Static Analysis Plugins
White Mountain Release

Ullrich Hafner
About me

Prof. Dr. Ullrich Hafner
ullrich.hafner@hm.edu
Professor for Software Engineering
University of Applied Sciences Munich

- Developer
- Architect
- Consultant
- > 20 years
- Quality
- Testing
- Usability
- Jenkins Committer since 2008
- Professor Software Engineering since 2012
Static analysis plugins – typical use case

Checkout → Compile → Test → Deploy

Test Result
0 failures (0), 3 skipped (0)

All Tests

<table>
<thead>
<tr>
<th>Package</th>
<th>Duration</th>
<th>Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>edu.hm.hafner</td>
<td>3 sec</td>
<td>0</td>
</tr>
<tr>
<td>edu.hm.hafner.analysis</td>
<td>2.3 sec</td>
<td>0</td>
</tr>
<tr>
<td>edu.hm.hafner.analysis.logic</td>
<td>7.5 sec</td>
<td>0</td>
</tr>
</tbody>
</table>
Static analysis plugins visualize compile step results
Support for almost 100 static analysis reports

- Checkout
- Compile
- CheckStyle
- SpotBugs
- JSLint

Details | File | Package
---|---|---
AreaStyle.java:9 | io.jenkins.plugins.analysis.core.charts
LineSeries.java:13 | io.jenkins.plugins.analysis.core.charts
LineSeries.java:14 | io.jenkins.plugins.analysis.core.charts
Jenkins' static analysis suite

- Android Lint
- SpotBugs
- CheckStyle
- Warnings
- PMD
- DRY
Jenkins' static analysis suite Warnings Next Generation
Warnings Next Generation plugin content

almost 100 tools
Static Analysis Plugins
White Mountain Release

Features (Demo)
Individual tool reports or aggregated reports

Report for each tool

Aggregated report

- Static analysis results from: FindBugs, CPD, CheckStyle, PMD, JavaDoc, Java, SpotBugs
Severity, priority, warnings, errors, issues, etc.

- Same classification used for all issue types of all tools
- Errors indicate compile errors
- Warnings have a priority
  - High
  - Normal
  - Low
- Tool report parsers use hard-coded mapping (make it configurable?)
Detection of new, fixed, and outstanding issues

- Issues delta for each build
  - New issues since build #n
  - Fixed issues since build #n
- Build ‘#n’: Reference (or baseline)
- Reference is configurable
  - Only stable builds
  - Passed Quality Gate
  - Different Job (e.g., master Branch)
- Optimum in theory
  - SCM branch point
  - Not implemented yet
Details view with paging, filter and sorting
Details with embedded detail description

<table>
<thead>
<tr>
<th>Details</th>
<th>File</th>
<th>Package</th>
<th>Category</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HealthReportBuilderTest.java:186</td>
<td>io.jenkins.plugins.analysis.core.quality</td>
<td>CORRECTNESS</td>
<td>RV_RETURN_VALUE_IGNORED</td>
<td></td>
</tr>
</tbody>
</table>


The return value of this method should be checked. One common cause of this warning is to invoke a method on an immutable object, thinking that it updates it, for example, in the following code fragment:

```java
String dateString = getHeaderField(name);
dateString.trim();
```

The programmer seems to be thinking that the trim() method will update the String referenced by dateString. But since Strings are immutable, the trim() method returns a new String value, which is being ignored here. The code should be corrected to:

```java
String dateString = getHeaderField(name);
dateString = dateString.trim();
```
Source Code View

```java
/**
 * A blamer that does nothing.
 * @author Ullrich Hafner
 */

public class NullBlamer implements Blamer {
    @Override
    public Blames blame(final Report report) {
        report.logInfo("Skipping blaming as requested in the job configuration");
        return new Blames();
    }
}
```

This class implements the Serializable interface, but does not define a serialVersionUID field. A change as simple as adding a reference to a .class object will add synthetic fields to the class, which will unfortunately change the implicit serialVersionUID (e.g., adding a reference to String.class will generate a static field class$java$lang$String). Also, different source code to bytecode compilers may use different naming conventions for synthetic variables generated for references to class objects or inner classes. To ensure interoperability of Serializable across versions, consider adding an explicit serialVersionUID.
Baseline examples – simple case

- **#1**
  - Outstanding: 2
  - New: 0
  - Fixed: 0

- **#2**
  - Outstanding: 2
  - New: 1
  - Fixed: 0

- **#3**
  - Outstanding: 2
  - New: 0
  - Fixed: 1
Baseline examples – tracking of new warnings

#1
Outstanding: 2
New: 0
Fixed: 0

#2
Outstanding: 2
New: 1
Fixed: 0

#3
Outstanding: ?
New: ?
Fixed: 0
Warnings Plugin – Reference Build

• Warnings supports both ways
  • Delta computation: current with previous
  • Delta computation: current with reference

• Reference
  • Baseline where to start
  • Automatically computed
Reference build – previous build

- Delta computation compares current with previous

- Builds will be skipped if job fails (optional)
Reference build – baseline build

• Delta computation compares current with baseline

• Baseline will automatically adjust if no new warnings
Reference build – baseline from other job

- Delta computation compares current with latest from another job

```
master   ...   #20   #21   #22
     Δ   Δ   Δ
#1   #2   #3
```

- Ideal solution: baseline will use branch point from SCM
Quality gates

- Provide a quality gate for each tool
  - Count number of warnings (by severity)
    - New warnings
    - Total warnings
- Set build result
  - Quality gate passed: Success
  - Quality gate failed: Unstable or Failed
- Quality gate operates on delta to baseline
- Baseline will be adjusted if quality gate passed
- Jenkins health report is supported as well
Pipeline support

**Simple Mode**
- Single Step scans for warnings and publishes report

**Advanced Mode**
- Step to scan for warnings
- Step to publish report
Declarative Pipeline example

```java
post {
  always {
    junit testResults: '*/target/surefire-reports/TEST-*.xml'
    recordIssues enabledForFailure: true, tool: mavenConsole()
    recordIssues enabledForFailure: true, tool: errorProne()
    recordIssues enabledForFailure: true, tool: java()
    recordIssues enabledForFailure: true, tool: checkStyle()
    recordIssues enabledForFailure: true, tool: spotBugs()
    recordIssues enabledForFailure: true, unstableNewAll: 1,
      tool: cpd(pattern: '*/target/cpd.xml')
    recordIssues enabledForFailure: true,
      tool: pmdParser(pattern: '*/target/pmd.xml')
    recordIssues enabledForFailure: true,
      tool: taskScanner(includePattern:'**/*.java', highTags:'FIXME',
                         normalTags:'TODO')
  }
}
```
Scripted Pipeline example

```java
stage ('Build') {
    def mvnHome = tool 'mvn-default'
    sh "${mvnHome}/bin/mvn --batch-mode -V -U -e clean verify
    junit testResults: '*/target/surefire-reports/TEST-*.xml'
    def java = scanForIssues tool: [$class: 'Java']
    def javadoc = scanForIssues tool: [$class: 'JavaDoc']
    publishIssues issues:[java,javadoc], unstableTotalAll:1
}
```
Static Analysis Plugins
White Mountain Release

Summary
Additional features

- Filtering of warnings
  - Type and category
  - Module, package or file name
  - Include or exclude filters

- View for internal messages
  - No need to open console log
  - Error and info messages

- Source code view
  - Highlighting of all languages
  - Navigation to affected line

- Trend charts
  - Uses modern charting library
  - Rendering is done on the client
  - More to come soon

- Improved overall user experience
- Parsers do not depend on Jenkins API
- Add Git author and commit blame
Next steps

Coding

• Responsive UI (screen size)
• Configurable trend charts
• Analysis dashboard
• More dashboard view portlets
• Server side paging
• Internationalization

Research

• New warnings computation
  • Based on Abstract Syntax Tree
  • Based on plagiarism algorithm
  • SCM delta analysis
• Technical debt for warnings
  • File change frequency
  • Number of authors
  • Size of file
Static Analysis Plugins
White Mountain Release