**ASSIGNMENT-15.2:**

Question : Explain the working and the differences between Maven, Gradle and SBT in detail.

**MAVEN :**

It was released in 2004. Its goal was to improve upon some of the problems developers were facing when using Ant. Maven continues using XML as the format to write build specification. However, structure is diametrically different. While Ant requires developers to write all the commands that lead to the successful execution of some task, Maven relies on conventions and provides the available targets (goals) that can be invoked. As the additional, and probably most important addition, Maven introduced the ability to download dependencies over the network (later on adopted by Ant through Ivy). That in itself revolutionized the way we deliver software.

However, Maven has its own problems. Dependencies management does not handle well conflicts between different versions of the same library (something Ivy is much better at). XML as the build configuration format is strictly structured and highly standardized. Customization of targets (goals) is hard. Since Maven is focused mostly on dependency management, complex, customized build scripts are actually harder to write in Maven than in Ant.

Maven configuration written in XML continuous being big and cumbersome. On bigger projects it can have hundreds of lines of code without actually doing anything “extraordinary”. Main benefit from Maven is its life-cycle. As long as the project is based on certain standards, with Maven one can pass through the whole life cycle with relative ease. This comes at a cost of flexibility.

In the mean time the interest for DSLs (Domain Specific Languages) continued increasing. The idea is to have languages designed to solve problems belonging to a specific domain. In case of builds, one of the results of applying DSL is Gradle.

**GRADLE:**

Gradle combines good parts of both tools and builds on top of them with DSL and other improvements. It has Ant’s power and flexibility with Maven’s life-cycle and ease of use. The end result is a tool that was released in 2012 and gained a lot of attention in a short period of time. For example, Google adopted Gradle as the default build tool for the Android OS.

Gradle does not use XML. Instead, it had its own DSL based on [Groovy](http://groovy.codehaus.org/) (one of JVM languages). As a result, Gradle build scripts tend to be much shorter and clearer than those written for Ant or Maven. The amount of boilerplate code is much smaller with Gradle since its DSL is designed to solve a specific problem: move software through its life cycle, from compilation through static analysis and testing until packaging and deployment.

Initially, Gradle used [Apache Ivy](http://ant.apache.org/ivy/) for its dependency management. Later own it moved to its own native dependency resolution engine.

Gradle effort can be summed as “convention is good and so is flexibility”.

**SBT:**

**SBT** is an [open Source](https://en.wikipedia.org/wiki/Open_source) [Build Tool](https://en.wikipedia.org/wiki/Build_tool) for [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)) and [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) projects, similar to Java's [Maven](https://en.wikipedia.org/wiki/Apache_Maven) or [Ant](https://en.wikipedia.org/wiki/Apache_Ant).

The functionality of sbt can be extended through a plugin architecture. A dedicated website was set up for community contributed plugins, which cover various areas such as signing, packaging, publishing and releasing artifacts, connecting to other services such as blogs and databases, or integrating with other technologies such as deploying to the Android platform.

There are plugins to automatically create project files for the Eclipse and IntelliJ IDEA IDEs. On the other hand, an IntelliJ IDEA plugin allows the sbt console to be integrated into IDEA, and projects can choose to use sbt for building.

Its main features are:

* continuous compilation, testing, and deployment
* integration with the Scala interpreter for rapid iteration and debugging
* support for mixed Java/Scala projects
* native support for compiling Scala code and integrating with many Scala [test frameworks](https://en.wikipedia.org/wiki/Test_automation)
* build descriptions written in Scala using a [DSL](https://en.wikipedia.org/wiki/Domain_Specific_Language)
* dependency management using [Ivy](https://en.wikipedia.org/wiki/Apache_Ivy) (which supports Maven-format repositories)