45426: Teste e Qualidade de Software

Continuous integration

Ilídio Oliveira

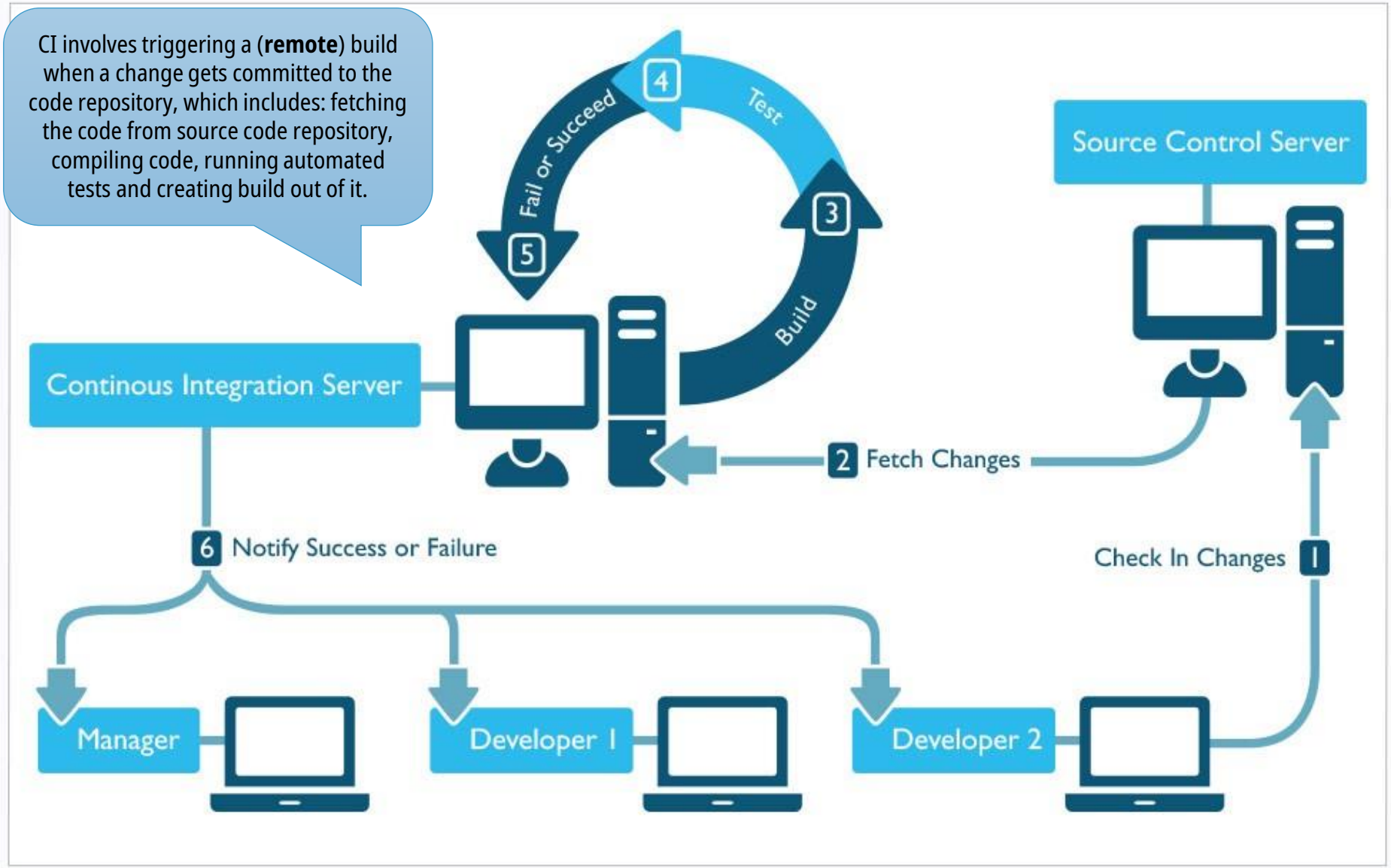
v2020-04-14

universidade de aveiro
departamento de eletrónica, telecomunicações e informática
deti

Learning objectives

- Explain the practices proposed by Fowler to implement a continuous integration system.
- Explain them meaning of “continuous” in the context of CI/CD
- Describe the development workflow when a team adopts CI/CD.
- How is the culture of continuous integration beyond the tools?
- Compare Continuous integration, Continuous Delivery and Continuous Deployment.
- Discuss the typical steps/stages in a CI/CD pipeline.

CI involves triggering a (**remote**) build when a change gets committed to the code repository, which includes: fetching the code from source code repository, compiling code, running automated tests and creating build out of it.



https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html

Continuous integration practices

- Developers commit to a shared repository regularly
- Changes in SCM are observed and trigger automatically builds
- Immediate feedback on build failure (broken builds have high-priority)
- Optional: deploy of artifacts into a reference repository
- Optional: trigger deployment for integration/acceptance tests

The most frequent the integration process is, the less painful

Continuously integrating the "units"

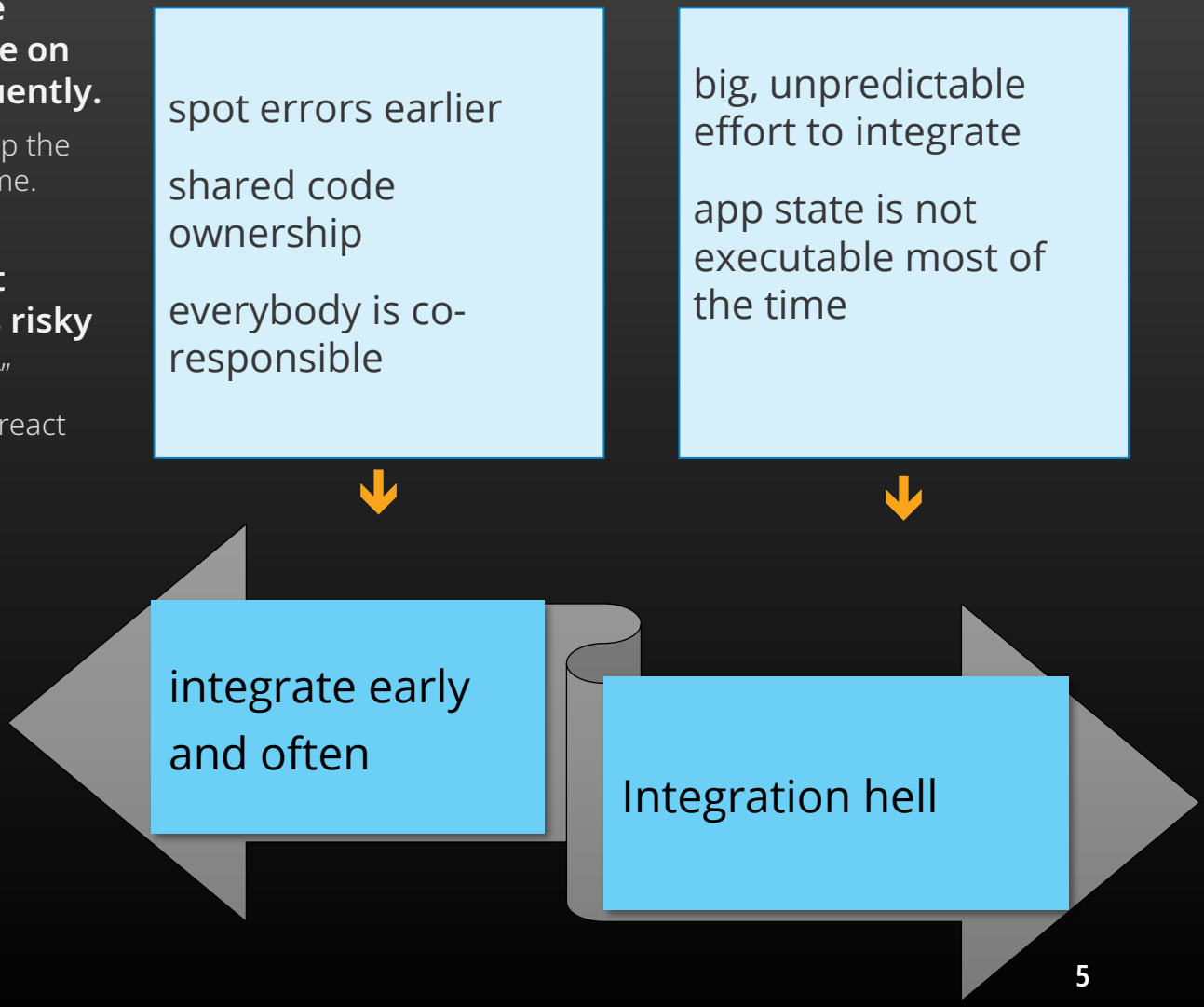
The essence of it lies in the simple practice of everyone on the team integrating frequently.

Feel comfortable and set up the tools to integrate at any time.

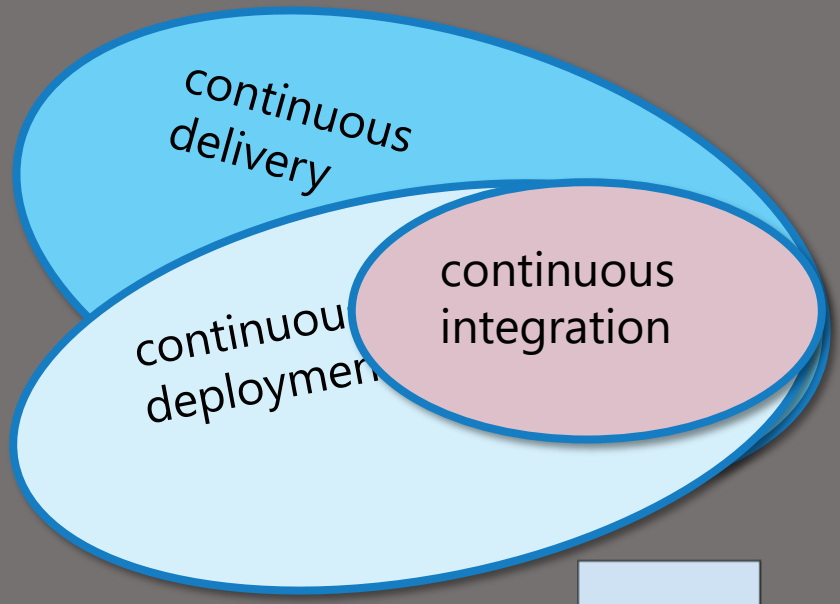
CI makes the development process smoother and less risky

↓ "it runs on my computer"

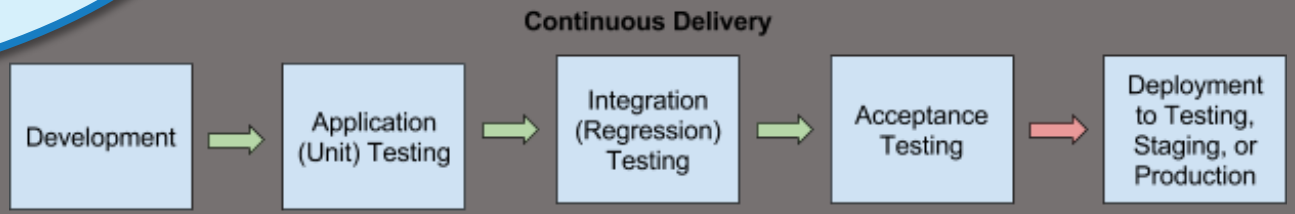
Early detection of failures (react quickly)

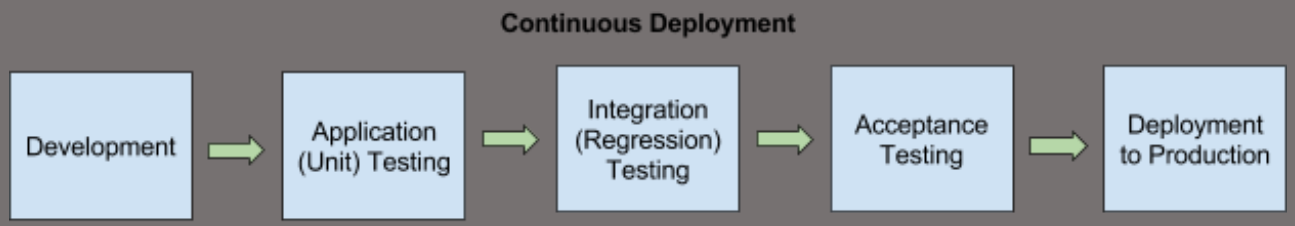


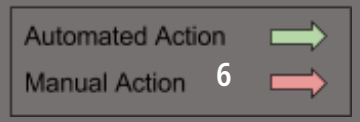
Related (yet different) **terms**



You can do frequent deployments but may choose not to do it (usually related to businesses strategy)







Continuous...

Continuous Delivery

sw development practice in which you build software in such a way that it can be released to production at any time.

You're doing continuous delivery when:

Focus on quality of working software

Your software is deployable throughout its lifecycle

Your **team prioritizes keeping the software deployable over working on new features**

Anybody can get fast, automated feedback on the production readiness

Continuous Deployment/release

every change goes through the pipeline and automatically gets put into production.

Focus on speed and agility to deploy to production

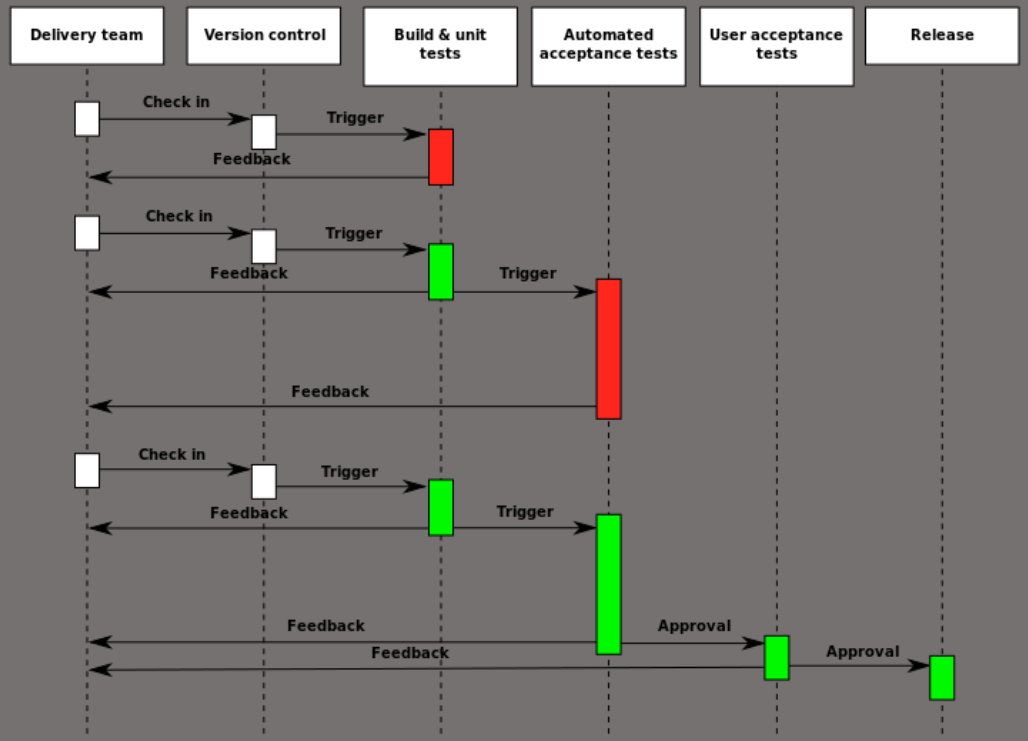
Continuous Integration

Automatically integrating, building, and testing code within the development environment.

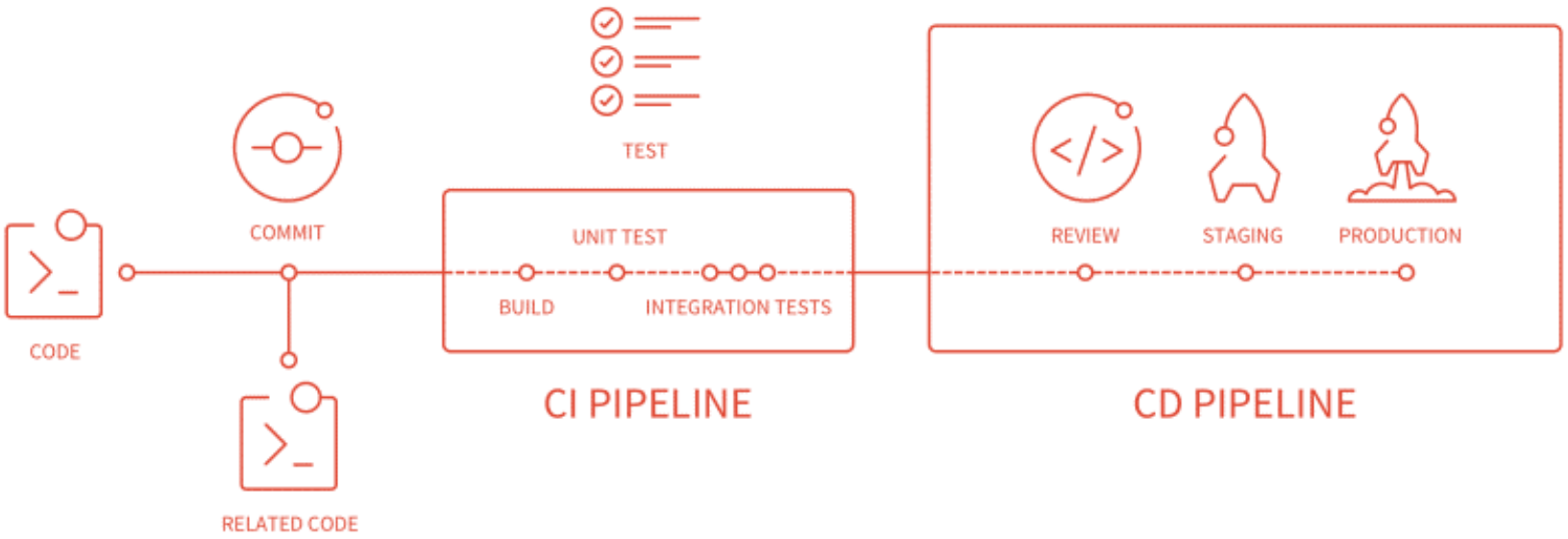
Pre-delivery steps.

Oliveira (2018)

Continuous delivery



https://about.gitlab.com



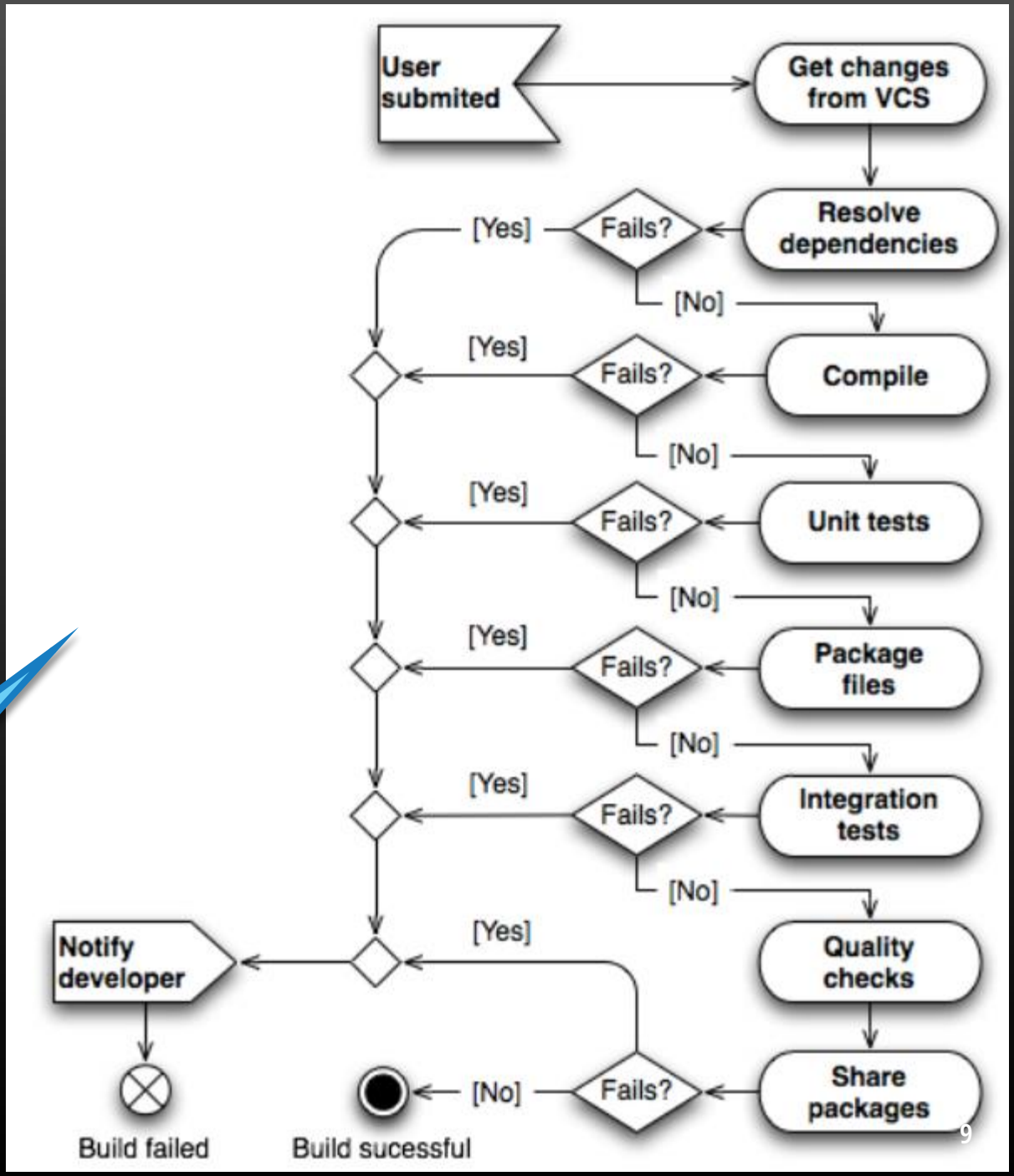
The build process

A build has several stages (goals in Maven terms).

A successful build implies success in code correctness and quality checks.

Automatic build tools run quality checks (e.g.: unit testing, code inspections)

Not just compiling...

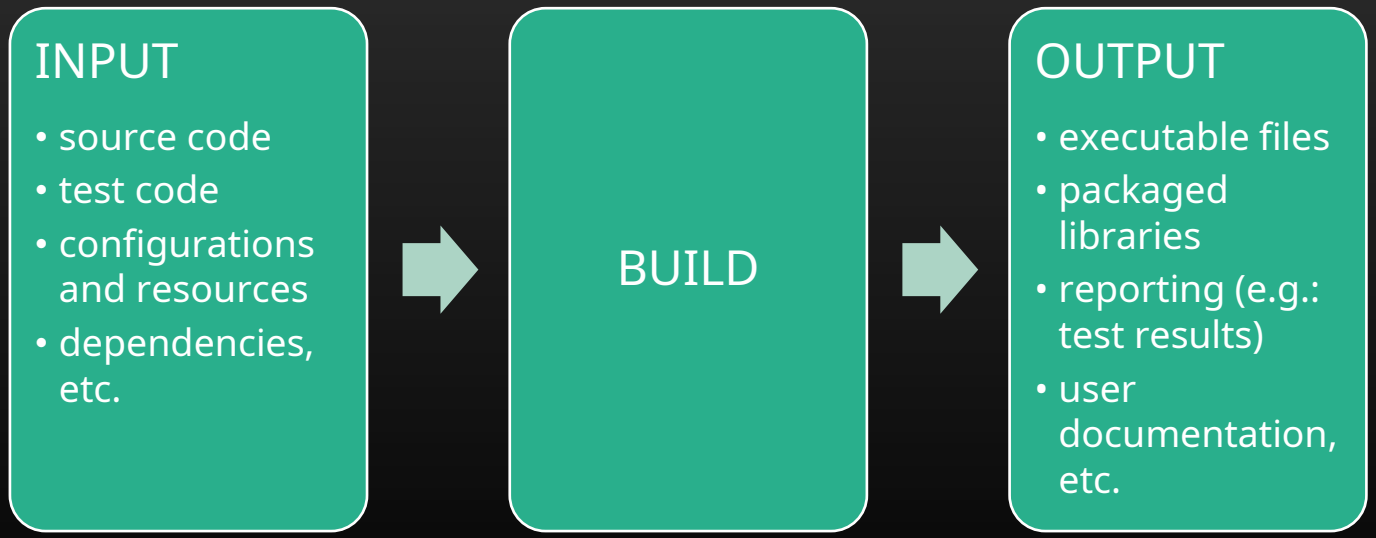


"Continuous" building: the build process

Build process is a series of steps that transforms the various project components in an application ready to be deployed

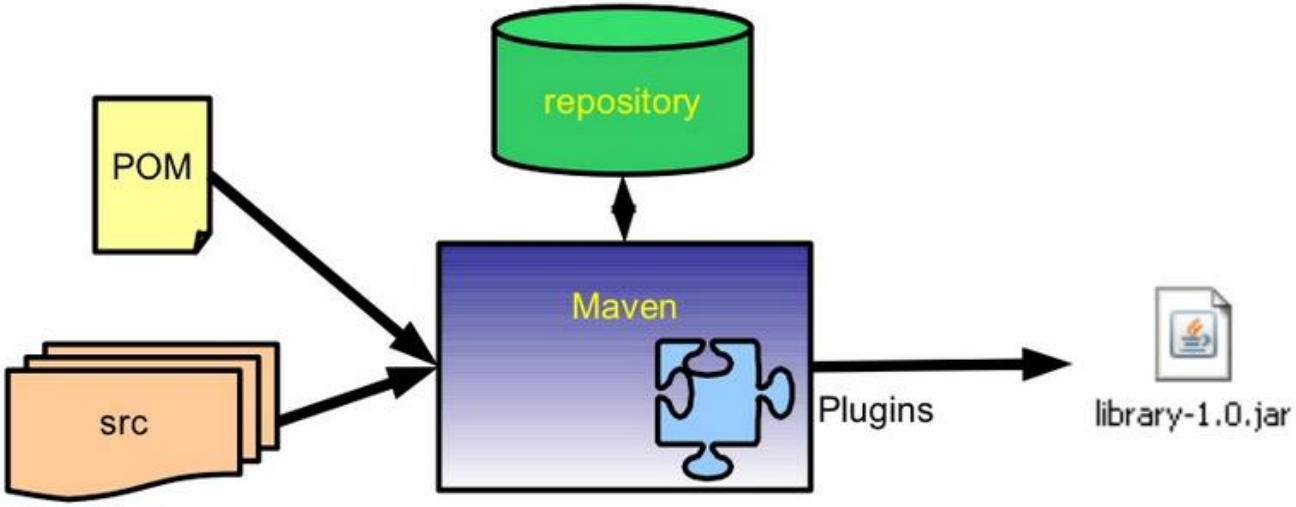
Build instructions are outlined in one or more description files

e.g.: POM.xml



Key component: build tool

- Maven is a modular automation system built around 4 main elements



  - input: project src/resources + POM
  - output: tested and packaged artifact

Carlo Bonamico - carlo.bonamico@gmail.com – JUG Genova

Maven lifecycle

- Default life cycle

```
validate
generate-sources
process-resources
compile
test-compile
test
package
integration-test
verify
install
deploy
```

  - (some skipped for clarity)

- Every goal implies all the previous ones

```
mvn compile
```

  - actually executes

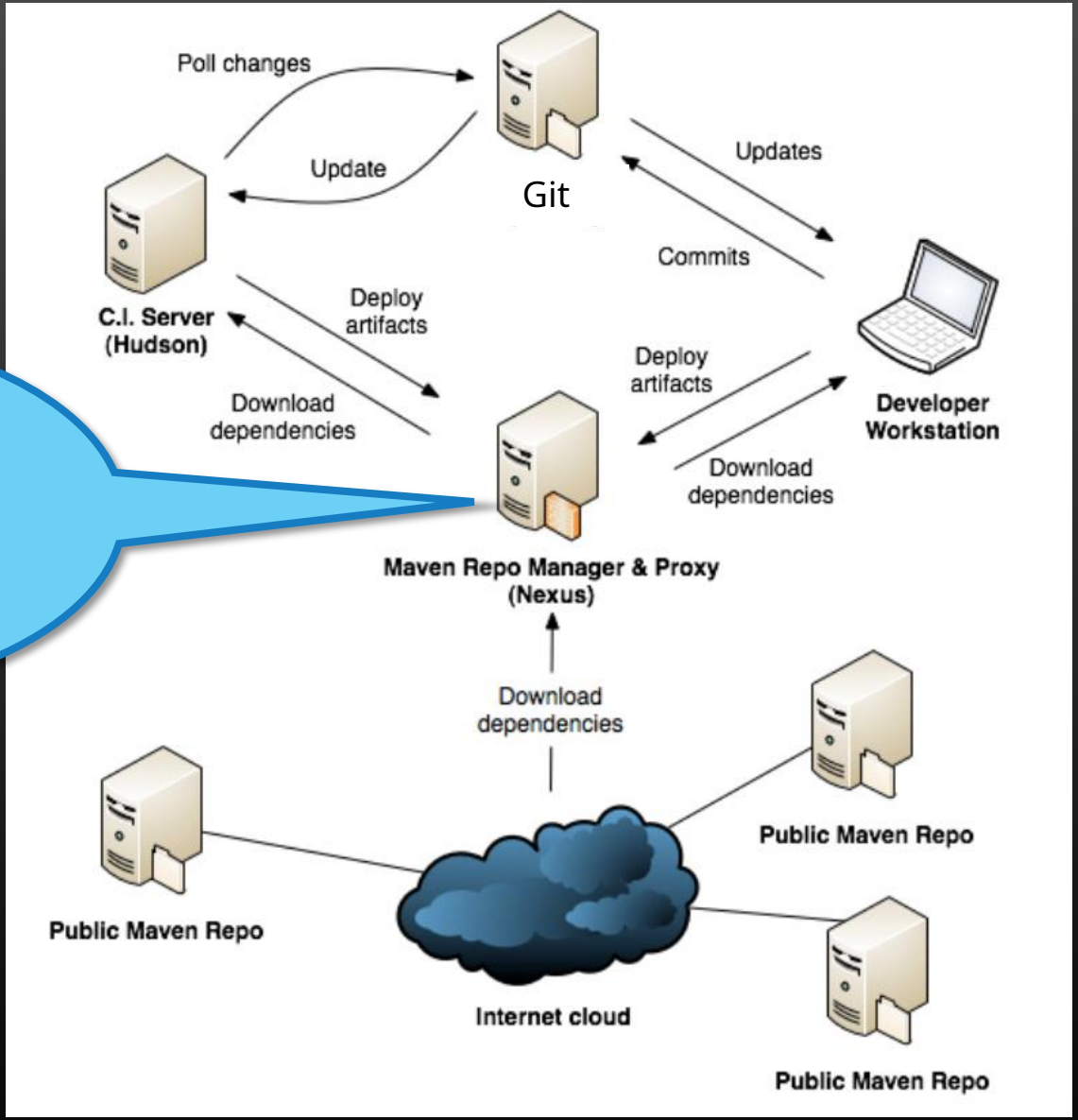
```
validate
generate-sources
process-resource
compile
```

- Stand-alone goals

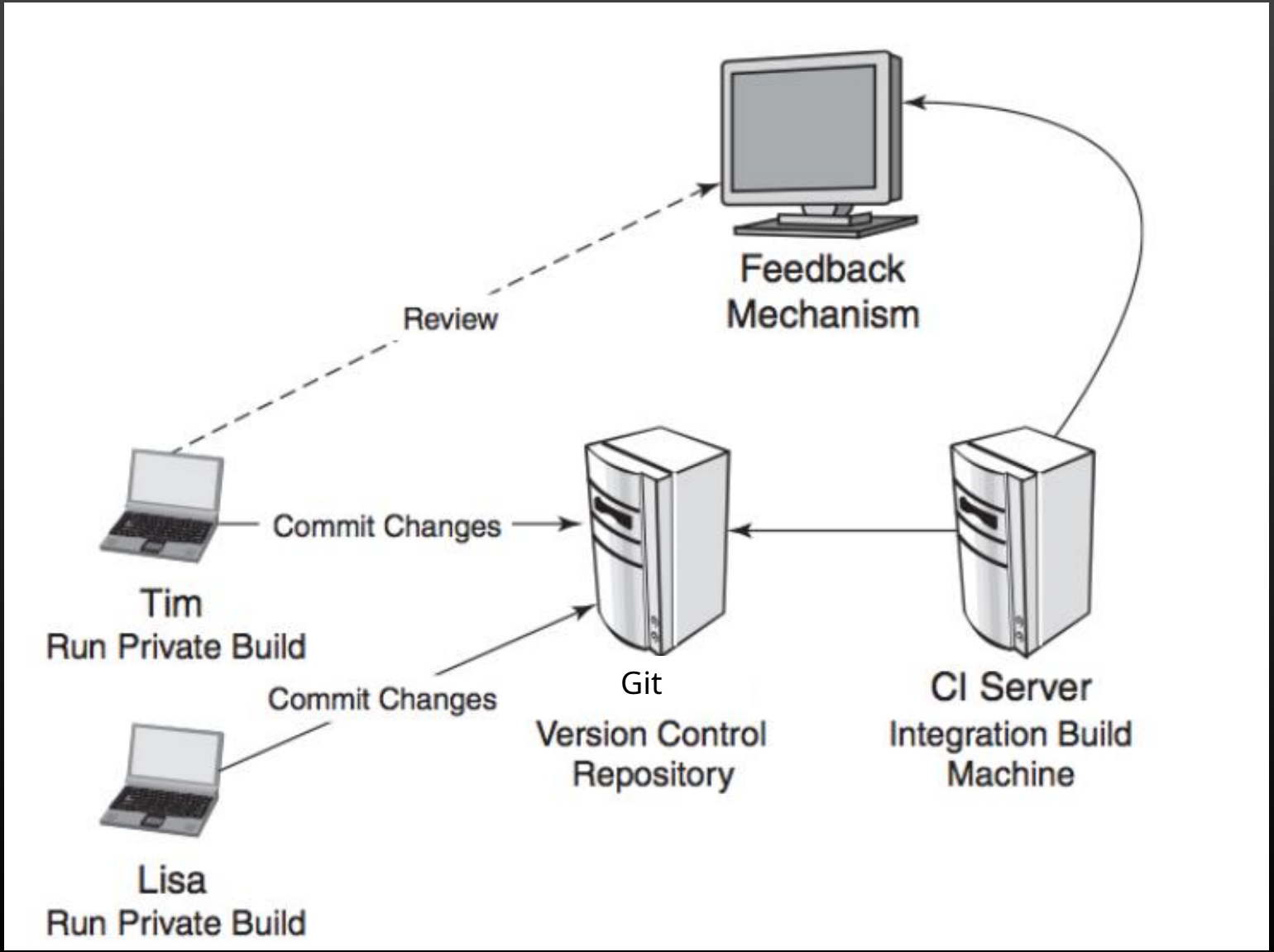
```
mvn scm:update
```

Role of (dependencies) repositories

Efficiently sharing and caching binaries
e.g.: JFrog Artifactory



Components of an integration system



Generic development workflow

1- Checkout (or update) from SCM

2- Code a new feature

3- Run automated build on local machine

Repeat #2 and #3 till tests pass

4- Merge local copy with latest changes from SCM

Fix and rebuild till tests pass

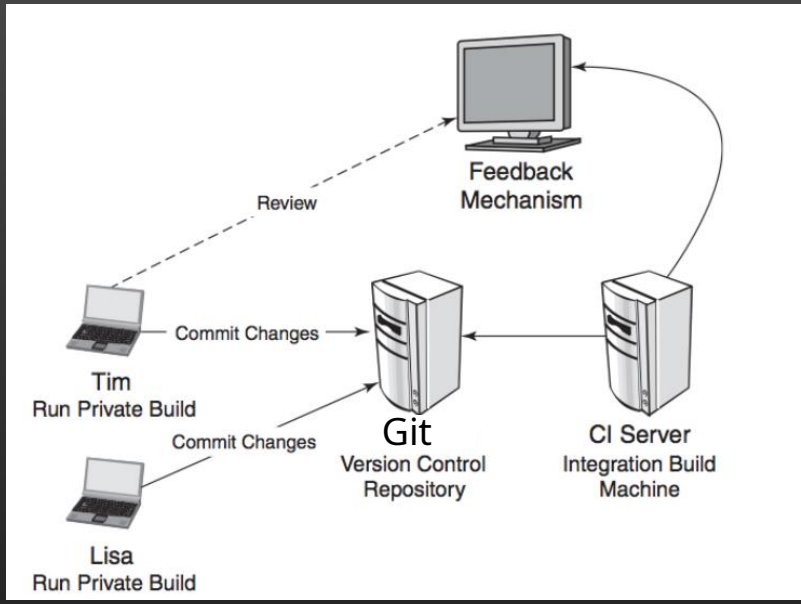
5- Commit (integrate with “central”)

6- Run a build on a clean machine

Update artifacts, evidence build status

Immediately fix bugs and integration issues

Not only tools: CI culture required!



[Oliveira (2018)]

Fowler's 10 CI practices

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

http://martinfowler.com/articles/continuousIntegration.html

CI culture

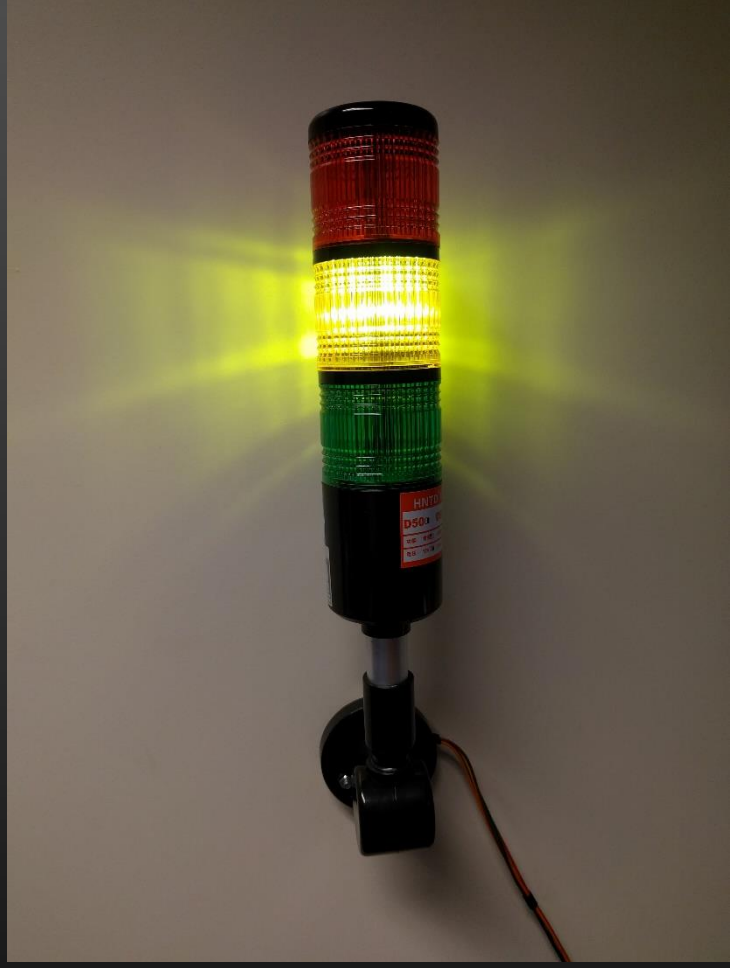
CI is a toolset and a mindset

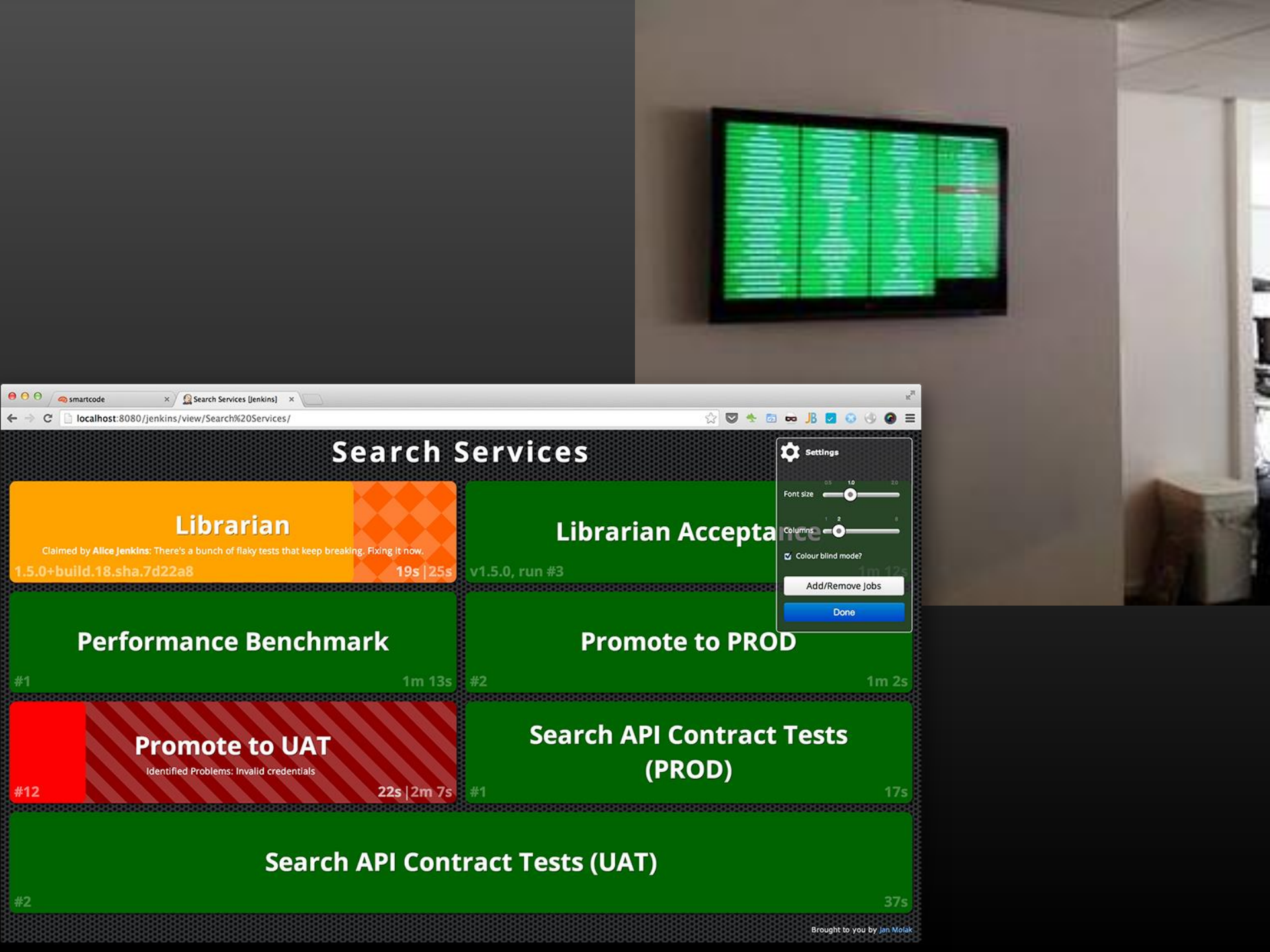
Broken builds are high-priority

No manual steps in the build process

Everybody must supply the CI with good tests

The most frequent the integration process is, the less painful




smartcode
Search Services [Jenkins]
localhost:8080/jenkins/view/Search%20Services/
Search Services
Settings
Font size
Columns
Colour blind mode?
Add/Remove Jobs
Done
Librarian
Claimed by Alice Jenkins: There's a bunch of flaky tests that keep breaking. Fixing it now.
1.5.0+build.18.sha.7d22a8
19s | 25s
Librarian Acceptance
v1.5.0, run #3
1m 12s
Performance Benchmark
#1
1m 13s
Promote to PROD
#2
1m 2s
Promote to UAT
Identified Problems: Invalid credentials
#12
22s | 2m 7s
Search API Contract Tests (PROD)
#1
17s
Search API Contract Tests (UAT)
#2
37s
Brought to you by Jan Molak

Continuous feedback

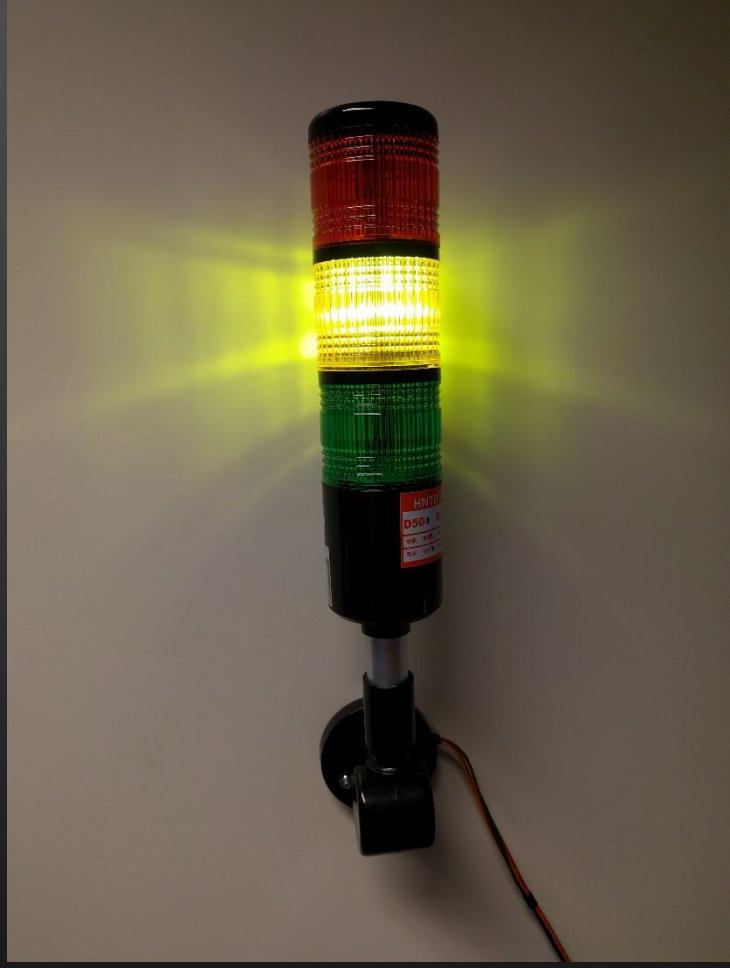
Errors are easier to detect in an earlier stage, near the point where they have been introduced:

The detection mechanism of such bugs becomes simpler because the natural step in diagnosing the problem is to check what was the latest submitted change.

problems followed by atomic commits are easiest to correct than to fix several problems at once, after bulk commits

There must be an effective mechanism that automatically informs programmers, testers, database administrators and managers about the status of the build

Feedback → generate reaction in a more accurate and prompter way



[Oliveira (2018)]

Continuous testing

Quality checks at all system levels and involve all individuals, not just the elements of the QA team

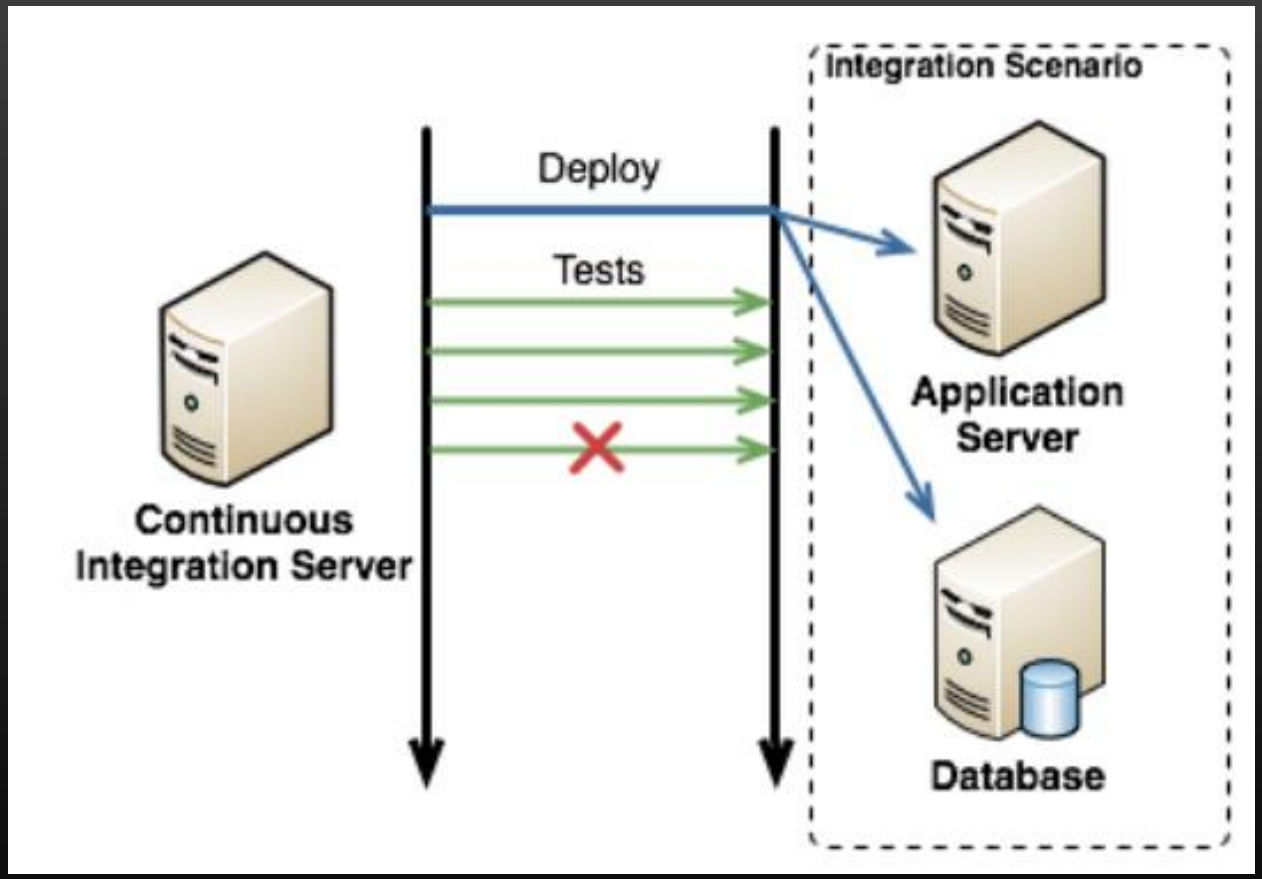
Most of the tests can be automated and should be run in the CI pipeline to be carried out repeatedly:

unit testing, integration testing, regression testing, system testing, load and performance testing, etc.

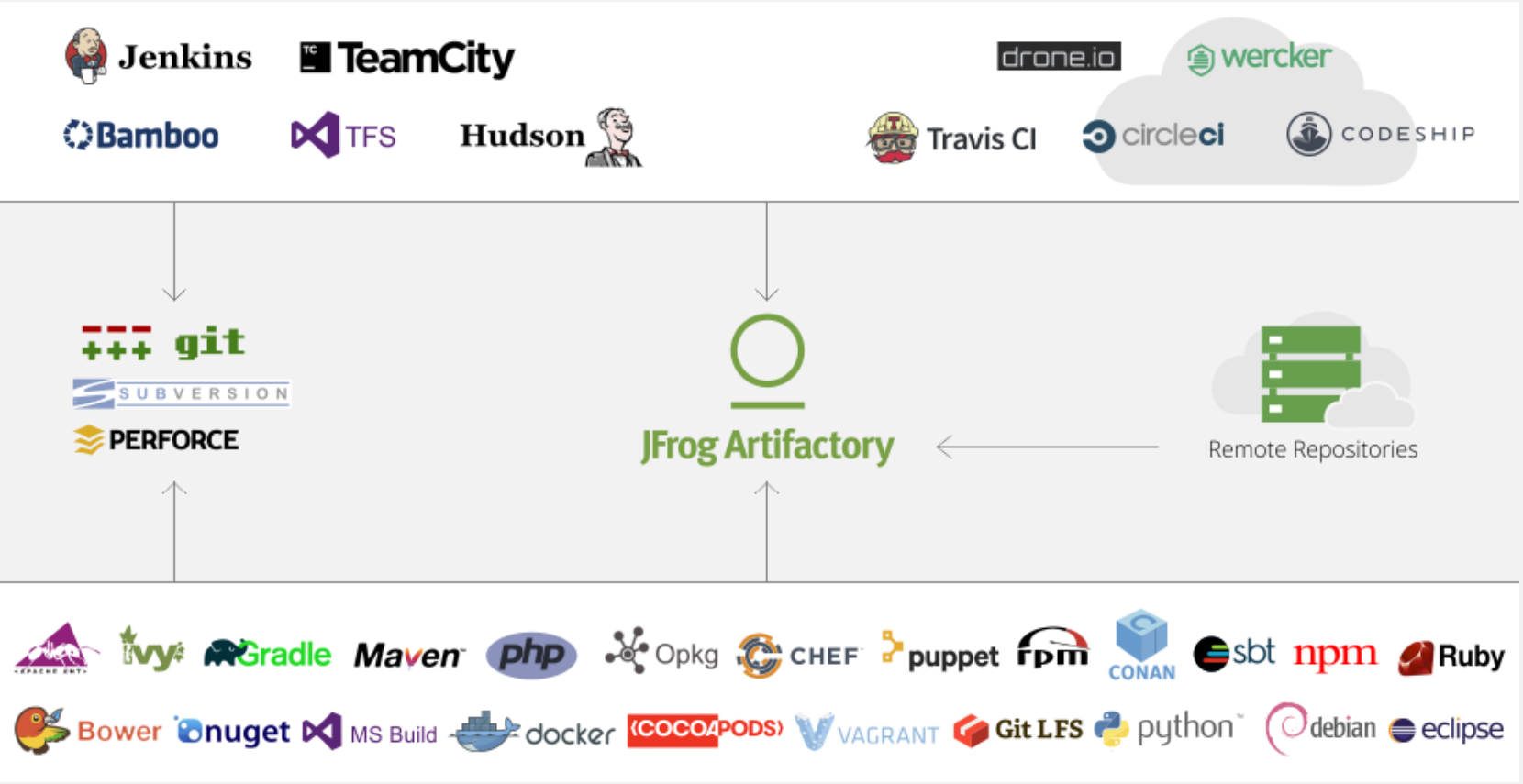
Build tools can take a crucial role on automating tests

[Oliveira (2018)]

Integration tests



Related technologies



https://www.jfrog.com/artifactory/

Jenkins

Easy of use and extremely extensible

Plugins-oriented

Hosted

vs cloud-centric

Distributed builds

Master/slaves architecture

Jenkins vocabulary

Job: a runnable task

Node: a master or slave machine

Build Executor: a stream of builds to run

Plugin: module that extends the core functionality.

Pipeline: definition of the steps to be executed

Jenkins Downloads Blog Documentation Plugins Use-cases Participate Sub-projects Resources Security Press Conduct

Fork me on

Jenkins supports building Java projects since its inception, and for a reason! It's both the language Jenkins is written in, plus the language in use by many if not all the projects Kohsuke Kawaguchi wanted to watch out when he created the tool many years ago.

If you want to build a Java project, there are a bunch of different options. The most typical ones nowadays are generally Apache Maven, or Gradle.

Apache Maven

In any FreeStyle job, as **currently** Maven is supported in standard, you can use the dedicated step. One advantage is, as for all Jenkins tools, that you can select a specific Maven version and have Jenkins automatically install it on the build node it's going to run on.

image::/images/solution-images/jenkins-maven-step.png

Gradle

As the associated plugin is not installed by default, first install the Gradle plugin. Once done, you should be able to add a Gradle step.

image::/images/solution-images/jenkins-gradle-step.png

Java plugins for Jenkins

JUnit plugin
publishes JUnit XML formatted test reports for trending and analysis

Gradle plugin
support invoking Gradle as a build step and listing executing tasks per build

Findbugs plugin
generate trending and analysis for FindBugs reports

PMD plugin
generate trending and analysis for PMD reports

Cobertura plugin
publish and trend code coverage reports from Cobertura

SonarQube plugin
integrate reporting from the SonarQube code quality/inspection platform

Repository Connector plugin
adds features for resolving artifacts from a Maven repository such as Nexus or Artifactory.

→ https://jenkins.io/solutions/java/

Pipeline as Code with Jenkins

The default interaction model with Jenkins, historically, has been very web UI driven, requiring users to manually create jobs, then manually fill in the details through a web browser. This requires additional effort to create and manage jobs to test and build multiple projects, it also keeps the configuration of a job to build/test/deploy separate from the actual code being built/tested /deployed. This prevents users from applying their existing CI/CD best practices to the job configurations themselves.

Pipeline

With the introduction of the Pipeline plugin, u
/deploy pipeline in a `Jenkinsfile` and store th
another piece of code checked into source co

Jenkins ♥ Continuous Delivery Articles

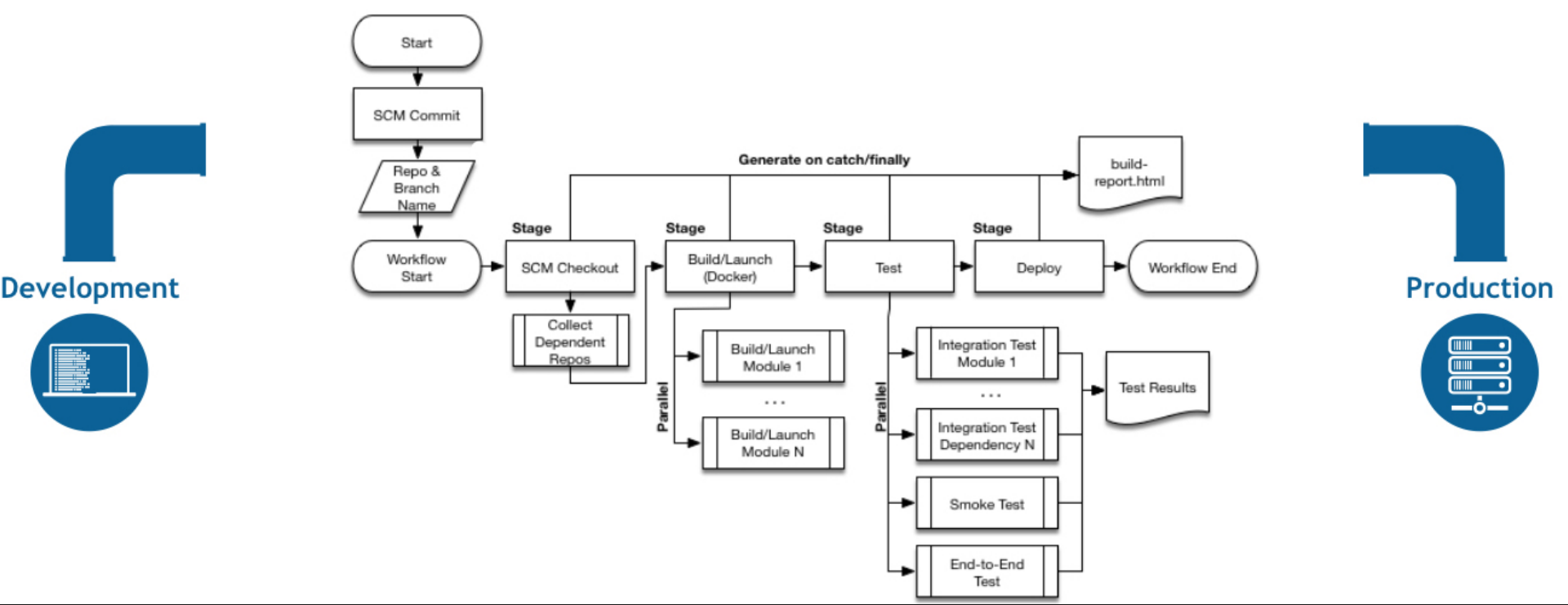
Multibranch Workflows in Jenkins
jenkins-ci.org

Continuous Delivery

fining
pipeline

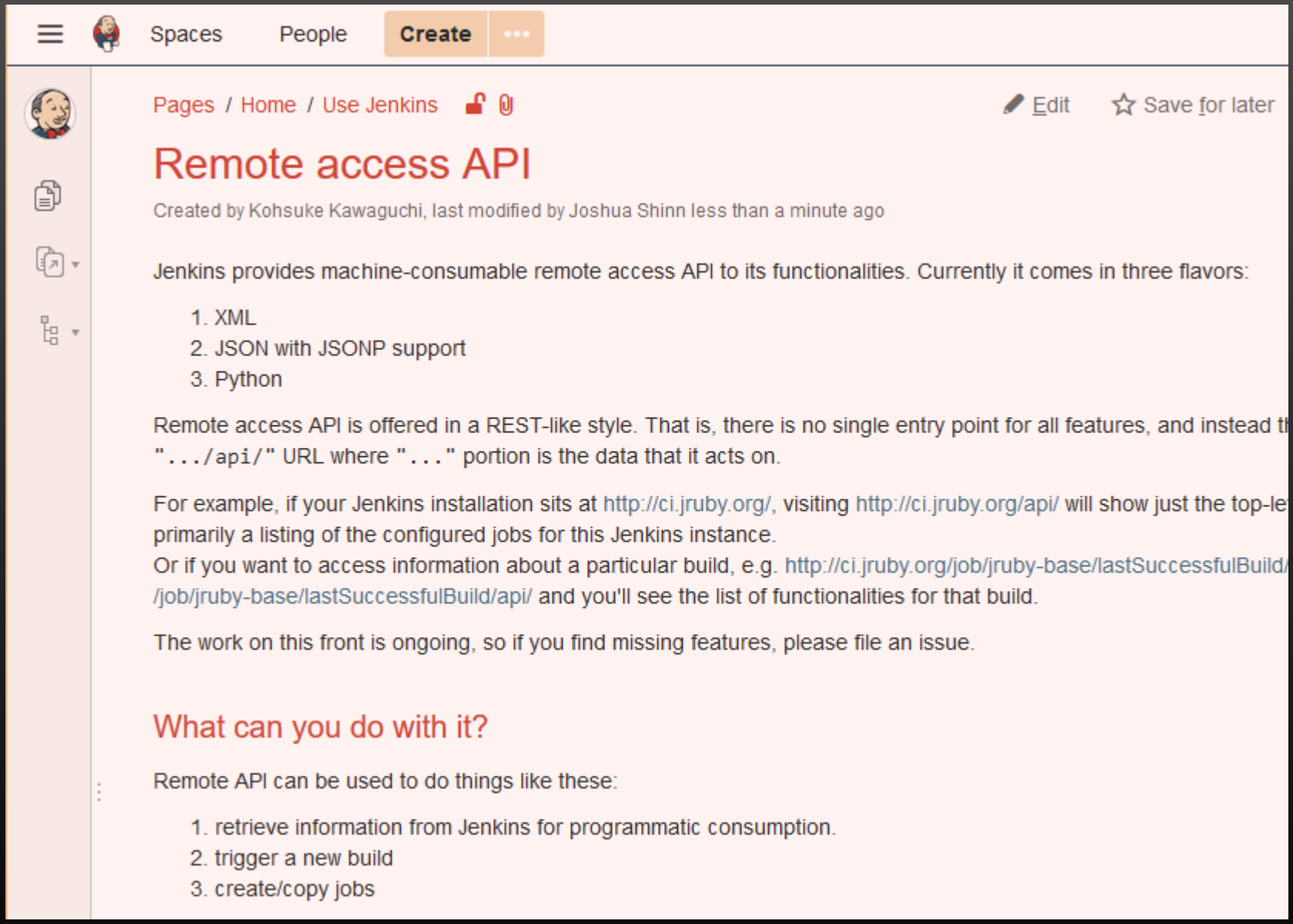
A *continuous delivery (CD) pipeline* is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline domain-specific language (DSL) syntax.

Jenkins pipelines



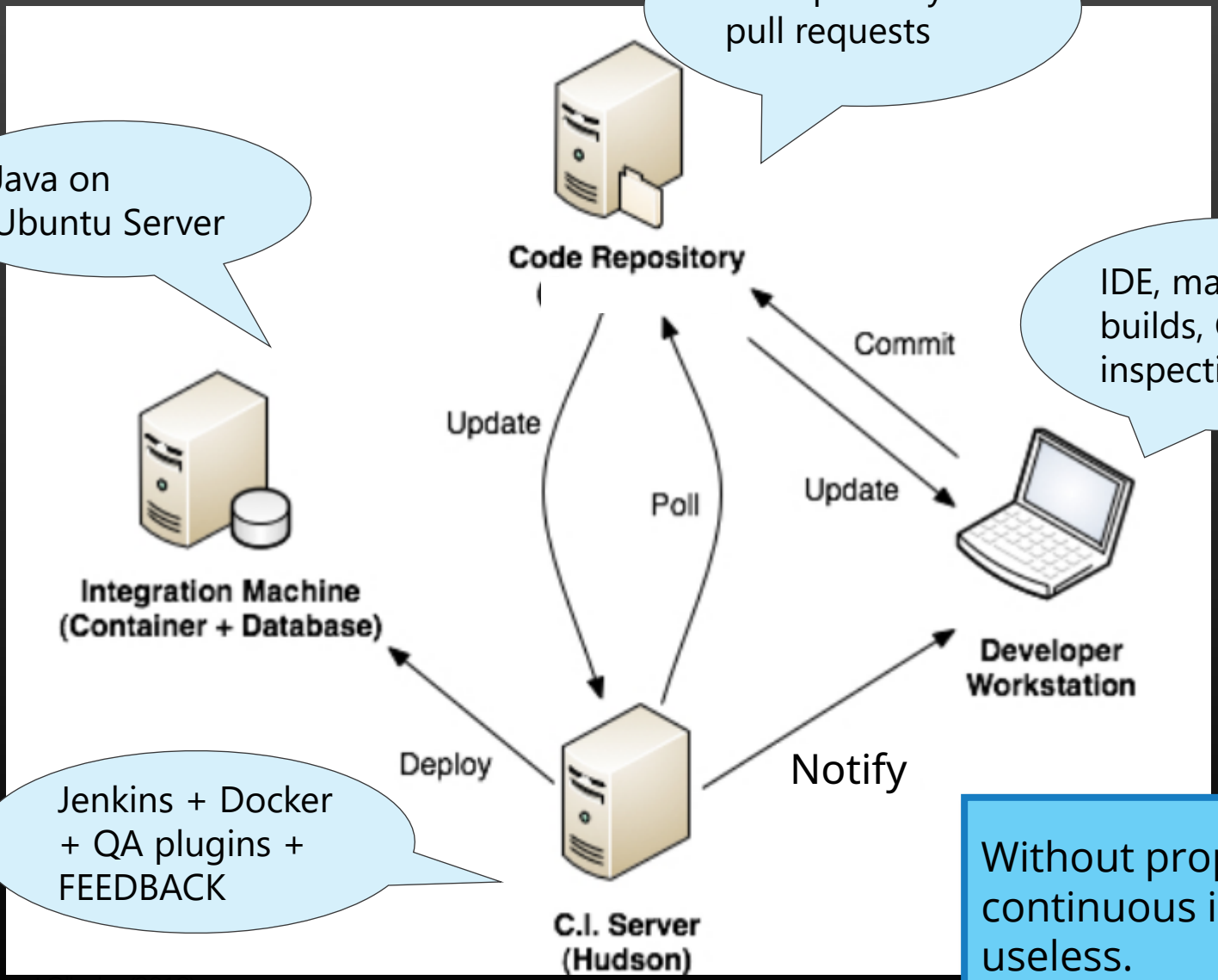
```
pipeline {
    agent {
        docker {
            image 'maven:3-alpine'
            args '-v /root/.m2:/root/.m2'
        }
    }
    options {
        skipStagesAfterUnstable()
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }
        stage('Deliver') { (1)
            steps {
                sh './jenkins/scripts/deliver.sh' (2)
            }
        }
    }
}
```

```
pipeline {
    agent any
    stages{
        stage('Build'){
            steps {
                sh 'mvn clean package'
            }
            post {
                success {
                    echo 'Now Archiving...'
                    archiveArtifacts artifacts: '**/target/*.war'
                }
            }
        }
        stage ('Deploy to Staging'){
            steps {
                build job: 'Deploy-to-staging'
            }
        }

        stage ('Deploy to Production'){
            steps{
                timeout(time:5, unit:'DAYS'){
                    input message:'Approve PRODUCTION Deployment?'
                }

                build job: 'Deploy-to-Prod'
            }
            post {
                success {
                    echo 'Code deployed to Production.'
                }

                failure {
                    echo ' Deployment failed.'
                }
            }
        }


    }
}
```



→https://wiki.jenkins.io/display/JENKINS/Remote+access+API

Sample scenario



Without proper feedback, continuous integration is useless.

More to explore

Books on Continuous integration:

Duvall's Continuous Integration: http://www.amazon.com/Continuous-Integration-Improving-Software-Reducing/dp/0321336380

Humble's "Continuous Delivery": http://www.amazon.com/Continuous-Delivery-Deployment-Automation-Addison-Wesley/dp/0321601912

Hudson/Jenkins

Extensive information: http://www.youtube.com/watch?v=6k0S4O2PnTc#!

Maven:

Free ebook: http://www.sonatype.com/books/mvnref-book/reference/public-book.html