Table of Contents

[What you will learn 1](#_Toc41899582)

[Waterfall model 1](#_Toc41899583)

[Agile methodology 1](#_Toc41899584)

[Key benefits 1](#_Toc41899585)

[Different Styles of Agile 1](#_Toc41899586)

[Scrum 1](#_Toc41899587)

[User Stories 1](#_Toc41899588)

[3 C’s Elements 1](#_Toc41899589)

[Example 1](#_Toc41899590)

[Sprint Artifacts 1](#_Toc41899591)

[Sprint Ceremonies 1](#_Toc41899592)

[Sprint Planning 1](#_Toc41899593)

[Daily Scrum Meeting 1](#_Toc41899594)

[Sprint Review 1](#_Toc41899595)

[Sprint Retrospection 1](#_Toc41899596)

[DevOps 1](#_Toc41899597)

[Continuous Integration 1](#_Toc41899598)

[Tools used 1](#_Toc41899599)

[Continuous Delivery 1](#_Toc41899600)

[Tools used 1](#_Toc41899601)

[Git 1](#_Toc41899602)

[Workflow 1](#_Toc41899603)

[Commands 1](#_Toc41899604)

[Maven 1](#_Toc41899605)

[POM 1](#_Toc41899606)

[Repositories Tag 1](#_Toc41899607)

[Dependencies Tag 1](#_Toc41899608)

[Build tag 1](#_Toc41899609)

[Plugins Tag 1](#_Toc41899610)

[Build Life Cycle 1](#_Toc41899611)

[Maven Build 1](#_Toc41899612)

[Jenkins 1](#_Toc41899613)

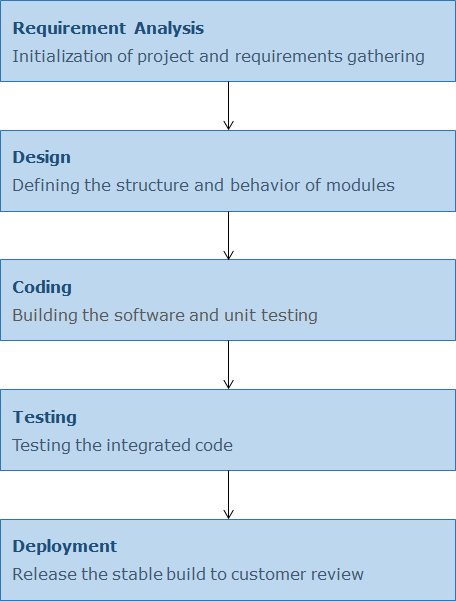
[Azure Deployment 1](#_Toc41899614)

[ESLint 1](#_Toc41899615)

# What you will learn

* Understand project execution using Agile methodology.
* Design the user interface to provide the best user experience
* Describe the concepts of Continuous Integration (CI) process involved in Software Development Life Cycle (SDLC)
* Execute the build of a project using Maven for *compiling*, running *test cases* and *deploying* the project
* Discuss the need for a *source code management system* and demonstrate the capabilities of Git
* Implement the Continuous Integration process using Jenkins

# Waterfall model

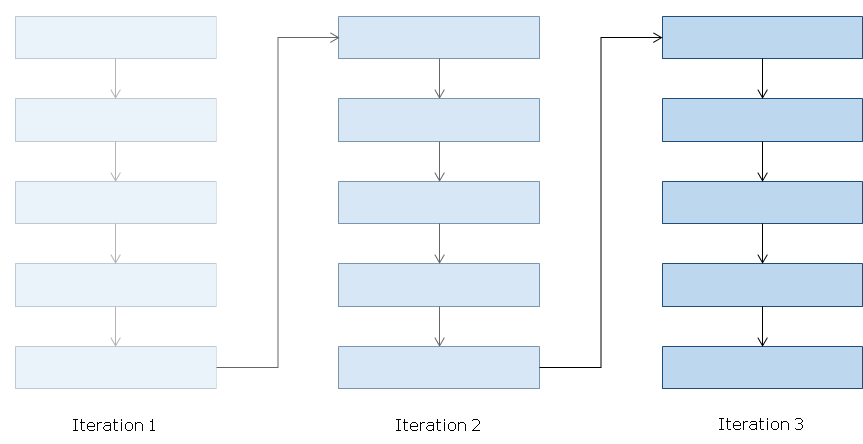
SRS: Software Requirement Stage – Using in Requirement Analysis

UML – Unified Modeling Language – Plan on how to implement and test project

Choosing Data Structures, Unit Testing, …

Integrating, Performance, System tests.

# Agile methodology



Projects are developed in a short time-boxed duration, during which a set of pre-determined *features are developed and delivered*. These *time intervals* are called Sprints/Iterations. They typically range between *2 to 4 weeks*. Through Agile methodology, projects are developed with multiple iterations and a working incremental versions of the software is delivered at the end of each iteration. At the end of the final iteration, the product with all pre-determined features will get delivered.

* Instead of focusing on processes and tools, one focuses on *individuals and interactions*.
* *Working* software gets priority over comprehensive documentation.
* It is more important to *react to change* than to stick to a plan.
* *Collaboration* with customer is more important than negotiating contract.

Key benefits

* *Reduction of risk associated* with the development of software
* Allows frequent and continuous releases of the product.
* The customers are able to view the working product and provide feedback for the functionalities, falling short of expectations.
* The delivery time is quicker compared to the traditional Waterfall approach.
* In Agile methodology there are *frequent interactions* *between the customer and the development team* which *increases collaboration between the team members*, as well with the *client*.
* There is flexibility in incorporating changes in the current sprint or in the upcoming sprints.

## Different Styles of Agile

* Scrum: An agile process that allows self-organizing teams to focus on delivering the *highest business value* in the *shortest time*.
* Kanban: Kanban keeps a *constant flow* of work in motion, ensuring that everyone is allocated just the right amount of work at any given time. This is called Limiting Work in Progress.
* Lean software development: Lean is more of a *group of principles* and tools for building your own processes. These tools and tenets were derived out of manufacturing process developed by Toyota.
* Extreme Programming: Extreme programming (XP) is rooted in frequent releases and short development cycles, with opportunities to adopt new customer requirements. It includes *programming in pairs*, extensive *code review* and simplicity and *clarity in code*.

### Scrum

Statistis say:

* 87% say Scrum improves teams' quality of work life.
* 81% believe certifications improves the practice.
* On average, Scrum is successful 62% of the time.
* 95% will continue to use Scrum.

Attributes:

* Cross-functional team *works together as a single unit* having specialization in different areas.
  + The core of the team involves architects, business analyst, coders, testers, UI Designer etc.
* Self-organized and focused team which is empowered to *take collective decisions* on its own.
* The *team has* *shared targets* and it decides what and how much work is to be done.
* Close communication with the client representative to understand the features.
* Development happens in time boxed short development cycles known as Sprints, at the end of which working software is delivered to the customer.
* The Quality of product and process has to be met in every sprint which should satisfy the Definition of Done. It is the criteria against which the working software is evaluated by the client. The product is either 100% done or not done.

Phases:

1. Plan - In this phase, project is planned and high level design is made.
2. Produce - Actual development of project is done in this phase.
3. Inspect - The client inspects the product and delivers feedback to the team.
4. Adapt - The project team incorporates changes as per feedback received.

Roles:

* Product Owner
  + Creates and prioritizes the requirements.
  + Understands the customer's needs and business value.
  + Who all can be a Product owner (PO)?
    - Customers, Product managers, Program managers, Project managers, Business analysts,
* Development Team
  + Organizes itself to perform the work and deliver business value.
  + Comprises of 5-9 members.
  + Who all can be part of a development team?
    - Architects, Programmers, Testers, Business analysts, UI designers, Document writers
* Scrum Master
  + Organizes the process.
  + Keeps a track of team progress.
  + Problem solver
  + Who all can become a scrum master?
    - An experienced member of the team, Ex-project manager

## User Stories

User Stories are the requirements / functionalities which has to be implemented. These are the features which the client *might like* to see in their software. By definition, an User Story is a short and simple description of a feature, explained from the perspective of the person who desires the new capability (usually the client or other stakeholders of the application). But often these user stories are wrongly interpreted by the development team which often leads to wrong implementation of a requirement.

"As a <User / type of user> I want to <achieve certain goal / target> so that <some result or reason of doing the thing is fulfilled>"

Here three things are important :

1. The role of the user.
2. The functionality desired by the user.
3. The reason behind requesting the functionality

Deriving hours to complete for each story. Story Point units are given according to a Fibbonacci sequence. These story points help prioritize tasks to complete in next Sprint.

### 3 C’s Elements

* Card: The optimal size of a user story is when it can be represented on a small index card. In the words of *Ron Jeffries* who proposed the three C's model, a user story should not contain all the information about the functionality but rather just enough to initiate planning for the story.
* Conversation: In-depth conversations with the customers are necessary throughout the sprint cycles as a user story might be difficult to interpret, background knowledge might be needed or requirements might change since the story was written. This conversation or discussion is valued more than what is captured in the user story itself.
* Confirmation: The final component of a user story which serves as a measure of whether the story was implemented correctly. It is also known as Acceptance Criteria and is arrived at through *discussions with customers and the project team*.

### Example

“As a registered customer, I should be able to delete the saved cards, so that I can remove my old cards.”

Acceptance Criteria :

* Should allow the customer to choose and delete the card.
* Customer should be prompted to confirm the delete operation before the delete operation is performed.
* On successful deletion, a success message should be displayed to the end user.
* In case of any error or exception, appropriate error message should be displayed.

To sum up: user stories should be written such that it contains the absolute minimum amount of information needed to enable start of development and testing

## Sprint Artifacts

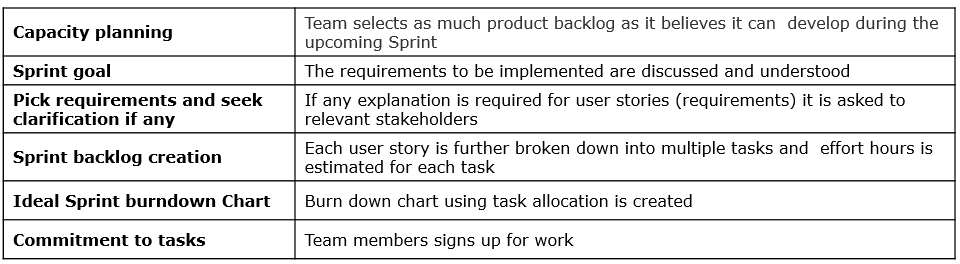
*Sprint Artifacts* provide key information that the team and the client need to be aware of in order to understand the product under development and the activities being planned in the project.

* Product Backlog: Contains a full list of user stories that describe functionalities and features of the product.
* Sprint Backlog: Contains a list of the features to be implemented in the current sprint.
* Potentially Shippable Increment: Is a working software containing the features implemented in that sprint.

## Sprint Ceremonies

* Sprint Planning - Conducted at the beginning of each Sprint. Team members decide what needs to be implemented in the current Sprint.
* Daily Scrum Meeting - Conducted every day, for 15 minutes during Sprint execution. Team members discuss the things they have done or going to do on that day.
* Sprint Review - Conducted at the end of each Sprint where the shippable increment developed during the Sprint will be reviewed.
* Sprint Retrospection - Conducted after the Sprint Review where the team members identify the things which are done well and the things which did not go as planned, so as to make the next Sprint better.

### Sprint Planning

Arrive at a list of tasks for the stories chosen for a particular sprint.

* Seek clarifications from Product Owner on the chosen user stories.
* Divide the user stories into tasks and estimate effort.

Daily Scrum Meeting

* What did I do since the last Scrum meeting?
* What am I going to do today?
* What are the obstacles I am facing?

Sprint Review

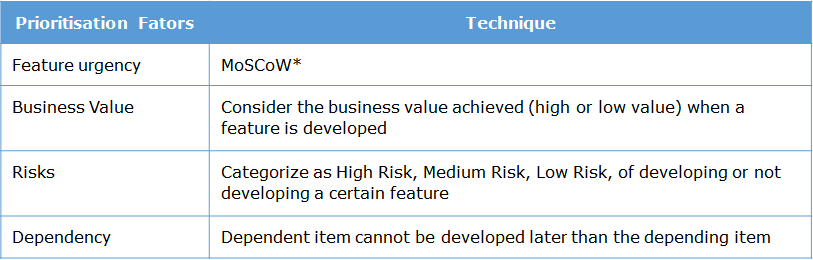
The ceremony held at the end of each sprint where the team members submit the increment produced in the sprint for review. The Product Owner along with other stakeholders check the produced increment for the done-ness and discuss whether the sprint goal has been met and provide feedback and suggest improvements.

### Sprint Retrospection

Takes place immediately after the Sprint Review is done. While review focuses on what has been done, this ceremony is more focused on how things are done. The agenda of this ceremony is improving the development process by sharing experiences and observations of the team members, encountered during the sprint. Three major things are discussed in this meeting:

* What went well?
* What could be improved?
* How it can be improved?

Prioritzing User Stories



MoSCoW

* Must Have: A critical requirement that must be in the solution. The solution would be invalid without it.
* Should Have: A critical requirement that should be in the solution if at all possible, but which can be met by other means.
* Could Have: A desirable / 'nice to have' requirement, like one which would improve the user experience.
* Won't Have: A requirement that has been agreed as out of scope for the current release / time box, but the client would like to consider it in future.

Development Phase:

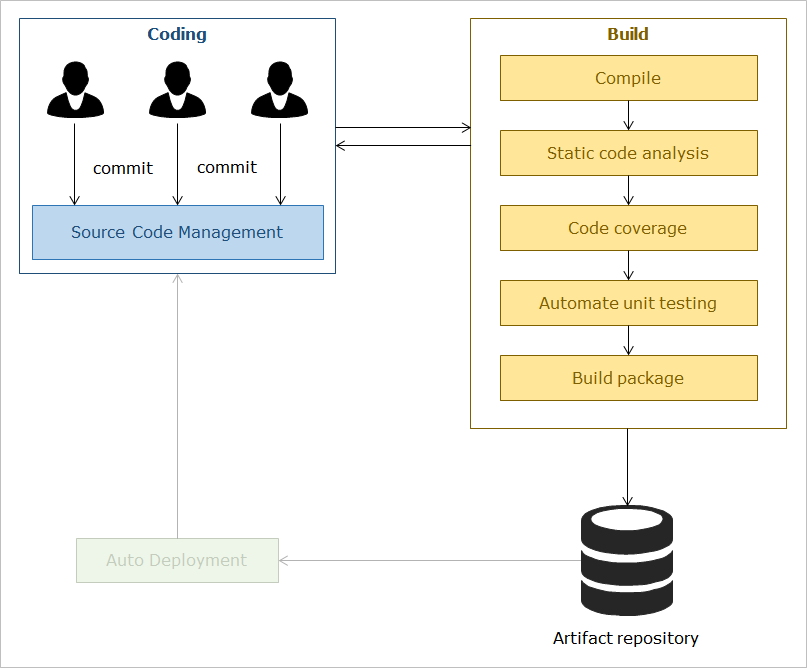
Code is integrated and test cases are run and later packaged into binary artifacts before deploying

# DevOps

Development of software done in three teams:

* Business Representatives :
  + Responsible for requirements and release management.
  + Part of the customer team.
* Development Team (Dev):
  + Responsible for development activities, testing, quality of code and deliverable software.
* Operations Team (Ops):
  + Responsible for deploying the application developed by development team as per release management plan and maintaining it.
  + Automating Dev team tasks like Code Integration, running test cases, packaging into jar or war.

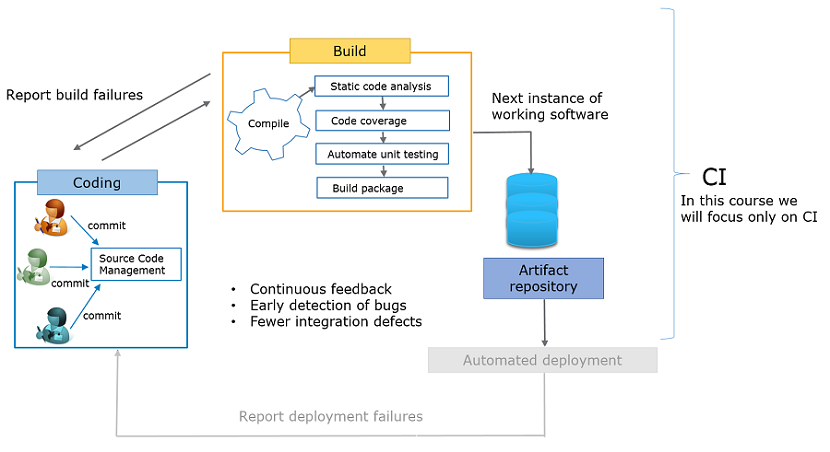
Devops:

* DevOps methodology will make rapid deployment a reality through seamless collaboration between Development team(Dev) & Operations team(Ops) and *automation of the activities* involved in development and delivery.
* This approach involves developers and operations executives in a project starting from development to deployment enabling rapid delivery. It uses various tools throughout the life cycle process to automate project release.
* DevOps expands the continuous development targets of Agile methodology to Continuous Integration(CI) and Continuous Delivery(CD).
* Developers *integrate* the code using the Source Control Management (SCM) system.
* Build is automatically triggered by the Continuous Integration (CI) system when there is a change in SCM.
* If build fails it will be reported back to the developers.
* After successful build the software is placed in the Artifact Repository.

When automated unit tests and build are successful for a software, it will deploy the application to an environment where more in-depth system testing and the performance testing are done.

Auto deployment comes under Continuous Deployment.

## Continuous Integration



Continuous Integration(CI) is a software development practice adopted as part of eXtreme Programming (XP). It helps in automating the build process, enabling frequent integration, code quality checks and unit testing without any manual intervention. Each time a developer integrates his code, automatically build process is triggered.

CI in practice:

* supports team members to integrate their work frequently.
* automates build process used for verification of the integration.
* uses regression tests.
* leads to fewer integration problems.
* helps fast development of cohesive software.
  + software which constitutes of multiple interdependent modules.

### Tools used

1. Git is a free and open source distributed Source Code Management system designed to handle everything from small to very large projects with speed and efficiency.
2. Cobertura is a free Java tool that calculates the percentage of code accessed by tests. It can be used to identify which parts of your Java program are lacking *test coverage*.
3. Artifactory is a software used to manage the binary artifacts (jar/war).
4. Apache Maven is an intelligent, comprehensive, project management and build tool to build deployable relics from the source code.
5. Jenkins *CI server* is the leading open-source continuous integration server. Built with Java, it provides 1115 plugins to support building and testing of virtually any project.

## Continuous Delivery

CD takes CI to one step further. Automated deployment of the working software from the artifact repository happens. The software undergoes system and performance testing before deployment/delivery. Any deployment failures are also reported.

Then the software is *ready to be released* to the customer environment.

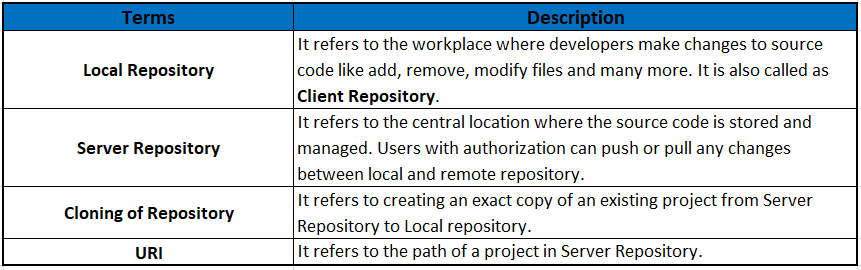
Here is the overall CD workflow.

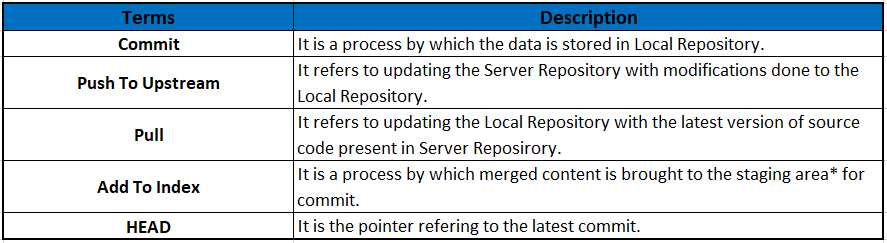
### Tools used

1. Jenkins is a versatile open-source platform that can be configured to be used as a continuous delivery platform along with its powerful continuous integration capabilities.
2. CircleCI provides a state of the art platform for integration and delivery. It has garnered a following of hundreds of thousands of teams across the globe.
3. AppVeyor is a Windows-only cloud-based all-in-one continuous delivery platform that provides software like Visual Studio pre-installed. You can quickly set up your CI/CD environment and get started on developing your apps.
4. GoCD is a open-source continuous delivery server that can automate and streamline several processes like testing and building. It even lets you set up multiple servers to keep your application consistently available even in case of server failure.

# Git

We will learn about the most commonly used Source Code Management system Git and it has lot advantages than other SCM tools such as SVN - Subversion, TortoiseSVN, CVS and VSS.

* Git is an open source distributed SCM tool.
* Git provides fast collaboration as most operations are performed offline at client side.
* Git has provision for implicit backup.
* We do not need powerful central server.
* It’s highly secured as it uses SSH\* keys to connect with server.
  + SSH(Secure Socket Shell) keys is a network protocol which provide a secure way to access remote server.



## Workflow

Git has three main states that the files can reside in : Modified, Staged, Committed.

* In a Working Directory(workspace)
  + Modify a file, now it is in *Modified state*.
  + *Add* the modified files into *staging area*. Now the file is in the Staged State
* In Staging Area (temporary location for modified files)
  + Do *commit* the changes to the local repository, now the file is in *Committed State*.
* At GIT Repository (local repository)
  + Perform *push* operations to update the server repository permanently.

## Commands

Fork – A branch of a repository that can be worked on individually and merged later.

Clone

* Eclipse - import in with git-type project. Follow instructions to enter URI and credentials.
* Visual Studio Code - Ctrl+Shift+P, enter ‘git:clone’, enter https repository path, select local repository in empty folder in users>git

Commit

* Eclipse – Right click project in Project Explorer, Team>Commit.., Drag added files to stage box, write in commit message
* VS Code – Source Control, Stage changes, Enter commit message, click Commit, click Push.

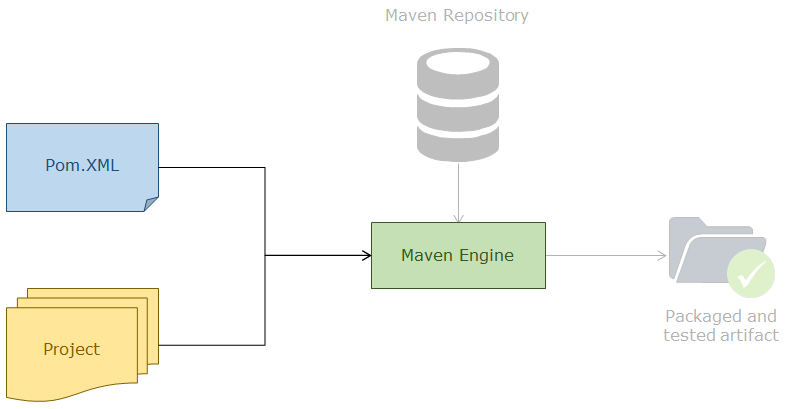
Pull

* Eclipse – Right click project in Project Explorer, ‘Team>Synchronize Project’, Team>Pull
* VS Code – Sync,

Merge

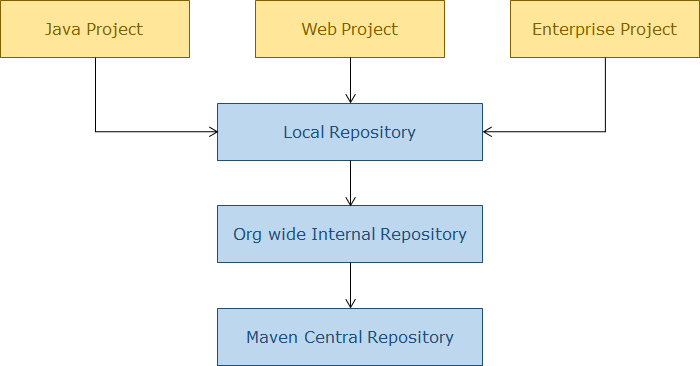
* Eclipse -- Right click project in Project Explorer, Team>Synchronize, Team>Pull, Team>Add to Index, Team>Commit
  + Conflicts need to be resolved in each conflicting file.
* VS Code – After merge conflict -> Sync, accept any changes, Stage, Commit, Push.

# Maven

Apache Maven is an intelligent, comprehensive project management tool used to automate and manage the build of a project (primarily Java projects), its reporting and documentation.

Advantages of Maven:

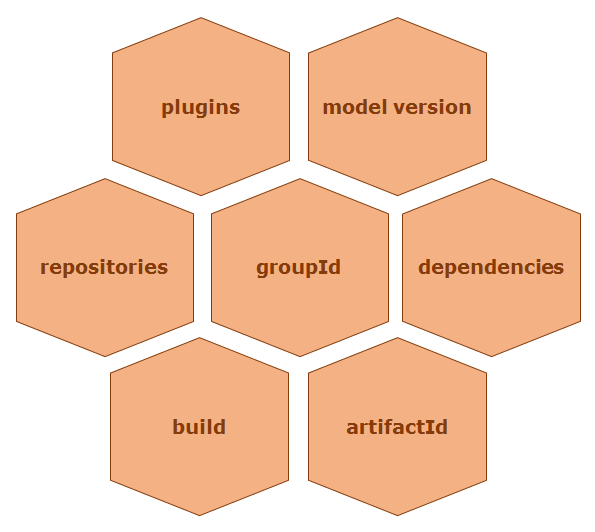
* Maven provides better *dependency management*, more powerful builds, better debugging and more consistent project structure.
* Maven helps us build *deployable executables* from the source code.
* It has build capabilities like creating pre-processing, compilation, packaging, testing and distribution.
* All Maven projects have a common structure, thus it is easier to understand each project.



## POM

Project Object Model

* POM is a declarative descriptor of project.
* POM describes the type of the project and provides the details about project name, customizations about the build for the project. Maven refers to POM before executing the goals.
* It also describes and defines various dependencies and adapts plugin activities.
* It also defines the associations among the project and prerequisites.



The minimal POM file should contain the following Project Information:

* Model Version: Tells about the version of the POM model being used in the project. Version 4.0.0 is compatible with Maven 2 and Maven 3.
* groupId: This element normally has group name or organization that owns the project. Ex: org.Infosys or com.infy.
* artifactId: A distinctive identifier to represent a single project. It is generally the name of the Project.
* version: This element distinguishes the particular version of project. “*Snapshot*” denotes active developments of projects
  + note: *groupId*, *artifactId* and *version* are also called as GAV co-ordinates or Maven co-ordinates
  + Convention for development and stable versions respectively: infycart-0.0.1 – SNAPSHOT, Spring Context - 4.3.14.RELEASE.jar

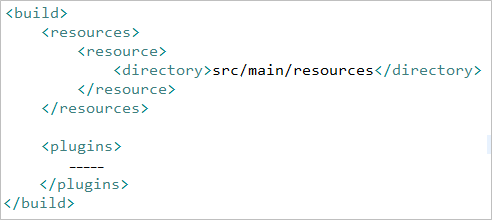
### Repositories Tag

* If the artifact (jar files, war, parent-pom) required to build the project is not available in the local maven repository, Maven redirects the artifact request to the org wide *repository*.
* If the dependencies are not found, the request will be *redirected* to central maven repository.
* This central repository available over the internet can be specified using the repository tag

### Dependencies Tag

* The build of a particular project is *dependent* on the artifacts such as jar, war and parent-pom.
* These essential artifacts are supplied using the *dependencies* tag.
* Maven is capable of locating the required artifact (jar, war, pom) in a remote/local repository based on Maven coordinates. Maven co-ordinates are used to define dependencies in the POM of project.
* The <dependencies> tag can contain multiple <dependency> tags
* <scope>
  + Compile – default, needed at compile,test, and run
  + Provided – available at compile,test,run but not designated for exporting.
  + Runtime - available at test, run
  + Test – available at test
  + System – like provided, does not retrieve from remote repositories.
  + import

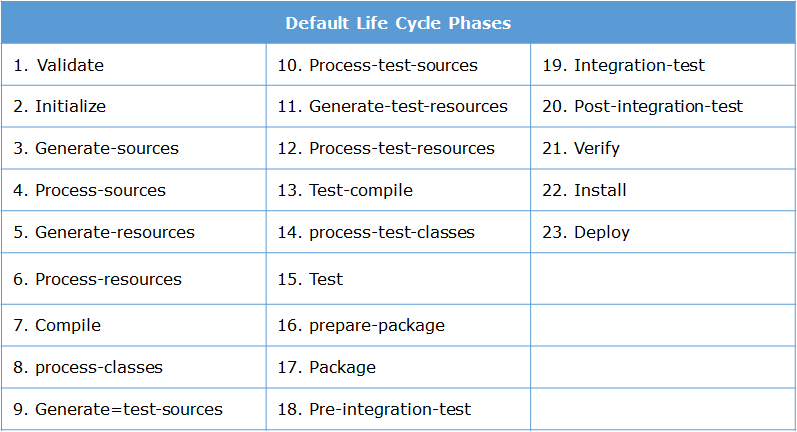
### Build tag

* Build tag is used to inform Maven about the Project Structure. Build tag is used to organize the plugins.
* **<resources>** will inform about non Java files such as database.properties, configuration.properties

### Plugins Tag

* Plugins tag provide support to create jar, war, ear files, compile, unit test code, generate project documentation etc.
* Plugins plays a major role in executing the goals of projects.
* Goal: A goal is a precise task (a small unit of work) to be executed as single or along with various other goals in making the build.
* One plugin can execute multiple goals.
* Goal is also called as MOJO (Maven plain Old Java Object).
* The sample plugin shown here is used to run all the test classes available within testjunit package.
* Name of the plugin used to run test classes is maven-surefire-plugin as defined in the <artifactId>
* Test classes will get executed during test phase of Maven life cycle as <phase> is defined as test
* **Cobertura -** cobertura-maven-plugin - Calculates the percentage of application code-base accessed by test-cases.
  + Has the following important goals:
    - **cobertura** - Instruments the compiled classes, executes the unit test-cases and generates a Cobertura report.
    - **clean** - Cleans up the files that cobertura-maven-plugin has created
* **Sonar** – sonar-maven-plugin – tests for code vulnerabilities.
  + Goals:
    - **sonar** - Analyzes the project. For this, SonarQube server must be started
    - **help** - Displays help information on sonar-maven-plugin
* **Javadoc** – maven-javadoc-plugin – Automates source code documentation
  + Goals:
    - **javadoc** generates the Javadoc files for the project. It executes the standard Javadoc tool and supports the parameters used by the tool.
    - **test**-**javadoc** generates the test Javadoc files for the project. It executes the standard Javadoc tool and supports the parameters used by the tool.

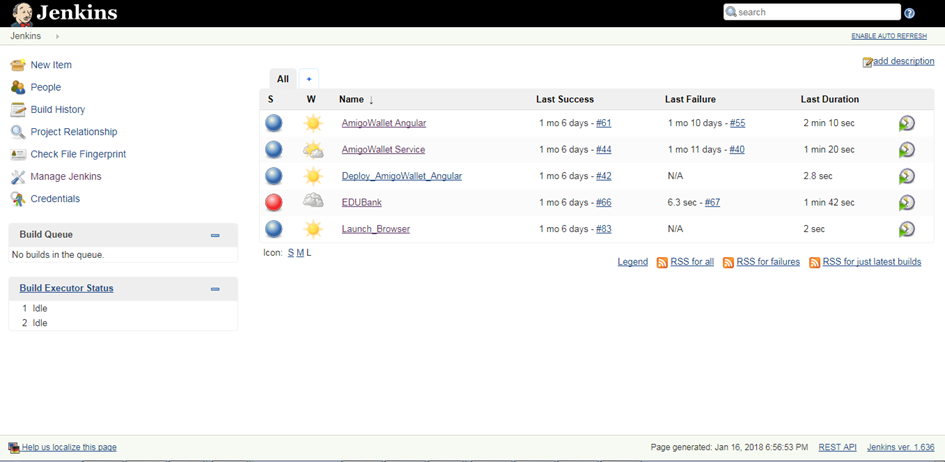
## Build Life Cycle

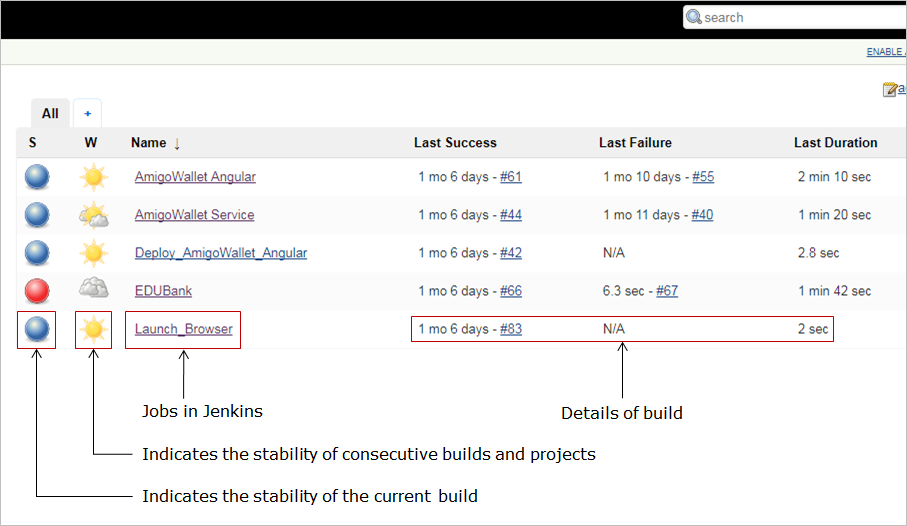
* Clean – Deals with project cleaning
  + Pre-Clean - executes tasks required prior to the actual cleaning of project
  + Clean - remove the outcomes generated by the previous build - target folder
  + Post-Clean - execute tasks required to finalize the project cleaning
* Site – Phases that help in deploying the project
  + Pre-Site - execute tasks needed prior to the actual project site generation
  + Site - generate the project's site documentation
  + Post-Site - execute processes needed to finalize the site generation, and to prepare for site deployment
  + Site-Deploy - deploy the generated site documentation to the specified web server
* Default – creating the project site’s documentation.
  + validate - It validates that the project is correct and all the necessary information is available for the build process to complete.
  + compile - Compilation of source code is done in this phase
  + test - The compiled source code is tested using suitable test framework (eg: JUnit). Reports can be viewed inside ${basedir}/target/surefire-reports folde
  + package - The compiled code is packaged into a JAR/WAR.
  + verify - Run checks on the integration test results to make sure that the quality criteria are met.
  + install - It installs the package into the local/remote Maven repository.
  + deploy - The final package is copied to the remote repository.
  + If a build phase is requested for execution then all the build phases up to and including that phase will execute sequentially.
    - Example: If phase install is requested then all the build phases before and including install phase (validate, initialize…install) will be executed sequentially.

## Maven Build

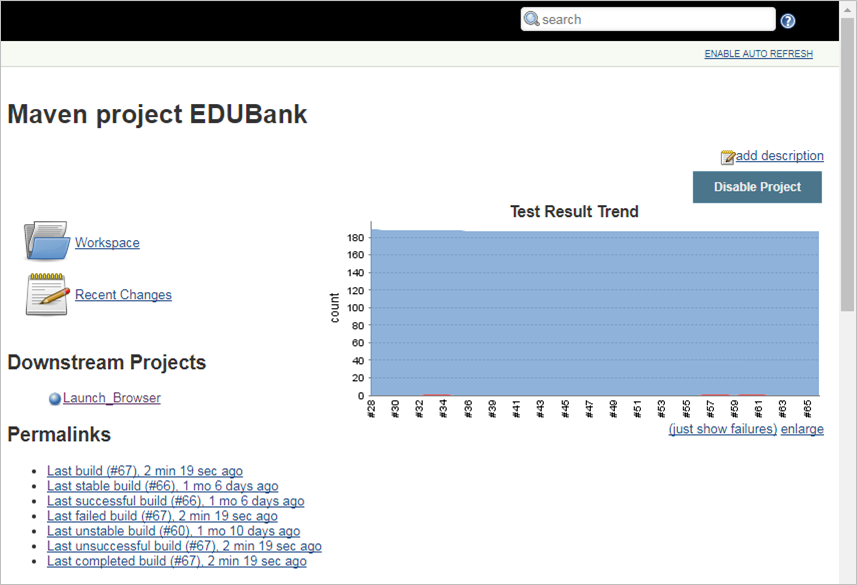
* Package Explorer > Project\_Name > Run As > Maven Build… > Goals:”clean install” > Apply & Run
  + “Clean install’ deletes previous build data from the target data folder present in the project root folder.
* Target > site > open with web browser: Details PMD violations in project.
* Jacoco-ut > index.html > open with web browser: Details code coverage
* **Goals**: logical, small, precise unit of work. Eg: clean goal in maven-clean-plugin
  + maven-compiler-plugin : has goals for compiling source code and unit tests
  + maven-surefire-plugin : has goals for execution of unit tests and publishing its reports
  + maven-jar-plugin : has goals for creating JAR files
  + maven-clean-plugin : has goal for the deletion of target folder

# Jenkins

* Jenkins is an open source Continuous Integration server written in Java
* It provides the user to configure jobs, also referred as *builds*.
* Each build will contain a task related to a software or step in the project lifecycle. These can be linked together thereby forming a *pipeline workflow*.
* Jenkins triggers a particular build depending on the *trigger* *criteria* provided by user.
* To perform a particular task, one should ensure that the related software plugin is added to the Jenkins.
* All the executions will be triggered from Jenkins. As it is the central tool which holds all other tools involved in Continuous Integration process, it is known as CI server
* The page gives details about the jobs created.
* It also provides options for the different functionalities to be performed



The details about the *jobs* created, including the *name* of the job, *stability* of the build and *time details* of build are explained in the image.



Following image shows the details of the job selected from the home page

*Using Jenkins*:

1. Programmer *commits* the code to the source code control(SCM) repository
2. Jenkins *polls the SCM repository for changes* and triggers a build action
3. *Build* action executes the Maven goals
4. Jenkins performs the *post-build* actions as configured
5. Jenkins *generates reports* and moves artifacts created by jobs as per post build configuration

Let us see the best practices to be followed in the process of CI/CD.

* Every developer must *commit the code* at least once in a day
* Developer must not submit the broken code.
* Developer must ensure that 100% of test cases must pass for every build
* Developer must fix the failed builds on the highest priority
* Developer should review the reports generated by the build
* Developer must write automated tests
* Developer must run private builds before committing code to central repository
* Developer must take into consideration the automated deployment

Run Jenkins:

1. C:/FSM\_Jar and Servers/Tomcat/bin/startup
2. Open browser to localhost:3333/Jenkins
3. New Items > Maven Project > Name > Git SCM URL > Add credentials > username/password > Build Triggers (Poll SCM ) > “H/2\*\*\*\*” > Pre Steps > Add pre-build step > Execute Windows batch command “cd “”C:\jenkins\_configure” Jenkins\_pre.bat 4000”” > Build Root POM “Application/pom.xml” > Goals and options = “clean install” > add post-build action > script > C:\jenkins\_configure\jenkins.bat “ServerSide\_InfyMovie” InfyMovieApp-0.0.1-SNAPSHOT > Apply > Save
4. Manually: Build Now option

# Azure Deployment

**Cloud computing** is a process of using *remote servers* via the internet for storing, managing and processing the data.

* In Cloud computing; the computing hardware and software resources that you need to process your tasks are provided for you, "as a service" over the internet, by a vendor instead of you owning and maintaining them.
* It is the responsibility of the vendor, the Cloud Service Provider (CSP), to develop, own and maintain these resources and make them available to the consumers over the internet.
* Some famous cloud across the globe are Azure by Microsoft, AWS by Amazon, Oracle cloud by Oracle, Softlayer by IBM

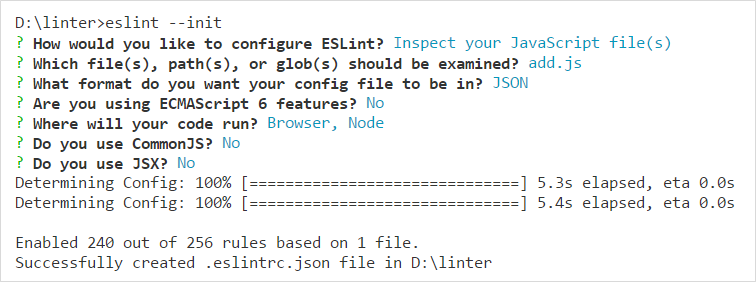
Azure Resource Group

* Resource groups enable you to manage all your resources in an application together.
* Resource groups are enabled by Azure Resource Manager.
* Resource Manager allows you to group multiple resources as a logical group which serves as the lifecycle boundary for every resource contained within it.
* Typically a group will contain resources related to a specific application.
  + For example, a group may contain a Website resource that hosts your public website, a SQL Database that stores relational data used by the site, and a Storage Account that stores non-relational assets.

Creating an Azure MySQL Database Server Instance

* Create a resource > Azure Database for MySQL

# ESLint

Enforces good coding practices with javascript files

* Install: npm install eslint –save-dev
* Create configuration file: eslint –init
* Run: eslint “javascript\_File”