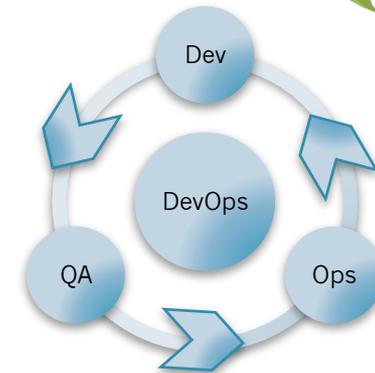
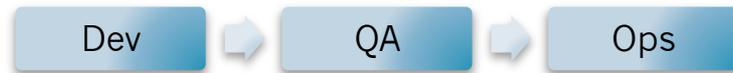
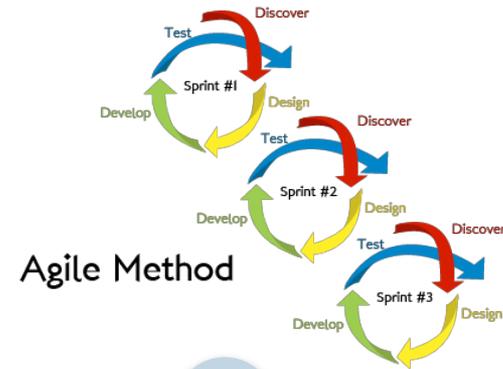
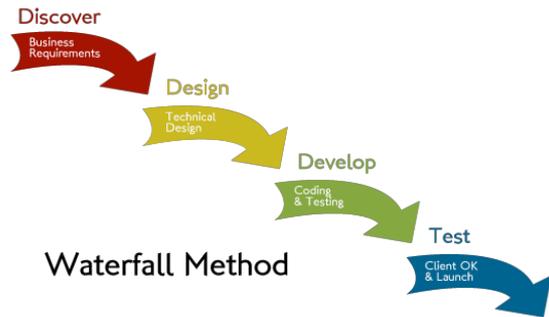


# Practicing Continuous Delivery using Hudson

Winston Prakash  
Oracle Corporation

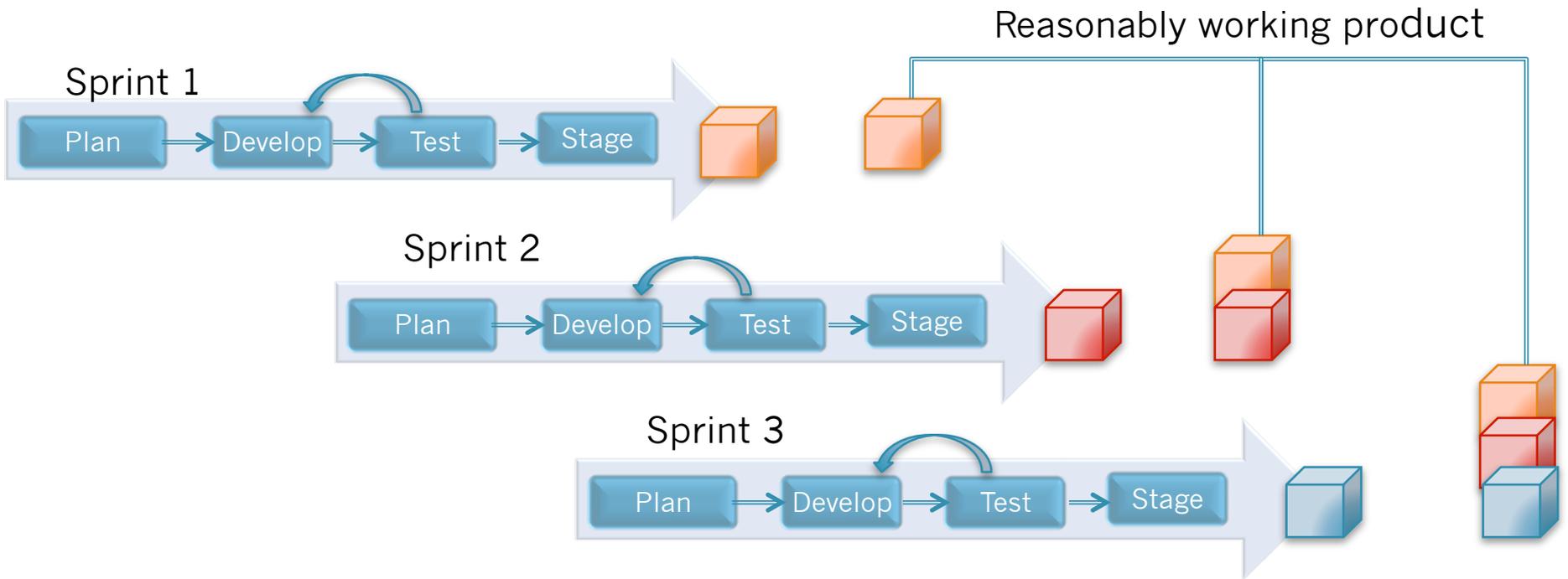
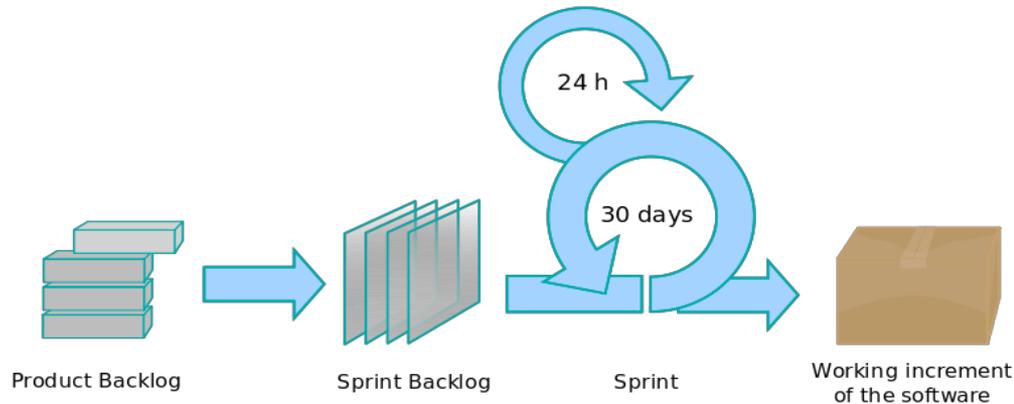
# Development Lifecycle



Typical turn around time is  
6 months to 1 year

Sprint cycle is typically  
2 weeks to 4 weeks

# Typical Sprint Cycle



# Further Thought Process

- There are some greedy people out there
- They can't wait until the end of the sprint cycle to get a working product
- They want a working product on every commit



Commit



Build  
Magic

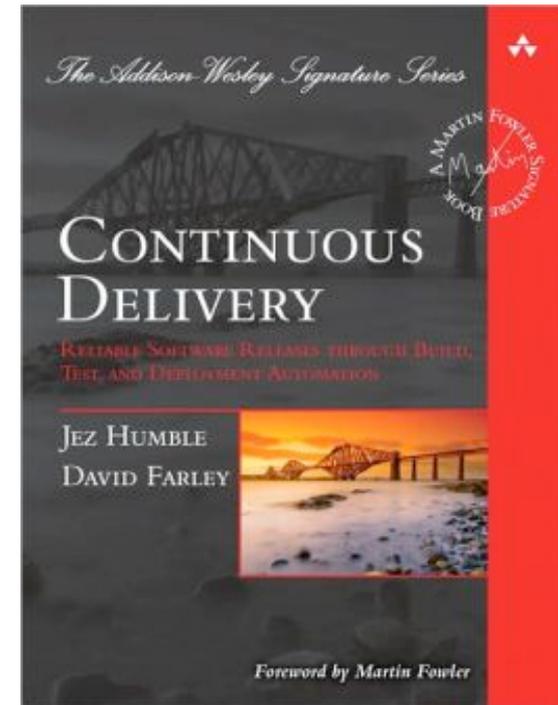


Production  
Ready

Thus the **Continuous Delivery** concept was born

# What is Continuous Delivery?

- A set of **practices and principles** aimed at building, testing and releasing software faster and frequently.
- Produce a **deployable-to-production** build regularly, probably on each commit.
- Every build is a **potential** release.



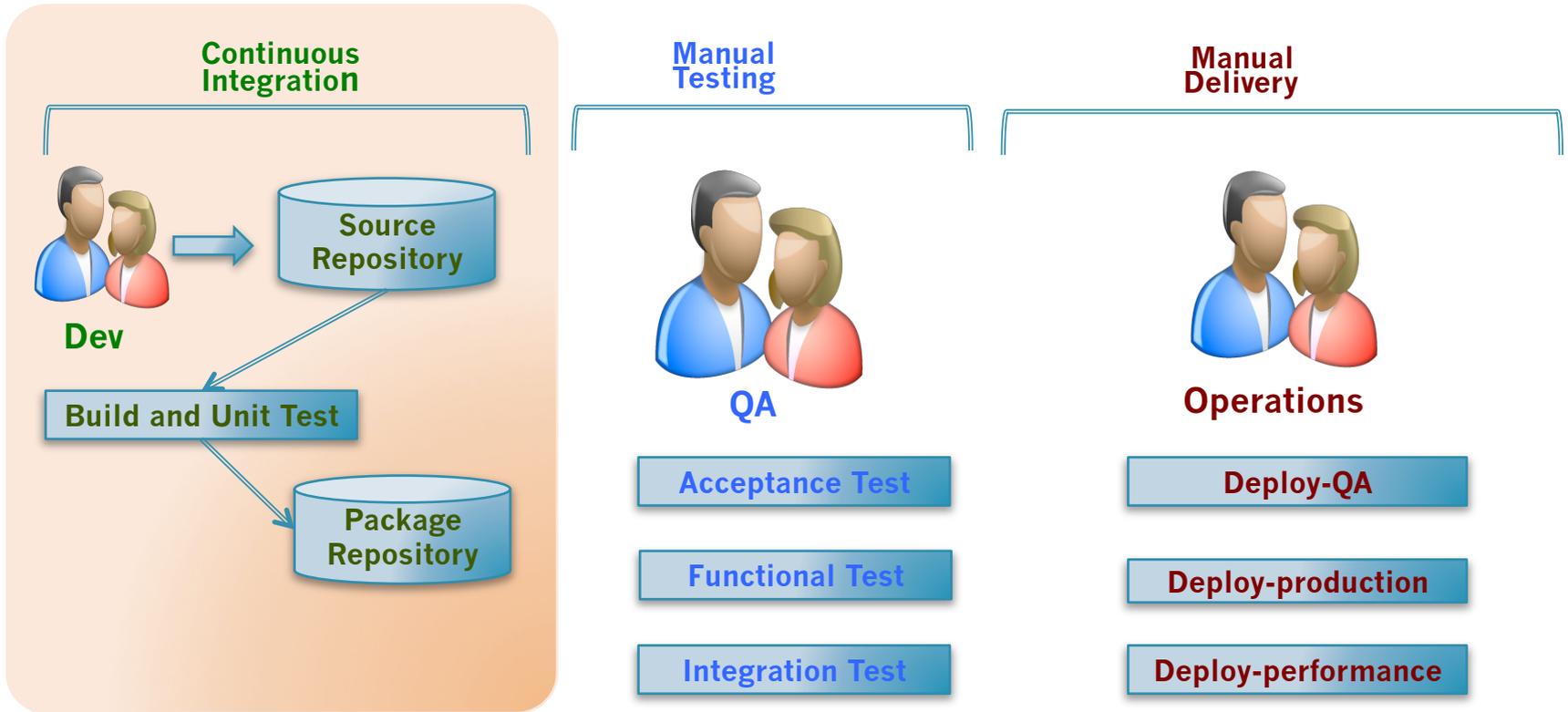
# Commit to deploy



In a **Test Driven Development** build pipeline, **Continuous Integration** is the first step and the end result is the **Continuous Delivery**.

While **Continuous Delivery** promotes the concept of keeping your product in a deliverable state on each commit, **Continuous Deployment** takes it further. On each commit, the deliverable can be deployed to a production environment.

# Typical Hudson CI Server Usage



Hudson is mostly tuned to focus on **development teams**

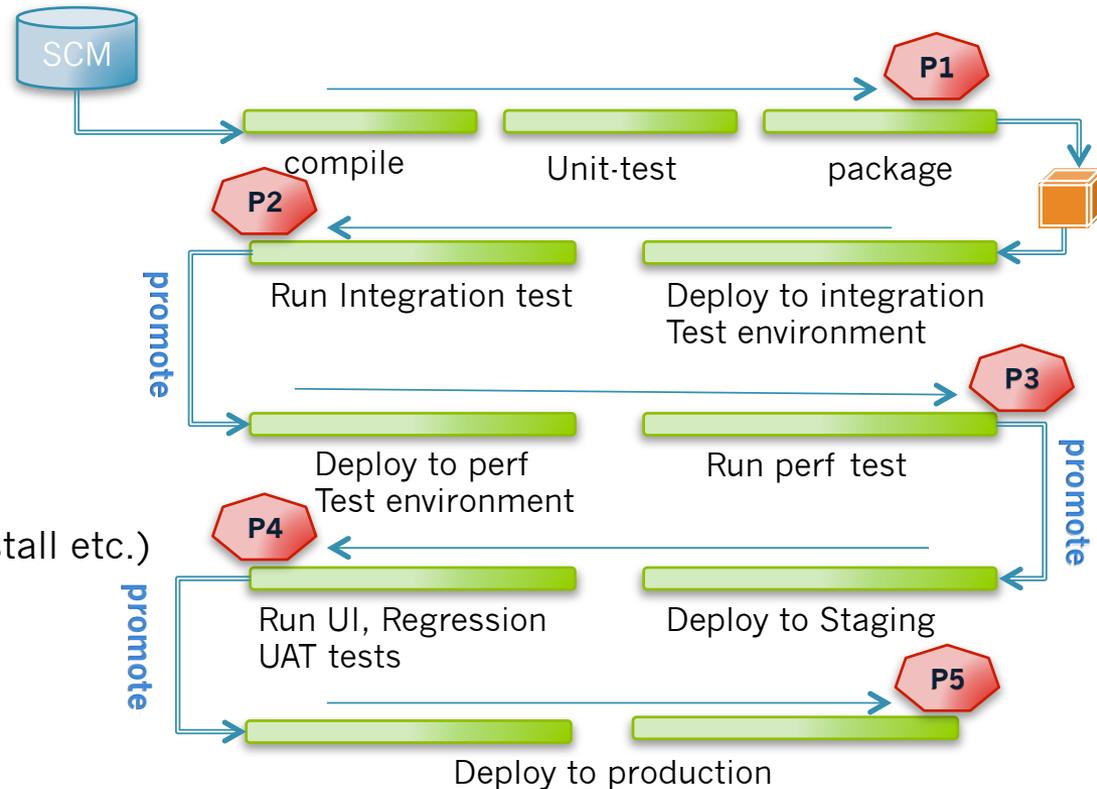
# Build Pipeline

## ➤ Commit

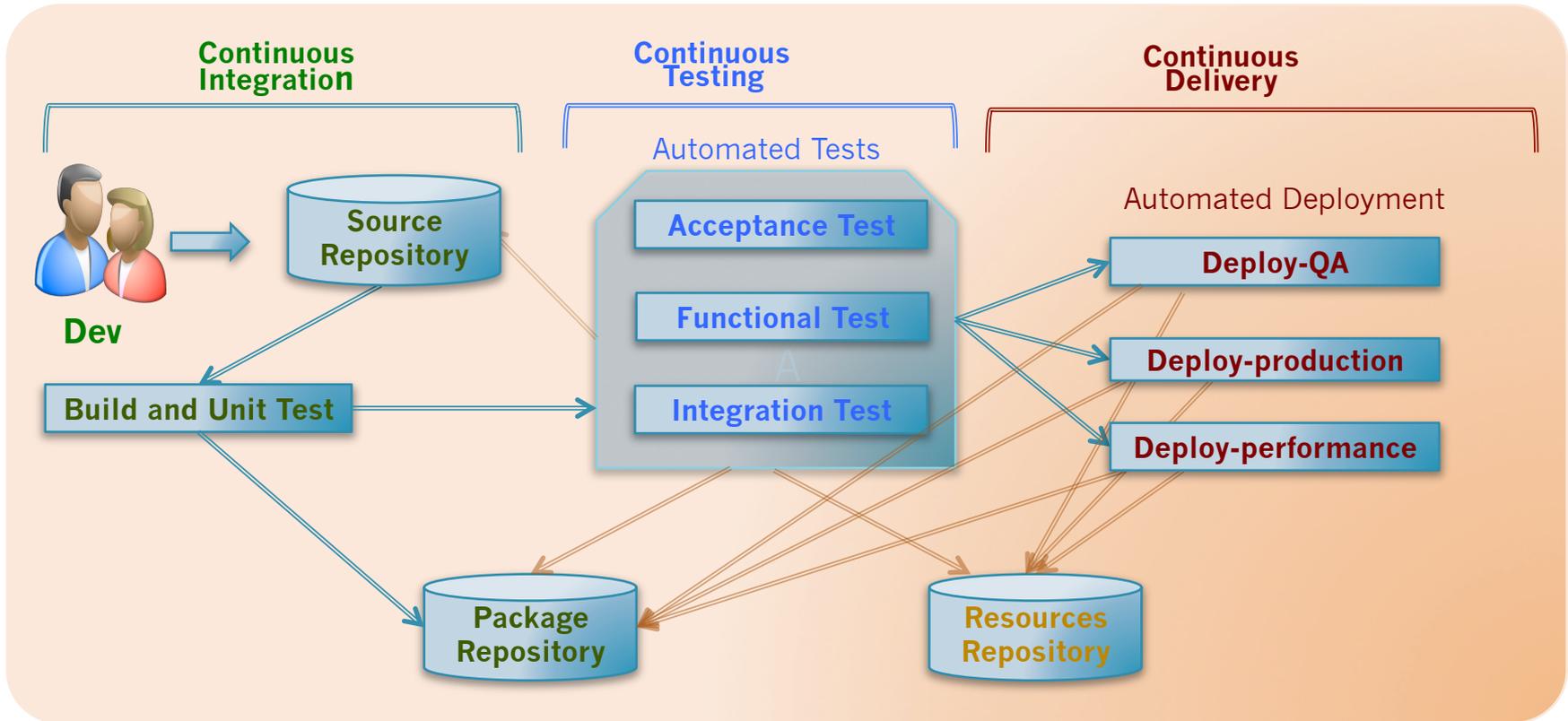
## ➤ Build and Test

- Unit tests
- Static code coverage
- Packaging
- Integration tests
- UI tests
- Performance tests
- Regression tests
- Deployment tests (install, uninstall etc.)
- Manual exploratory tests
- Regulatory, compliance checks
- Clearance from UAT

## ➤ Stage and Deploy



# Setting up Hudson to do Continuous Delivery

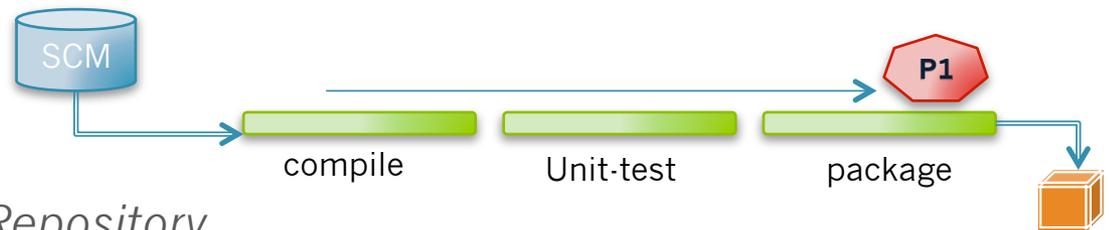


Identifying the relevant plugins and configuring the jobs to participate in the pipeline is critical.

# Setting up CI Environment

- A Centralized SCM repositories (Git, SVN, CVS etc)
- Dedicated build servers
- Continuous Integration software (Hudson)
- Unit testing framework (JUnit, nUnit etc)
- Build tool (Maven, Gradle, Ivy, Ant etc)
- Deployment environment (Application Servers)
- Build Dashboard (Hudson UI)
- Communication tool (E-mail, twitter, IRC etc)
- Deployment Tool

# Effective practicing of CI



- *Maintain a Single Source Repository.*
- *Automate the Build*
- *Make Your Build Self-Testing*
- *Everyone Commits To the Mainline Every Day*
- *Every Commit Should Build the Mainline on an Integration Machine*
- *Keep the Build Fast*
- *Test in a Clone of the Production Environment*
- *Make it Easy for Anyone to Get the Latest Executable*
- *Everyone can see what's happening*
- *Automate Deployment*

# Choosing Hudson Plugin for Effective practicing of CI

[http://wiki.eclipse.org/Hudson-ci#Hudson\\_Plugins](http://wiki.eclipse.org/Hudson-ci#Hudson_Plugins)

## Maintain a Single SCM

This principle encourages the project team to use SCM to maintain their source code. Hudson supports

99% of Hudson users use one of

- Git
- CVS
- SVN
- Perforce
- Clearcase
- Mercurial

Hudson supports ~20 additional SCM wh



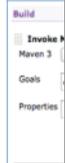
## Automate the Build

Automating the build using a single of a CI build. Hudson supports various

99% of Hudson users use one of

- Ant
- maven
- gradle
- MSBuild
- Nant
- Rake

Hudson supports ~40 additional build tools



## Make your build self-testing

CI build is not about catching errors more quickly and efficiently. Hudson supports various **Frameworks** via Plugins.

99% of Hudson users use one of

- jUnit
- nUnit
- Selenium
- CppUnit
- TestNg
- xUnit

Hudson supports ~10 additional Unit Test wh

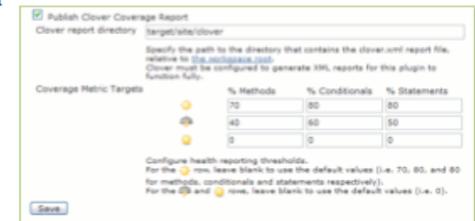


## Make your build self-testing (Code Coverage)

Self testing is best achieved if there is uniform code coverage. Hudson supports various **Code Coverage Tools** via Plugins

99% of Hudson users use one of

- Clover
- Cobertura
- Emma
- Serenity
- Sonar
- NCover



Hudson supports ~2 additional Code Coverage which are used by less than 1% of the users

# Buildable Units

Important guideline of CI is to build fast and give back feedback quickly. To achieve this

- Rather than building the entire source in one single job, divide the project sources into buildable chunks. Each chunk of software must be able to build independent of each other.
- The dependent chunks must be built separately and stored in an artifact repository manager for other software chunks to use them as dependencies.
- Each of the software chunks is a buildable unit and is built by a single Hudson job.



# Setting up upstream-downstream builds

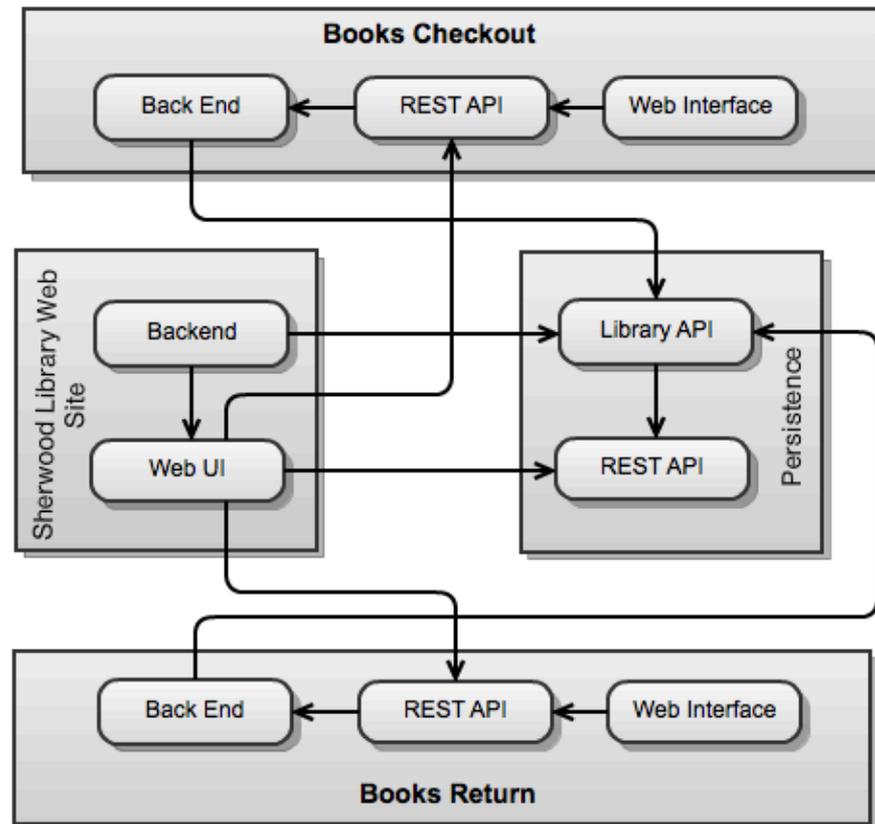
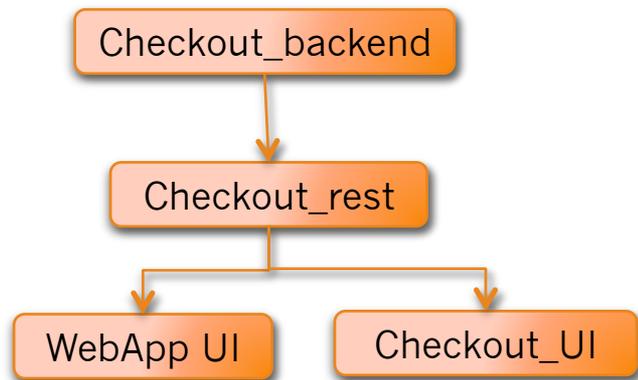
## Build Triggers

- Build after other projects are built

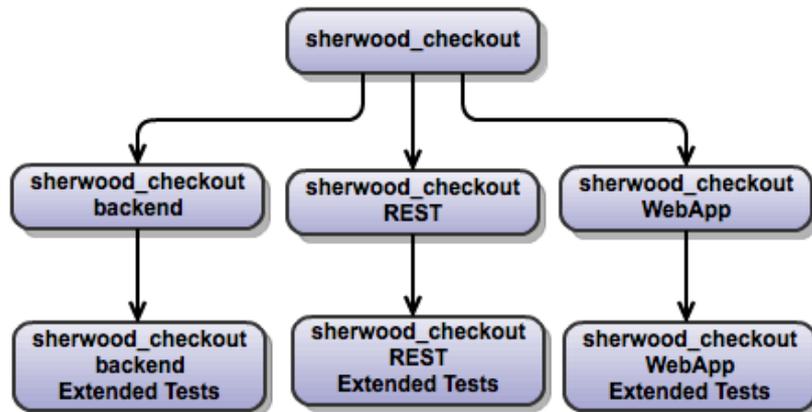
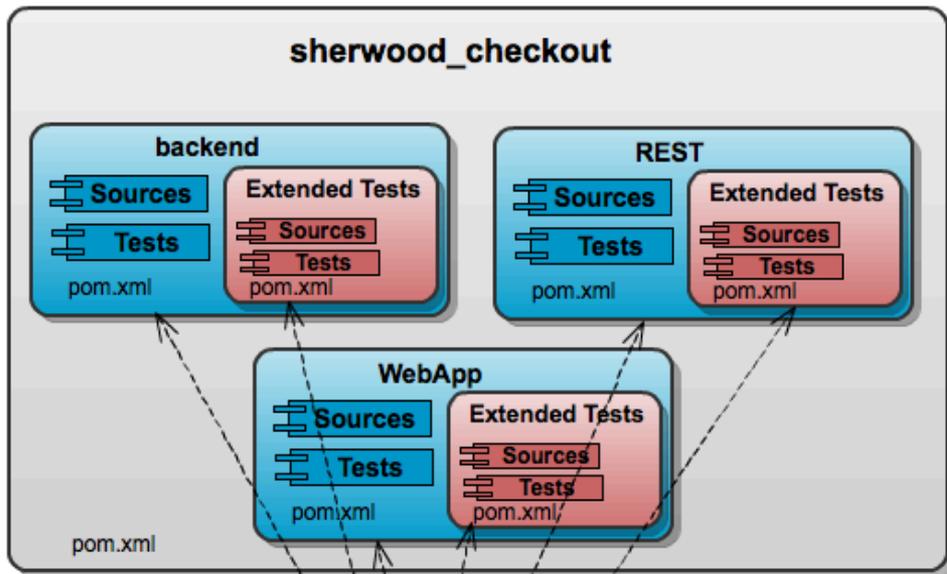
Set up a trigger so that when some other projects finish building, you can conveniently run an extensive test after a build.

This configuration is the opposite view of the "Build on change" configuration. You can change the other automatically.

Projects names  Multiple projects can be specified like 'abc, def'



# Speeding up CI Builds

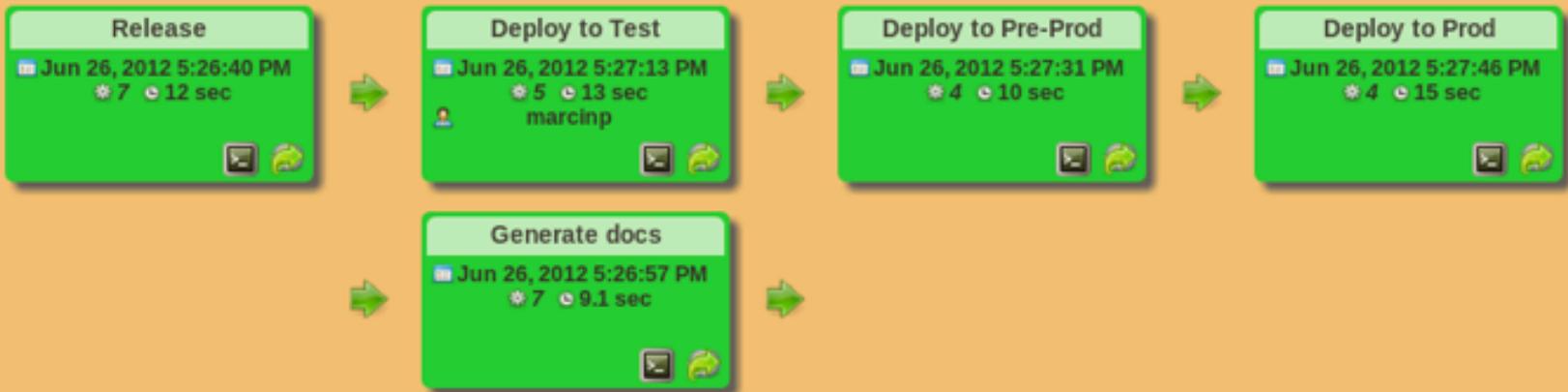
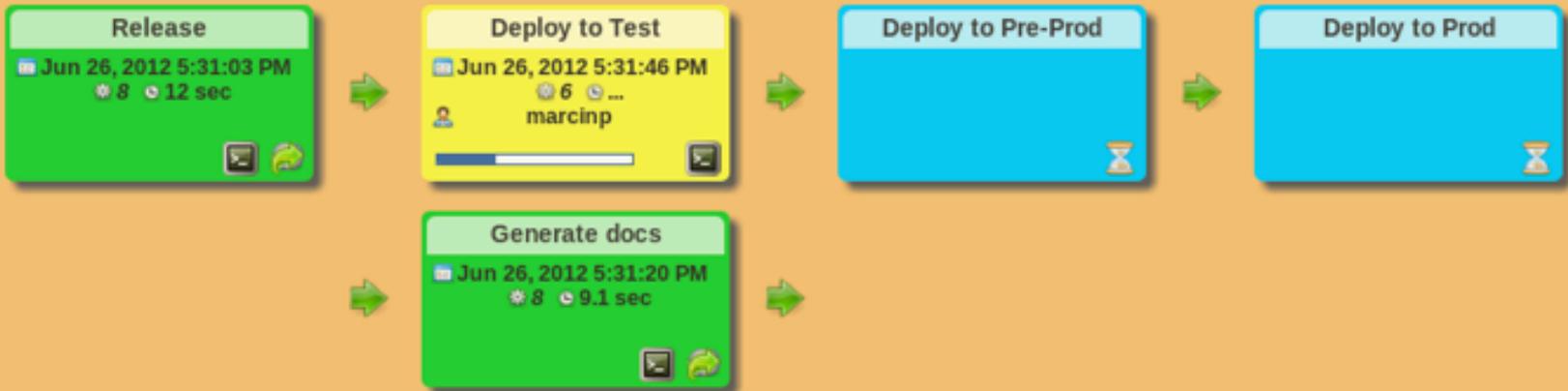


**Buildable Sub Units**

**Staged Builds**

- The first stage would do the compilation and localized unit tests. The unit tests may be created with out any real time database or server connections to keep it fast. (Mockito, Powermock)
- In the second stage, the extended builds run different suits of tests, may be with real time server and database connections.

# Build pipeline plugin



# Cascading jobs



## Job Configurations

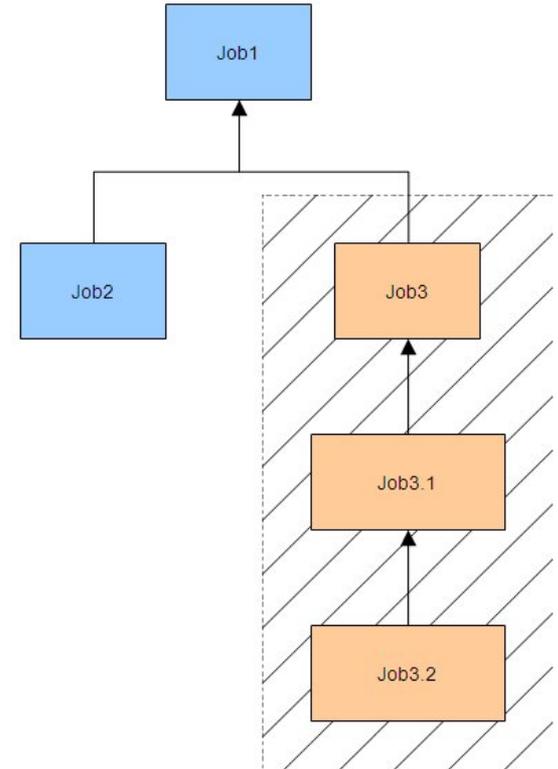
Project name

Cascading Project

Description

Discard Old Bu  This build is pa

- sherwood\_checkout
- sherwood\_checkout\_rest
- sherwood\_checkout\_test\_harness
- sherwood\_checkout\_top\_level**
- sherwood\_checkout\_webapp



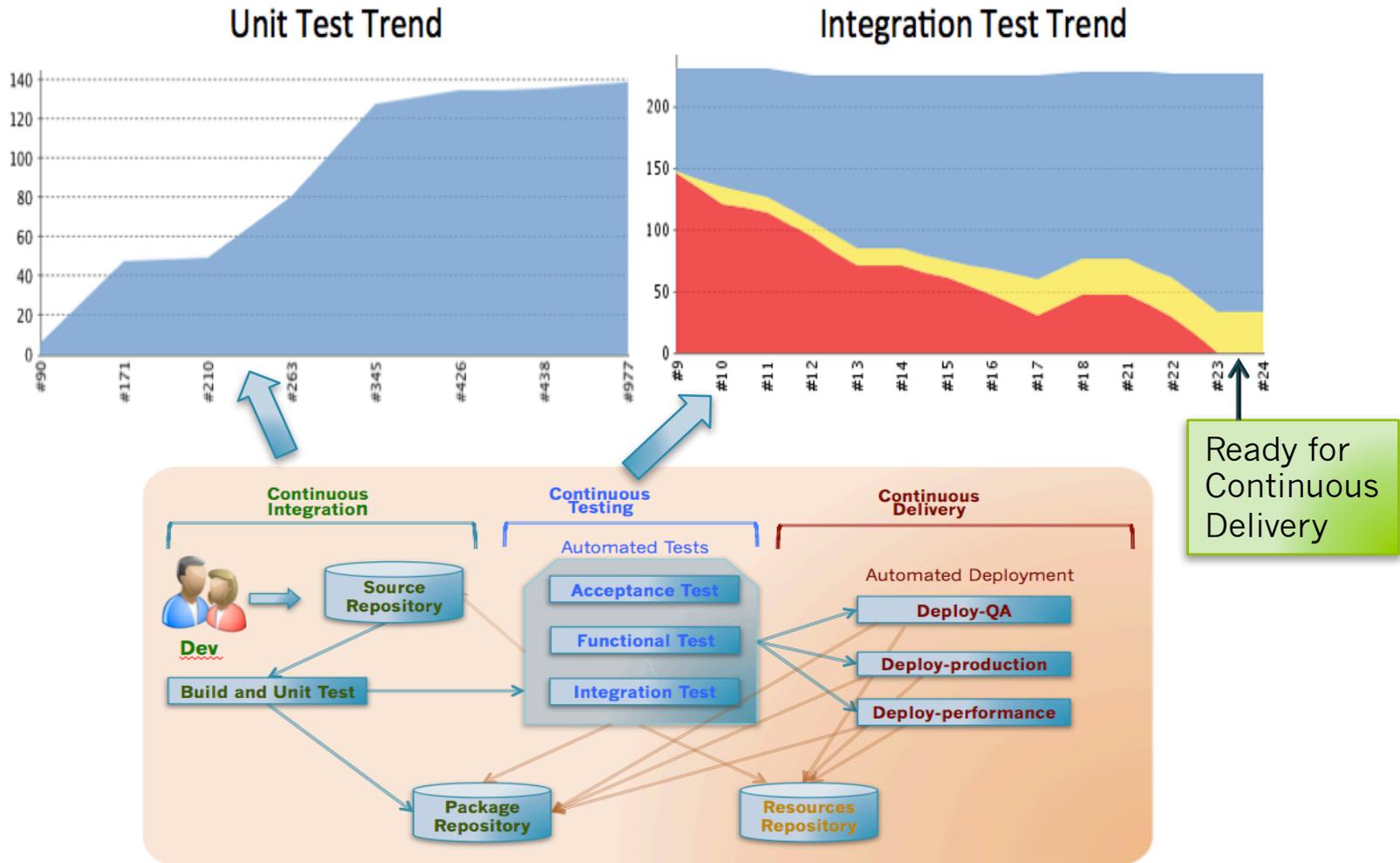
E-mail Notification

Recipients

Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

- Send e-mail for every unstable build
- Send separate e-mails to individuals who broke the build

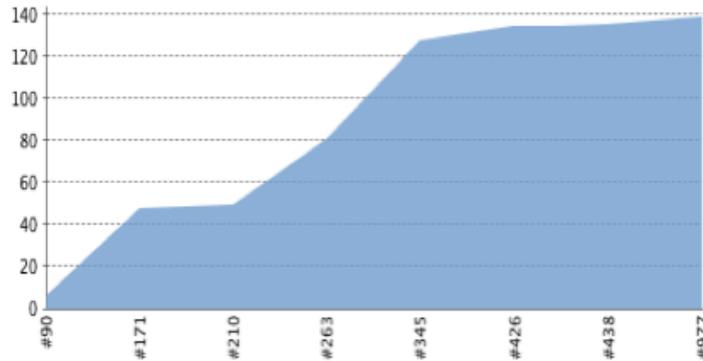
# Monitor Test Trends



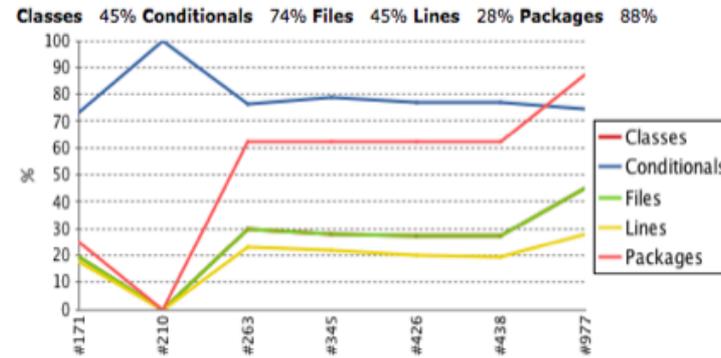
In a CI build, the unit tests should never fail.  
During the initial stage of the project, the integration test may be in flux.

# Monitor Quality Metrics Trend

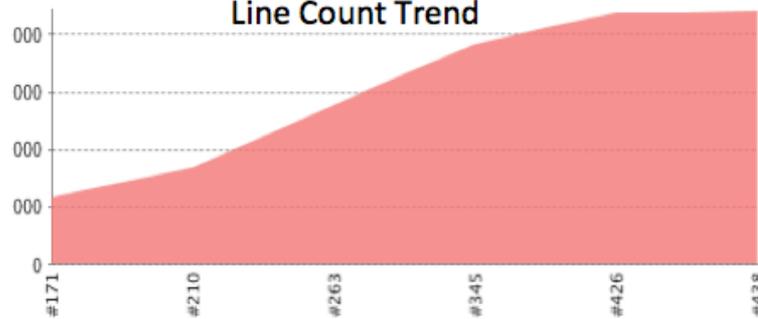
## Tests Result Trend



## Code Coverage Trend



## Line Count Trend



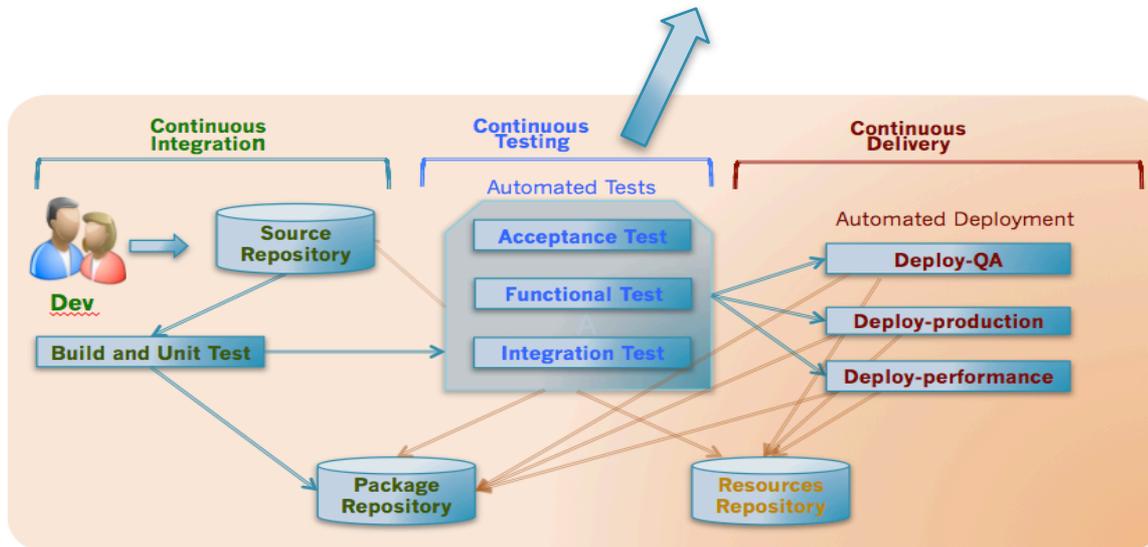
Code quality measurement is important in Continuous delivery. Improves the confidence of the product being deliverable state.

# Automated Upload

(FTP)

As part of Build pipeline, often there may be requirements to copy configuration files or database schemas, test scripts, properties files, install scripts etc., which are part of a build to another machine to facilitate additional test run

The screenshot shows a Jenkins configuration window for publishing artifacts to FTP. It includes a checked checkbox for 'Publish artifacts to FTP', a dropdown for 'FTP site' set to 'acceptance-test-machine', and two entries under 'Files to upload'. The first entry has 'Source' as 'target/sql/\*.sql' and 'Destination' as 'sql'. The second entry has 'Source' as 'target/properties/\*.txt' and 'Destination' as 'properties'. Each entry has a 'Delete' button, and there is an 'Add' button at the bottom.



# Automated Execution

(SSH)

Execute commands on that remote machine to ready the machine for automatic deployment.

Execute shell script on remote host using ssh

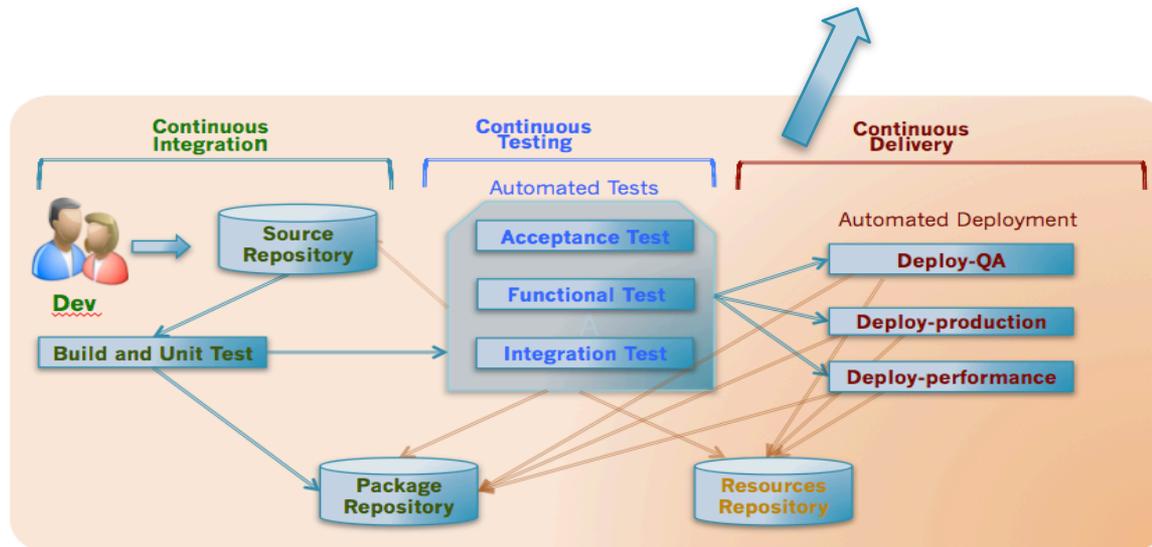
SSH site

Pre build script 

```
/usr/local/tomcat6/bin/shutdown.sh  
rm /usr/local/tomcat6/wepapp/test-app.war  
rm -rf /usr/local/tomcat6/wepapp/test-app
```

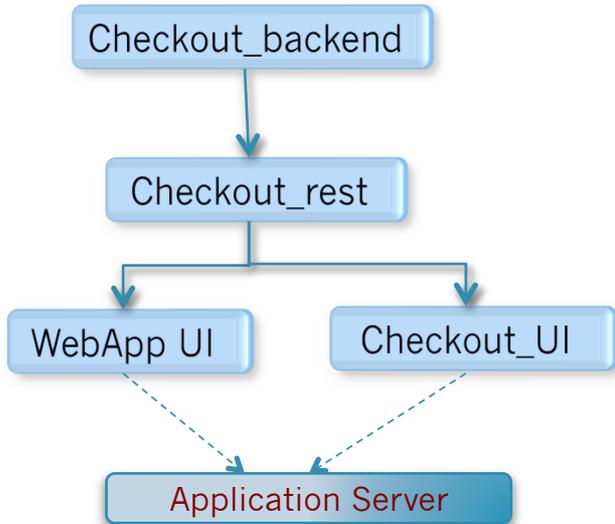
Post build script 

```
/usr/local/tomcat6/bin/startup.sh
```



# Automated Deployment

(Application Servers)



Deploy war/ear to a container

WAR/EAR files

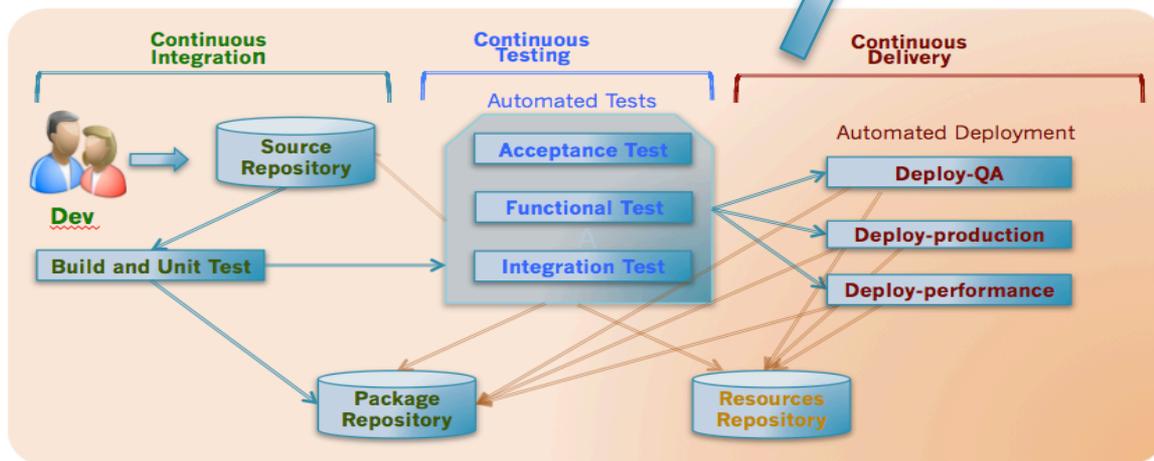
Container

Manager user name

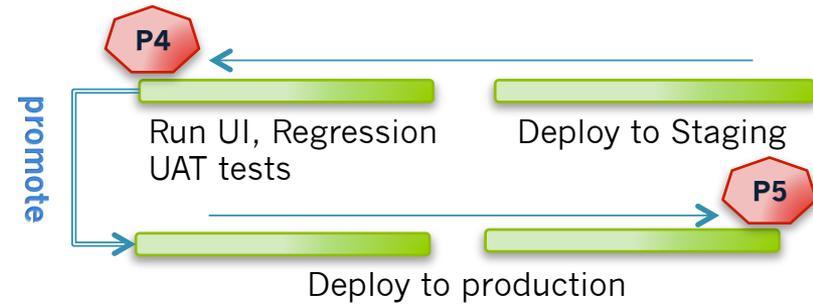
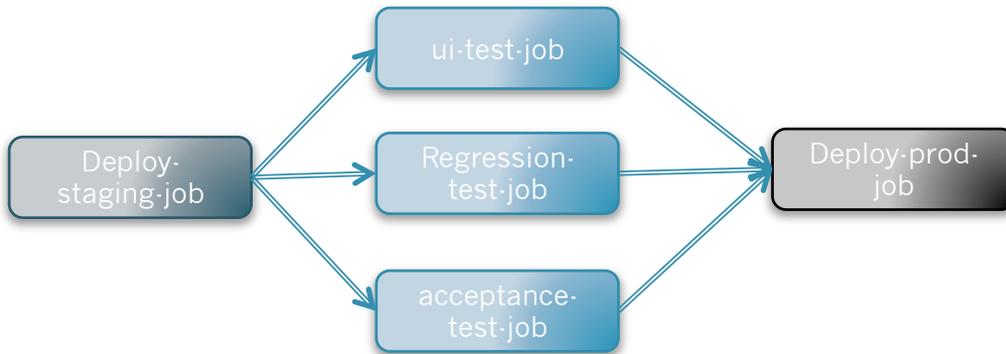
Manager password

Tomcat URL

Deploy on failure



# Diamond build pattern



**Join plugin** triggers a job after all the downstream jobs are completed in parallel. This allows a pipeline to branch out to perform many steps in parallel, and then run another job after all the parallel jobs are finished.

Build other jobs

Jobs to build: ui-test-job, regression-test-job

Trigger even if the build is unstable

Publish Javadoc

Aggregate downstream test results

Join Trigger

Trigger even if some downstream projects are unstable

Projects to build once, after all downstream projects have finished: prod-deploy-job

Run post-build actions at join

# Automated Promotion

Though promoted build plugin provides opportunity to promote every build, typically the promotion process is done for a pipeline

Promote builds when...

Promotion process

Name

Icon

Restrict where this promotion process can be run

**Criteria**

- When the number of builds success is satisfied
- Promote immediately once the build is complete
- When the following downstream projects build successfully
- When the following upstream promotions are promoted

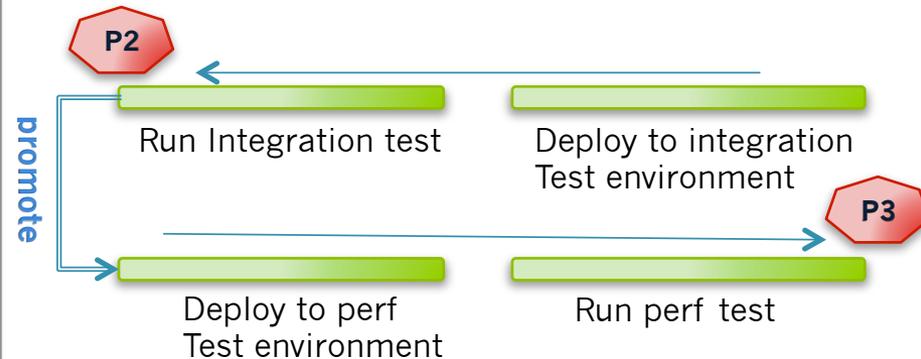
Promotion names

Only when manually approved

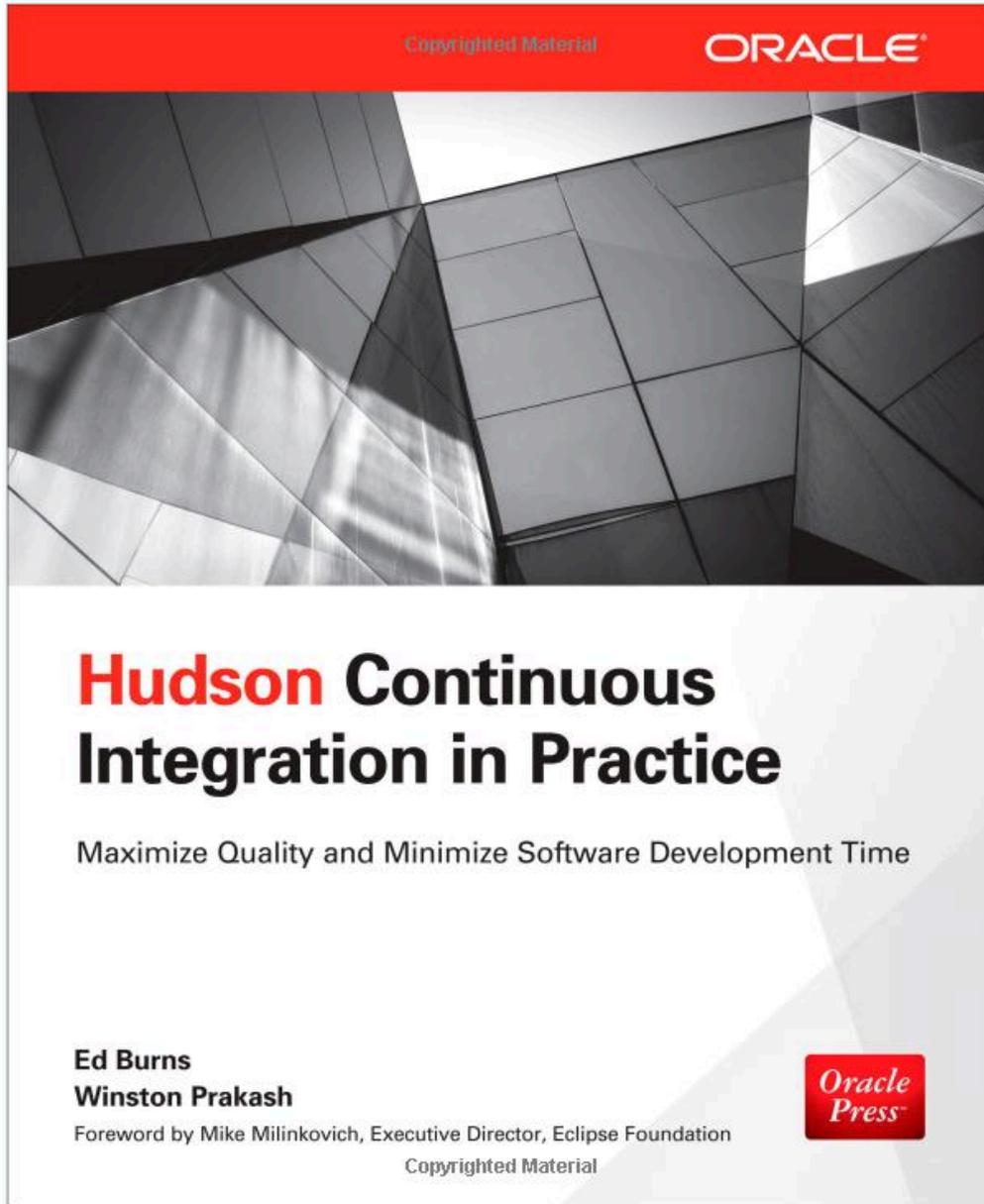
**Actions**

**Build other jobs**

Jobs to build



# Hudson Book



### Part1:

Practicing effective  
CI using Hudson

### Part2:

Hudson plugin  
development

[http://www.amazon.com/  
Hudson-Continuous-Integration-  
Practice-Burns/dp/  
0071804285](http://www.amazon.com/Hudson-Continuous-Integration-Practice-Burns/dp/0071804285)