DZone Refcardz

Continuous Integration: Servers and Tools

By Paul M. Duvall

ABOUT CONTINUOUS INTEGRATION

When implementing automated Continuous Integration (CI), you have a number of CI servers and tools from which to choose. Here, one server (Hudson) and many tools/platforms (e.g., Linux, Java, Ant, Subversion, MySQL and Sonar) are covered and linked to CI Patterns and Antipatterns (see CI Patterns and Antipatterns Refcard #84). The goal is to demonstrate how you can use a CI server to create working software with every change. Furthermore, team members need to be notified when something goes wrong, a vital element to CI. It is important to note that while this Refcard uses the Java platform and Linux for example purposes, servers and tools do exist that support other development platforms (e.g., CruiseControl.NET, cruisecontrol.rb, MSBuild and so on).

Relevant Patterns

The patterns below are from the CI patterns and anti-patterns Refcard (#84) that are relevant to ci servers and tools and are covered in this Refcard.

Pattern	Description
Repository	Commit all files to a version-control repository
Automated Build	Automate all activities to build software from source without manual configuration
Minimal Dependencies	Reduce pre-installed tool dependencies to the bare minimum
Label Build	Label the build with unique name
Continuous Feedback	Send automated feedback from CI server to development team
Independent Build	Separate build scripts from the IDE
Dedicated Machine	Run builds on a separate dedicated machine
Continuous Inspection	Run automated code analysis to find common problems
Build Threshold	Use thresholds to notify team members of code aberrations
Headless Execution	Securely interface with multiple machines without typing a command
Protected Configuration	Files are shared by authorized team members only
Automated Tests	Write an automated test for each unique path

CONFIGURING THE MACHINE

Minimal Dependencies Pattern: Reduce pre-installed tool dependencies to the bare minimum. Reduce required environment variables from the Automated Build and Scripted Deployment to the bare minimum.

Antipatterns: Requiring developer to define and configure environment variables. Requiring developer to install numerous tools in order for the build/deployment to work.

Configure the Operating System and Servers

Configuring Environment Variables

Download the Java development kit zip distribution to a temporary directory on your workstation and extract the file to a directory such as `/usr`.

http://www.java.com/en/download/manual.jsp

Download Ant build tool zip distribution to a temporary directory on your machine and extract the file to a directory such as `usr/local`.

http://ant.apache.org/bindownload.cgi

Create or open your user profile by using the vi editor as shown below to open/create a `.bash_profile` file.

```
vi .bash_profile
```

Add environment information to the `.bash_profile` file. Examples are shown below.

```
ANT_HOME=/usr/local/ant-1.7.0
JAVA_HOME=/usr/jdk1.5.0_10/
HUDSON_HOME=$HOME/hudson_data
export ANT_HOME JAVA_HOME HUDSON_HOME
export PATH=$ANT_HOME/bin:$JAVA_HOME/bin:$PATH
```

Save the profile to the system by sourcing the profile as shown below.

```
. .bash_profile
```

Test Java and Ant were installed by typing:

```
java –version

ant -version
```

Install MySQL using yum as shown here. If you are using another flavor of Linux that doesn't support yum, you can use rpm. Or, if you're using Windows, you can download the MySQL installation from http://dev.mysql.com/downloads/ and ensure MySQL is available in your system path.

```
yum -y install mysql mysql-server
```

Start the MySQL server. An example is shown below.

```
/etc/init.d/mysqld start
```

Test MySQL was installed by typing:

```
mysql –help
```

Install Version-Control System Client

Install Subversion client software using yum.

```
yum -y install subversion
```

Test svn client installation by typing:

```
svn help
```

Install Tomcat Web Container

Download Tomcat by visiting Apache Tomcat's download site at:

http://tomcat.apache.org/download-60.cgi

Save the zip file to the machine where Tomcat will be hosted (e.g. apache-tomcat-6.0.20.zip). Go to the command prompt on the machine and type:

```
unzip apache-tomcat-6.0.20.zip
```

Start the Tomcat server by typing:

```
cd [tomcat-download-location]/apache-tomcat-6.0.20/bin

chmod ugo+rx *.sh

./startup.sh
```

Test Tomcat is running by launching your browser and typing:

```
http://localhost:8080/
```

INSTALLING THE CI SERVER

Dedicated Machine Pattern: Run builds on a separate dedicated machine.

Antipatterns: Existing environmental and configuration assumptions can lead to the "but it works on my machine" problem.

When considering which CI server to use, consider the following server evaluation features:

Feature	Explanation
Version-control system integration	If a tool doesn't support your particular version-control system, do you really want to write a custom integration for it?
Build-tool integration	In choosing a CI server, you need to consider which build tools you already use or will be using. For Java™ programming, there are a couple of clear favorites, Ant and Maven, and most all CI tools support them. If your build system isn't either Ant or Maven, does the CI tool support the ability to run a program from the command line?
Feedback and reporting	Consider the old adage, "If a tree falls in the forest, does anyone hear it?" If a build fails, does anyone hear about it? If no one does, what's the purpose of having a CI tool? All CI tools offer some notification mechanism, but which one will work best for you? E-mail? Instant messenger? RSS?
Labeling	Some development teams like to track builds by giving them unique labels so they can refer to a particular build instance at a later date. If this is important to you, be aware that only some CI servers provide this capability.
Project dependencies	In some cases, after you build one project, you may need to build another dependent project. Certain CI servers support this feature and some don't.
Ease of extensibility	How easy is it to extend the current functionality of the tool? Are there plug-ins that allow for simple extension or is it always a code change?

One CI server that satisfies these criteria well is Hudson, which will be the focus here. Below you will learn how to download, install, configure Hudson, and configure a Hudson job with CI patterns in mind.

Download Hudson

Hudson is free and openly available at hudson-ci.org. Hudson supports numerous version-control systems, including Subversion (see Repository Pattern in Refcard #84).

http://hudson-ci.org/

Download the latest from the website to your machine where Hudson will be hosted. An example is shown below:

http://hudson-ci.org/latest/hudson.war

Install Hudson

To install Hudson you will need Java version 1.5 or higher and the Hudson installation file, which is a Java EE Web archive (WAR) file. Typically, you can use a web container, such as Tomcat, and deploy the hudson.war file to the web container.

Copy the Hudson.war to Tomcat.

```
cd ~/hudson/application/webapps/apache-tomcat-6.0.20/webapps

cp [hudson-download-location]/hudson.war ~/hudson/
application/webapps/apache-tomcat-6.0.20/webapps
```

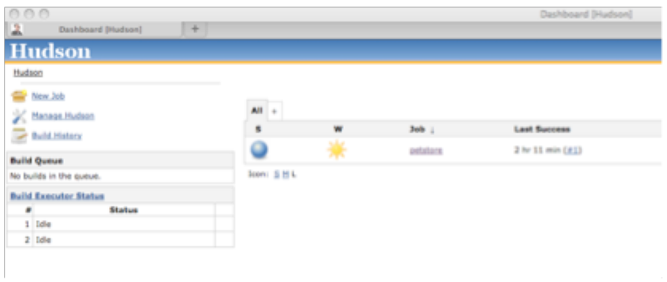
Restart the Tomcat container.

```
cd ~/hudson/application/webapps/apache-tomcat-6.0.20/bin

./shutdown.sh

./startup.sh
```

Verify the Hudson CI server is running by launching a web browser and typing:

http://localhost:8080/hudson

The Hudson server dashboard should be displayed and look similar to the following figure.



Dashboard [Hudson]

Hudson

Hudson

New Job
Manage Hudson
Build History

Build Queue
No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

All +

S	W	Job ↓	Last Success
●	☀	petstore	2 hr 11 min (#1)

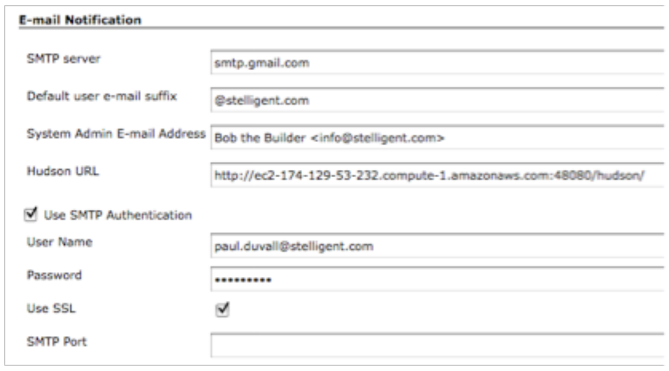
Icon: S M L

Configure Hudson CI server

You will need to configure Hudson to refer to the Java JDK and Ant installation on the machine where you have installed Hudson. Go to the main Hudson page and click on the *Manage Hudson* link. Click *Configure System*. To configure JDK and Ant instances click on the *Add* button under the relevant sections, which will display a set of fields for configuration.

Configuring the Email Server Information

An example of configuring how email is sent from your Hudson server for all jobs is shown below. After applying the changes, click the *Save* button.



E-mail Notification

SMTP server	smtp.gmail.com
Default user e-mail suffix	@stelligent.com
System Admin E-mail Address	Bob the Builder <info@stelligent.com>
Hudson URL	http://ec2-174-129-53-232.compute-1.amazonaws.com:48080/hudson/
✓ Use SMTP Authentication	
User Name	paul.duvall@stelligent.com
Password	*********
Use SSL	✓
SMTP Port	

CONFIGURING A HUDSON JOB

After installing and configuring a Hudson server, you can create one or more Hudson jobs. A job polls a version-control repository for changes and runs a build to create software. The corresponding CI patterns are described below.

Automated Build Pattern: Automate all activities to build software from source without manual configuration. Create build scripts that are decoupled from IDEs. Later, these build scripts will be executed by a CI system so that software is built at every repository change.

Antipatterns: Continually repeating the same processes with manual builds or partially automated builds requiring numerous manual configuration activities.

Headless Execution Pattern: Securely interface with multiple machines without typing a command.

Antipatterns: People manually access machines by logging into each of the machines as different users; then they copy files, configure values, and so on.

Independent Build Pattern: Separate build scripts from the IDE. Create build scripts that are decoupled from IDEs. Later, these build scripts will be executed by a CI system so that software is built at every repository change.

Antipatterns: Automated Build relies on IDE settings. Build cannot run from the command line.

Protected Configuration Pattern: Using the repository, files are shared by authorized team members only.

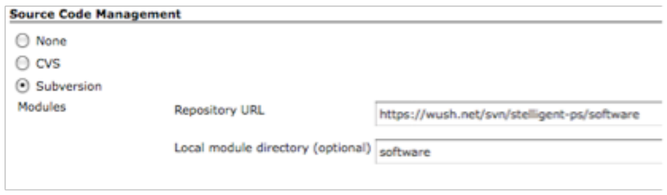
Antipatterns: Files are managed on team members' machines or stored on shared drives accessible by authorized team members.

Continuous Inspection Pattern: Run automated code analysis to find common problems. Have these tools run as part of continuous integration or periodic builds.

Antipatterns: Long, manual code reviews or no code reviews.

Configure Version-Control Repository

From the Hudson dashboard, select the job you are configuring, then select the *Configure* link. One of the configuration options is called *Source Code Management*. From this section, select the *Subversion* radio button. Then, you will enter the Subversion URL where that contains the build file you're using to run your build in the `Repository URL` field. To indicate the directory name where this repository will be represented locally on the Hudson server, enter a value in the `Local module directory` field. Enter this information and click the *Save* button. The figure below demonstrates these actions.



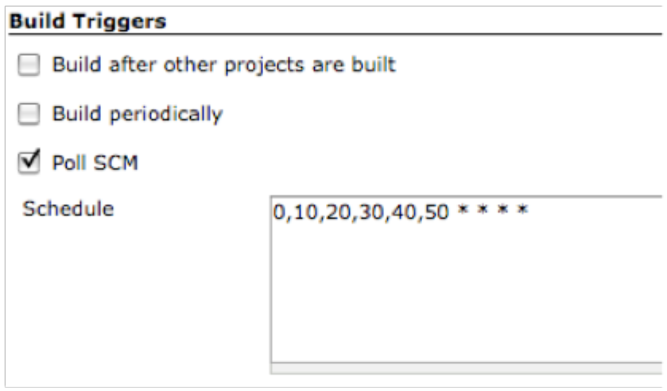
Source Code Management

○ None
○ CVS
◉ Subversion

Modules		
	Repository URL	https://wush.net/svn/stelligent-ps/software
	Local module directory (optional)	software

Set the Polling Frequency

From the Hudson dashboard, select the job you are configuring, and then select the *Configure* link. One of the configuration options is called *Build Triggers*. Select the *Poll SCM checkbox* and enter the `0,10,20,30,40,50 * * * *` in the *Schedule text area* and click the *Save* button. This means that Hudson will check for any changes to your Subversion repository every 10 minutes. If no changes are found, it won't run a build. If it discovers changes, it runs the build file and targets described in the next section.



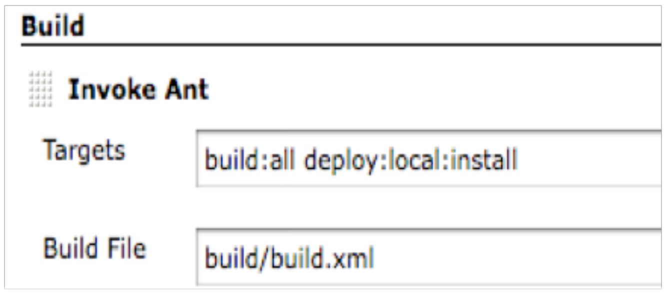
Build Triggers

☐ Build after other projects are built
☐ Build periodically
☑ Poll SCM

Schedule	0,10,20,30,40,50 * * * *

Configure the Build Target and Build File location

From the Hudson dashboard, select the job you are configuring, then select the ***Configure*** link. One of the configuration options is called ***Build***. Under ***Invoke*** Ant, enter values in the ***Targets*** and ***Build File*** fields. The targets are a space-delimited list of targets you wish to call for a particular build file the build file name is relative to the `Repository URL` you configured in the configure version-control repository section above. Enter this information and click the ***Save*** button.



Build

Invoke Ant

Targets	build:all deploy:local:install
Build File	build/build.xml

Continuous Inspection

The Continuous Inspection pattern is an approach to running automated code analysis as part of a build in order to find code quality problems. Continuous Inspection can help reduce the time spent in Long, manual code review sessions. While there are numerous tools you can use to implement this pattern, this example shows a tool called Sonar, which collects the information from several code quality analysis tools into comprehensive graphs and reports.

Using Sonar

Sonar provides code quality reports and graphs using several of the widely used open source static analysis tools on the market. The benefit is that Sonar aggregates the data and displays it as information in an easy-to-understand way. Using Sonar is quite simple in Hudson by downloading ***Hudson's Sonar Plugin***.

Add the Sonar plugin to Hudson

From the main Hudson dashboard, Select the ***Manage Hudson*** link, then ***Manage Plugins***. From ***Manage Plugins***, select the ***Available*** tab. From the numerous plugins, select the ***Hudson Sonar Plugin*** checkbox and select the ***Install*** button.

Restart the Tomcat container

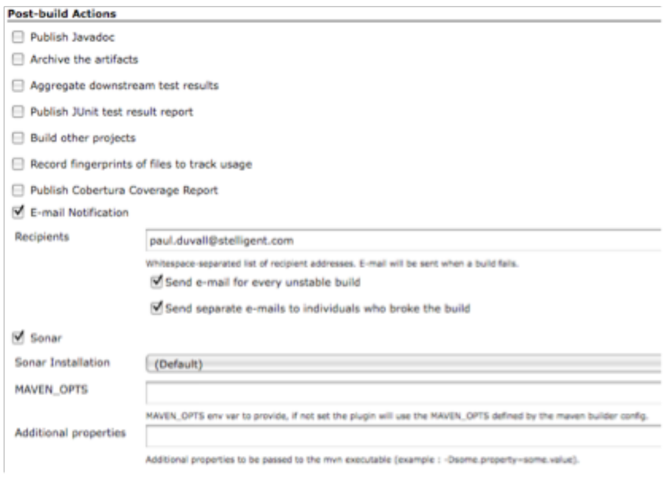
Go back to the ***Manage Hudson*** link and select the ***Prepare for Shutdown*** option. This prevents any other jobs from running while you restart the server. Because the Tomcat server is hosting the Hudson CI application, you will access the host where Tomcat is installed and go to the Tomcat bin directory.

```
cd ~/hudson/application/webapps/apache-tomcat-6.0.20/bin
./shutdown.sh
./startup.sh
```

Configure Sonar

To verify the ***Sonar Plugin*** was installed. Go back to the main Hudson dashboard and Select the ***Manage Hudson*** link, then select ***Manage Plugins***. From here, select the Installed tab. You should see the ***Sonar Plugin*** listed on this tab.

To configure Sonar for a particular job, select a specific Hudson job and then select ***Configure***. From the Post-build Actions section on this page, you will see a ***Sonar*** checkbox. Select this checkbox and click the ***Save*** button. Typically, you will also need to configure other project-specific options as well. This is illustrated in the screenshot below.



Post-build Actions

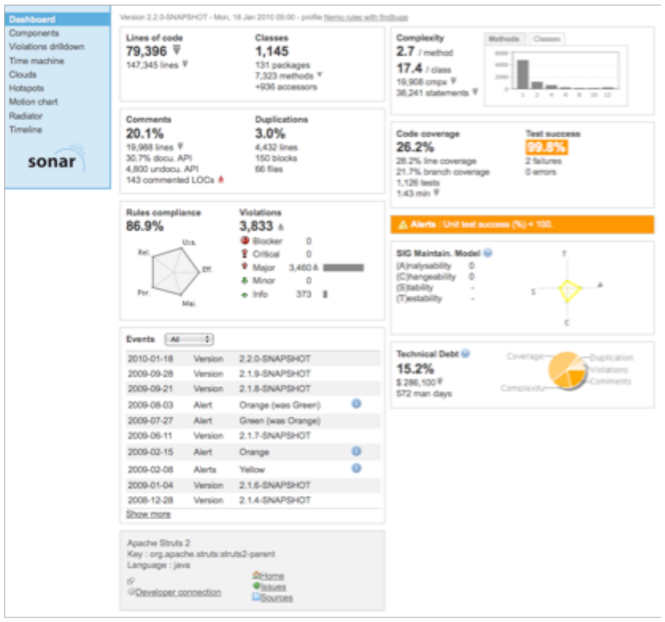
- ☐ Publish Javadoc
- ☐ Archive the artifacts
- ☐ Aggregate downstream test results
- ☐ Publish JUnit test result report
- ☐ Build other projects
- ☐ Record fingerprints of files to track usage
- ☐ Publish Cobertura Coverage Report
- ☑ E-mail Notification

Recipients: paul.duvall@stelligent.com
Whitespace-separated list of recipient addresses. E-mail will be sent when a build fails.
☑ Send e-mail for every unstable build
☑ Send separate e-mails to individuals who broke the build

☑ Sonar

Sonar Installation: (Default)
MAVEN_OPTS:
MAVEN_OPTS env var to provide, if not set the plugin will use the MAVEN_OPTS defined by the maven builder config.
Additional properties:
Additional properties to be passed to the mvn executable (example : -Dsome.property=some.value).

An example of a dashboard report provided by Sonar is shown below.



Version 2.2.0-SNAPSHOT - Mon, 18 Jan 2010 05:00 - profile Nemo rules with findbugs

Lines of code: 79,396 — 147,345 lines
Classes: 1,145 — 131 packages, 7,323 methods, +936 accessors
Complexity: 2.7 / method, 17.4 / class — 19,908 cmpx, 38,241 statements
Comments: 20.1% — 19,988 lines, 30.7% docu. API, 4,800 undocu. API, 143 commented LOCs
Duplications: 3.0% — 4,432 lines, 150 blocks, 66 files
Code coverage: 26.2% — 28.2% line coverage, 21.7% branch coverage, 1,126 tests, 1:43 min
Test success: 99.8% — 2 failures, 0 errors
Rules compliance: 86.9%
Violations: 3,833

Blocker	0
Critical	0
Major	3,460
Minor	0
Info	373

Alerts : Unit test success (%) < 100.

SIG Maintain. Model

(A)nalysability	0
(C)hangeability	0
(S)tability	-
(T)estability	-

Technical Debt: 15.2% — $ 286,100, 572 man days

Events: All

2010-01-18	Version	2.2.0-SNAPSHOT
2009-09-28	Version	2.1.9-SNAPSHOT
2009-09-21	Version	2.1.8-SNAPSHOT
2009-08-03	Alert	Orange (was Green)
2009-07-27	Alert	Green (was Orange)
2009-06-11	Version	2.1.7-SNAPSHOT
2009-02-15	Alert	Orange
2009-02-08	Alerts	Yellow
2009-01-04	Version	2.1.6-SNAPSHOT
2008-12-28	Version	2.1.4-SNAPSHOT

Show more

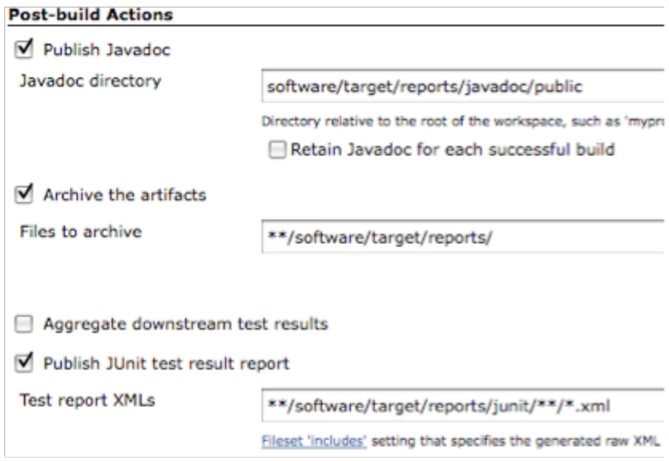
Apache Struts 2
Key : org.apache.struts:struts2-parent
Language : java
Developer connection
Home
Issues
Sources

RUNNING TESTS

Automated tests Pattern: Write an automated test for each unique path.

Antipatterns: Not running tests, no regression tests, manual testing

Once you've written some texts, you can configure your CI server – Hudson, in this case – to display the unit test results. The figure below shows the checkbox to select and an example of the fileset include – it uses the xml file to display the test reports on the Hudson dashboard.

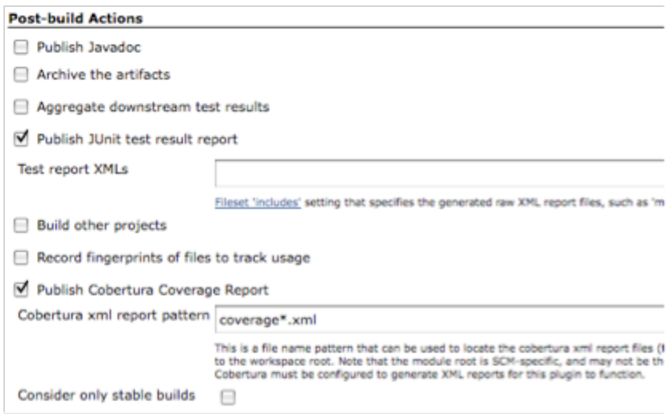


Post-build Actions

- ✓ Publish Javadoc
  - Javadoc directory: software/target/reports/javadoc/public
  - Directory relative to the root of the workspace, such as 'mypr
  - ☐ Retain Javadoc for each successful build
- ✓ Archive the artifacts
  - Files to archive: **/software/target/reports/
- ☐ Aggregate downstream test results
- ✓ Publish JUnit test result report
  - Test report XMLs: **/software/target/reports/junit/**/*.xml
  - Fileset 'includes' setting that specifies the generated raw XML

Code Coverage

Once you've written some automated tests, you can use a CI server tool like Hudson to determine your overall code coverage - either in the form of line or branch coverage. There are several code coverage tools including Emma, Cobertura, NCover, Clover, etc. This example shows how to configure Hudson to run your existing Cobertura reports. The example below assumes that you have already installed the Cobertura plugin for Hudson. Once this plugin has been installed, the Publish Cobertura Coverage Report checkbox is displayed. NOTE: Sonar aggregates Cobertura results as well.



Post-build Actions

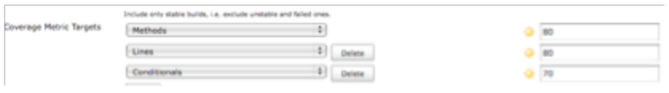
- ☐ Publish Javadoc
- ☐ Archive the artifacts
- ☐ Aggregate downstream test results
- ✓ Publish JUnit test result report
  - Test report XMLs:
  - Fileset 'includes' setting that specifies the generated raw XML report files, such as 'm
- ☐ Build other projects
- ☐ Record fingerprints of files to track usage
- ✓ Publish Cobertura Coverage Report
  - Cobertura xml report pattern: coverage*.xml
  - This is a file name pattern that can be used to locate the cobertura xml report files (
    to the workspace root. Note that the module root is SCM-specific, and may not be th
    Cobertura must be configured to generate XML reports for this plugin to function.
  - Consider only stable builds ☐

Build Thresholds

Build Threshold Pattern: Notify team members of code aberrations such as low code coverage or high cyclomatic complexity. Fail a build when a project rule is violated. Use continuous feedback mechanisms to notify team members.

Antipatterns: waiting for numerous code quality issues to go undiscovered or build up to the point where software maintenance increases or functional features are affected.

Hudson provides a way to fail the build based on specific thresholds. This is an implementation of the build threshold pattern. In the example below, you can see that Hudson lets you configure method, line and conditional code coverage targets. If the code in the build doesn't meet these criteria, the build fails. This is an effective to discover and fix potential problems earlier in the development process.



Include only stable builds, i.e. exclude unstable and failed ones.

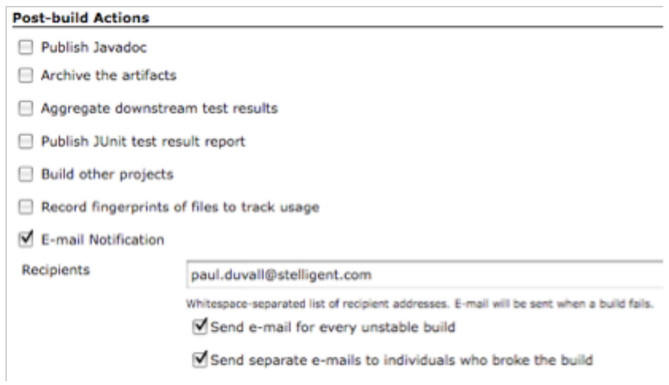
Coverage Metric Targets			
Methods		●	80
Lines	Delete	●	80
Conditionals	Delete	●	70

SENDING EMAIL NOTIFICATIONS

Continuous Feedback Pattern: Sending automated feedback from CI server to development team. Setting up the server to send email notifications will be covered here, but there are other notification mechanisms available (e.g., RSS, SMS, X10, Monitors, Web Notifiers).

Antipatterns: Minimal feedback, which prevents action from occurring. Receiving spam feedback, which causes people to ignore messages.

From the Hudson dashboard, select the job you are configuring, then select the ***Configure*** link. One of the configuration options is called ***Post-build Actions***. Then, select the ***E-mail Notification*** checkbox. Here you can enter the email addresses for the people who will receive emails after builds are run. You can choose to send an email for every unstable build and/or to send separate emails to individuals who broke the build. The latter option requires you configure your domain suffix in the Hudson system configuration (See the configuring the email server information section above).



Post-build Actions

- ☐ Publish Javadoc
- ☐ Archive the artifacts
- ☐ Aggregate downstream test results
- ☐ Publish JUnit test result report
- ☐ Build other projects
- ☐ Record fingerprints of files to track usage
- ✓ E-mail Notification
  - Recipients: paul.duvall@stelligent.com
  - Whitespace-separated list of recipient addresses. E-mail will be sent when a build fails.
  - ✓ Send e-mail for every unstable build
  - ✓ Send separate e-mails to individuals who broke the build

TOOLS

The following are *not* recommended *tools*, but recommended tool *types* - with example tools that you might use. There are many more tools available than the list provided here. It's important to realize that you need numerous types of tools to effectively create working software in a single command with every change applied to the version-control repository.

Tool Name	Tool Type	Platform
CruiseControl Hudson CruiseControl.net	Continuous Integration Server	Java .Net Others
Checkstyle PMD FindBugs JavaNCSS FxCop NCover Cobertura Ratproxy	Code Quality Static Analysis	Java .Net
JUnit DbUnit Selenium AntUnit JMeter	Automated Unit, Database Seeding, Functional, Build, Load, Web Services Testing	Java

Ant Maven Buildr NAnt MSBuild	Build Tool	Java .Net Ruby Others
Ivy Artifactory	Dependency Manager, Repository	Ant Java
Liquibase	Automated Database Upgrades	Java Others
Java Secure Channel	Deployment	Java
Grand	Build Diagrams	Ant
UMLGraph	UML Documentation	Java
SchemaSpy	ERD Documentation	Any

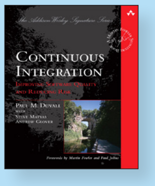
JDepend	Dependency Analysis	Java
Sonar	Code Quality Reporting & Aggregation	Java
Subversion Perforce Accurev TFS Git	Version-Control	Any
IzPack	GUI Installer	Java Ant
Email X10 SMS Jabber (IM)	Feedback Mechanisms	Any

ABOUT THE AUTHOR

Paul M. Duvall is the CEO of Stelligent, a company that provides products and services to help customers create production-ready software every day. A featured speaker at many leading software conferences, he has worked in virtually every role on software projects: developer, project manager, architect, and tester. He is the principal author of Continuous Integration: Improving Software Quality and Reducing Risk (Addison-Wesley, 2007) and a 2008 Jolt Award Winner. Paul contributed to the UML 2 Toolkit (Wiley, 2003), wrote a series for IBM developerWorks called Automation for the People, and contributed a chapter to No Fluff Just Stuff Anthology: The 2007 Edition (Pragmatic Programmers, 2007). His company provides a cloud-based product called CI as a Service, which recently released its beta version at http://ciasaservice.com/. He is passionate about automating software development and release processes and actively blogs at http://blog.stelligent.com/

Some of the concepts and material in this Refcard were adapted from Continuous Integration: Improving Software Quality and Reducing Risk (Addison-Wesley, 2007) - http://www.amazon.com/gp/product/0321336380/?tag=integratecom-20

RECOMMENDED BOOK



For any software developer who has spent days in "integration hell," cobbling together myriad software components, *Continuous Integration: Improving Software Quality and Reducing Risk* illustrates how to transform integration from a necessary evil into an everyday part of the development process. The key, as the authors show, is to integrate regularly and often using continuous integration (CI) practices and techniques.

BUY NOW
books.dzone.com/books/continuous-integrations

ISBN-13: 978-1-934238-67-7
ISBN-10: 1-934238-67-8
50795
9 781934 238677
$7.95

Version 1.0