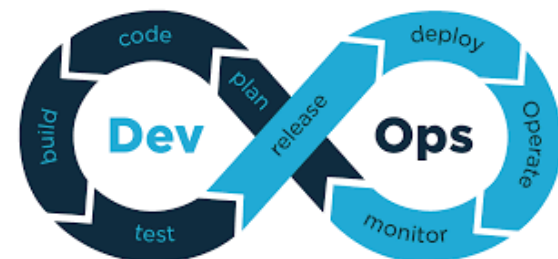


Automated testing

Tools and Techniques for Software Testing - Barbara Russo
SwSE - Software and Systems Engineering group

DevOps

- The term DevOps emerged a decade ago as an amalgamation of Development and Operations
- A reaction to a perceived disconnect between developers and operators within the same organization
- Adopted to develop and operate software solutions in the cloud



Automation in development

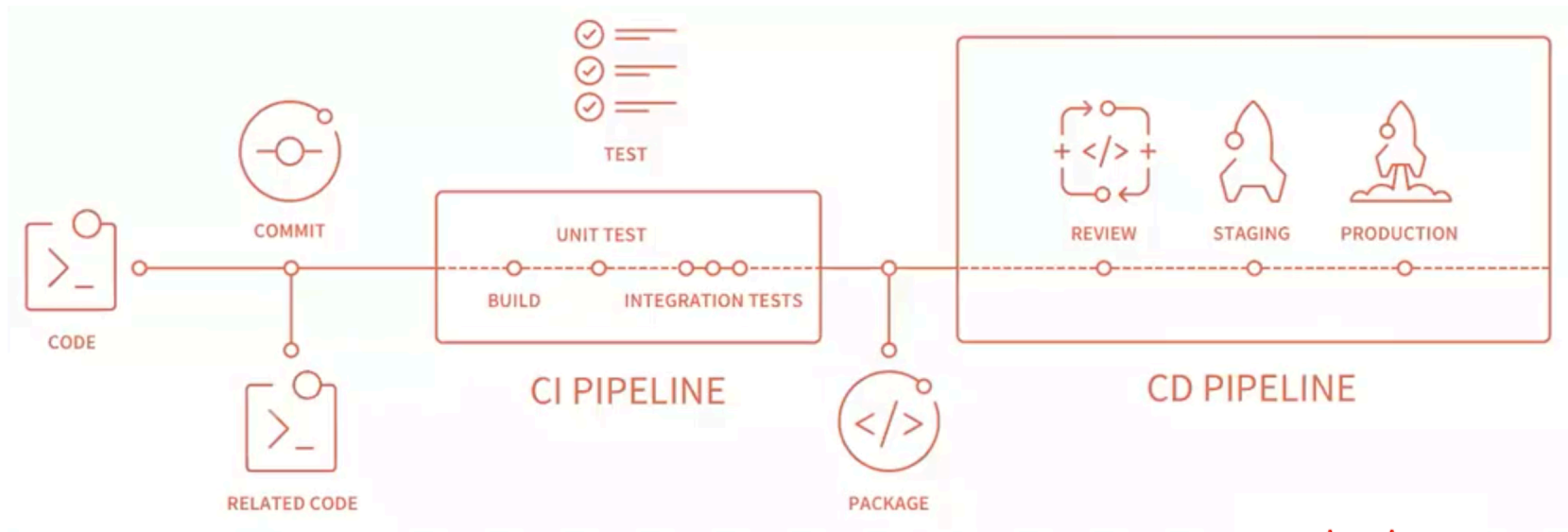
- **Virtualized infrastructure** enables perform deployment and configuration using pre-configured scripts
- This allows organizations to deploy new versions **several times a day**
- Operations can be configured to provide **feedback from daily operations**, which in turn supports development and deployment of new and enhanced features at a **rapid pace**



Why did we set up an automated system to run tests?

- Manual testing is more prone to human error
- To reduce QA effort
- Reduce waiting time to know build quality
- Categorize tests and run against build (smoke/regression test)
- Schedule tests and make them under control (nightly regression)

Continuous Development lifecycle



DevOps

- Continuous Integration
 - Automated Testing and Artifact Creation (e.g., build creation)
- Continuous Delivery
 - Automated deployment to test and staged environments
 - Manual deployment to production
- Continuous Deployment
 - Automated Deployment to production

Continuous Development

- In Continuous Integration, several builds in a day are executed (nightly builds as well)
- Each build should pass through various sandboxed environments leading up to production and stages (stable, develop, master)

Continuous Delivery

- In Continuous Delivery, code changes are continuously deployed, although the **deployments are triggered manually**
- If the entire process of moving code from source repository to production is fully automated, the process is called Continuous Deployment

Continuous Deployment

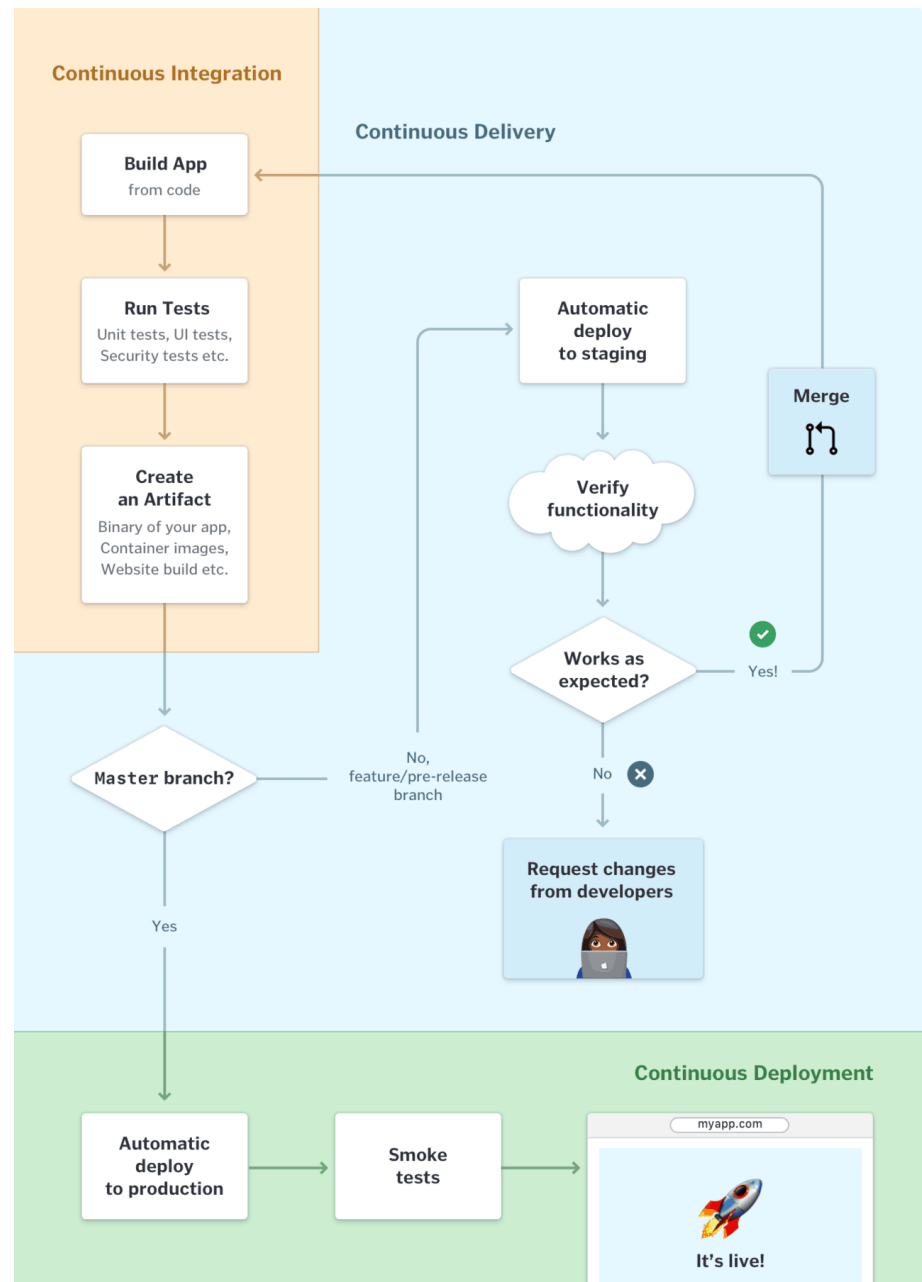
- Strategy for software releases for which commits that pass **automated testing** are **automatically released** into the production environment

Continuous deployment

- Deployment process involves
 - Continuous development and
 - Continuous testing
- Developers and testers work together to release high-quality product in shorter time to market

Test and staging environments

- A test environment is where tests are performed.
No real environment bounds tests
- A staging environment (stage) is a **nearly exact replica** of a **production environment**
- Staging environments are made to test codes, builds, and updates to ensure quality under a production-like environment before application deployment



Exercise

- Clean up your project
- How many branches do you have?
- Did you merge your branch into master before deploy?
- Did you create test environment? (e.g., test package)

Automation in distributed environment - Pipelines

Tools and Techniques for Software Testing - Barbara Russo
SwSE - Software and Systems Engineering group

Pipelines

- Pipelines are top-level components of continuous integration, delivery, and deployment
- A pipeline is a **group of job/tasks** that get executed in *stages* also called *batches*
- All jobs in stages run sequentially/parallel based on the runners available

Pipelines

- If all jobs succeed, then the pipeline moves to the next stage
- If it fails, next stage will not get executed

Typical stages

- Build
- Test
- Deploy
- Review
- Dynamic Application Security Testing (DAST)
- Staging
- Canary

Canary test

- Progressively deploy (e.g., push to master) code changes to a small number of users (unaware and not volunteers)



```
void main(String[] args) {  
  int foo;  
  // do something  
  bar(foo);  
  System.out.println("Hi");  
}
```

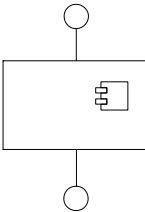
Implementation

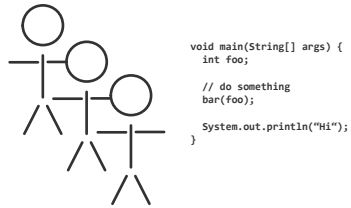
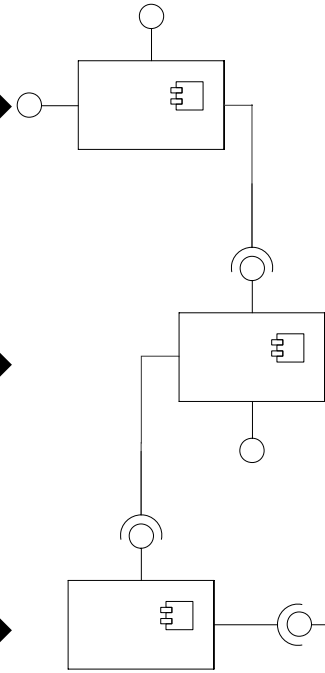
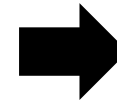
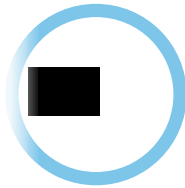
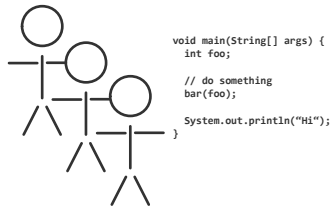


Build

Functional Testing

Performance Testing



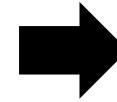
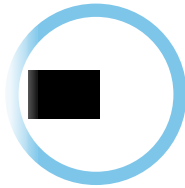
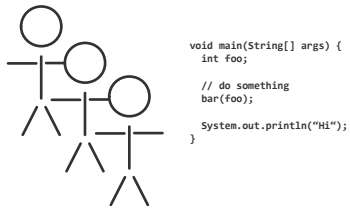


Implementation

Build

Functional Testing

Performance Testing



Fast & frequent releases

```
void main(String[] args) {  
    int foo;  
  
    // do something  
    bar(foo);  
  
    System.out.println("Hi");  
}
```

```
void main(String[] args) {  
    int foo;  
  
    // do something  
    bar(foo);  
  
    System.out.println("Hi");  
}
```

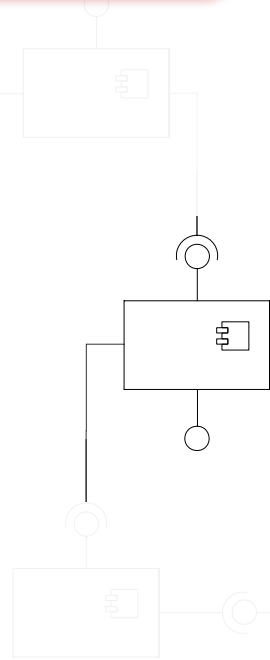
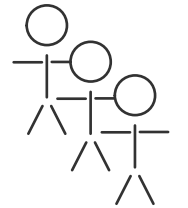
Implementation

```
void main(String[] args) {  
    int foo;  
  
    // do something  
    bar(foo);  
  
    System.out.println("Hi");  
}
```

Build

Functional Testing

Performance Testing

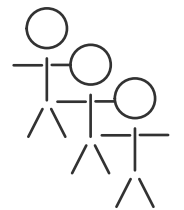


Pipeline automation

Fast & frequent releases



```
void  
int  
// do something  
bar(foo);  
System.out.println("Hi!");
```



```
void main(String[] args) {  
    int foo;  
    // do something  
    bar(foo);  
    System.out.println("Hi!");  
}
```



```
void main(String[] args) {  
    int foo;  
    // do something  
    bar(foo);  
    System.out.println("Hi!");  
}
```

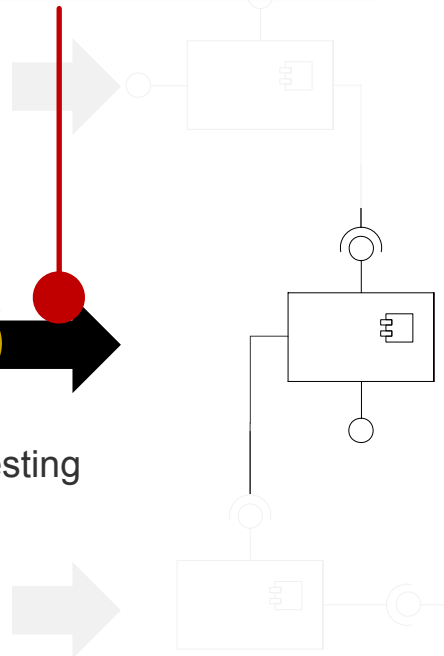


Implementation

Build

Functional Testing

Performance Testing



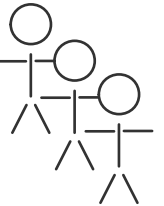
Pipeline automation

Fast & frequent releases

```
void  
int  
// do something  
bar(foo);  
System.out.println("Hi");
```

```
void main(String[] args) {  
    int foo;  
    // do something  
    bar(foo);  
    System.out.println("Hi");  
}
```

```
void main(String[] args) {  
    int foo;  
    // do something  
    bar(foo);  
    System.out.println("Hi");  
}
```

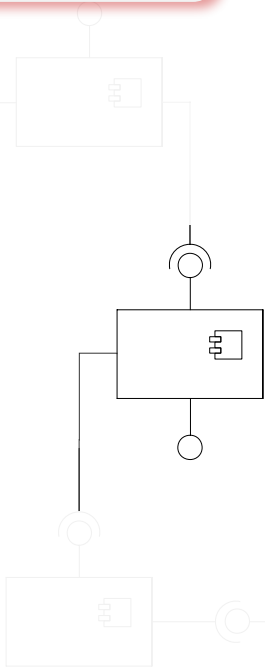


Implementation

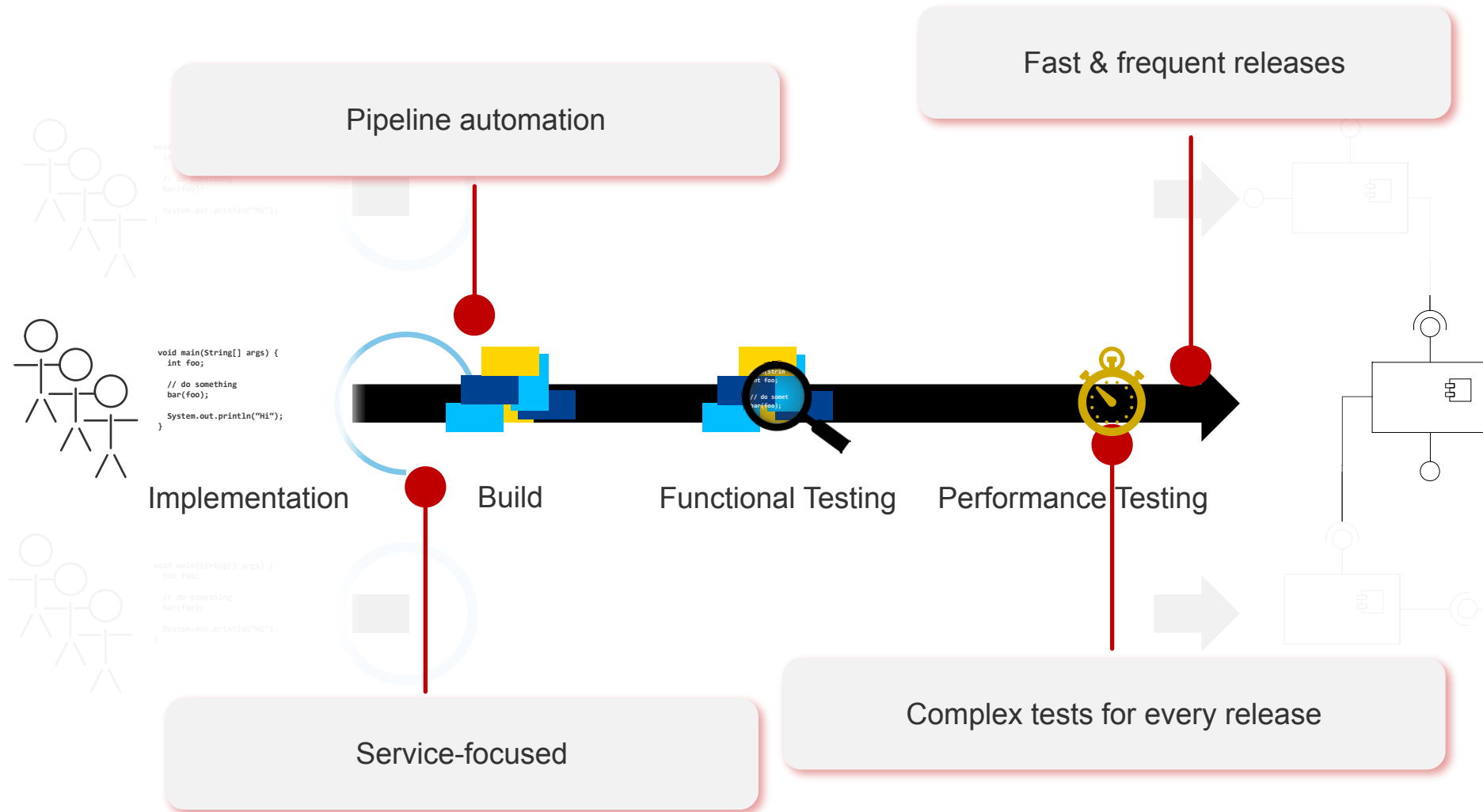
Build

Functional Testing

Performance Testing

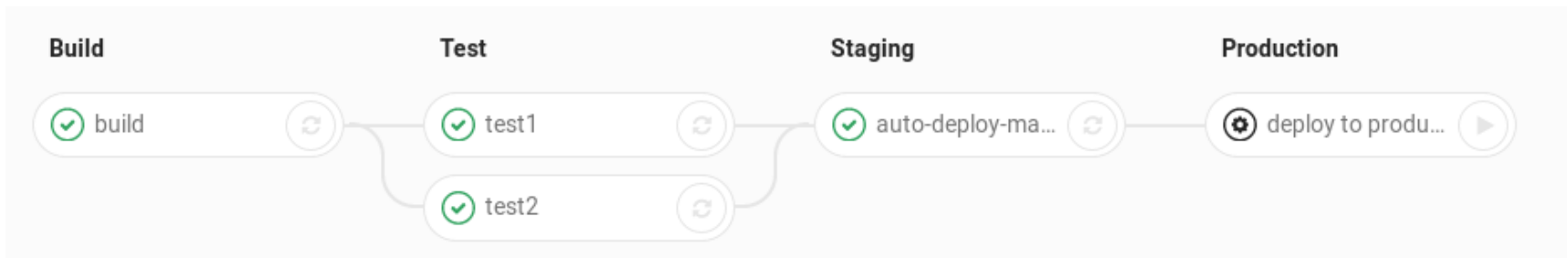


Service-focused



Simple pipeline example

- Imagine a pipeline consisting of four stages, executed in the following order:
 - build, with a job called build
 - test, with two jobs called test1 and test2.
 - staging, with a job called deploy-to-stage.
 - production, with a job called deploy-to-production



Pipeline and testing

- About two thirds of the overall build time is spent on testing (ICSE2021)

Maven

Tools and Techniques for Software Testing - Barbara Russo
SwSE - Software and Systems Engineering group

Build

- **Build** is the process of converting code files into deployable software

Maven Build Lifecycle - phases

- **validate**: validate the project is correct and all necessary information is available
- **compile**: compile the source code of the project
- **test**: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed, **test-compile** Compile test but not execute
- **package**: take the compiled code and package it in its distributable format, such as a JAR
- **verify**: run any checks on results of integration tests to ensure quality criteria are met
- **install**: install the package into the local repository, for use as a dependency in other projects locally
- **deploy**: done in the build environment, copies the final package to the remote repository for sharing with other developers and projects
- **clean**: remove all files generated by the previous build

A lifecycle phase

- A **phase** is responsible for a specific step in the lifecycle
- The manner in which it carries out those responsibilities may vary
- To specify a phase

```
mvn <goal>
```

- or for plugin

```
mvn <plugin-group-id>:<plugin-artifact-id>[:<plugin-version>]:<goal>
```

- example

```
mvn org.openclover:clover-maven-plugin:check
```

Build Goals

- *A phase is made up of goals*
- A goal represents a specific **task** (finer than a build phase) which contributes to the building and managing of a project
- This is done by declaring the goals bound to those build phases

Example

Goal of the plug-in

phase

```
mvn compile org.openclover:clover-maven-plugin:clover clean
```


in Eclipse

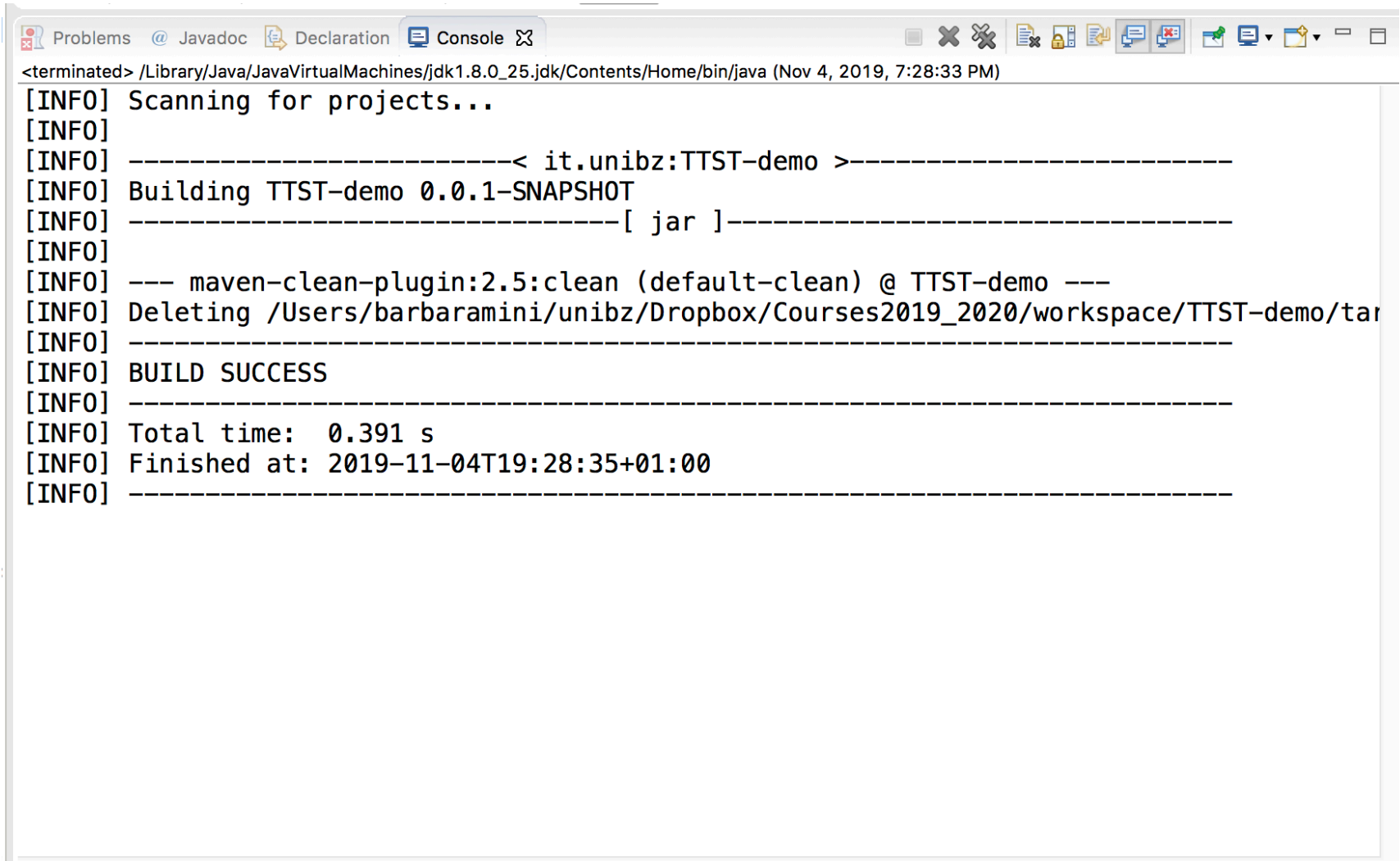
The screenshot shows the Eclipse IDE interface. The top bar includes the Eclipse logo, the title 'Eclipse', and system tray icons for network, volume, and battery, along with the date and time 'Mon 3:05 PM'. Below the title bar, the window title is 'Run Configurations'. The main area is titled 'Create, manage, and run configurations' and features a green play button icon.

The left sidebar displays a project tree with a search filter 'type filter text'. The tree is expanded to show the 'JUnit' category, which contains numerous test classes. The 'm2 Maven Build' category is also expanded, showing several build configurations, with 'SimpleTest' selected and highlighted.

The main configuration area for 'SimpleTest' is visible. It includes tabs for 'Main', 'JRE', 'Refresh', 'Source', 'Environment', and 'Common'. The 'Base directory' is set to '\$\${workspace_loc}/TTST-demo'. The 'Goals' field contains 'compile org.openclover:clover-maven-plugin:clover clean'. The 'User settings' field is set to '/Users/barbaramini/m2/settings.xml'. There are checkboxes for 'Offline', 'Update Snapshots', 'Debug Output', 'Skip Tests', 'Non-recursive', and 'Resolve Workspace artifacts'. The 'Threads' field is set to '1'. A table for 'Parameter Name' and 'Value' is empty. The 'Maven Runtime' is set to 'EMBEDDED (3.6.1/1.12.0.20190529-1915)'. At the bottom, there are 'Revert' and 'Apply' buttons.

At the very bottom of the screen, there are 'Close' and 'Run' buttons.

in Eclipse



```
<terminated> /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java (Nov 4, 2019, 7:28:33 PM)
[INFO] Scanning for projects...
[INFO]
[INFO] -----< it.unibz:TTST-demo >-----
[INFO] Building TTST-demo 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ TTST-demo ---
[INFO] Deleting /Users/barbaramini/unibz/Dropbox/Courses2019_2020/workspace/TTST-demo/target
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.391 s
[INFO] Finished at: 2019-11-04T19:28:35+01:00
[INFO] -----
```

Note

- If you do not know a goal of a plugin try a term, the console will list all the available ones!

Exercise

- Write in any of the phases we have seen a goal for a plugin of your choice

Goals

- It may be bound to zero or more build phases

```
mvn compile org.openclover:clover-maven-plugin:clover clean
```

- The order of execution depends on the order in which the goals and the build phases are invoked
- A goal not bound to any build phase could be executed outside of the build lifecycle by direct invocation

```
mvn org.openclover:clover-maven-plugin:clover
```

Profiles

- Profiles are a natural way of addressing the problem of different build configuration requirements for different target environments

```
mvn groupId:artifactId:goal -P profile-1,profile-2
```

```
mvn test -P unit
```

in Eclipse

The screenshot displays the Eclipse IDE interface. At the top, the title bar reads 'Eclipse' and 'Run Configurations'. Below the title bar, a subtitle says 'Create, manage, and run configurations'. The main window is divided into several panes:

- Left Pane (Project Explorer):** Shows a tree view of projects and test classes. The 'JUnit' folder is expanded, listing various test classes like 'AppUnitTest', 'BAUnitTest', etc. The 'Maven Build' folder is also expanded, showing configurations like 'cleanBuildConfiguration' and 'OneClassMethodTest'.
- Right Pane (Run Configuration Editor):** Shows the configuration for 'OneClassMethodTest'.
 - Name:** OneClassMethodTest
 - Base directory:** `${workspace_loc:/TTST-demo}`
 - Goals:** `clean -Dtest=FirstExampleUnitTest test`
 - Profiles:** `unit`
 - User settings:** `/Users/barbaramini/.m2/settings.xml`
 - Options:** Includes checkboxes for 'Offline', 'Update Snapshots', 'Debug Output', 'Skip Tests', 'Non-recursive', and 'Resolve Workspace artifacts'. A 'Threads' spinner is set to 1.
 - Parameter Table:** A table with columns 'Parameter Name' and 'Value'. It is currently empty.
 - Maven Runtime:** `EMBEDDED (3.6.1/1.12.0.20190529-1915)`

Setting a test environment

```
mvn -Denv=test integration-test
```

- Available lifecycle phases for this environment are:
- validate, initialize, generate-sources, process-sources, generate-resources, process-resources, compile, process-classes, generate-test-sources, process-test-sources, generate-test-resources, process-test-resources, test-compile, process-test-classes, **test**, prepare-package, package, pre-integration-test, **integration-test**, post-integration-test, verify, install, deploy, pre-clean, **clean**, post-clean, pre-site, site, post-site, site-deploy

Running all Unit Tests - one way

The screenshot shows the Eclipse IDE's 'Run Configurations' dialog for a project named 'TTST-demo'. The 'Goals' field is populated with '-Dtest=**/*UnitTest test', which is a Maven command to run all unit tests. The 'Maven Runtime' is set to 'EMBEDDED (3.6.1/1.12.0.20190529-1915)'. The 'Threads' field is set to 1. The 'Parameter Name' and 'Value' table is empty. The 'Launch Group' is set to 'm2 Maven Build'. The 'Maven Build' configuration is set to 'cleanBuildConfiguration'.

Name: TTST-demo

Base directory: \${workspace_loc:/TTST-demo}

Goals: -Dtest=**/*UnitTest test

Profiles:

User settings: /Users/barbaramini/.m2/settings.xml

Offline Update Snapshots

Debug Output Skip Tests Non-recursive

Resolve Workspace artifacts

Threads: 1

Parameter Name	Value
----------------	-------

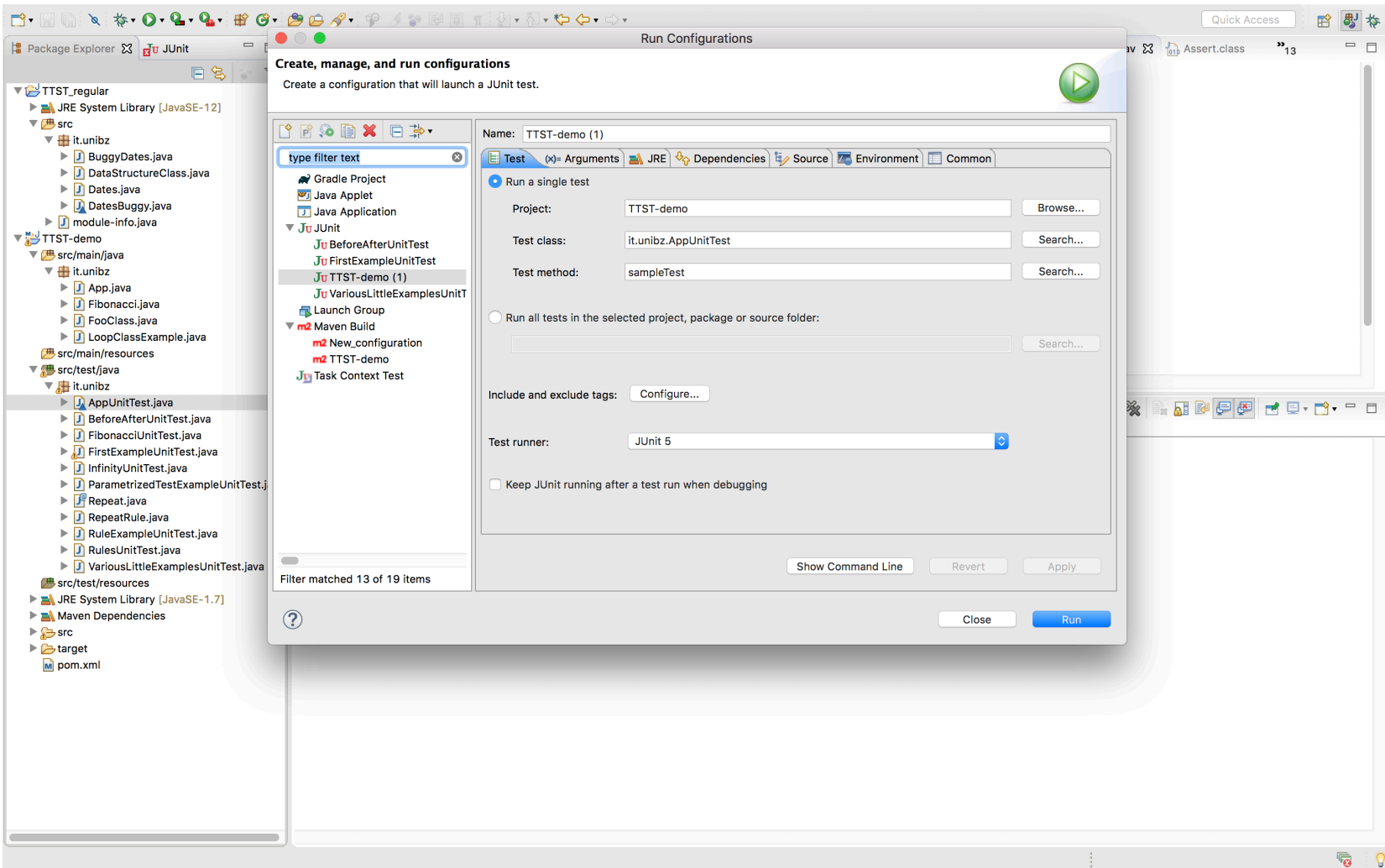
Maven Runtime: EMBEDDED (3.6.1/1.12.0.20190529-1915)

Launch Group: m2 Maven Build

Maven Build: cleanBuildConfiguration

Filter matched 62 of 68 items

Run a single test class or method - JUnit



Run it with Maven

The screenshot displays the Eclipse IDE interface. On the left, the Package Explorer shows the project structure for 'TTST_demo', including source files and test files. The 'Run Configurations' dialog is open, showing the configuration for the 'OneClassMethodTest' goal. The 'Goals' field is set to '-Dtest=AppUnitTest#sampleTest test'. The 'User settings' field is set to '/Users/barbaramini/.m2/settings.xml'. The console output at the bottom shows the following text:

```
<terminated> OneClassMethodTest [Maven Build] /Library/Java/JavaVirtualMachines/jdk-12.0.1.jdk/Contents/Home/bin/java (Oct 14, 2019, 12:04:35 PM)
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running it.unibz.AppUnitTest
[ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.026 s <<< FAILURE! - in it.unibz.AppUnitTest
[ERROR] sampleTest Time elapsed: 0.02 s <<< FAILURE!
java.lang.AssertionError: sample
    at it.unibz.AppUnitTest.sampleTest(AppUnitTest.java:19)
```

How to build pipelines in Maven

Tools and Techniques for Software Testing - Barbara Russo

SwSE - Software and Systems Engineering group

Project Object Model

- POM stands for *Project Object Model*
- It is an XML representation of a Maven project held in a file named pom.xml

POM - design

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0"
2.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4.     http://maven.apache.org/xsd/maven-4.0.0.xsd">
5.   <modelVersion>4.0.0</modelVersion>
6.
7.   <!-- The Basics -->
8.   <!-- Mandatory -->
9.   <groupId>...</groupId>
10.  <artifactId>...</artifactId>
11.  <version>...</version>
12.
13.  <scope>...</scope>
14.  <dependencies>...</dependencies>
15.  <properties>...</properties>
16.
17.  <!-- Build Settings -->
18.  <build>...</build>
19.  <reporting>...</reporting>
20.
21.  <!-- Environment Settings -->
22.  <profiles>...</profiles>
23. </project>
```

Example - library in dependencies

```
<dependency>  
  <groupId>org.junit.vintage</groupId>  
  <artifactId>junit-vintage-engine</artifactId>  
  <version>5.5.2</version>  
  <scope>test</scope>  
</dependency>
```

POM profiles

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
5
6     <modelVersion>4.0.0</modelVersion>
7     <groupId>it.unibz</groupId>
8     <artifactId>TTST-demo</artifactId>
9     <version>0.0.1-SNAPSHOT</version>
10
11 <dependencies>
35
36 <properties>
40
41 <build>
67
68 <profiles>
105
106
107
```


Scope

It refers to the classpath of the task at hand (compiling and runtime, testing, etc.)

We will use:

- **compile** - the default scope, used if none is specified. Compile dependencies are available in all classpaths.
- **runtime** - the dependency is not required for compilation, but only for execution
- **test** - the dependency is only available for the test compilation and execution phases

Build

- Project build

```
<build>...</build>
```

- Profile build

```
<profiles>
  <profile>
    <id>test</id>
    ...
    <build>
      <!-- "Profile Build" contains a subset of "Project Build"s elements -->
      ...
    </build>
  </profile>
</profiles>
```

Build - example 'install'



workspace

```
<build>  
  <defaultGoal>install</defaultGoal>  
  <directory>${basedir}/target</directory>  
  <finalName>${artifactId}-${version}</finalName>  
</build>
```

Build - example 'plugins'

```
<plugins>  
  <plugin>  
    <groupId>org.apache.maven.plugins</groupId>  
    <artifactId>maven-jar-plugin</artifactId>  
    <version>2.6</version>  
  </plugin>  
</plugins>
```

Automatically activate profiles

```
1. <settings>
2.   ...
3.   <activeProfiles>
4.     <activeProfile>profile-1</activeProfile>
5.   </activeProfiles>
6.   ...
7. </settings>
```

Surefire plugin to test

- If you want your JUnit 5 test cases to be executed with maven build, you will have to configure maven-surefire-plugin with junit-platform-surefire-provider dependencies

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.22.1</version>
  <configuration>
    <excludes>
      <exclude>**/*IntegrationTest.java</exclude>
    </excludes>
  </configuration>
</plugin>
```



wildcards

Maven Surefire Plugin

- To execute unit tests
- It generates reports in two different file formats:
 - Plain text files (*.txt)
 - XML files (*.xml)
- By default, these files are generated in target/surefire-reports folder
 - `${basedir}/target/.../TEST-*.xml`
- The Surefire Plugin can be invoked by calling the test phase

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left lists various Java files, with a red circle highlighting the 'surefire-reports' folder containing 'it.unibz.FirstExampleUnitTest.txt' and 'TEST-it.unibz.FirstExampleUnitTest.xml'. The main editor displays the code for 'InfinityUnitTest.java':

```
1 package it.unibz;  
2 import org.junit.Test;  
3  
4 public class InfinityUnitTest {  
5     @Test(timeout=100)  
6     // @Test  
7     public void infinity() {  
8         while(true);  
9     }  
10 }  
11
```

The Console window at the bottom shows the following output:

```
<terminated> SimpleTest [Maven Build] /Library/Java/JavaVirtualMachines/jdk-12.0.1.jdk/Contents/Home/bin/java (Nov 11, 2019, 3:23:07 PM)  
[INFO] Total time: 1.089 s  
[INFO] Finished at: 2019-11-11T15:23:10+01:00  
[INFO] -----
```

Name	Date Modified	Size	Kind
it.unibz.FibonacciUnitTest.txt	Today at 3:14 PM	297 bytes	Plain
it.unibz.FirstExampleUnitTest.txt	Today at 3:14 PM	762 bytes	Plain
it.unibz.Foo1UnitTest.txt	Today at 3:14 PM	683 bytes	Plain
it.unibz.FooUnitTest.txt	Today at 3:14 PM	1 KB	Plain
it.unibz.InfinityUnitTest.txt	Today at 3:14 PM	511 bytes	Plain
it.unibz.MyFirstClassUnitTest.txt	Today at 3:14 PM	513 bytes	Plain
it.unibz.MySecondClassUnitTest.txt	Today at 3:14 PM	305 bytes	Plain
it.unibz.ParametrizedTestExampleUnitTest.txt	Today at 3:14 PM	330 bytes	Plain
it.unibz.RuleExceptionsUnitTest.txt	Today at 3:14 PM	5 KB	Plain
it.unibz.RulesUnitTest.txt	Today at 3:14 PM	292 bytes	Plain
it.unibz.VariousLittleExamplesUnitTest.txt	Today at 3:14 PM	325 bytes	Plain
TEST-it.unibz.FibonacciUnitTest.xml	Today at 3:14 PM	15 KB	XML
TEST-it.unibz.FirstExampleUnitTest.xml	Today at 3:14 PM	16 KB	XML
TEST-it.unibz.Foo1UnitTest.xml	Today at 3:14 PM	16 KB	XML
TEST-it.unibz.FooUnitTest.xml	Today at 3:14 PM	17 KB	XML
TEST-it.unibz.InfinityUnitTest.xml	Today at 3:14 PM	15 KB	XML
TEST-it.unibz.MyFirstClassUnitTest.xml	Today at 3:14 PM	15 KB	XML
TEST-it.unibz.MySecondClassUnitTest.xml	Today at 3:14 PM	15 KB	XML
TEST-it.unibz.ParametrizedTestExampleUnitTest.xml	Today at 3:14 PM	16 KB	XML
TEST-it.unibz.RuleExceptionsUnitTest.xml	Today at 3:14 PM	25 KB	XML
TEST-it.unibz.RulesUnitTest.xml	Today at 3:14 PM	15 KB	XML
TEST-it.unibz.VariousLittleExamplesUnitTest.xml	Today at 3:14 PM	15 KB	XML

ReportDettaglioRiepilogoConsumiPrepagato-3.xlsx Nov 5, 2019 at 2:49 PM 66 KB Microsoft Excel workbook Nov 5, 2019 at 2:49 PM

Add a profile - UnitTest

```
69  <profile>
70    <id>unit</id>
71    <build>
72      <plugins>
73        <plugin>
74          <groupId>org.apache.maven.plugins</groupId>
75          <artifactId>maven-surefire-plugin</artifactId>
76          <configuration>
77            <excludes>
78              <exclude>**/*IntegrationTest.java</exclude>
79              <exclude>**/*RegressionTest.java</exclude>
80            </excludes>
81          </configuration>
82        </plugin>
83      </plugins>
84    </build>
85  </profile>
```

Naming convention

- UnitTest
- IntegrationTest
- RegressionTest

Time to watch a video

- <https://www.youtube.com/watch?v=R2ok6mKU0TI>
-

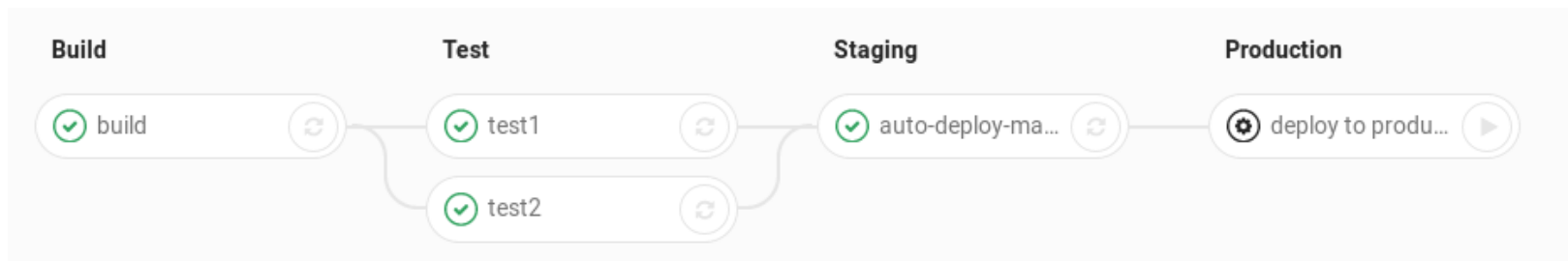
Pipelines in shared environments

- We have seen that we can implement pipelines in Maven
- **Exercise** run Maven to test your project
 - Import the example of POM file from ole
 - Customize and run for your project
 - Refactor some of the classes changing the name with postfix IntegrationTest
 - Re-run test with Maven excluding Unit Tests
 - Report any issue

Pipelines in gitlab

Pipelines in gitlab

- A pipeline is a **group of job/tasks** that get executed in *stages* also called *batches*
- All jobs in stages run sequentially/parallel based on the runners available



Pipeline configuration

- The au of automation pipeline is performed with a *yaml file* for the configuration and a *runner* for the execution

YAML - Yet Another Markup Language

- YAML is a data serialization language that is utilized to create configuration files and works with any programming language
- It's a superset of JSON
- It can do everything that JSON can and more
- Newlines and indentation mean something in YAML (not in JSON)

Pipelines' configuration



- Pipelines' major components: jobs and stages
- Pipelines are configured using a YAML file called **.gitlab-ci.yml** within each project
 - Jobs and stages are defined in the `.gitlab-ci.yml` file for each project
 - <https://docs.gitlab.com/ee/ci/yaml/README.html>

Stages (careful not for environments!)

- “stages” term is used to define stages that can be used by jobs
- Stages allow multiple pipelines
- The ordering of elements in stages defines the ordering of jobs’ execution:
 - Jobs of the same stage are run in parallel
 - Jobs of the next stage are run after the jobs from the previous stage complete successfully

.gitlab-ci.yml

- The .gitlab-ci.yml file defines the structure and order of the pipelines and determines:
 - What to execute using GitLab Runner.
 - What decisions to make when specific conditions are encountered
 - For example, when a process succeeds or fails

Typical stages

- Build (default if no stages are defined in yml)
- Test (default if no stages are defined in yml)
(assigned to a job when no stage specified for a job)
- Deploy (default if no stages are defined in yml)
- Review
- Dynamic Application Security Testing (DAST)
- Staging
- Canary

Jobs

- Jobs are defined with constraints stating under what conditions they should be executed
- They must contain at least the script clause

```
job1:  
  script: test.sh
```

```
job2:  
  script: mvm install
```

Jobs

- Not limited in how many can be defined
- If all the jobs in a stage
 - Succeed, the pipeline moves on to the next stage
 - Fail, the next stage is not (usually) executed and the pipeline ends early

Job duration

- Job A(1,3), B(2,4) and C(6,7)



- Total duration: $4-1 + 7-6 = 4$

Examples

You can use all sorts of scalar types as values: numbers, booleans, and strings (quoted or not).

```
[ version: 2
```

If you have a list of things (like images), you can denote that sequence using dashes.

```
docker:  
- image: ubuntu:14.04  
- image: mongo:2.6.8  
  command: [mongod, --smallfile]  
- image: postgres:9.4.1
```

Key

Value

```
image: maven:latest
```

```
variables:
```

```
  MAVEN_CLI_OPTS: "-s ./Applications/maven/conf/settings.xml --batch-mode"
```

```
cache:
```

```
  paths:
```

```
    - target/
```

```
stages:
```

- build
- test
- doc
- deploy
- trigger



Globally defined

```
build:
```

```
  stage: build
```

```
  script:
```

```
    - mvn $MAVEN_CLI_OPTS compile
```



shell commands

```
test:
```

```
  stage: test
```

```
  script:
```

```
    - mvn $MAVEN_CLI_OPTS test
```

```
  artifacts:
```

```
  paths:
```

```
    - target/
```

Create the yml file and push it

Push `.gitlab-ci.yml` to GitLab

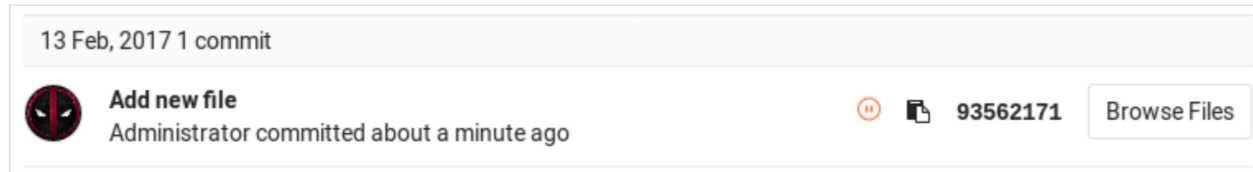
After you've created a `.gitlab-ci.yml`, you should add it to your Git repository and push it to GitLab.

```
git add .gitlab-ci.yml  
git commit -m "Add .gitlab-ci.yml"  
git push origin master
```

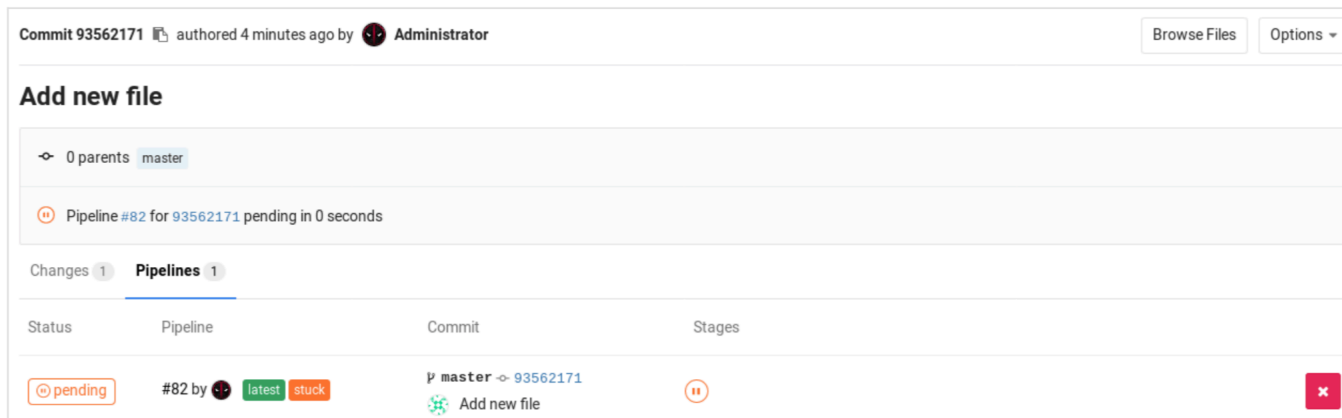
Now if you go to the **Pipelines** page you will see that the pipeline is pending.

Note: If you have a [mirrored repository where GitLab pulls from](#), you may need to enable pipeline triggering in your project's **Settings > Repository > Pull from a remote repository > Trigger pipelines for mirror updates**.

You can also go to the **Commits** page and notice the little pause icon next to the commit SHA.



Clicking on it you will be directed to the jobs page for that specific commit.



Notice that there is a pending job which is named after what we wrote in `.gitlab-ci.yml`. "stuck" indicates that there is no runner configured yet for this job.

Runner

- The next step is to configure a runner so that it picks the pending jobs

Runner

- An agent written in Go
- It runs jobs in stages
- After installation, Runners **must be registered**
 - as *shared* (multiple projects with same requirements) or *specific* (individual project)
- gitlab has a default Runner, but you can create yours

GitLab Runner

- GitLab Runner is used to **run jobs** and **send the results back to GitLab**
- Runners run the code defined in **.gitlab-ci.yml**
- Jobs are executed within the environment of the Runner
- Multiple jobs in the same stage are executed by Runners in parallel, if there are enough concurrent Runners

Configuring a runner

In GitLab, runners run the jobs that you define in `.gitlab-ci.yml`. A runner can be a virtual machine, a VPS, a bare-metal machine, a Docker container, or even a cluster of containers. GitLab and the runner communicate through an API, so the only requirement is that the runner's machine has network access to the GitLab server.

A runner can be specific to a certain project or serve multiple projects in GitLab. If it serves all projects, it's called a *shared runner*.

Find more information about runners in the [runner](#) documentation.

The official runner supported by GitLab is written in Go. View [the documentation](#).

For a runner to be available in GitLab, you must:

1. [Install GitLab Runner](#).
2. [Register a runner for your group or project](#).



When a runner is available, you can view it by clicking **Settings > CI/CD** and expanding **Runners**.

- T ToolsTechniquesST
- Project
- Repository
- Issues 1
- Merge Requests 0
- CI / CD
- Operations
- Packages
- Snippets
- Settings
 - General
 - Members
 - Integrations
 - Repository
 - CI / CD
 - Operations
- Collapse sidebar

Specific Runners

Set up a specific Runner automatically

You can easily install a Runner on a Kubernetes cluster. [Learn more about Kubernetes](#)

1. Click the button below to begin the install process by navigating to the Kubernetes page
2. Select an existing Kubernetes cluster or create a new one
3. From the Kubernetes cluster details view, install Runner from the applications list

Install Runner on Kubernetes

Set up a specific Runner manually

1. Install GitLab Runner
2. Specify the following URL during the Runner setup: `https://gitlab.inf.unibz.it/`
3. Use the following registration token during setup: `h36TyiYQXSBzGsErWGs3`

Reset runners registration token

4. Start the Runner!

Variables ?

Shared Runners

GitLab Shared Runners execute code of different projects on the same Runner unless you configure GitLab Runner Autoscale with MaxBuilds 1 (which it is on GitLab.com).

Disable shared Runners for this project

Available shared Runners: 1

afedc506 shared-docker #34

Group Runners

GitLab Group Runners can execute code for all the projects in this group. They can be managed using the Runners API.

This project does not belong to a group and can therefore not make use of group Runners.

Expand



Environment

Linux

macOS

Windows

Docker

Kubernetes

Architecture

amd64 ▾

Download and install binary

[Download latest binary](#)

```
# Download the binary for your system
sudo curl --output /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-darwin-amd64

# Give it permissions to execute
sudo chmod +x /usr/local/bin/gitlab-runner

# The rest of commands execute as the user who will run the Runner
# Register the Runner (steps below), then run
cd ~
gitlab-runner install
gitlab-runner start
```



Command to register runner

```
sudo gitlab-runner register --url https://gitlab.inf.unibz.it/ --registration-token $REGISTRATION_TOKEN
```



Close

Register the runner

- Registering a Runner is the process that binds the Runner with a GitLab instance
- Read more:
- <https://docs.gitlab.com/runner/register/index.html>

Start the Runner

- `cd ~`
- `gitlab-runner install`
- `gitlab-runner start`

Executing the pipeline

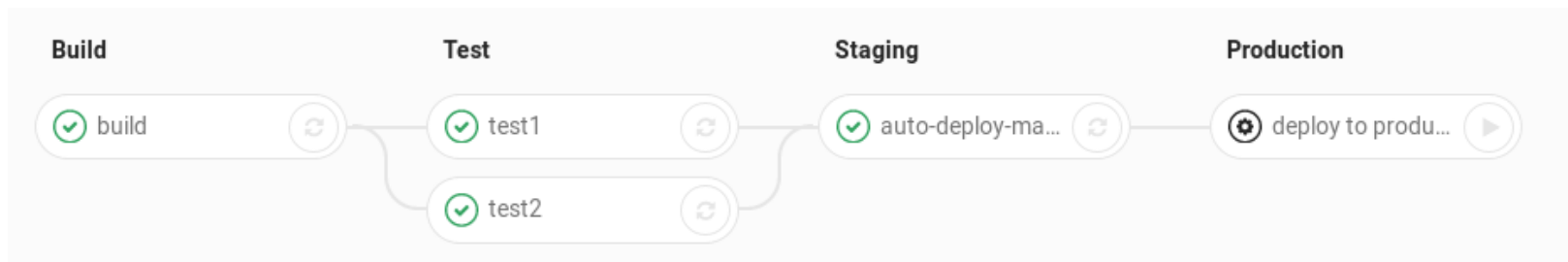
- When commit is pushed to the repository, GitLab will look for `.gitlab-ci.yml` from the root directory and trigger a build according to the settings configured
- GitLab Runner uses this file to manage project's jobs which defines how the project should be built

Lint

- Each instance of GitLab CI has an embedded debug tool called Lint, which validates the content of your `.gitlab-ci.yml`

Exercise

- Write the yml file for the simple pipeline example
- build, with a job called compile
- test, with two jobs called test1 and test2
- staging, with a job called deploy-to-stage
- production, with a job called deploy-to-prod



Pipelines in gitlab

- <https://gitlab.inf.unibz.it/help/ci/yaml/README.md>

Exercise

- create you first yml file

```
image: maven:latest
```

```
variables:
```

```
  MAVEN_CLI_OPTS: "-s .m2/settings.xml --batch-mode"
```

```
  MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository"
```

```
cache:
```

```
  paths:
```

```
    - .m2/repository/
```

```
    - target/
```

```
stages:
```

```
  - build
```

```
  - test
```

```
  - deploy
```

Issue trackers

Tools and Techniques for Software Testing - Barbara Russo
SwSE - Software and Systems Engineering group

How can we increase our ability to discover and fix bugs?

Tools and Techniques for Software Testing - Barbara Russo
SwSE - Software and Systems Engineering group

top 10 bugs

- <https://www.youtube.com/watch?v=AGI371ht1N8>

Big data shared over the Internet

- We can increase our ability to fight bugs by learning from data shared over the Internet
- Examples:
 - Q&A websites: Stack overflow
 - Code repositories: github
 - Issue trackers: JIRA, Bugzilla, Issuezilla, Google issue tracker

Issue Tracker

- A software package and a database that tracks and maintain lists of bugs and their fixes over time. Bug history:
 - Chats (comments) on bugs and how to fix them
 - Bug fixes ...
- Most of the software projects publicly share over the Internet the information stored in their issue trackers

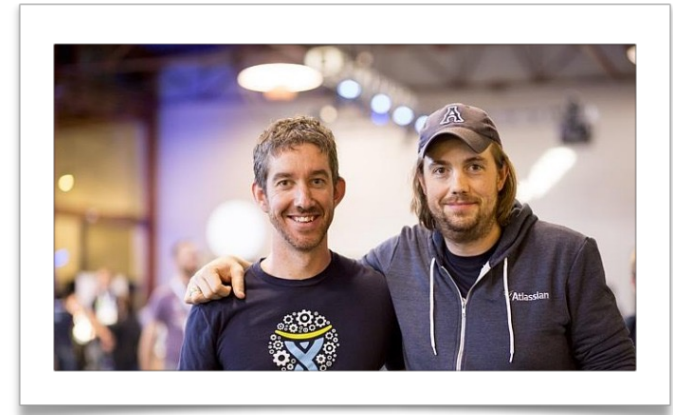
Features

- Entering of dysfunctions, errors and requests
- **Moderation**: distribution and assignment of issues
- Statistical analysis of the number of tickets
- Automatic generation of tickets by alarming systems
- Per issue **comments**
- Priority assignment
- **Report**: Detailed descriptions of the problem being experienced, attempted solutions or workarounds, and other relevant information. Reference to **commits**
- Maintaining of a **history of changes (file change set)**

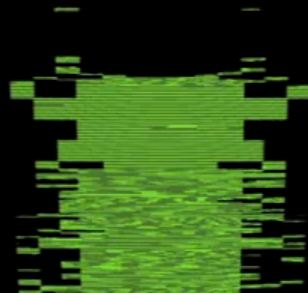
Atlassian



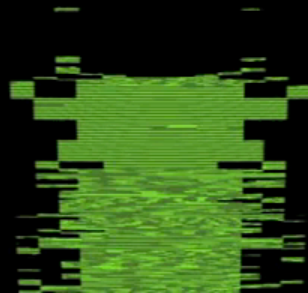
- Funded: 2002
- Founders: Scott Farquhar, Mike Cannon-Brookes
- Revenue: 619.9 million USD (July 2017)



Using bugs to catch bugs: JIRA



Using bugs to catch bugs: JIRA



FILTERS <<

New filter

Find filters

My Open Issues

Reported by Me

Recently Viewed

All Issues

FAVOURITE FILTERS

You must be logged in to view favourite filters.

Search

Save as

Export Tools

Project: All Bug, Defect Closed Assignee: All Contains text More Advanced

Order by

SWS-520
HTTP Accept header field contains invalid ty...

SWS-518
Echo sample application fails to build due to...

SWS-517
Soap response validation errors after updat...

SWS-514
SpringPlainTextPasswordValidationCallback...

SWS-509
Namespace prefix in attribute value not reso...

SWS-506
All-in-one jar contains different namespace i...

SWS-502
Soap envelope rpc-encoded namespace issue

SWS-501
Maven dependency for oxm 1.5.5.A non-exi...

SWS-500
remove amazonaws repository urls from spr...

SWS-497
JDOM exception when parsing odd SOAP m...

SWS-496
SpringWS client does not send Mtom attach...

SWS-494
AbstractHttpSenderConnection getErrorMes...

< 1 2 3 4 5 >

repository/org/springframework/ws/spring-ws-parent/1.5.5/spring-ws-parent-1.5.5.pom:
<url>http://s3.amazonaws.com/maven.springframework.org/external</url>
I think these are outdated and should be removed.or at least replaced by something that works.

Agile
[View on Board](#)

Activity

All **Comments** Work Log History Activity Transitions

- Arjen Poutsma added a comment - 13/May/09 9:26 PM
Quite frankly, I have no idea what's causing this. I can duplicate the warnings, but the build does not hang for me.
I think it might be due to Amazon S3 returning non-standard HTTP headers or bodies, which isn't something I can fix.
- Wouter de Vaal added a comment - 13/May/09 9:53 PM
My point was that the amazonaws repositories aren't used anywhere in Spring anymore, but for this parent pom. My suggestion is removing this repository, as it's not valid anymore
- Arjen Poutsma added a comment - 13/May/09 10:04 PM
You are right: other projects use maven.springframework.org/external or maven.springframework.org/release, but these resolve to the same Amazon S3 host/directory as the ones presently there. I will fix.
- Arjen Poutsma added a comment - 13/May/09 10:09 PM
I replaced the s3.amazonaws.com URLs with maven.springframework.org ones.
- Arjen Poutsma added a comment - 04/May/12 7:03 AM
Closing old issues



XNIO / XNIO-15

ConnectionString produces a warning when used in vararg lists

183 of 149308 [Return to search](#)

Agile Board [More Actions](#)

Views

Details

Type: Bug
Priority: Major
Affects Version/s: None
Component/s: None
Labels: None
Estimated Difficulty: Low
Similar Issues: [Show 10 results](#)

Status: Closed
[\(View Workflow\)](#)
Resolution: Done
Fix Version/s: [1.0.0.CR1](#)

People

Assignee: David Lloyd
Reporter: David Lloyd
 Vote (0) Watch (0)

Description

Need to come up with a cleaner solution here.

Dates

Created: 10/Jun/08 10:21 PM
Updated: 11/Jun/08 4:53 AM
Resolved: 11/Jun/08 4:53 AM

Activity

All [Comments](#) [Work Log](#) [History](#) [Activity](#) [Commits](#) [Source](#) [Reviews](#)

David Lloyd made changes - 10/Jun/08 10:21 PM

Field	Original Value	New Value
Fix Version/s		1.0.0.CR1 [12312410]

david.loyd@boss.com submitted changeset 34 to trunk in XNIO (2 files) - 11/Jun/08 4:53 AM

[XNIO-15](#) - fix lame warning by changing API

xnio-base/trunk/api/src/main/java/org/boss/xnio/oUtils.java (+10 -7)
 xnio-base/trunk/api/src/test/java/org/boss/xnio/test/oUtilsTestCase.java (+3 -2)

David Lloyd made changes - 11/Jun/08 4:53 AM

Status	Open [1]	Closed [6]
Resolution		Done [1]

Agile

[View on Board](#)

Exercise

- Enable issue tracker on your project
 - goto Settings>General and enable Issues
 - Create a set of labels for your project containing at least
 - Bug, feature, enhancement
 - Create the issue for Milestone ‘Unit tests are complete’
 - ‘Create unit test for method placeBid’
 - Assign it to one member

Exercise - unit test

- Select a project of another team
- Read the work done and create an issue with “bug” “feature” or “enhancement”