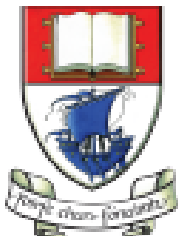


DevOps and Maven

Produced
by:

Eamonn de Leastar (edeleastar@wit.ie)

Dr. Siobhán Drohan (sdrohan@wit.ie)



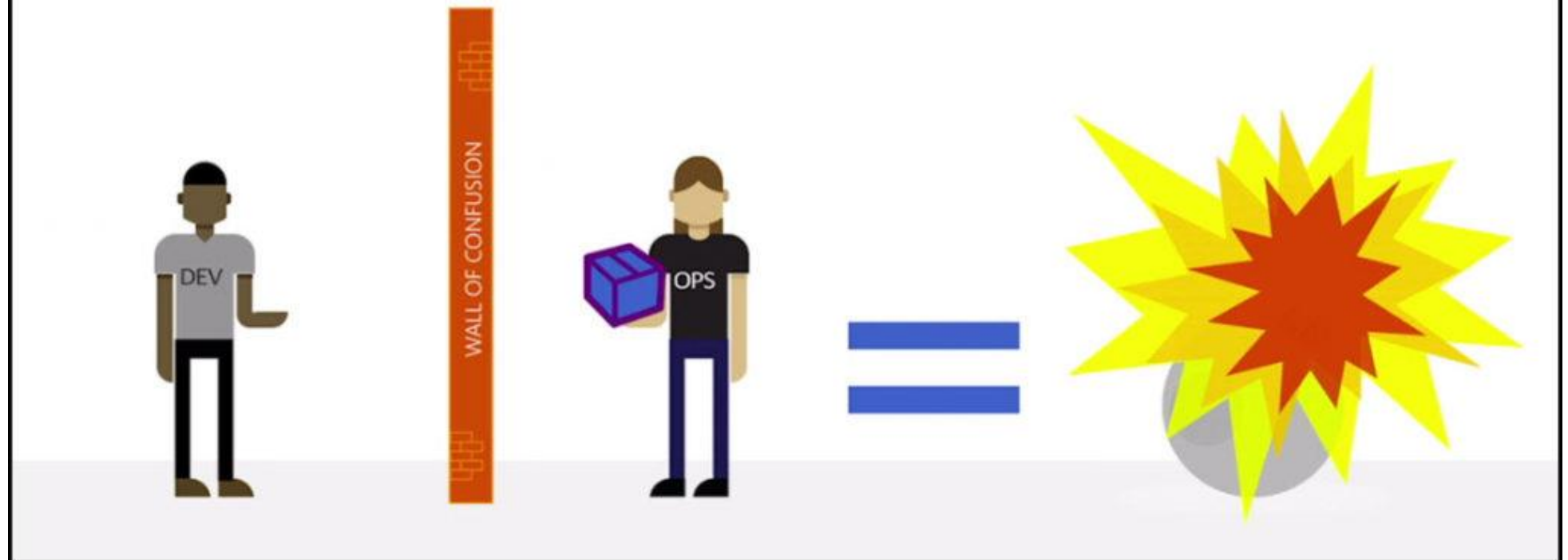
Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
<http://www.wit.ie/>

Traditional Development and Operations

Dev team
created a
solution for
production.

When it was
finished they
handed it over to
the ops team.



Ops job is to implement
the project in production
by manually changing
configuration files and
other data in order to
comply for deployment.



Development



Operations



Development
Wants agility



Operations
Wants stability

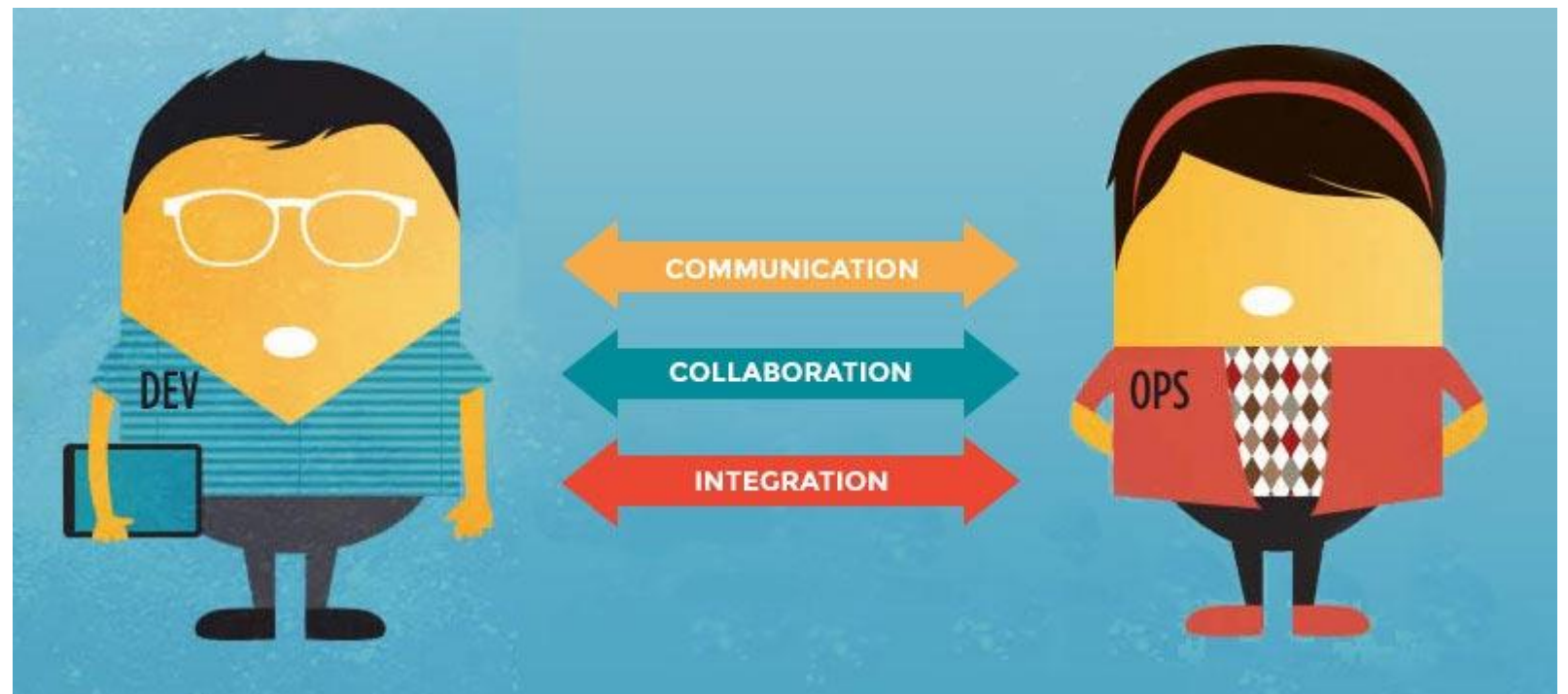
*“The idea of shipping code
faster has been a priority
since the practice of
software development began”*

*“DevOps is about
more frequent,
higher quality releases.”*

What is DevOps?

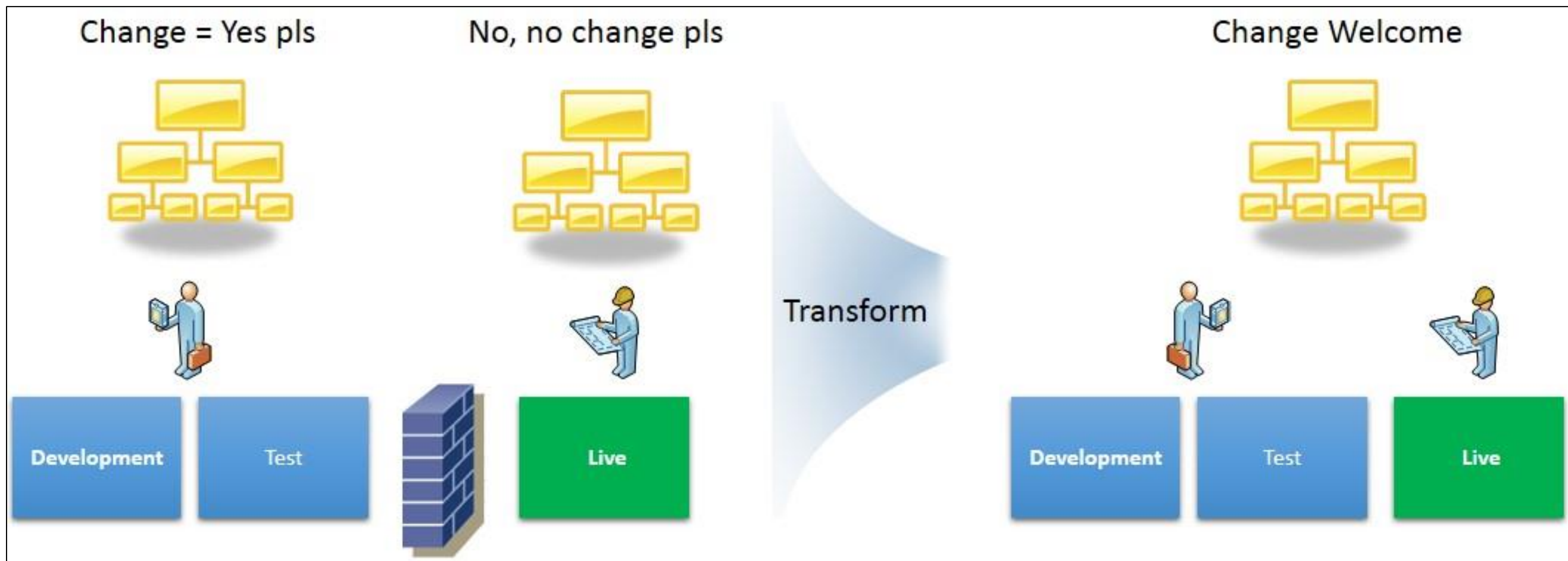
- DevOps is a **software development approach** that stresses:

- Communication
- Collaboration
- Integration
- Trust



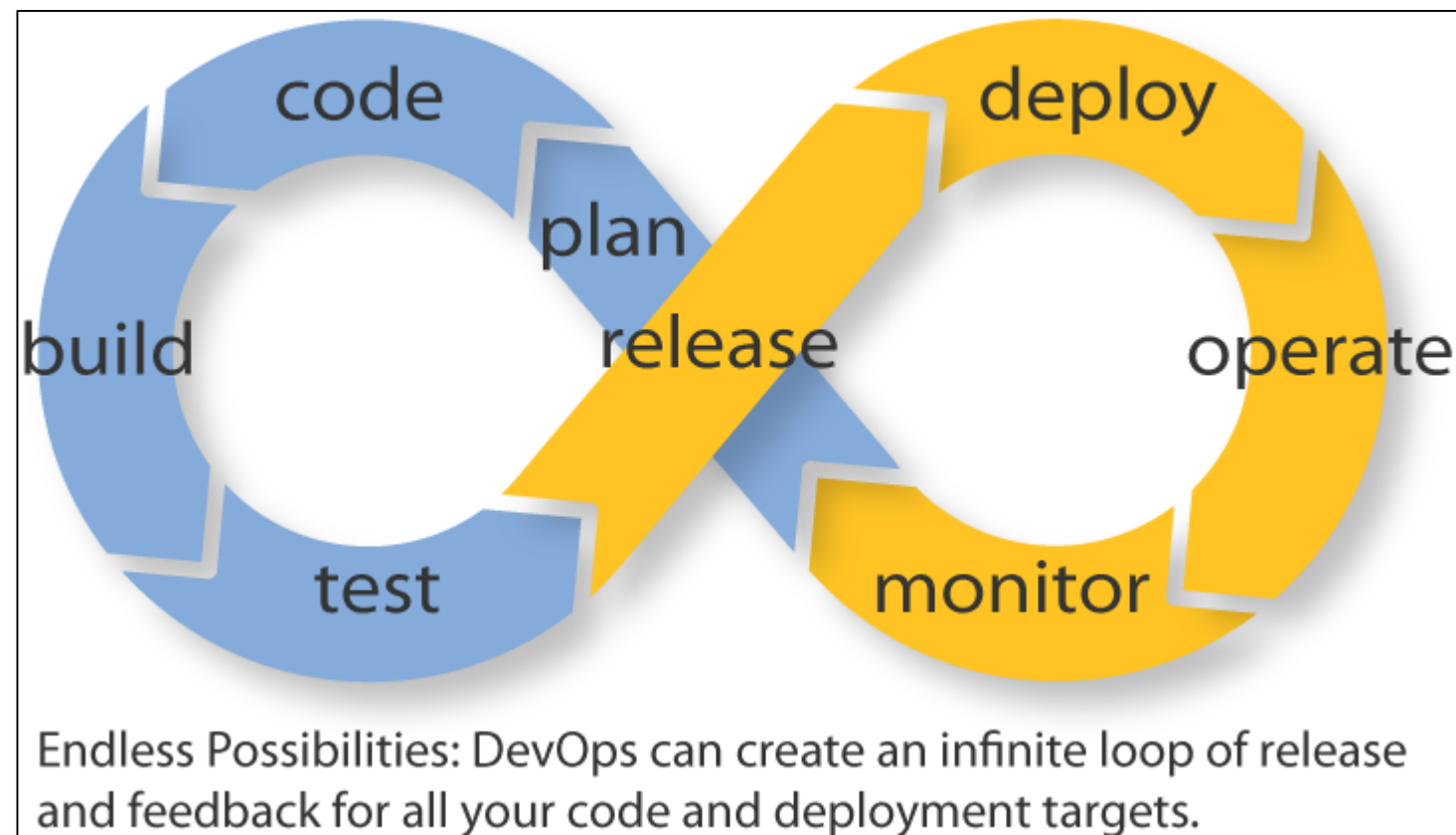
- between software developers and information technology (IT) professionals i.e. the merging of two different disciplines.

With DevOps, change is welcome

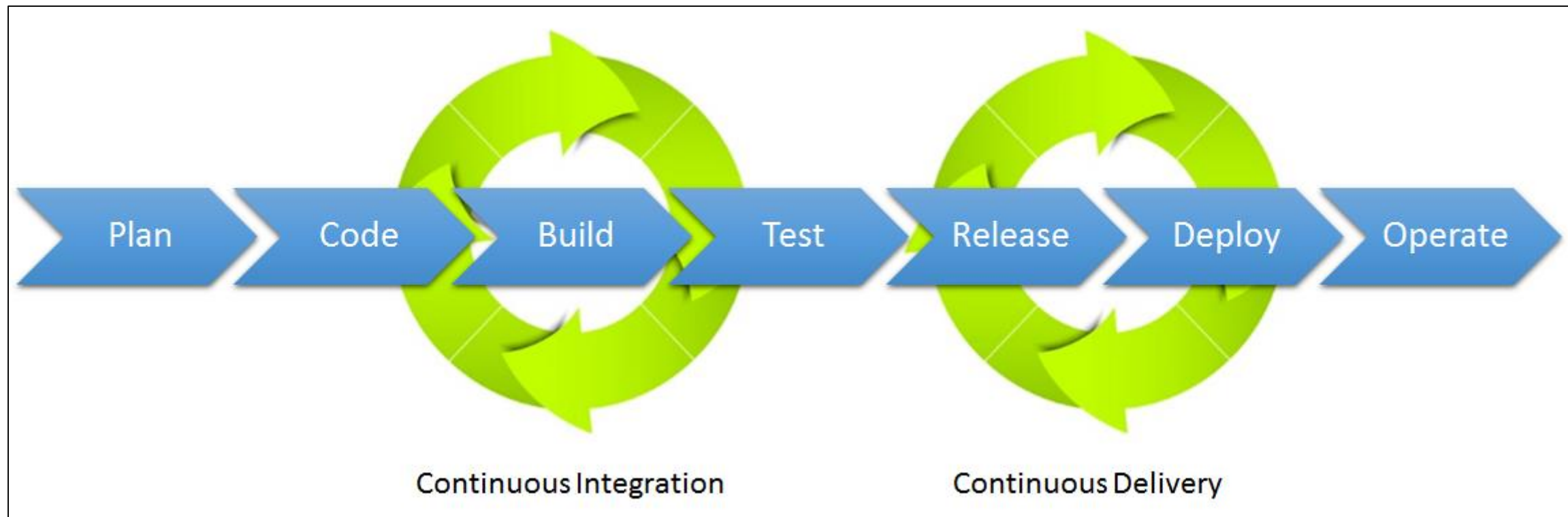


What is DevOps?

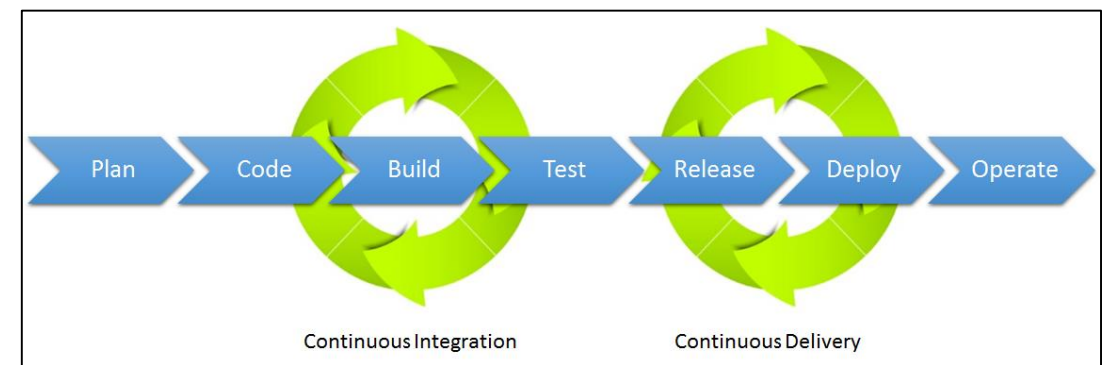
- DevOps allows us to build, deploy, and change our software with accelerated delivery cycle times.
- DevOps integration targets product delivery, quality testing, feature development, and maintenance releases in order to improve reliability and security and faster development and deployment cycles.



DevOps enables the merging of Continuous Integration and Continuous Delivery

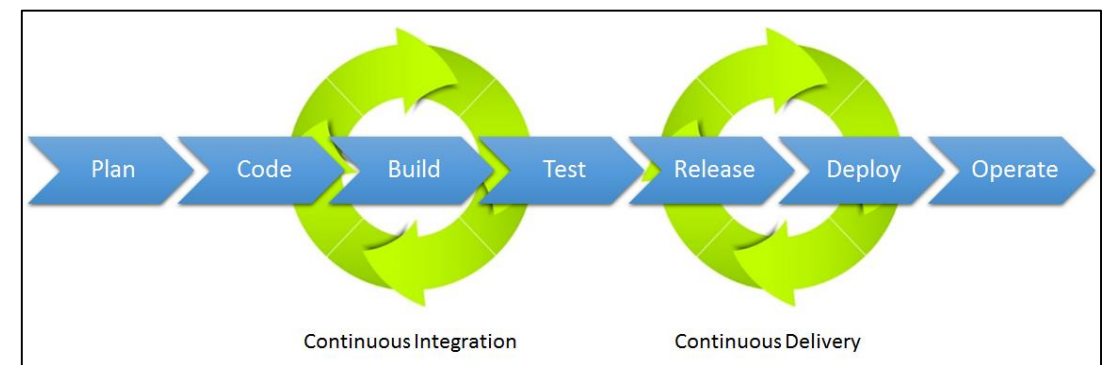


Continuous Integration



- The process of steadily adding new code commits to source code.
- Originally, a daily build was the standard for continuous integration.
- Today, the usual rule is for each team member to submit work as soon as it is finished and for a build to be conducted with each significant change.
 - Usually, a certain baseline of automated unit and integration testing is performed to ensure that new code does not break the build.
 - This way developers know as soon as they're done if their code will meet minimum standards and they can fix problems while the code is still fresh in their minds.
- An important advantage of continuous integration is that it provides developers with immediate feedback and status updates for the software they are working on.

Continuous Delivery



“Continuous Delivery is the ability to get changes of all types — including new features, configuration changes, bug fixes and experiments — into production, or into the hands of users, safely and quickly in a sustainable way.”

<https://www.continuousdelivery.com>

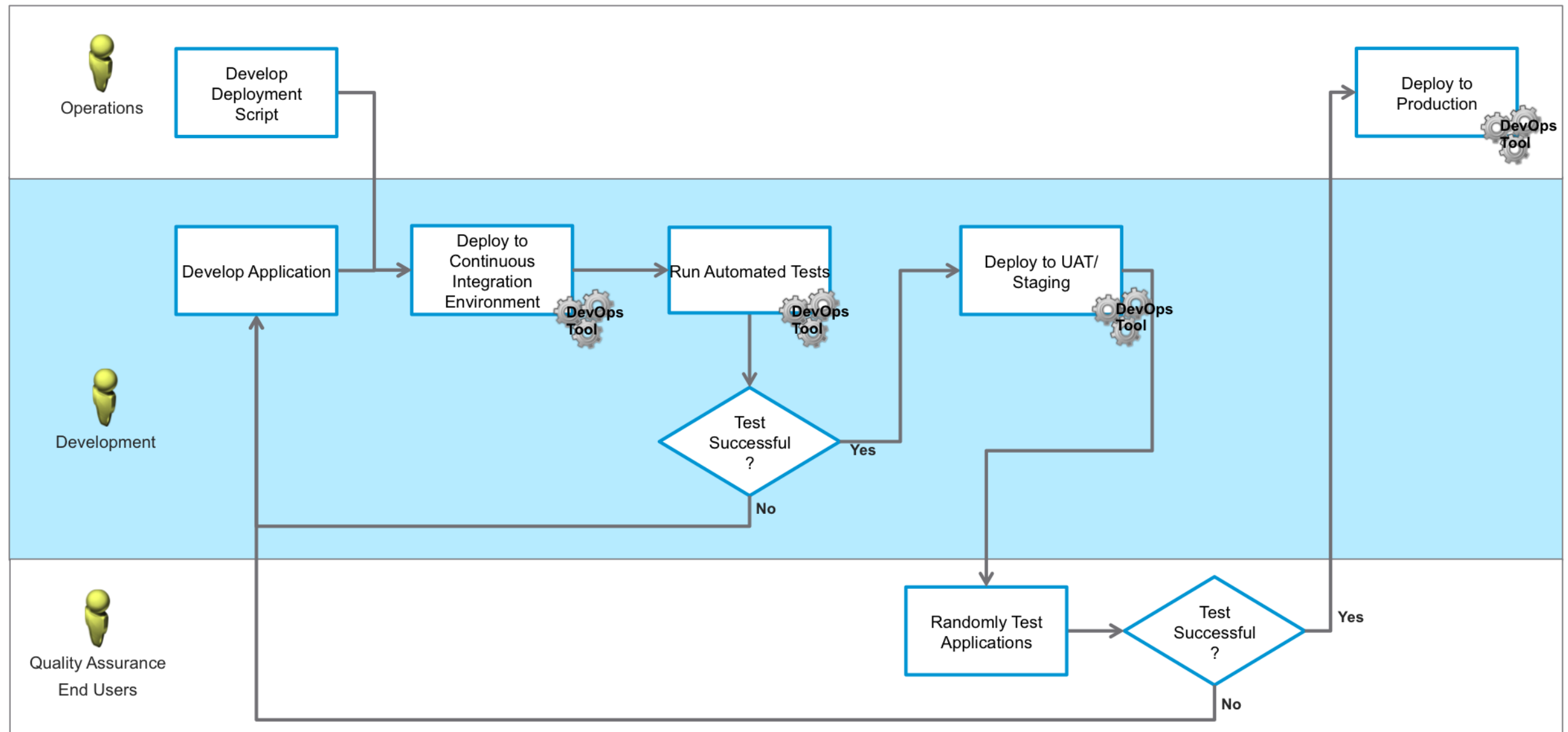
- Common goal of faster time to market for new services / releases.
- Approach whereby teams ensure that every change to the system can be released, and that any version can be released at the push of a button.

<http://automic.com/blog/whats-the-difference-between-devops-and-continuous-delivery>

Why Continuous Delivery?

Low-risk releases	Make software deployments painless, low-risk events that can be performed at any time, on demand
Faster time to market	Integration and test/fix phase of the traditional phased software delivery lifecycle to consume weeks or even months. When teams work together to automate the build and deployment, environment provisioning, and regression testing processes, developers can incorporate integration and regression testing into their daily work and we completely remove these phases.
Higher quality	When developers have automated tools that discover regressions within minutes, teams are freed to focus their effort on user research and higher level testing activities such as exploratory testing, usability testing, and performance and security testing.
Lower costs	Software changes. By investing in build, test, deployment and environment automation, we substantially reduce the cost of making and delivering incremental changes to software by eliminating many of the fixed costs associated with the release process.
Better products	Continuous delivery makes it economic to work in small batches.
Happier teams	Continuous delivery makes releases less painful and reduces team burnout. By removing the low-value painful activities associated with software delivery, we can focus on what we care about most—continuously delighting our users.

Sample DevOps LifeCycle



API Technologies



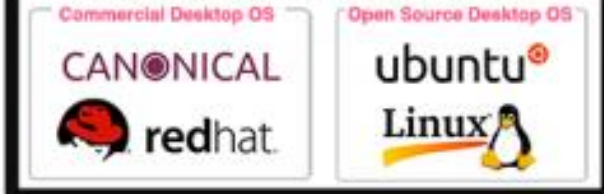
Analytics and Testing



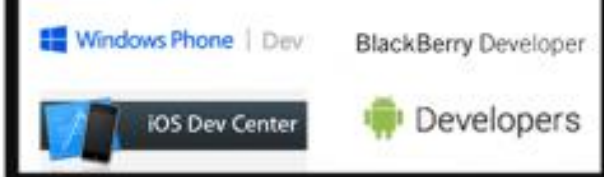
Coding Tools



Operating System Dev



Mobile Device Dev



Data Dev Technologies



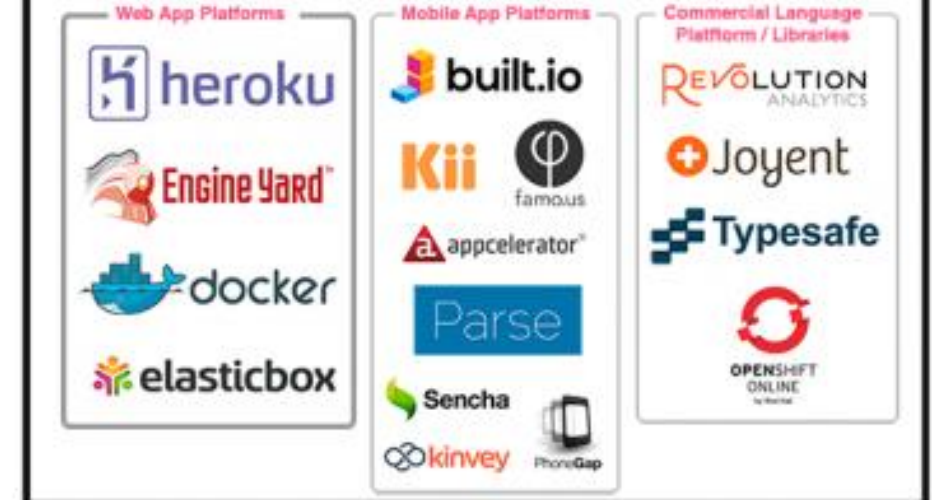
Cloud Tools



DevOps Technologies



Dev Platforms





Apache Maven Project

<http://maven.apache.org/>

A software project management and comprehension tool

What is Maven?

- Maven is a software project management and comprehension tool.
- Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly.
- Maven increases reusability and takes care of most of build related tasks.
- In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven

- Objectives:
 - Provide a standard development infrastructure across projects.
 - Make the development process transparent.
 - Decrease training for new developers.
 - Bring together tools in a uniform way.
 - Prevent inconsistent setups.
 - Divert energy to application development activities.
- Maven is a process of applying patterns to a build infrastructure in order to provide a coherent view of software projects.

Maven Principles

- Convention over configuration
- Reuse of build logic
- Declarative execution
- Coherent organization of dependencies

Convention over configuration

- This means developers are not required to create the build process themselves.
- Maven provides sensible default behaviour for projects.
- The primary conventions to promote a familiar development environment are:
 1. Standard directory layout for projects
 2. Standardised set of build phases – lifecycle phases
 3. A single Maven project produces a single output/artifact
 4. Standard naming conventions
- Maven leverages its built-in project knowledge to help users understand a complex project's structure and potential variations in the build process.

Maven Principles

- Convention over configuration
- Reuse of build logic
- Declarative execution
- Coherent organization of dependencies

Reuse of build logic

- Maven encapsulates build logic into modules called plugins.
- A plugin's components, called mojos, perform build tasks.
 - MOJO - **M**aven plain **O**ld **J**ava **O**bjects
- Maven acts as a framework which coordinates the execution of plugins in a well defined way.
- Some plugins are standard, others are downloaded on demand.

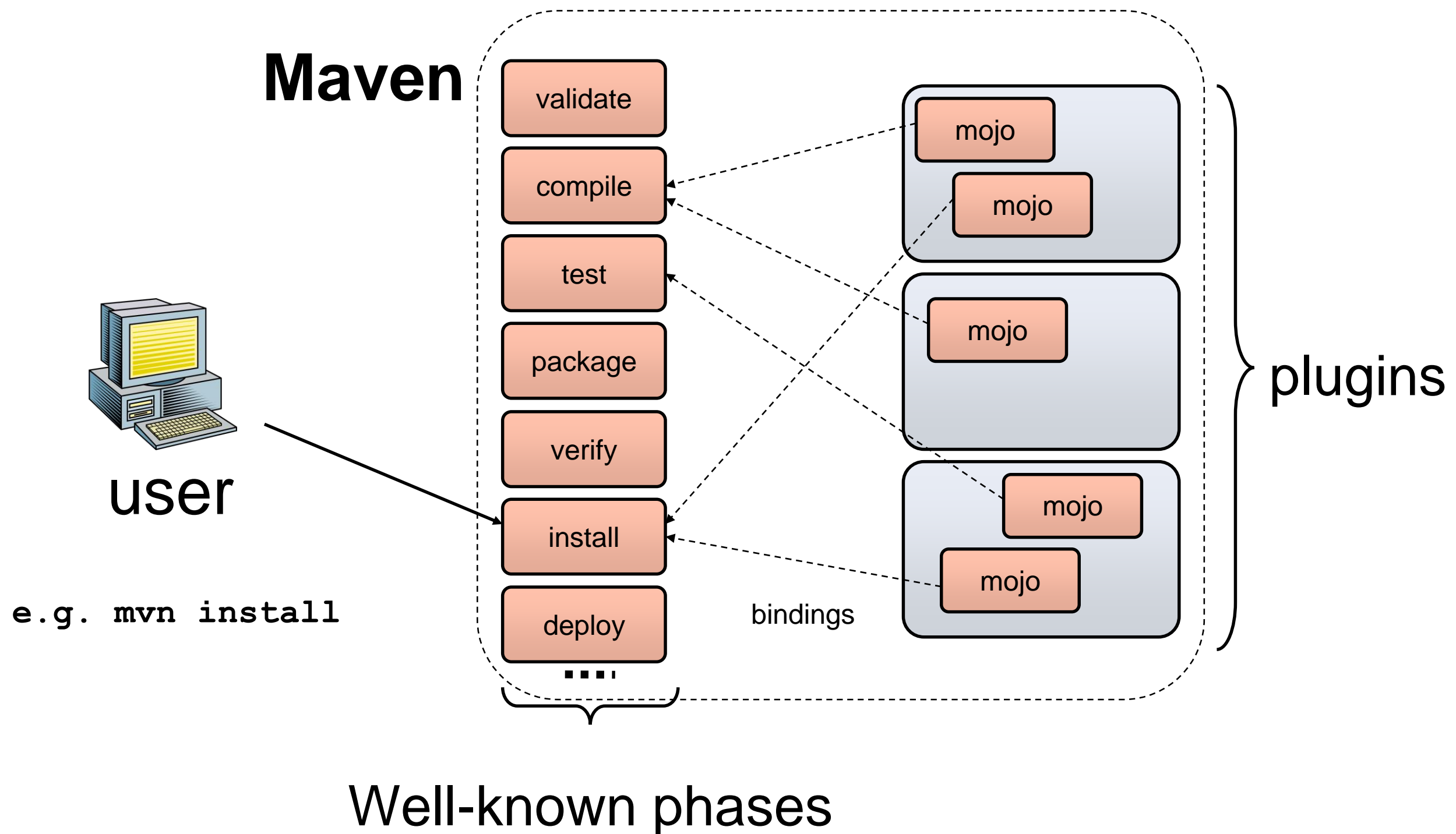
Maven Principles

- Convention over configuration
- Reuse of build logic
- Declarative execution
- Coherent organization of dependencies

Declarative Execution

- Maven's typical target names are standardised into a set of well-defined and well-known build lifecycle phases.
- A lifecycle phase invokes the relevant plugins (the mojos) to do the work.
- The phase to plugin bindings are hardwired (for standard plugins).
- User configures a plugin declaratively in the POM (Project Object Model) file.
- Configuration only necessary for non-standard cases.

Declarative Execution



Maven – Default 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.
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.

Full lifecycle reference:

https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference

Declarative Execution

- When user invokes a lifecycle phase, all its predecessors are also executed, if necessary, e.g. *mvn package*.
- You can also invoke plugins directly.

Format: *mvn plugin-name:goal*

e.g. mvn archetype:generate

Declarative Execution

- The Project Object Model, POM file (pom.xml in base directory of project), is Maven's description of a single project.
- It drives Maven's execution for a project:
 - e.g configuring a plugin for a particular phase.
- Contains metadata about the project
 - Location of directories, Developers/Contributors, Extra plugins required, Special plugin configuration, Jars required (3rd party and in-house), Repositories to search for plugins/jars, etc.
- A project's POM inherits from the Super POM.
 - All standard project information (e.g. directory structure) is held in the Super POM (principle).

Minimalist POM

```
<project>
```

```
  <modelVersion>4.0.0</modelVersion>
```

```
  { <groupId>com.mycompany.app</groupId>  
    <artifactId>my-app</artifactId>  
    <packaging>jar</packaging>  
    <version>1.0</version>  
  }
```

Uniquely identify
project in the
repo

```
  <dependencies>
```

```
    <dependency>
```

```
      <groupId>com.thoughtworks.xstream</groupId>
```

```
      <artifactId>xstream</artifactId>
```

```
      <version>1.3.1</version>
```

```
    </dependency>
```

```
  </dependencies>
```

```
</project>
```

Maven Principles

- Convention over configuration
- Reuse of build logic
- Declarative execution
- Coherent organization of dependencies

Coherent organization of dependencies

- Three related concepts: Artifact; Dependencies; Repositories

```
<project>
  .....
  <dependencies>

    <dependency>
      <groupId>com.thoughtworks.xstream</groupId>
      <artifactId>xstream</artifactId>
      <version>1.3.1</version>
    </dependency>

  </dependencies>
</project>
```

This project has a dependency on version 1.3.1 of the artifact with id xstream, produced by the com.thoughtworks.xstream group.

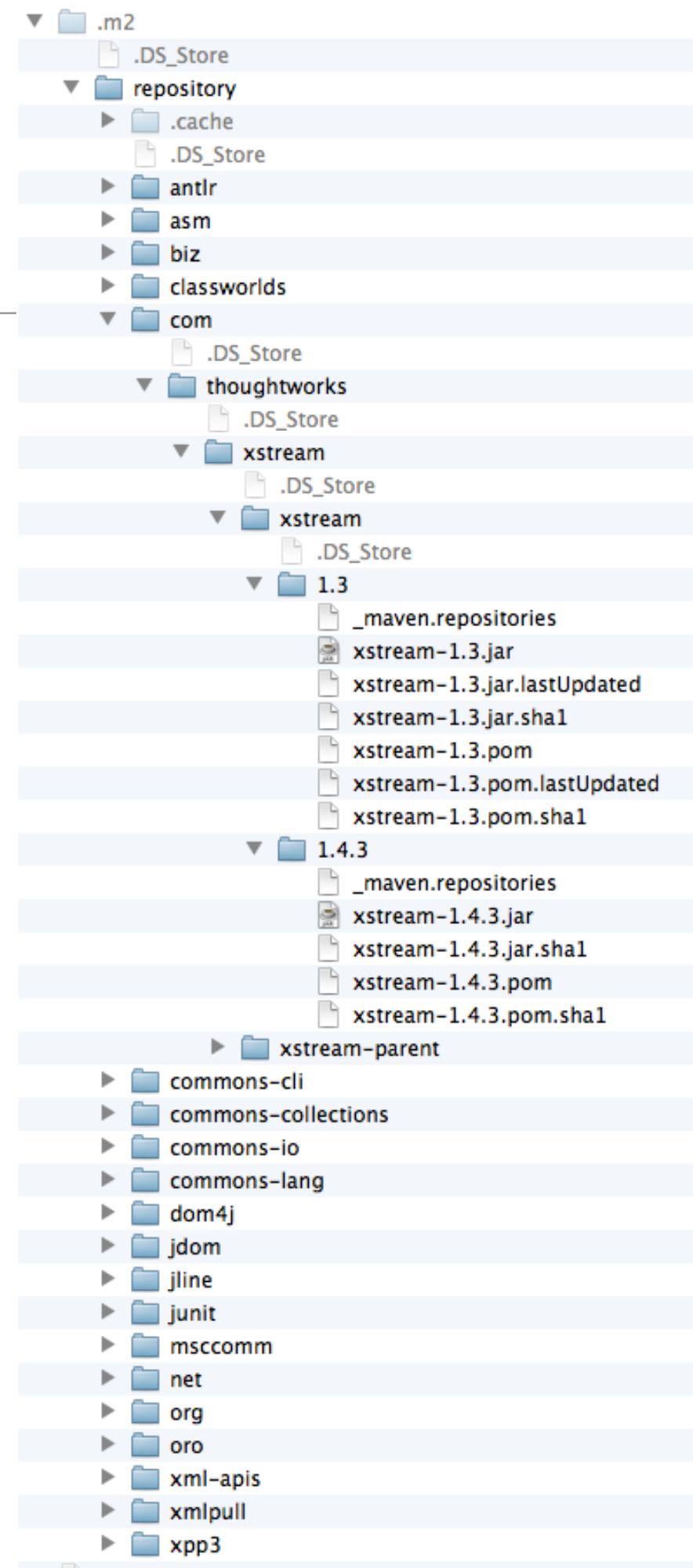
Coherent organization of dependencies

- All artifacts/dependencies are stored in repositories
 - Local and remote repositories
- The local repository is searched first, then remote ones
- Dependencies are automatically downloaded (from remote repositories) and installed (in local repository) for future use
- Maven knows about some remote repositories, e.g.

<http://www.ibiblio.org/archive/2013/02/ibiblio-tagged-in-maven/>
- Other remote repositories can be listed in the project POM or in Maven's configuration file (setting.xml)

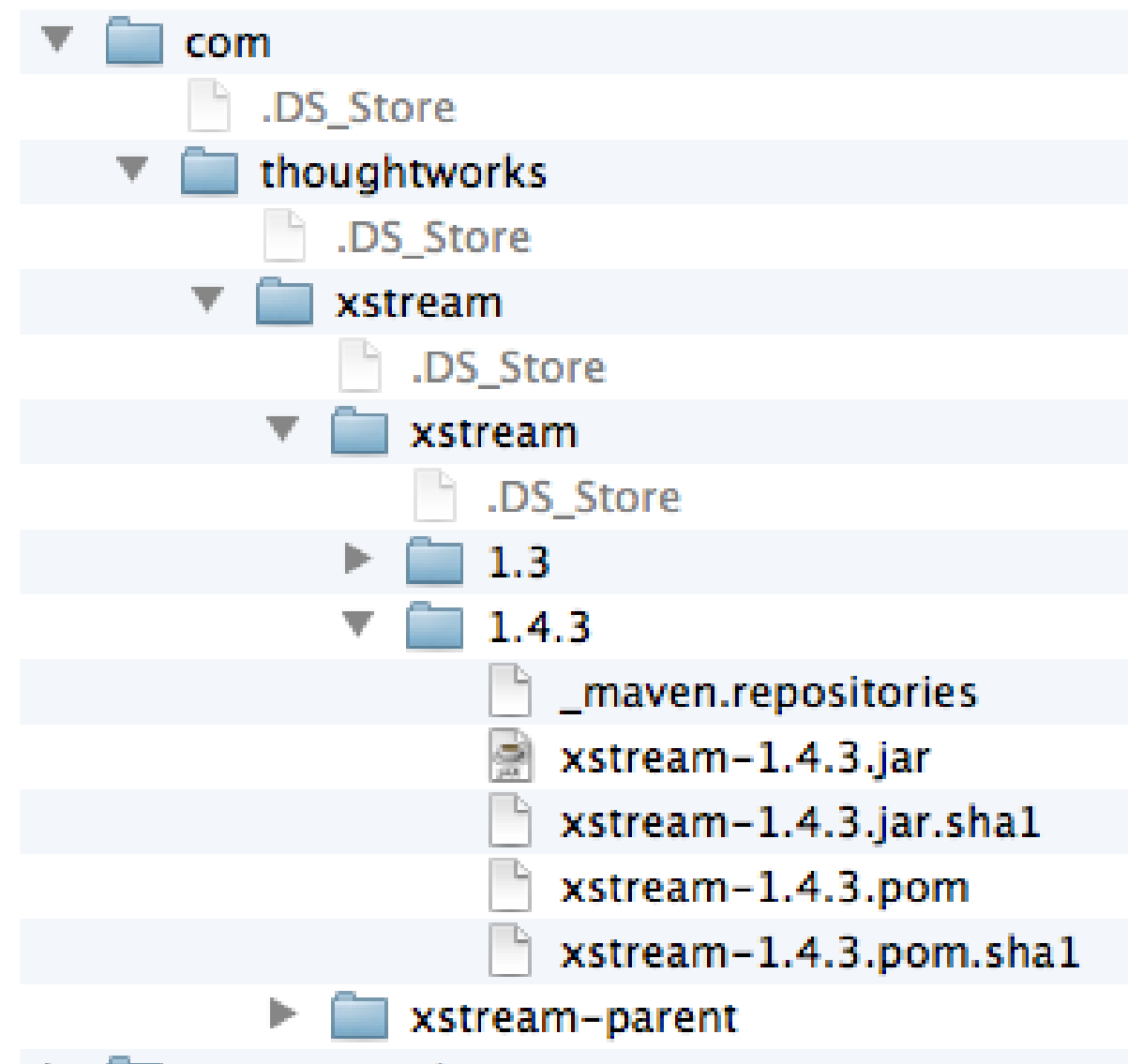
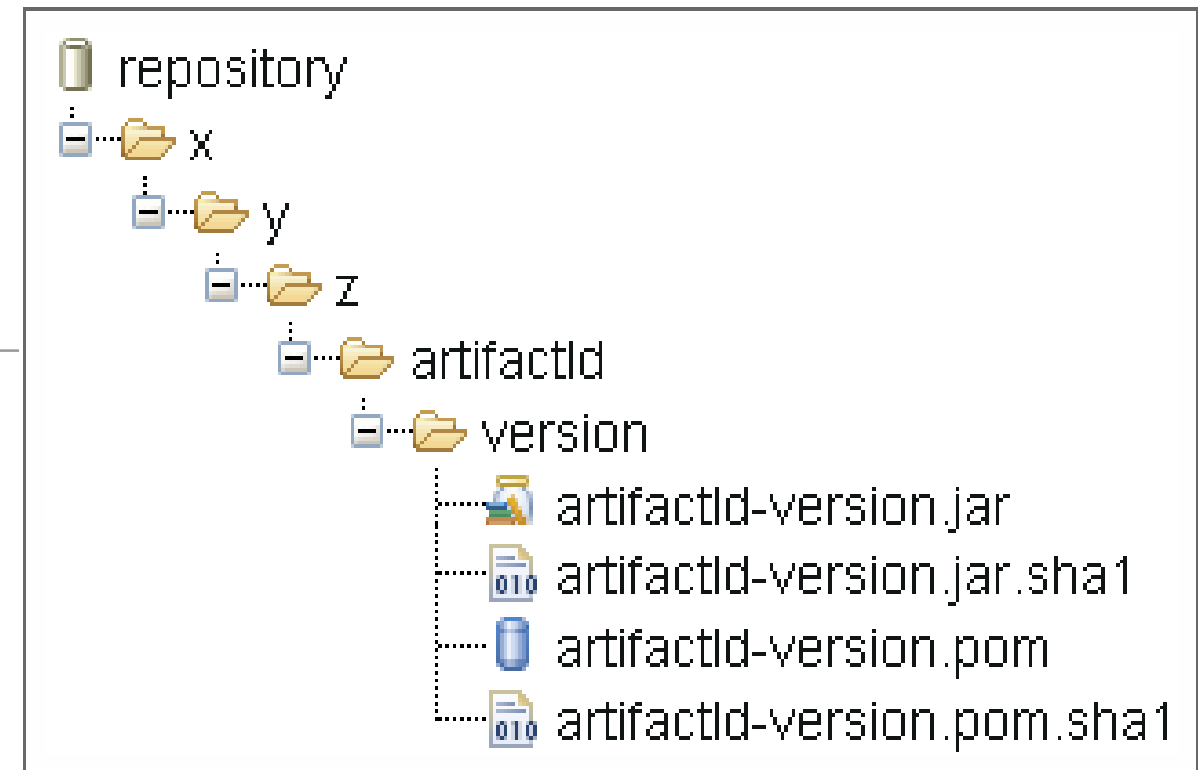
Local repositories

- After installing and running Maven for the first time a local repository is automatically created and populated with some standard artifacts.
- Default Local repository location:
Home/.m2/repository
- Plugins are also stored in repositories.
- In theory a repository is an abstract storage mechanism, but in practice it is a directory structure in your file system

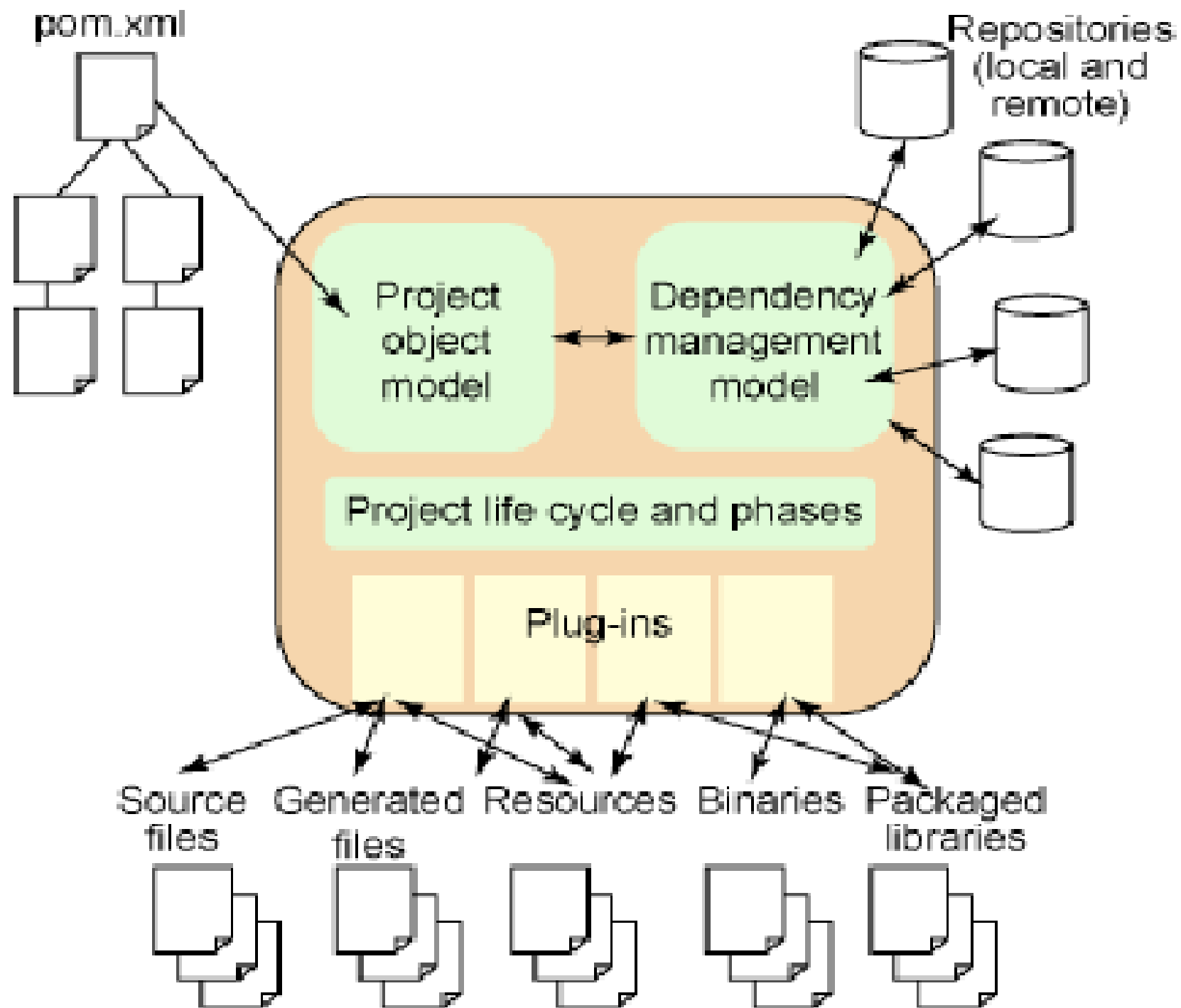


Repository structure

- Repository structure centered around dependency coordinates schema.
- Maven uses artifact's id, group id, and version to navigate to the correct folder.
- If the groupId is a fully qualified domain name such as x.y.z then it is fully expanded.



The full picture



Archetypes

- An archetype is a template project structure.
- Many archetype options:
 - maven-archetype-webapp – Web application (WAR) project template
 - maven-archetype-j2ee-simple – J2EE project (EAR) with directories and subprojects for the EJBs, servlets, etc.
 - maven-archetype-quickstart (default) – simple Java project (JAR)
- Create a new project folder structure with the archetype plugin, invoking the generate goal

```
mvn archetype:generate
```

```
-DgroupId=[your project's group id]  
-DartifactId=[your project's artifact id]  
-DarchetypeArtifactId=[artifact type]
```

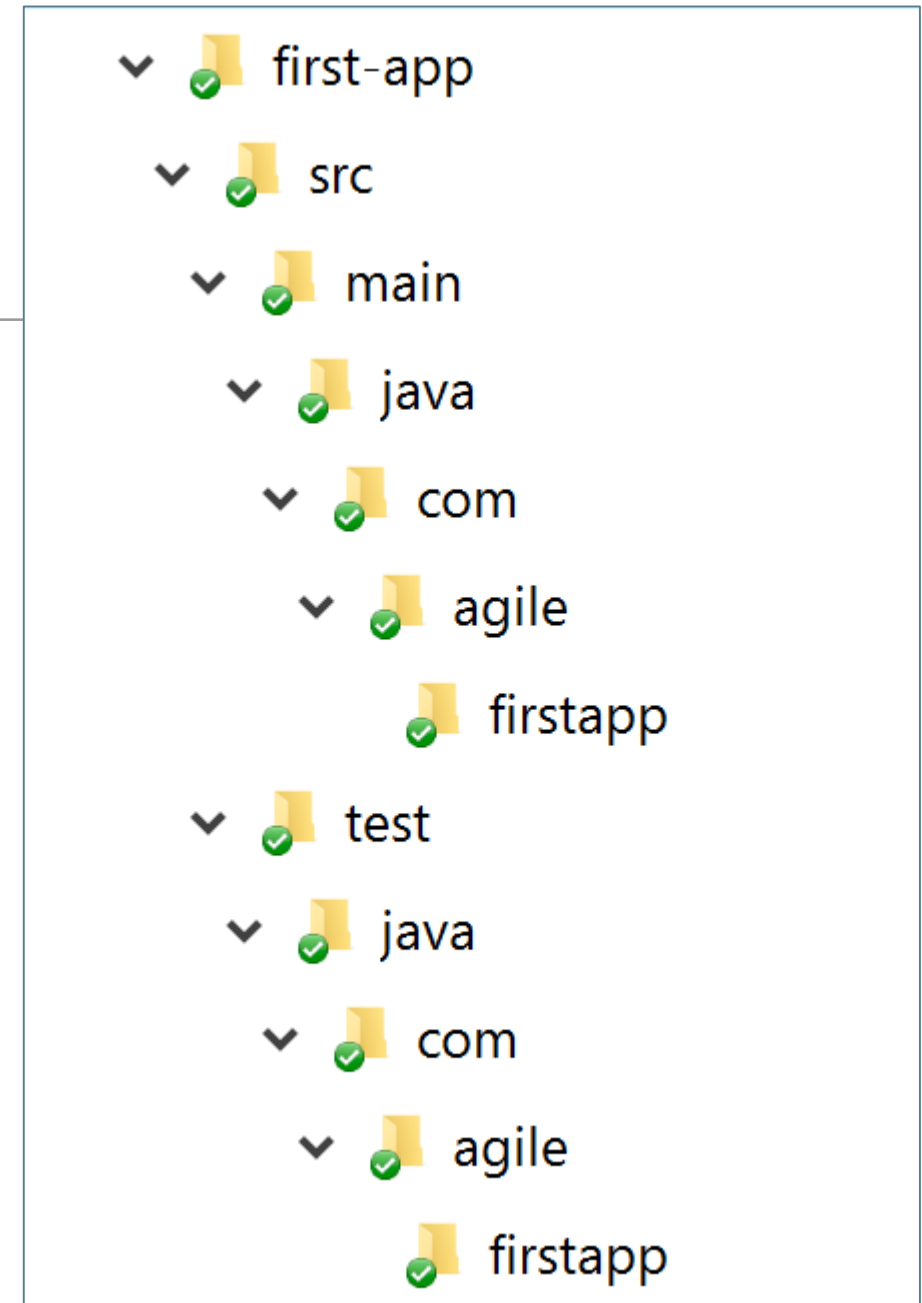
Archetypes

- An archetype is a template project structure.
- Many archetype options:
 - maven-archetype-webapp – Web application (WAR) project template
 - maven-archetype-j2ee-simple – J2EE project (EAR) with directories and subprojects for the EJBs, servlets, etc.
 - **maven-archetype-quickstart (default) – simple Java project (JAR)**
- Create a new project folder structure with the **archetype** plugin, invoking the **generate** goal

```
mvn archetype:generate
  -DgroupId=com.agile.firstapp
  -DartifactId=first-app
  -DarchetypeArtifactId=maven-archetype-quickstart
```


Quickstart archetype

- Folder structure for 'quickstart' archetype.
- The base directory name is taken from artifactId.
- A minimal POM is included in base directory.



```
mvn archetype:generate
```

```
-DgroupId=com.agile.firstapp
```

```
-DartifactId=first-app
```

```
-DarchetypeArtifactId=maven-archetype-quickstart
```

Quickstart archetype

```
package com.agile.firstapp;  
  
/**  
 * Hello world!  
 */  
public class App  
{  
    public static void main( String[] args )  
    {  
        System.out.println( "Hello World!" );  
    }  
}
```

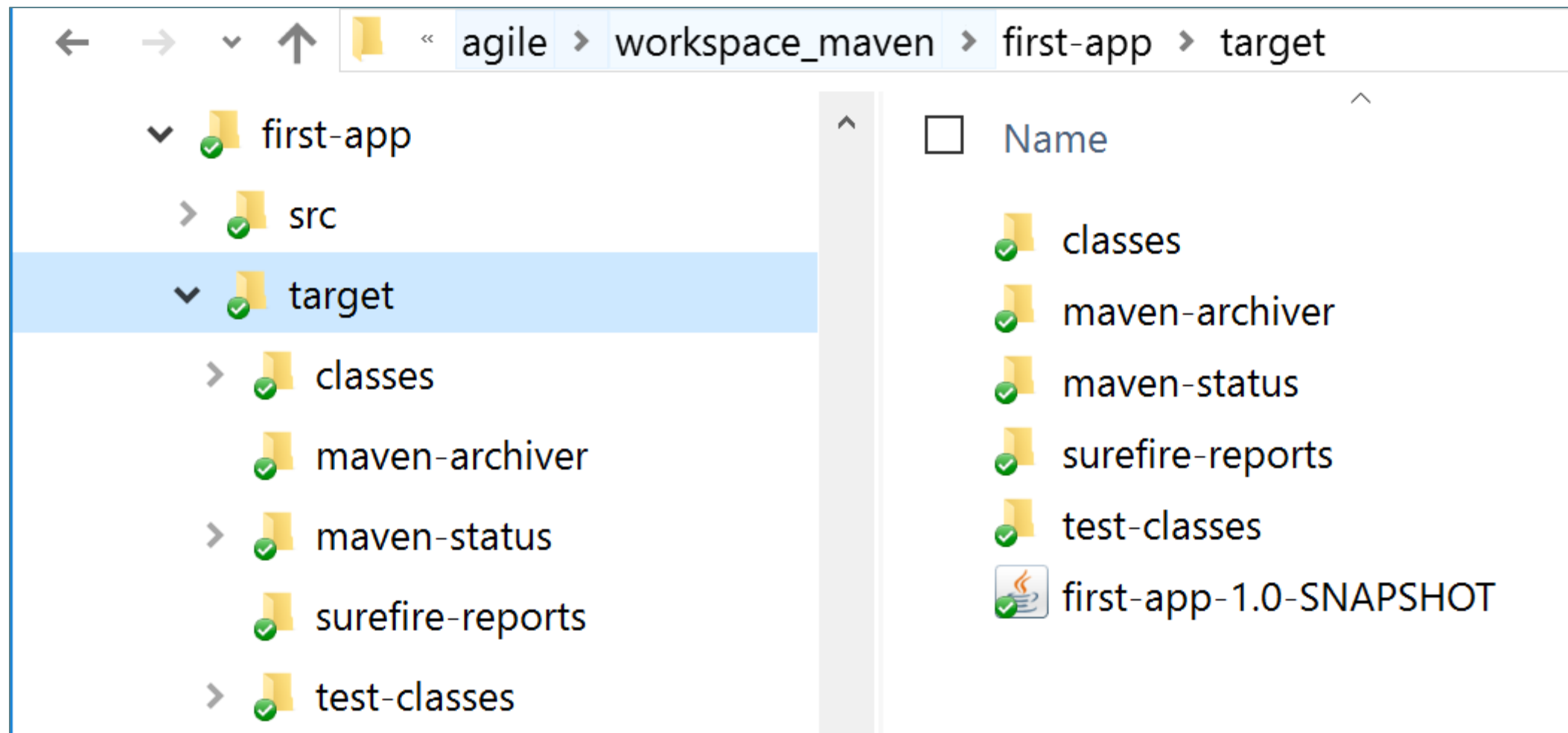
mvn archetype:generate

-DgroupId=**com.agile.firstapp**

-DartifactId=first-app

-DarchetypeArtifactId=maven-archetype-quickstart

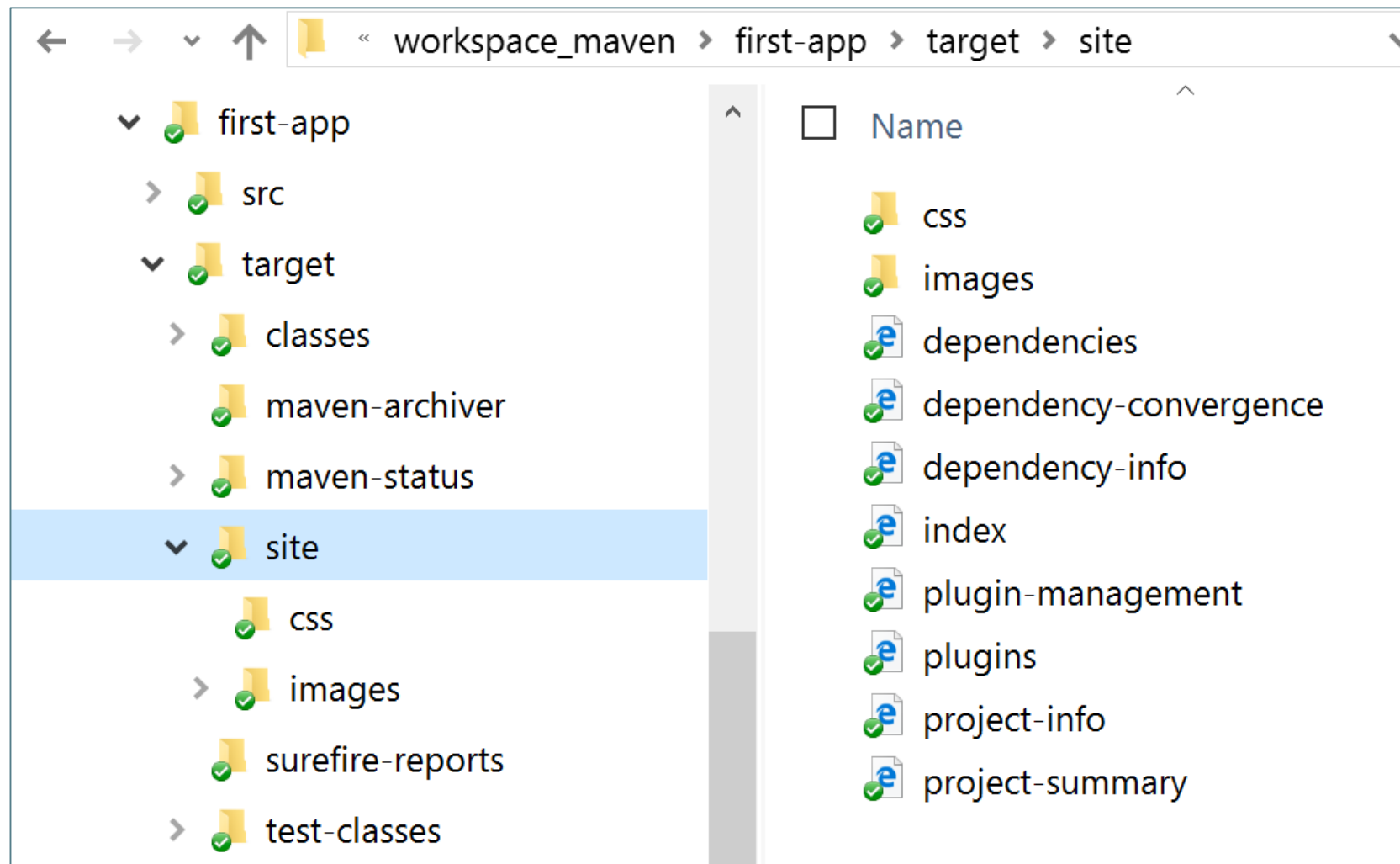
mvn package



mvn package

takes the compiled code and package it in its distributable format, such as a JAR.

mvn site



mvn site

generate the project's site documentation

mvn site

C:\Users\Siobhan\Documen | Maven – Maven Features | Maven – Introduction to the | first-app - Project Inforr X +

← → ↺ | file:///C:/Users/Siobhan/Dropbox/2016-2017/agile/workspace_maven/first-app/target/site/project-info.html | ☆ | ≡ | ✎ | 🔔 | ⋮

Find on page | Enter text to search | No results | < > | Options ▾ | X


first-app

Last Published: 2016-10-12 | Version: 1.0-SNAPSHOT first-app

Project Documentation

▼ Project Information

- Dependencies
- Dependency
- Convergence
- Dependency Information
- About
- Plugin Management
- Plugins
- Summary

Built by: 

Project Information

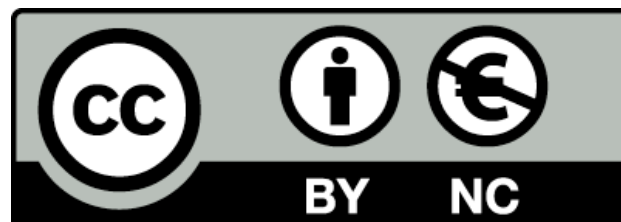
This document provides an overview of the various documents and links that are part of this project's general information. All of this content is automatically generated by [Maven](#) on behalf of the project.

Overview

Document	Description
Dependencies	This document lists the project's dependencies and provides information on each dependency.
Dependency Convergence	This document presents the convergence of dependency versions across the entire project, and its sub modules.
Dependency Information	This document describes how to to include this project as a dependency using various dependency management tools.
About	There is currently no description associated with this project.
Plugin Management	This document lists the plugins that are defined through pluginManagement.
Plugins	This document lists the build plugins and the report plugins used by this project.
Summary	This document lists other related information of this project

Copyright © 2016. All Rights Reserved.

generate the project's site documentation



Except where otherwise noted, this content is licensed under a [Creative Commons Attribution-NonCommercial 3.0 License](http://creativecommons.org/licenses/by-nc/3.0/).

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

