# Build Tools

## Introduction

A build tool is a software application designed to streamline and automate source code compilation, testing, packaging, and deployment with minimal manual intervention. It enhances developer productivity, allowing developers to allocate more time to core software development tasks like coding rather than getting caught up in routine and time-consuming processes.

Build tools are essential in managing and orchestrating various tasks required to produce a working software application.

## A. Key Aspects Of A Build Tool

1. **Automation**: A build tool automates repetitive and time-consuming tasks in the software development process. They reduce the chances of human error and ensure consistent builds.

2. **Dependency Management**: A build tool manages project dependencies. They download, manage, and configure libraries and frameworks needed for an application.

3. **Compilation**: A build tool compiles the source code, converting it into executable code or an intermediate representation.

4. **Packaging**: A build tool packages the compiled code and other required resources into a deployable format (e.g., JAR, WAR, or Docker image).

5. **Testing**: A build tool may execute testing suites and report the results, ensuring code quality.

6. **Deployment:** Some build tools can automate the deployment of applications to various environments, such as development, testing, and production.

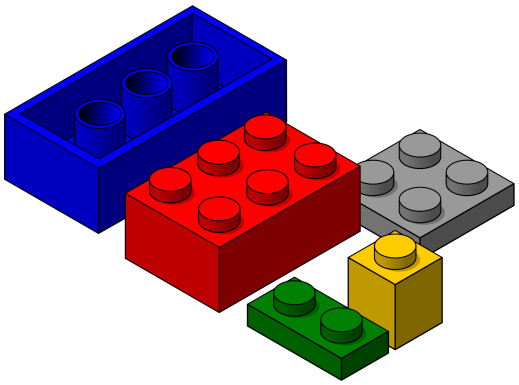
7. **Task Automation:** A build tool allows the creation and execution of custom build tasks, catering to project-specific requirements.

## B. How a Build Tool Works (An Analogous Understanding)

Let us understand how a build tool works by using a real-life example to have a proper understanding.

Imagine you are a master builder working with LEGO bricks to create unique structures and models. Building with LEGO bricks can be like writing computer programs, and here's how it relates to how a build tool works.

1**. LEGO Pieces (Source Code):**



In software development, your code is like a set of LEGO pieces. Each piece represents a part of your program, like a function or a class.

2**. Building Instructions (Build Script):**

To create something incredible with LEGO, you need building instructions that tell you how to arrange the pieces. These instructions are like your "build script" in programming.

**3. Manual Building (Manual Compilation):**

Imagine following the LEGO instructions by hand, carefully connecting each piece individually. This resembles how you might compile your code without a build tool. It's doable but takes a lot of time and attention to detail.

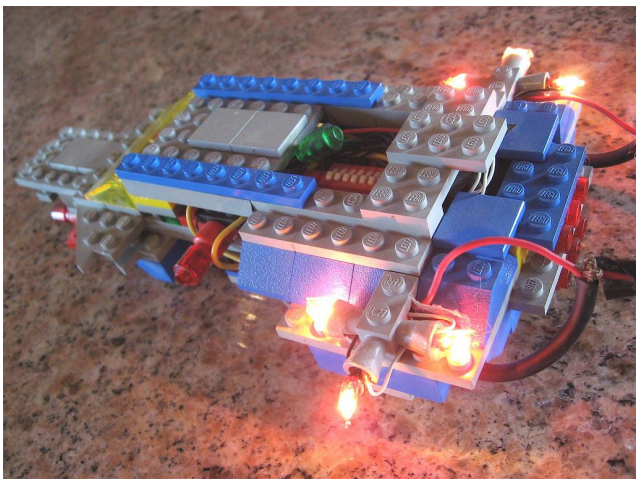
**4. Building Tool (Build Tool):**

Now, think of having a magical LEGO building tool. You give it your LEGO pieces and the building instructions, and it assembles your model. This tool is like a "build tool" in software development.

**5. Quick Assembly (Automated Compilation):**

Creating your model with the LEGO building tool is much faster and less error-prone. It knows how to arrange the pieces just right, so you don't have to worry about making mistakes. Similarly, a build tool compiles your code automatically, so you don't have to do it manually.

**6. Adding Extra Features (Plugins):** Sometimes, you want to add cool features to your LEGO creation, like lights or moving parts. For this, you might use unique LEGO add-ons. In the software world, we have "plugins" that you can add to your build tool to do extra tasks, like testing your code or creating a web version of your program.



**7. Building a Castle (Complex Project):**


Imagine building something complex, like a LEGO castle with hundreds of pieces. Doing this by hand could be a huge task. With the LEGO building tool, it's much more manageable. Similarly, in software development, a build tool helps organize, compile, and manage everything efficiently for large and complex projects.

**8. Saving Time and Effort:**

So, just like a LEGO building tool saves you time and effort in creating unique LEGO models, a build tool in programming automates many tasks, like compiling your code, managing project dependencies, and even deploying your software. It makes the development process smoother, faster, and less error-prone.
A build tool is like having a LEGO-building assistant that assembles your code pieces into a working program, saving you time and making software development more fun and efficient, just like creating awesome LEGO creations with ease.

However, many different build tools are used based on the project's requirements; we will look at the two widely used build tools, namely Maven and Gradle.

## C. Maven Build Tool

Maven is an open-source project management tool designed to facilitate the creation of software projects throughout their lifecycles. Its primary emphasis lies in standardizing the software development process, allowing for the rapid construction of applications in a consistent framework. While it is commonly used for Java projects, it also offers compatibility with various other programming languages. Maven employs Extensible Markup Language (XML) to structure applications.

**Key Concepts and Components:**

1. **Project Object Model (POM):** Maven's heart is the POM file (Project Object Model), represented in XML format. The POM file defines the project structure, configuration, and dependencies. It serves as the blueprint for the project, capturing essential project information.

2. **Dependency Management:** Maven simplifies the management of project dependencies. It maintains a central repository (Maven Central Repository) that houses many libraries and frameworks. Developers specify project dependencies in the POM, and Maven automatically resolves and downloads the required JAR files.
   A sample Lombok dependency definition in the pom.xml file of a project built with Maven looks like this:

```
<dependency>
     <groupId>org.projectlombok</groupId>
     <artifactId>lombok</artifactId>
     <optional>true</optional>
   </dependency>
```

3. **Conventions and Best Practices:** Maven promotes "convention over configuration." It enforces best practices and project structure conventions, allowing developers to focus on coding rather than creating complex build scripts. A consistent project structure aids collaboration and ensures uniformity.

4. **Plugin Ecosystem:** Maven boasts a rich ecosystem of plugins that extend its functionality. These plugins execute tasks such as compiling code, running tests, packaging applications, and more. Plugins enable integration with other tools and frameworks, making them versatile for different development needs.

5. **Build Lifecycle and Phases:** Maven organizes the build process into lifecycles, each composed of phases. Example lifecycles include "clean," "validate," "compile," "test," "package," and "install." Developers can execute specific phases or the entire lifecycle.

6. **Central Repository:** Maven relies on a central repository for storing and retrieving project dependencies. It connects to repositories like Maven Central to download required libraries. This approach ensures consistency, reliability, and accessibility of dependencies.

## D. Advantages Of Using Maven

- **Consistency**: Maven enforces project structure and best practices, fostering consistency across development teams.
- **Dependency Management**: Simplifies project dependencies' inclusion and version management, reducing potential conflicts.
- **Build Lifecycle**: Maven defines a clear build lifecycle with predefined phases, making it easier to manage the build process.
- **Plugin Ecosystem**: Offers various plugins for integrating with other tools and frameworks, enhancing project functionality.
- **Reusability**: Maven supports project sharing, enabling teams to reuse components in different projects.

## E. Disadvantages Of Using Maven

- **Limited Flexibility**: The convention-over-configuration simplifies development but may limit customization options for complex build scenarios.
- **Performance**: Downloading dependencies may need to be faster, mainly due to network latency.
- **Plugin Compatibility**: Not all plugins are regularly updated, leading to compatibility issues with newer versions of Maven.

## F. Gradle Build Tool

Gradle is an open-source build automation and project management tool for flexibility and performance. It is popular for building and managing software projects supporting multiple programming languages.

**Key Concepts and Components of Gradle Build Tool:**

1. **Build Scripts:** Gradle build scripts are typically written in Groovy or Kotlin, providing a high level of flexibility in configuration. Build scripts define tasks, dependencies, and custom-build logic for the project.

2. **Build File (build.gradle)**: The *build.gradle* file is the core configuration file in Gradle. It defines project settings, dependencies, and tasks. It can be highly customised to meet specific project requirements.

3. **Dependency Management:** Gradle simplifies dependency management by allowing developers to declare dependencies in the build file. A sample dependency for Lombok in the *build.gradle* file looks like this:

```
dependencies {
  implementation("org.springframework.boot:spring-boot-starter")
  compileOnly("org.projectlombok:lombok")
  annotationProcessor("org.projectlombok:lombok")

testImplementation("org.springframework.boot:spring-boot-starter-test")
}
```

4. **Task-Based Build System:** Gradle is known for its task-based approach. Tasks are units of work that can be executed individually or as part of a more extensive build process. Developers can create custom tasks to meet project requirements.

## G. Advantages of Gradle

- **Flexibility**: Gradle's build scripts are highly customisable, making them suitable for a wide range of projects, from simple to complex.

- **Performance**: Gradle is known for its efficient and fast build process. It uses incremental builds to avoid unnecessary recompilation.

- **Dependency Management:** It simplifies dependency resolution and management, supporting various repositories and dependency sources.

- **Plugin Ecosystem:** A vast selection of plugins enhances Gradle's capabilities, allowing integration with popular frameworks and tools.

- **Polyglot Support:** Gradle supports multiple programming languages, making it versatile for various projects.

## H. Disadvantages of Gradle

- **Learning Curve:** Building tasks with Gradle demands high technical proficiency.

- **Complexity**: Advanced customization can lead to complex build scripts that may be difficult to maintain.

- **Resource Intensive**: Gradle's performance may vary based on project size and complexity, and it can be resource-intensive for huge projects.

## I. Difference Between Maven and Gradle

| Basis | GRADLE | MAVEN |
|---|---|---|
| Language | Built on Groovy or Kotlin, offering flexibility. | Primarily based on XML for configuration. |
| Flexibility | Highly customizable, with no strict structure enforced. | Enforces a predefined project structure. |
| Dependency Management | Provides flexibility in dependency management. | Strong dependency management using POM files. |
| Build Efficiency | Efficient with incremental builds and optimized tasks. | Slower due to complete project rebuilds at times. |
| Project Complexity | Offers excellent flexibility for complex projects. | More structured and suitable for simpler projects. |
| Language Support | Supports Java, C, C++, Groovy, and more. | Primarily Java, but it supports other languages. |
| Build Approach | Task-based approach with customized lifecycles. | Predefined lifecycle phases (clean, compile, etc.). |

## Conclusion

In conclusion, building tools play a critical role in modern software development by automating tasks, managing dependencies, and improving the efficiency and reliability of the development process. They are widely used across different programming languages and platforms to streamline the building and deployment of software applications.

## References


1. Working of Maven
2. Features of Maven
3. Gradle in Android development.
4. Gradle
5. Differences between Maven and Gradle