakaleo Consulting
Email: john.smart@wakaleo.com
Web: http://www.wakaleo.com
Twitter: wakaleo
Presentation Goals

Learn about the latest evolutions in Java development tools. In particular, learn how to improve and automate your development process using tools like Maven, Hudson, and many others.
Speaker’s qualifications

- John Ferguson Smart
- Consultant, Trainer, Mentor, Author,...
- Works with Enterprise Java, Web Development, and Open Source technologies
- Author of ‘Java Power Tools’ (O’Reilly)
- Writes articles for sites like JavaWorld, DevX and Java.net, and blogs on Java.net
- Frequent speaker at conferences and Java User Groups
- Likes to write about himself in the third person
Agenda

What we will cover today:

- Industrializing your build process
- Organizing your internal artifacts
- Improving your release management strategy
- Automate the build process
- Better testing practices
- Monitoring code coverage and code quality metrics
HERE is Edward Bear, coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it.

-- A. A. Milne
Towards a better build process

Why bother?
- Each team and each project using different conventions?
- Build scripts ad-hoc and difficult to understand and maintain?
- Little reuse of components between teams and projects?
- Code quality metrics and reporting are not done systematically, or not at all?
- High learning curve and maintenance costs?

What can you do?
- Set up a development infrastructure: e.g. Maven 2, Nexus, CI..
Towards a better build process

- Maven as a standardization tool
- More than just your average build scripting tool!
  - A build tool
  - A dependency management tool
  - A documentation generation tool
  - A metrics/code quality reporting tool
  - A project management tool
  - And more…
Towards a better build process

- So what can Maven do for me?
  - Standardize your build and deployment process
  - A standard, but extensible, build lifecycle
  - A standard directory structure
  - Clear and clean dependency management
  - Good technical documentation and reporting
Towards a better build process

- **Maven 2 Highlights**
  - A standardized approach to building your software
  - Familiar commands across all projects
  - No re-inventing the wheel

```
$mvn compile
$mvn test
$mvn package
$mvn integration-test
$mvn install
$mvn deploy
```
Towards a better build process

- Maven 2 Highlights
  - A Standard Directory Structure
    - Familiar structure across all projects
    - Lower learning curve
    - Convention Over Configuration
    - Less low-level scripting
Towards a better build process

- Maven 2 Highlights
- A Standard Way of Identifying Artifacts

Each Maven artifact has a unique identifier, or “co-ordinates”

```xml
<project...>
  ...
  <groupId>com.mycompany.accounting</groupId>
  <artifactId>accounting-core</artifactId>
  <packaging>jar</packaging>
  <version>1.1</version>
  <name>Accounting Core package</name>
  ...
</project...>
```

`com/mycompany/accounting/accounting-core/1.1/accounting-core-1.1.jar`
Towards a better build process

- Maven 2 Highlights
- Dependency Management
Towards a better build process

Maven 2 Highlights

- Traditional Dependency Management
  - Each project has its own set of JAR files
  - Unnecessary duplication
  - Hard to keep track of versions
  - Errors due to incompatible JAR files
  - Overloads the source code repository
Towards a better build process

- Maven 2 Highlights
  - Declarative Dependency Management
    - Artifact versions are stored on a central server
    - Each project “declares” what libraries and versions it needs
    - All the required dependencies are automatically downloaded

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

Artifacts are downloaded as required
Maven 2 Highlights

- Declaring Dependencies in Maven
  - Declared in the build script itself
  - Dependencies with version numbers
  - Different types of dependencies
  - Compile, test, provided,

```xml
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.4</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.4</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>2.5.6</version>
  </dependency>
  ...
</dependencies>
```

Version numbers

For tests only

Provided by the application server
Towards a better build process

- Managing Maven Dependencies
- It’s easier with Eclipse...m2eclipse
  - Adding new dependencies
  - Visualizing dependencies
  - Handling dependency conflict
Towards a better build process

Managing Maven Dependencies
Viewing Maven project dependencies
Towards a better build process

- Managing Maven Dependencies
- Adding new dependencies...

...and selecting the correct scopes and versions of dependencies.
Towards a better build process

- Managing Maven Dependencies
- Visualizing the Dependency Hierarchy

Conflicting dependencies

Resolved dependencies
Towards a better build process

- Managing Maven Dependencies
- Visualizing the Dependency Graph
Towards a better build process

- Managing Maven Dependencies
- Visualizing conflicts in the Dependency Graph
Towards a better build process

- Managing Maven Dependencies
- Excluding Dependencies
A walk through a simple Maven 2 project
Organizing your artifacts

- Internal releases are hard to co-ordinate
  - How do I share my API with other teams?
  - Where is the latest version of that API?
  - What version am I using, anyway?

What can you do?

- Use an Enterprise Maven Repository
Organizing your artifacts

- Maven Repositories
  - JAR files are downloaded to a local cache
Organizing your artifacts

- Maven Repositories
  - Public web sites containing JARs for many open source projects
  - Maven automatically downloads the JARs you need onto your local machine
  - You can publish internal APIs on your own repository
Organizing your artifacts

- Maven Repositories
  - JAR files are downloaded as required into a local cache
Organizing your artifacts

Maven Repositories

- There are several public Maven repositories
- You can also share internal JARs in an Enterprise Maven Repository
Organizing your artifacts

Maven Repositories

- The Enterprise Maven Repository can also act as a proxy/cache to the public repositories
Organizing your artifacts

Key Best Practices

- Deploy your internal releases to a local Enterprise repository
- Store proprietary JAR files you need here as well
- Distinguish between snapshot and release versions
Organizing your artifacts

- Using the Nexus repository manager
  - Easy to set up
  - Easy to administer
  - Proxy/cache
An Enterprise Repository
Better Release Management

- Working with snapshots and releases
  - Snapshots
    - Work in progress
    - A new time-stamped version deployed with each deployment

```xml
<artifactId>tax-calculator-core</artifactId>
<packaging>jar</packaging>
<version>1.0.0-SNAPSHOT</version>
```

$mvn deploy
Better Release Management

- Working with snapshots and releases
- Releases
  - Stable, tested release
  - The deployed artifact is unique

```xml
<artifactId>tax-calculator-core</artifactId>
<packaging>jar</packaging>
<version>1.0.0</version>

$mvn deploy

tax-calculator-core-1.0.0-20081123.093826-1.jar
Working with snapshots and releases

- Setting it all up in Maven

- To deploy a snapshot version, you need:
  - A Snapshot repository
  - A SNAPSHOT version number
  - User authentication for the repository
  - A Snapshot configuration for the repository

```xml
<settings>
  <servers>
    <server>
      <id>wakaleo-snapshots</id>
      <username>johns</username>
      <password>secret</password>
    </server>
  </servers>
</settings>
```

```xml
<distributionManagement>
  <snapshotRepository>
    <id>wakaleo-snapshots</id>
    <name>Internal Snapshots</name>
    <url>http://wanaka:8081/nexus/content/repositories/snapshots</url>
  </snapshotRepository>
</distributionManagement>
```
Better Release Management

- Working with snapshots and releases
  - Automating snapshot deployments
    - Use the maven-release-plugin to automate SCM book-keeping
      - `mvn:prepare`
      - `mvn:perform`
      - `mvn:rollback`

```
<version>1.0.1-SNAPSHOT</version>
<version>1.0.1</version>
<version>1.0.2-SNAPSHOT</version>
```

Nexus

```
$mvn deploy
tax-calculator-core-1.0.1
```
Releases, Snapshots and Repositories
Automating the build process

Continuous Integration - what’s the issue?

- Integration is long and difficult
- Poor visibility on development progress
- Functional tests are done too late
- Raised issues are harder to fix
- The client gets a sub-optimal product
Automating the build process

Continuous Integration - what’s involved?
Automating the build process

- Continuous Integration - why bother?
- Smoother integration process
- Automatic regression testing
- Regular working releases
- Earlier functional testing
- Faster and easier bug fixes
- Better visibility
Automating the build process

Continuous Integration - what you need

- Automated build process (e.g. Maven)
- Automated tests (JUnit, Selenium, easyb...)
- Source code repository
- Continuous Build Server
Automating the build process

- Continuous Integration - what can it do?
  - More than just your average build scheduler!
  - Raise (and monitor) integration issues - fast!
  - Automatically publish Maven artifacts
  - Monitor your build process
  - Monitor and report on code quality and code coverage
Automating the build process

- Continuous Integration - raising issues fast!
  - Use an appropriate notification strategy, e.g.
    - The committer is notified for all build results
    - Team members are notified for any failed builds
    - Team leader gets special notification after 5 successive build failures
Automating the build process

- Continuous Integration - raising issues fast!
  - Use an appropriate (fast) notification mechanism
    - Instant Messaging, SMS etc. for fast notification
    - Email as a secondary channel
    - RSS for consulting build history
Automating the build process

- Continuous Integration - raising issues fast!
  - Use a good build plan strategy
    - Fast builds first, e.g.
    - Unit tests before integration tests
    - Integration tests before metrics
  - Use manual builds where appropriate
    - Release builds
    - Deployments
    - ...
Automating the build process

- Looking for a good O/S Continuous Integration tool?
  - Try Hudson!
  - Easy to set up and configure
  - Good build and code quality metrics
  - Lots of plugins
Automating the builds
Better Testing

- Improving your testing game
- Innovative testing techniques
- Automating unit tests
- Separate unit tests and integration tests
- Monitor test duration
- Keep tabs on test coverage
Better Testing

- Use innovative testing techniques
- Unit testing
  - A cornerstone of modern software development
  - Unit tests can help you:
    - Ensure that code behaves as expected
    - Make your code more flexible and easier to maintain
    - Detect regressions
    - Document your code
Better Testing

- Use innovative testing techniques
  - Innovative unit testing
    - More readable tests - Hamcrest asserts
    - More efficient tests - Parameterized tests
    - Cleaner tests - using Groovy
    - More accurate tests - Behavior Driven Development (BDD)
    - Functional and web tests
  - Use your imagination!
Use innovative testing techniques

- Hamcrest asserts
  - More expressive and readable test assertions
  - Easier to understand
  - Less chance of test errors
Better Testing

Use innovative testing techniques

- Hamcrest asserts
  - Traditional JUnit 3.x asserts are hard to read:
  - Parameter order is counter-intuitive for English-speakers
  - x=10 is written

```java
assertEquals(10, x);
```

- The statements don't read well for English-speakers

```java
assertTrue(color.equals("red") || color.equals("blue"));
```

- "Assert that are equal 10 and x"

- Default error messages are sometimes limited:
Better Testing

- Use innovative testing techniques
- Hamcrest asserts
  - JUnit 4.4 introduces the `assertThat` statement
  - Rather than writing:
    ```java
    import static org.junit.Assert.*;
    ...
    assertEquals(expectedTax, calculatedTax, 0);
    ```
  - You can write:
    ```java
    import static org.hamcrest.Matchers.*;
    ...
    assertThat(calculatedTax, is(expectedTax));
    ```
Better Testing

- Use innovative testing techniques
- Parameterized tests
  - Run several sets of test data against the same test case
  - Help reduce the number of unit tests to write
  - Encourage developers to test more thoroughly
Better Testing

- Use innovative testing techniques
- Parameterized tests

Example: Calculating income tax

<table>
<thead>
<tr>
<th>Taxable income</th>
<th>PAYE rate for every $1 of taxable income (excluding ACC earners' levy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>up to $14,000</td>
<td>12.5 cents</td>
</tr>
<tr>
<td>$14,001 to $40,000 inclusive</td>
<td>21 cents</td>
</tr>
<tr>
<td>$40,001 to $70,000</td>
<td>33 cents</td>
</tr>
<tr>
<td>$70,000 and over</td>
<td>39 cents</td>
</tr>
</tbody>
</table>
Better Testing

- Use innovative testing techniques
- Parameterized tests
  - Some test data
  - A test class with matching fields
  - And some tests
  - And an annotation

<table>
<thead>
<tr>
<th>Income</th>
<th>Year</th>
<th>Expected Tax</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0.00</td>
<td>2007</td>
<td>$0.00</td>
</tr>
<tr>
<td>$10,000.00</td>
<td>2007</td>
<td>$1,950.00</td>
</tr>
<tr>
<td>$20,000.00</td>
<td>2007</td>
<td>$3,900.00</td>
</tr>
<tr>
<td>$38,000.00</td>
<td>2007</td>
<td>$7,410.00</td>
</tr>
<tr>
<td>$38,001.00</td>
<td>2007</td>
<td>$7,410.33</td>
</tr>
<tr>
<td>$40,000.00</td>
<td>2007</td>
<td>$8,070.00</td>
</tr>
<tr>
<td>$60,000.00</td>
<td>2007</td>
<td>$14,670.00</td>
</tr>
<tr>
<td>$100,000.00</td>
<td>2007</td>
<td>$30,270.00</td>
</tr>
</tbody>
</table>

```java
@RunWith(Parameterized.class)
public class TaxCalculationTest {
    private int income;
    private int year;
    private double expectedTax;

    public TaxCalculationTest(int income, int year, double expectedTax) {
        this.income = income;
        this.year = year;
        this.expectedTax = expectedTax;
    }

    @Test
    public void shouldCalculateCorrectTax() throws InvalidYearException {
        TaxCalculator calculator = new TaxCalculatorImpl();
        BigDecimal calculatedTax = calculator.calculateIncomeTax(income, year);
        assertEquals(expectedTax, calculatedTax);
    }
}
```
Better Testing - Parameterized Tests and Hamcrest Asserts
Better Testing

- Use innovative testing techniques
- Groovy Testing
  - Write more expressive unit tests in Groovy
  - Expressive test code
  - Quicker to write
  - More incentive to write in-depth test cases
  - But
    - Relatively limited IDE support...
Better Testing

- Use innovative testing techniques
- Some examples...

```java
public class TaxCalculatorBusinessTest {

    private TaxCalculator calculator;

    @Before
    public void setup() {
        calculator = new TaxCalculatorImpl();
    }

    @Test
    public void shouldNotTaxLosses() throws InvalidYearException {
        BigDecimal tax = calculator.calculateIncomeTax(new BigDecimal("-100000"), 2009);
        assertThat(tax, is(new BigDecimal("0.00")));
    }
}
```

```groovy
class GroovyTaxCalculatorBusinessTest {
    def calculator = new TaxCalculatorImpl()

    @Test
    void shouldNotTaxLosses() {
        def tax = calculator.calculateIncomeTax(-100000, 2009)
        assertThat tax, is(0.00)
    }
}
```
Better Testing

Use innovative testing techniques
Some examples...

```java
public class TaxCalculatorBusinessTest {
    private TaxCalculator calculator;

    @Before
    public void setup() {
        calculator = new TaxCalculatorImpl();
    }

    @Test(expected=InvalidYearException.class)
    public void shouldNotAcceptUnknownYears() throws InvalidYearException {
        calculator.calculateIncomeTax(new BigDecimal("100000"), 1901);
    }
}
```

```groovy
class GroovyTaxCalculatorBusinessTest {
    final shouldFail = new GroovyTestCase().&shouldFail

    @Test
    void shouldNotAcceptUnknownYears() {
        shouldFail(InvalidYearException) {
            calculator.calculateIncomeTax(100000, 1901)
        }
    }
}
```
Better Testing

- Use innovative testing techniques
- Behaviour-Driven Development
  - It’s not about writing tests
  - TDD and BDD is about writing better code:
    - Maintainable
    - Flexible
    - Reliable
    - Simple
Better Testing

- Use innovative testing techniques
- Behavior-Driven Development
  - BDD uses words like “should” to describe the desired behavior
  - “Should transfer money from account A to account B”
  - Should deploy landing gear before touching ground”
  - ...
Better Testing

➢ Use innovative testing techniques

➢ Enter Easyb

☒ A BDD testing framework for Java
☒ Make testing clearer and easier to write
☒ Make tests self-documenting
☒ Help developers focus on the requirements
Use innovative testing techniques

Easyb stories:

- Use a narrative approach
- Describe a precise requirement
- Can be understood by a stakeholder
- Usually made up of a set of scenarios
- Use an easy-to-understand structure:
  - Given [a context]...
  - When [something happens]...
  - Then [something else happens]...
Use innovative testing techniques
Enter Easyb
A Groovy-based DSL for Behaviour-Driven Development

```java
import com.wakaleo.jpt.taxcalculator.InvalidYearException

scenario "Should not tax losses", {
    given "a correctly configured tax calculator",
    when "you calculate annual income tax for a negative income",
    then "the calculated tax should be zero",
}
```
Better Testing

- Use innovative testing techniques
- Easyb reporting
- Simple but readable...

![Screen shot of Easyb reporting dashboard with test results and stories list.](image-url)
Better Testing - Testing with Groovy and EasyB
Better Testing

- Use innovative testing techniques
- Integration Test strategies
  - Deploy to an embedded Jetty server for local integration tests
  - Run local integration tests with a web testing tool
    - Selenium
    - HtmlUnit
    - JWebUnit
    - Canoo WebTest
  - Deploy automatically to a publicly-visible integration server
Better Testing

Web Testing Tools

- Test before deployment
  - Use Jetty
  - Run web tests locally from within the normal build

- Test after deployment
  - Deploy to a test server
  - Run web tests against a remote server in a dedicated build job

You probably need both...
Examples of Web Testing Tools

- Selenium
  - Runs tests through a real browser
  - Accurate rendition of Javascript and AJAX behavior
  - High-level readable API
  - More complicated to set up
  - Slow
@Test
public void testDepositAmount() throws Exception {
    Selenium selenium = new DefaultSelenium("localhost", 4444,
                                           "*firefox", "http://localhost:8080");
    selenium.start();
    selenium.open("/ebank-web");
    selenium.waitForPageToLoad("10000");
    selenium.type("depositAmount", "100");
    selenium.click("deposit");
    selenium.waitForPageToLoad("10000");
    selenium.isTextPresent("Current balance: $100");
}
Better Testing

Examples of Web Testing Tools

- **HTMLUnit**
  - Simulates a browser
  - Lower level API
  - Easier to set up
  - Fast
Examples of Web Testing Tools

**JUnit**

```java
@Test
public void depositCash() throws Exception {
    WebClient webClient = new WebClient();
    HtmlPage page = webClient.getPage("http://localhost:8080/ebank-web");
    assert page.asText().contains("Current Balance: $0");
    HtmlForm form = page.getForms().get(0);
    HtmlSubmitInput depositButton = form.getInputByName("deposit");
    HtmlTextInput textField = form.getInputByName("depositAmount");

    textField.setValueAttribute("100");

    HtmlPage result = depositButton.click();
    assert result.asText().contains("Current Balance: $100");
}
```
Better Testing

- Examples of Web Testing Tools
  - JWebUnit
Better Testing - Web Testing
Why use code quality metrics

- Better quality code
- Code is easier to maintain
- Detect potential bugs
- Train new staff
Automated Code Quality

- Why automate code quality metrics
- Global picture of code quality
- Statistical code quality trends
- More value-added manual reviews
How automate code quality metrics

Integrate code quality metrics into your build scripts

Run a special build reserved for code metrics

TIP: Code quality metrics are easier with Maven!
Automated Code Quality

- Types of code quality metrics
  - Coding Standards
    - Naming conventions, Javadoc comments, layout,...
    - Tool: **Checkstyle**
  - Best Practices
    - Good programming habits, potential bugs,...
    - Tools: **Checkstyle, PMD, Findbugs**
  - Bug Detection
    - Broken code, dangerous code, bugs,...
    - Tool: **FindBugs**
  - Code Coverage
    - How much code is executed by your tests
    - Tool: **Cobertura, Emma, Clover,...**
Automated Code Quality

- Team code reviews
- Review code as a group
- Long and slow if done manually
- Benefits greatly from the use of tools
Automated Code Quality

- Automating code quality metrics with Hudson
- Using the Violations plugin
  - Reports on Checkstyle, PMD, Findbugs, and others
  - Uses data generated with Maven or Ant
Automated Code Quality

- Automating code quality metrics with Hudson
- Configuring the Violations plugin

Define thresholds appropriate for your project

<table>
<thead>
<tr>
<th>Type</th>
<th>Per file display limit</th>
<th>XML filename pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkstyle</td>
<td>10</td>
<td>planeshowcase/target/checkstyle-result.xml</td>
</tr>
<tr>
<td>cpd</td>
<td>10</td>
<td>planeshowcase/target/cpd.xml</td>
</tr>
<tr>
<td>findbugs</td>
<td>10</td>
<td>planeshowcase/target/findbugs.xml</td>
</tr>
<tr>
<td>fxcop</td>
<td>10</td>
<td>planeshowcase/target/pmd.xml</td>
</tr>
<tr>
<td>pmd</td>
<td>10</td>
<td>planeshowcase/target/pmd.xml</td>
</tr>
<tr>
<td>pylint</td>
<td>10</td>
<td>planeshowcase/target/pmd.xml</td>
</tr>
</tbody>
</table>

**/src/main/java

This is a file name pattern that can be used to resolve classes to sourcefiles (for example **/src/main/java).
Automated Code Quality

- Automating code quality metrics with Hudson
- Displaying the Violations reports

Number of violations over time
Automated Code Quality

- Automating code quality metrics with Hudson
- Displaying the Violations reports

Violations Report for build 105

<table>
<thead>
<tr>
<th>Type</th>
<th>Violations</th>
<th>Files in violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkstyle</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>mnd</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>findbugs</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>mrd</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Drilling down
Automated Code Quality

Automating code quality metrics with Hudson
Displaying the Violations details

Details for a particular issue
Automated Code Quality

Code Coverage

- Indicates what code is being executed by your unit tests.
- Can help isolate untested code
- Does not guarantee that the tests that are run are of high quality
- Can be an indicator of whether tests are being written at all
Automated Code Quality

- Code Coverage
  - Using the Hudson code coverage plugins
  - Generate data using Ant or Maven
  - Report coverage metrics in Hudson

Configure health reporting thresholds.
For the ☀️ row, leave blank to use the default value (i.e. 80).
For the ☁️ and ⛅️ rows, leave blank to use the default values (i.e. 0).
Automated Code Quality

- Code Coverage
- Using the Hudson code coverage plugins
- Cobertura coverage reports

```java
43 */
44 public Manifest getManifest() {
45  if (manifest == null) {
46   loadMetaInf();
47  }
48  return manifest;
49 }
50 
51 public void setManifest(Manifest manifest) {
52  this.manifest = manifest;
53 }
54 
55 public ServletContext getContex() {
56  return context;
57 }
58 
59 public void setContex(ServletContext context) {
60  this.context = context;
61 }
62 
63 /**
64  * Load the META-INF file.
65  */
66 public void loadMetaInf() {
67  String appServerHome = context.getRealPath("/!
68  File manifestFile = new File(appServerHome, "META-INF/MANIFEST.MF");
```
Automated Code Quality

- Code Quality Management
  - Check out Sonar!
    - Centralized code quality management
    - Generate code quality metrics in an automated build using Maven
    - Store code quality metrics in a database
    - Code quality metrics can be consulted on a web site
Automated Code Quality

- Code Quality Management
- Sonar architecture

- Run Sonar on a CI server
- The Hudson Sonar Plugin injects data into the database
- mvn sonar:sonar
- Sonar runs on a Java web server
- Sonar quality metrics data is stored in a database

The Maven Sonar Plugin injects data into the database
Automated Code Quality

- Code Quality Management
- Sonar centralizes many code quality metrics.

Source code metrics

- Code complexity metrics
- Test results and code coverage
- Build history
- Click anywhere to drill down
Automated Code Quality

- Code Quality Management
- You can drill down to view the details for each type of issue

Overview

Different types of violations

Violations in this class

Violation details
Automated Code Quality

- Code Quality Management
- You can also view historical data
Automated Code Quality

- Code Quality Management
- You can also view historical data

Which violations occur the most?
More advanced code quality metrics

- How long do your tests run
  - Overly-long tests slow down your build process
  - They may also indicate a performance issue

![Automated Code Quality](image)
Automated Code Quality

- More advanced code quality metrics
- Are your tests failing repeatedly
  - May indicate a difficult technical issue
  - Could result in unreliable “quick-fix” solutions

Issues that took a long time to fix
Automated Code Quality
Summary

- There are many ways to improve a development process
- Improve your build process and dependency management
- Install a CI server and publish your APIs automatically
- Automate as much as possible!
- Use code quality metrics to isolate bottlenecks and fix problems
Thanks for your attention!

John Ferguson Smart
Email: john.smart@wakaleo.com
Web: http://www.wakaleo.com
Twitter: wakaleo

http://www.wakaleo.com
http://www.wakaleo.com

Optimize Your SOFTWARE DEVELOPMENT

Wakaleo Consulting

Managing automated build dependencies with Maven and Hudson

http://weblogs.java.net/blog/john-smart/archive/2008/11/managing_automa.html