BUILD AUTOMATION TOOLS FOR SOFTWARE DEVELOPMENT
A COMPARATIVE STUDY BETWEEN MAVEN, GRADLE AND BAZEL

Mridula Prakash

L&T Technology Services, CTO Office, Mysore, India

ABSTRACT

The automated processes will play an increasingly vital role in continuous integration as the pace of design and development of new software picks up. With the importance of software build automation tools taking center stage, the present paper undertakes a comparative analysis of three best available solutions - Maven, Gradle and Bazel. We aim to evaluate the solutions based on their efficiency and performance in the context of software build automation and deployment. There are some fundamental differences in the way each tools approach builds. The aim of this study is also to provide the reader with a complete overview of the selected build automation tools and, the relevant features and capabilities of interest. In addition, the paper leads to a broader view on the future of the build automation tools ecosystem.

KEYWORDS

Automated process, Build automation tools, Maven, Gradle, Bazel.

1. INTRODUCTION

The build automation process involves automating tasks about software build, including the compilation of source code into binary code, packaging the binary code, and running the automated tests; as the final procedure. The process helps in reducing down time, optimizing costs, and simplifying the overall development process. Over the years, given the growing demand for software development globally, a series of tools have emerged to provide a streamlined continuous integration framework.

Build automation tools can be broadly classified into two types namely:

1. Build automation utility
2. Build automation servers

The build automation utilities consist of a range of solutions, including, Gradle, Maven, Bazel, and Cmake. Their primary purpose of these tools is to generate software builds by compiling and linking the source code. The build automation server, on the other hand, comprises of continuous integration-based web servers, with instances including continuous management tools, and continuous integration tools. We first provide a comprehensive overview of the emerging tools like Gradle, Maven and Bazel and then proceed to summarize with the latest trends and the way forward.

David C. Wyld et al. (Eds): EMSA, SEA, AIFU, NLCAI, BDML, BIoT, NCOM, CLOUD, CCSEA, SIPRO - 2022
pp. 73-89, 2022. CS & IT - CSCP 2022 DOI: 10.5121/csit.2022.120608

1.1. How to Select the Best Build Automation Tool?

An organization needs to select an appropriate tool because it will save a lot of money and time along with providing the best quality outcome. Before selecting an automation tool everyone has to go through the following simple steps,

- Step 1: Collect all the requirements.
- Step 2: Categorize your requirements (basic, technical, business, feature).
- Step 3: Create a list of tools against basic requirements.
- Step 4: Shortlist tools against other requirements.
- Step 5: Create a chart for comparison.
- Step 6: Scorecard.

The initial step is to collect all the requirements that are needed for our project. The next step is to categorize the requirements i.e split the requirements as basic, technical, feature and business.

- Basic Requirements deal with the basic needs of the project like what type of product or application we are going to build and the ease of execution.
- Technical Requirements consists of system requirements along with technical details like microprocessor details, configuration, the platform used and operating system specifics.
- Business Requirements concentrate on the cost, budget, time and deadline of the project.
- Features Requirements are all about what kind of feature we are looking for like alarm descriptions, feature from user perspective, user interface details, integration and reporting.

After categorizing the requirements, the tools must be mapped with all types of requirements to make sure whether the tool is satisfying all the needs.

At last, a comparison chart has to be plotted using all the details and generate the scorecard which will let us know which tool is best suited for our project.

Open source and multi-language, multi-module supported tools like Gradle, Maven and Bazel are becoming very popular in continuous integration. In this paper, we present an in-depth view into the framework and the specification of each of the tools to understand their feasibility and applicability across projects.

2. MAVEN

Maven is an open-source Java-based build automation tool, used mostly for the creation of Java projects. The tool was created by Jason Van Zyl in 2002, with its first release on July 13, 2004 by the Apache Software Foundation.

The tool can also be used to build and manage projects in other languages such as Ruby, Scala and C# respectively. The working mechanism of Maven consists of usage of declarative speech approach, where the project structure and the contents are described in the form of the project object model.

Maven attempts to apply patterns to any project's build structure. It promoted comprehension and productivity by making use of the best practices. It acts as a management tool in managing the documentation, builds and dependencies.

2.1. Project Creation in Maven

STEP-1: Download the Maven software from the official website:
https://maven.apache.org/download.cgi
STEP-2: Install the software by changing the appropriate environment variables.
STEP-3: Set path in the root directory or directory of your choice.
STEP-4: Check for successful installation using the command maven -v.

2.2. Maven Build Phases

Maven build lifecycle goes through a series of stages, called as default build phases – validate, compile, test, package, verify, install and deploy. These phases are executed sequentially when we run a maven build command. Let us take an example to understand Maven in detail.

Build a java project using maven

Maven supports two ways to build the project as below:

1. Using eclipse
2. Using command prompt

Using Eclipse

Create a Maven project and then check for the POM file which is very important for automation. In the POM file, we add the dependencies. As mentioned in Figure 1, we can take the dependencies from the system which we are using or we can use online repositories. Usually java projects use the test frames like junit or testing. We can use any test frame of our choice.

Example dependency used in my Pom file.

```
<!-- https://mvnrepository.com/artifact/org.testng/testng -->
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>7.4.0</version>
            <scope>test</scope>
        </dependency>
```

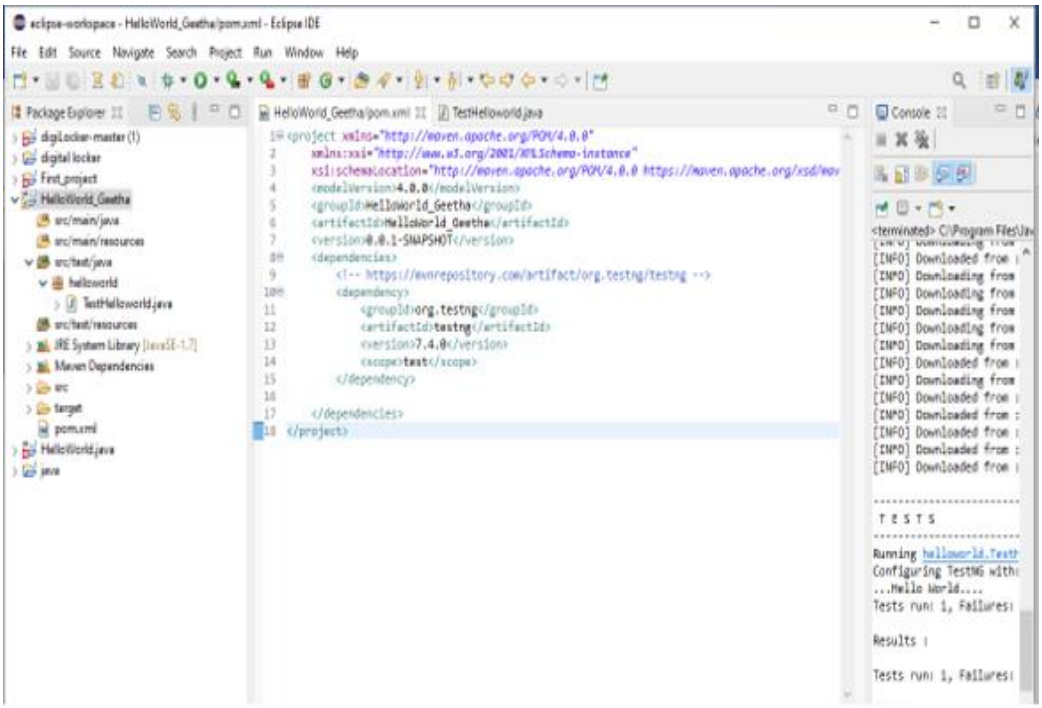

Figure 1: Adding dependencies to the POM file

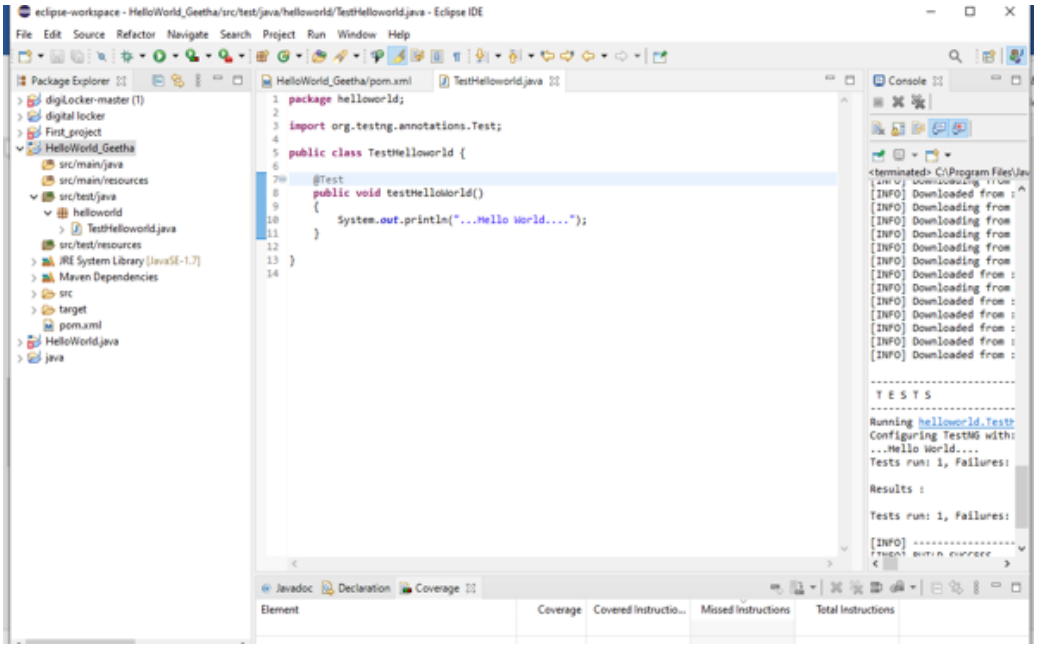

Figure 2: Project class creation in Eclipse

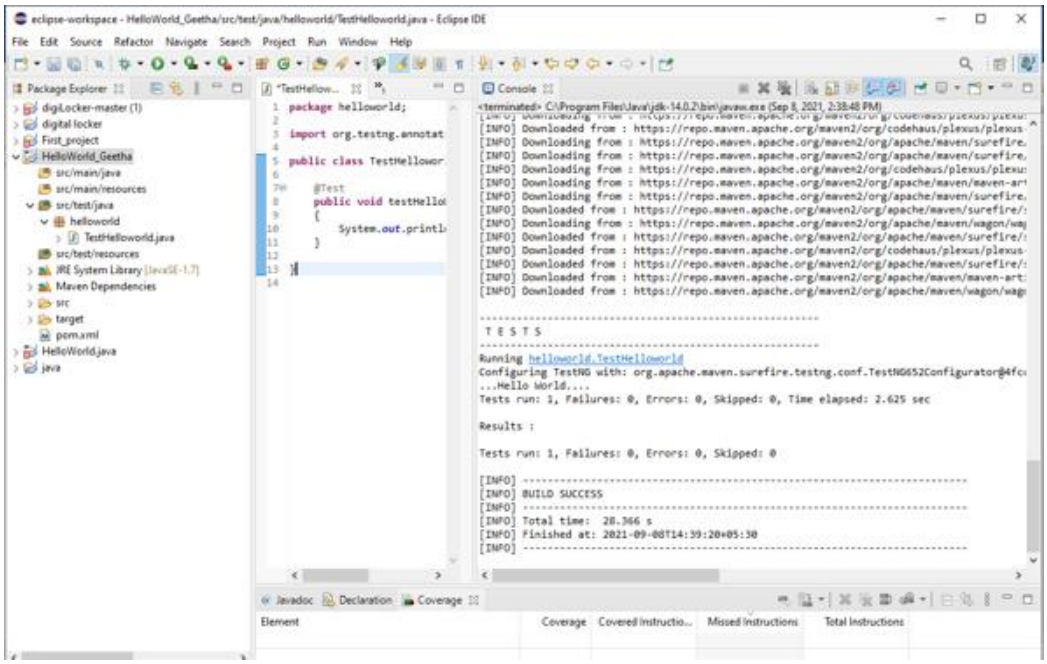

Figure 3: Project build compilation in Eclipse using Maven

To verify we can check using the command prompt. Go to the folder in which our project is saved and then use the maven test command.

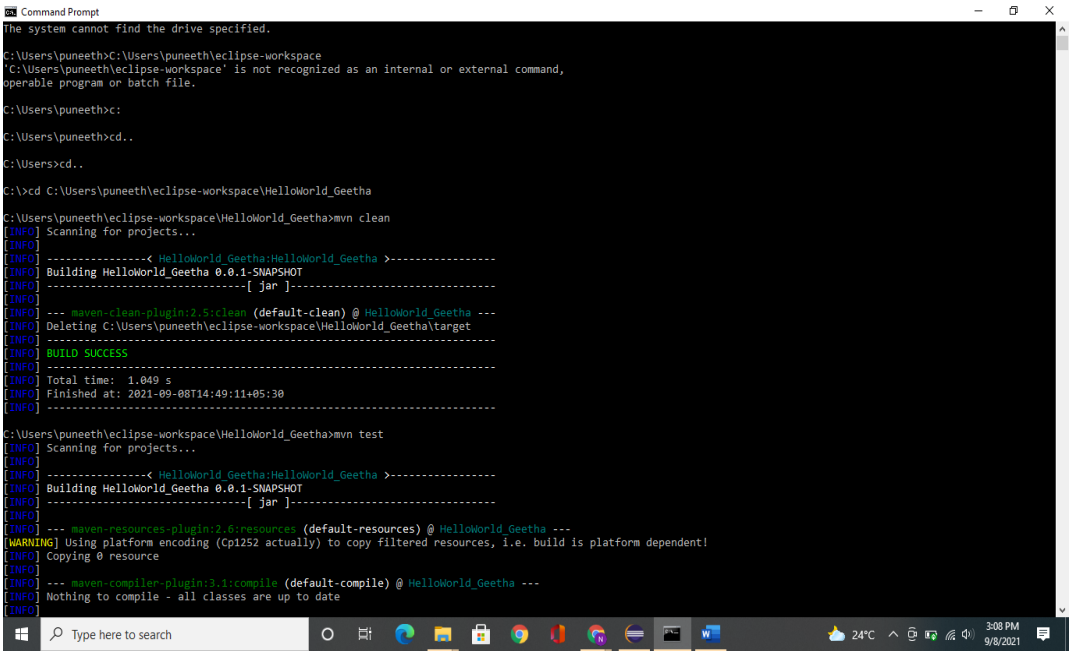

Figure 4: Project build compilation in command prompt using Maven

3. GRADLE

Gradle began as an open-source project led by Hans Docker and Adam Murdoch. The initial success of the Gradle build tool project paved the way for the establishment of the Gradle Enterprise in 2017.

As a build automation tool, Gradle supports multiple-language software development. The tool controls various processes involved in the software development processes. It automates processes such as the tasks of compilation, packaging, testing, deployment and publishing as well. The languages supported by the Gradle build automation tool, include:

- Java
- Kotlin
- Groovy
- Scala
- C/C++, and
- JavaScript

The working mechanism of the Gradle tool consists of inclusion of directed acyclic graph to detect the order in which each of the tasks are running. The tool operates by leveraging certain series of build tasks which can run either serially or in parallelly. An important part of any build tool is the ability to avoid doing work that has already been done. It supports incremental builds by reading the updated parts of the build tree. You need to tell Gradle which task properties are inputs and which are outputs. If a task property affects the output, be sure to register it as an input, otherwise the task will be considered up to date when it's not. Gradle supports cache conflicts and dependency management of the builds. It also supports multi-project organization.

3.1. Project Creation in Gradle

STEP-1: Download the Gradle software from the official website - https://gradle.org/install/
STEP-2: Install the software by changing the appropriate environment variables.
STEP-3: Set path in the root directory or directory of your choice.
STEP-4: Check for successful installation using the command gradle -version.
STEP-5: Once details of Gradle version is displayed, create a build using the gradle init command.
STEP-6: On pressing enter, select the language of your choice for project creation.
STEP-7: Project starts generated corresponding to the program in the build. Gradle file.
STEP-8: Project is successfully created in Gradle.

The building process includes compiling, linking and packaging the code. The tool is supported mostly for a groovy-based domain. This tool provides building, testing and deploying of the software. It is used to build any software and large projects. Gradle mainly focuses on maintenance, performance and flexibility.

3.2. Gradle Build Phases

A Gradle build has three distinct phases.

1 **Initialization**: Gradle supports single and multi-project builds. During the initialization phase, Gradle determines which projects are going to take part in the build, and creates a Project instance for each of these projects.

2 **Configuration**: During this phase, the project objects are configured. The build scripts of all projects which are part of the build are executed.

3 **Execution**: Gradle determines the subset of the tasks, created and configured during the configuration phase, to be executed. The subset is determined by the task name arguments passed to the Gradle command and the current directory. Gradle then executes each of the selected tasks.

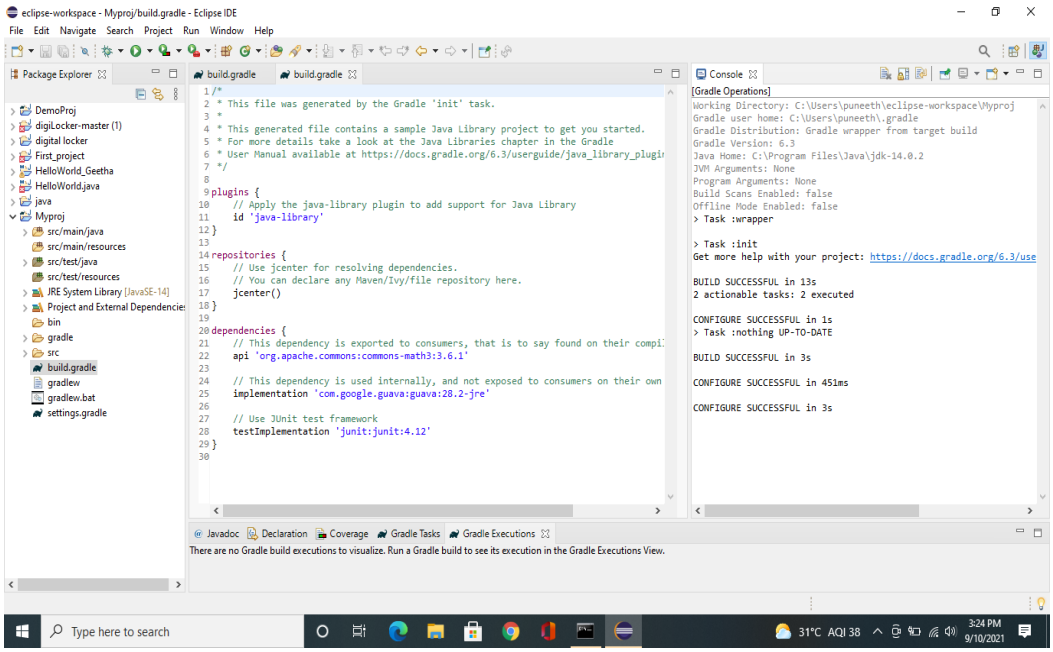

Figure 5: Project build compilation in eclipse using Gradle

Gradle project creation is performed using eclipse and the results are as shown in the above picture. The process to build the application using Gradle is almost the same as Maven. The difference is the Maven Project will be built only after modifying the POM file, but in Gradle, the application is built just after the creation of the project.

Gradle has many options in the command prompt to select the project details and all other requirements to start the build process. The command used to start the build process in Gradle is “Gradle init”.

After entering this command in the command prompt, it will initialize the task and starts showing all the options one by one and we have to select what are the requirements needed to build the application.

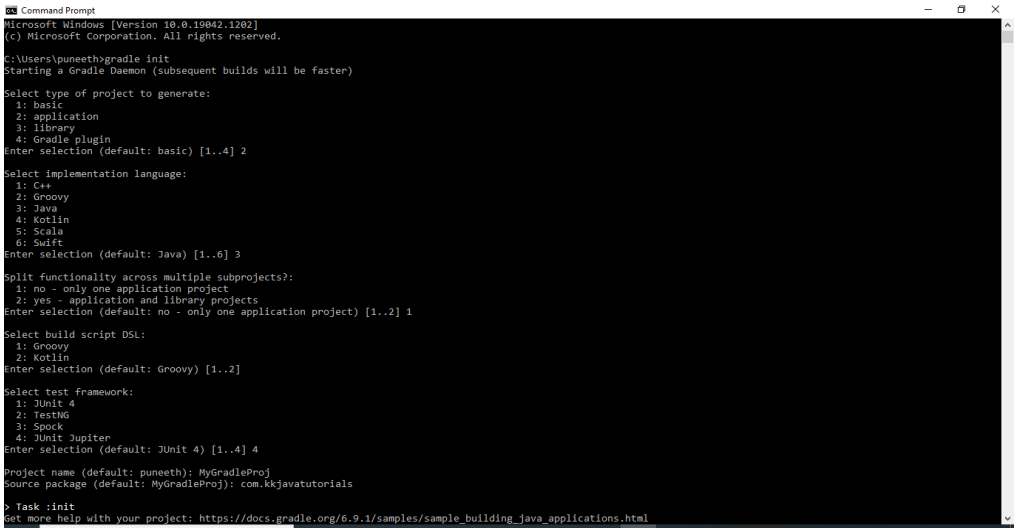

Figure 6: Options display to build a process in Gradle tool.

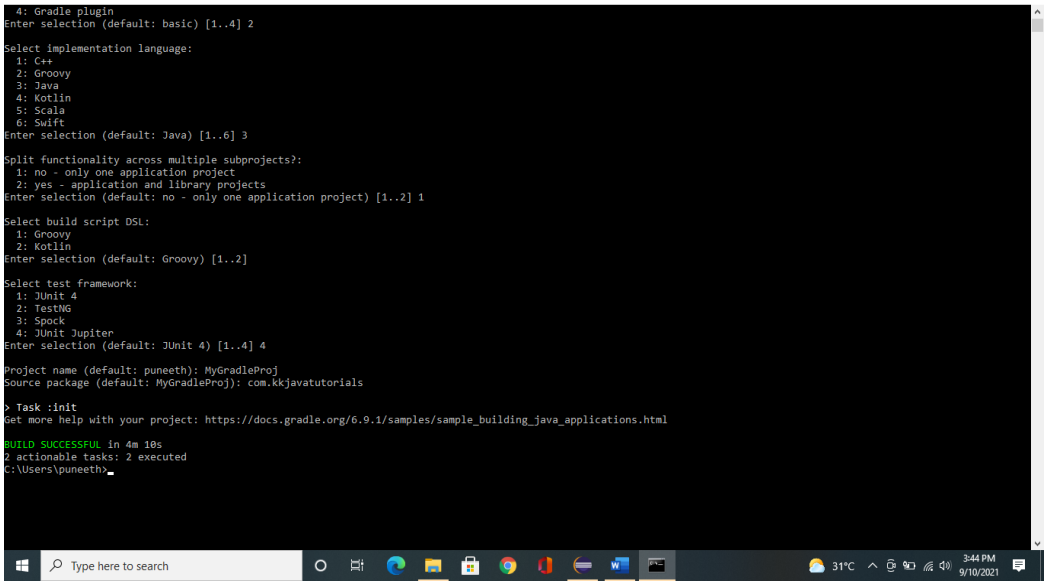

Figure 7: Complete testing of compilation process and build creation in Gradle tool.

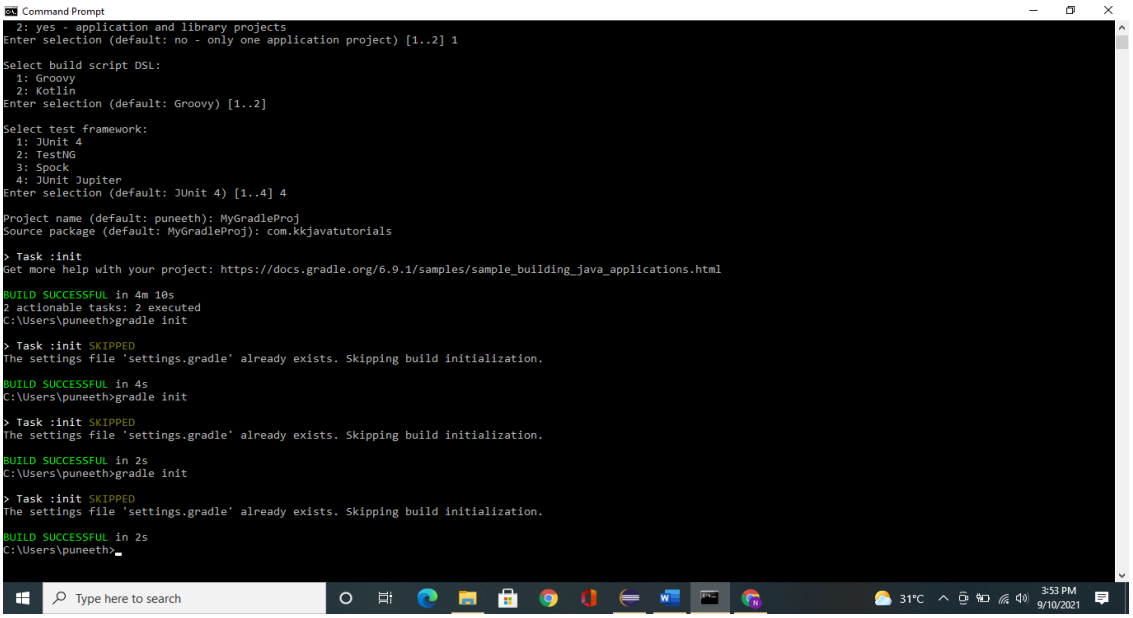

Figure 8: Complete testing of compilation process and build creation in Gradle tool.

If observed the build compilation time in Gradle is more than Maven in the command prompt, this is due to the extra time consumed while opting for the requirements and automatic cache cleaning function of Gradle when a new application is started.

In Eclipse IDE, we can see that the build time of Gradle is very less compared to Maven.

4. BAZEL

Bazel is an open-source version of Google's internal solution, Blaze which is used for automating software processes within the search engine giant. The tool, an anagram of Blaze was launched in March 2015.

In terms of build automation tools, Bazel is similar to Apache Maven and Apache Ant. It provides complete automation in build and testing of the software, across platforms. The working mechanism consists of building of software applications from the source-code by following a certain a set of rules. These rules are created in Starlark Language which serves as dialect of the Python programming language.

The software application packages produced by Bazel capable of being integrated with Android and iOS platforms, thereby paving way for real-time applications. Built-in rules are present in Bazel for building software applications with various types of language support namely: C++, Java, Go, Python, Objective-C and Bourne Shell script respectively.

Bazel leverages parallelization techniques to speed up the process of build and software application package creation as well. New build rules can be written in Bazel so as enhance cross-platform integration. The tool also provides plugin-in support with the following IDEs: IntelliJ, Android Studio, and CLion.

4.1. Project Creation in Bazel

STEP-1: Download the Bazel software from the official website.https://docs.bazel.build/versions/main/install-windows.html
STEP-2: Install the software by changing the appropriate environment variables.
STEP-3: Set path in the root directory or directory of your choice.
STEP-4: Check for successful installation using the command bazel -v.
STEP-5: Once details of Bazel version is displayed, create a build file with sub file called planning.java.
STEP-6: Create a sub-directory with a java file as source.java
STEP-7: Once build is created test the builds.
STEP-8: Project is successfully created in Bazel.

4.2. Bazel Build Phases

In Bazel, a build occurs in three distinct phases

1. **Loading Phase**: The first is loading during which all the necessary BUILD files for the initial targets, and their transitive closure of dependencies, are loaded, parsed, evaluated and cached. Errors reported during this phase include: package not found, target not found, lexical and grammatical errors in a build file, and evaluation errors.
2. **Analysis Phase**: The second phase, the analysis involves the semantic analysis and validation of each build rule, the construction of a build dependency graph, and the determination of exactly what work is to be done in each step of the build. Errors reported at this stage include: inappropriate dependencies, invalid inputs to a rule, and all rule-specific error messages.
3. **Execution Phase**: This phase ensures that the outputs of each step in the build are consistent with its inputs, re-running compilation/linking/etc. tools as necessary. Errors reported during this phase include: missing source files, errors in a tool executed by some build action, or failure of a tool to produce the expected set of outputs.

Figure 9: Step1: Installing the Bazel software.

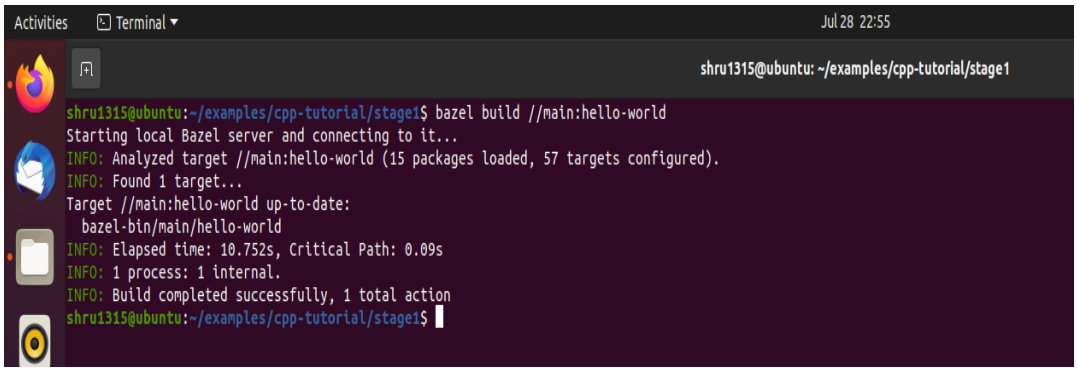

Figure 10: Step 2: Checking for complete compilation process of Bazel tool.

5. CURRENT SCENARIO: PRACTICES, AND CHALLENGES

Build automation techniques and tools have emerged as a significant component for ensuring seamless and continuous integration. In the present-day scenario of build automation, there are certain practices which are being currently deployed to improve the build automation processes. Some of these include:

- Maintenance of a central code repository.
- Making the build as self-testing.
- Testing in a clone of the production management.
- Quicken up the build time execution.
- Maintenance of revision control system for the project's source code.

Although, build automation tools have several benefits, they have their own unique challenges, including:

- Selection of the right tool.
- Correspondence of project requirements and tool features.
- Scalability of the builds and its deployment.
- Expectations of the build to detect occurrence of new defects during the project creation.

The industry-wide adoption of our three selected tools is shown below [Figure 11]. The adoption of Maven has continued to rise steadily over the years, Bazel has been in decline around. There is a sharp uptick in the adoption of Gradle.

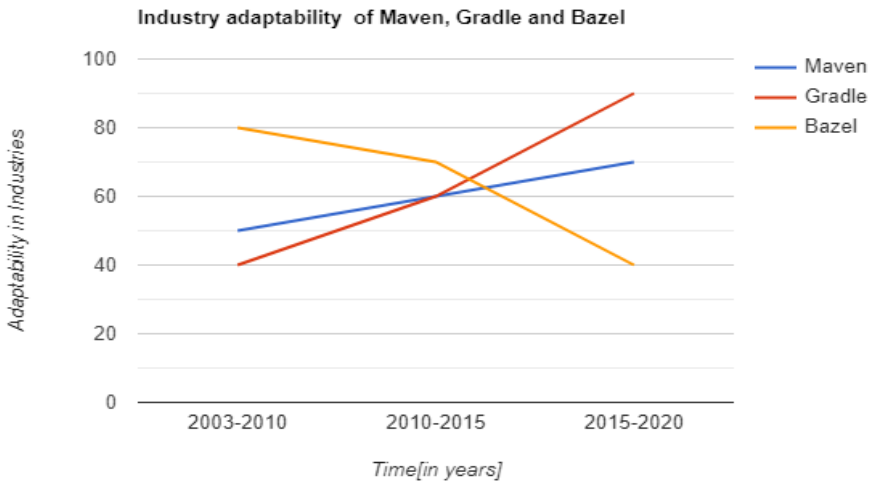

Figure 11: Line graph based on timeline analysis of tools

5.1. Maven Trends

1. Provides allowance to a developer to understand the entire state of a development within a short period of time.
2. Provision of easy guidance for project creation.
3. Maven provides more than 200+ plugin-in to work with which is one of the remarkable trends seen in the Maven tool.

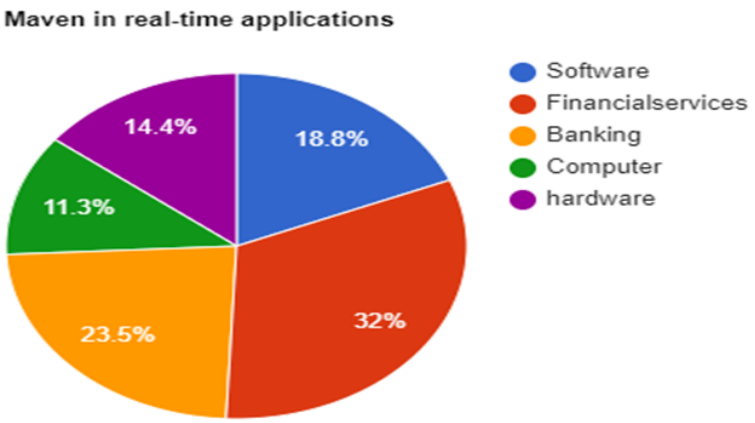

Figure 12: Maven real-time applications [Sample size: 100] (Source Material:From References*)

5.2. Maven Challenges

1. Tool is unreliable since it can collapse the software development lifecycle due to the limitless number of loops present in IDE.
2. Existence of pom.xml files make it difficult for usage when it comes to large codebases, since the number of XML files increases.

5.3. Gradle Trends

1. Establishment of incremental annotation processing to increase the effectiveness of incremental annotation.
2. Collaborative debugging in Gradle hugely helps in scanning of multiple scripts at one glance, reducing the processing time.
3. Supports user customization at a maximum rate.

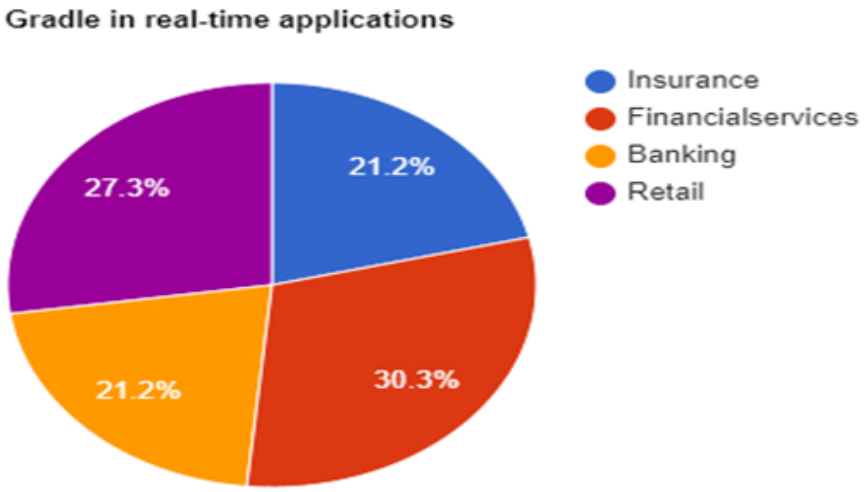

Figure 13: Gradle implementation for real-time applications [Sample size: 100] (Source Material: From References)

5.4. Gradle Challenges

1. Changes in environment variable pose a challenge for tracking it.
2. Lack of adequate documentation requirements of Groovy DSL [Dynamic structured language] to write the build configuration.

5.5. Bazel Trends

1. Supports multiple project integration with toolchains and platforms.
2. Aims to excel at supporting multiple platforms and mixed language basis of project development.
3. In architectural implications, Bazel avoids cyclic dependencies.

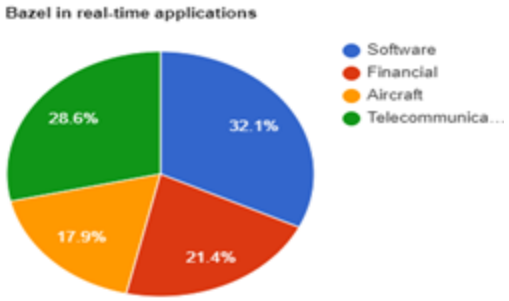

Figure 14: Bazel implementation for real-time applications [Sample size: 100] (Source Material: From References*)

5.6. Bazel Challenges

1. Explicit listing of dependencies makes the build process a tedious and a repetitive task.
2. Bazel uses sandboxing to ensure the build correctness, but sandboxing creates over head in performance.

5.7. User Analysis

It is therefore evident that Maven leads the field in terms of adoption by offering more user-friendly features for developers and programmers.

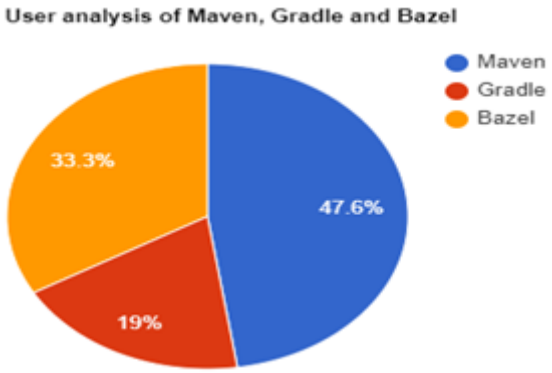

Figure 15: Usage Analysis of Maven, Gradle and Bazel Tools across industries [Sample size: 100](Source Material: From References*)

6. SUMMARY AND COMPARISON OF MAVEN, GRADLE AND BAZEL

Build Tool Critical Features	Maven	Gradle	Bazel
Open-Source	1	1	1
Java-Projects Enabled	1	1	1
Multiple Platform Support	1	1	1
Code Management Efficiency	1	1	1
Multiple Language Support	1	1	1
Build Execution	-1	1	0
Use of Xml File	1	-1	-1
Customization	-1	1	0
Ease of Understanding	0	1	1
Dependency Management	0	1	0
Total	**4**	**8**	**5**

***Weights** (-1 being weakest feature, 0 being average and 1 being strongest feature)*

6.1. Build Compilation Time Analysis of Maven, Gradle, and Bazel

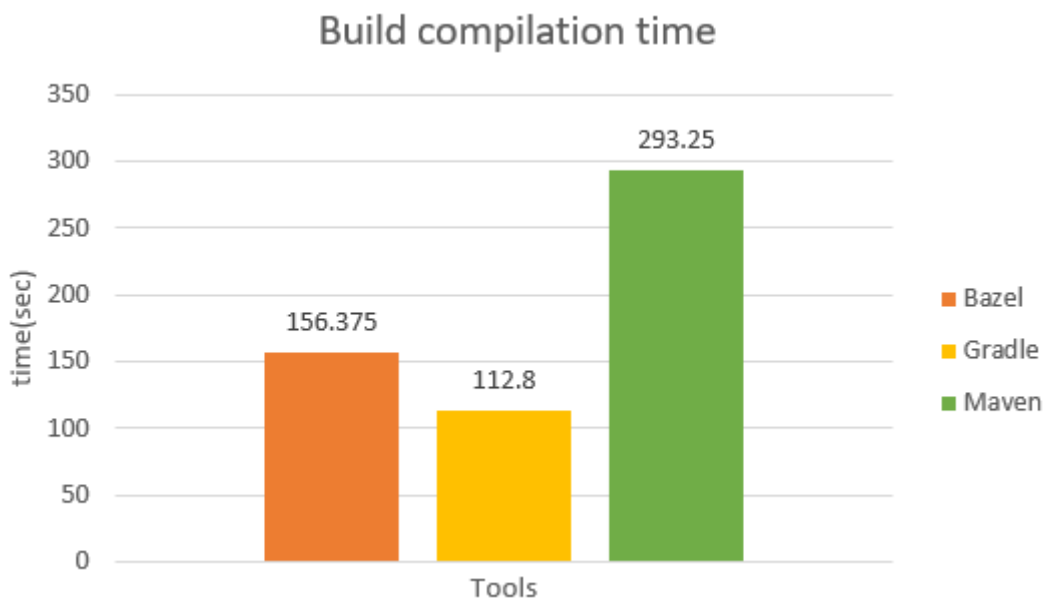

Figure 16: Bar-Graph representation of execution time analysis of Maven, Gradle and Bazel

The above Bar-Graph is drawn using the average time compilation of small, medium and large projects using Maven, Gradle and Bazel. The factors we use in a project decide the efficiency of a build system and its performance. The approach for making this comparison is based on so many features like performance, user experience, dependency management and flexibility. This graph concludes that Maven takes more time to perform the build compilation followed by Bazel and Gradle. This shows us that on an average Gradle is more effective for all types of project. Among these three build automation tools it is found that Gradle will complete the process within a short period of time, 112.8 sec (average of small, medium and large projects together). Therefore, Gradle is more effective and helpful for the developers.

7. CONCLUSION

The main purpose of this paper is to analyse and provide an overview of automation tools. Gradle and Maven are almost similar in building projects along with their dependencies, but we have seen Bazel follows different approaches.

Gradle gives best results in terms of performance and Maven is remarkable in providing an overall easy build experience for users.

Maven focus on developing a project within the deadline whereas Gradle develops the application by adding functionalities. Gradle has many advanced CLI features for effective build automation. Bazel supports IntelliJ IDEA(Java IDE) that works properly in MAC and is not supported in windows.

With reference to API's Maven and Gradle are good at providing flexible plugin provisions with the other IDEs and have created a benchmark for plugin user-facilities to a large extent. Maven provides overriding dependencies based on the version that too only on a single set of

dependencies. Whereas Gradle and Bazel both supports similar dependency management flow but Gradle has more advanced features and capabilities in dependency management.

Based on this comparison we can conclude that both Bazel and Gradle are faster than Maven. The studies suggest that Gradle is better than Bazel for most of the projects. Even if we take a million lines of code Gradle performance is fast and reliable than Bazel and Maven.

However, it is difficult to choose a single tool for project development, when all the tools have their own advantages and disadvantages. We can consider Maven while working on a small project where modularization, consistency and lots of plugins are required. Gradle is best suited for large projects, where the focus is on flexibility, speed, ease of use and integration builds. Bazel is designed purely to handle very large projects. Projects started in Maven, at any phase can be easily ported to Gradle and vice versa. Gradle can handle all the projects with utmost flexibility and speed. It also supports porting between the tools in minimum steps. Hence, Gradle can be considered as the most preferred automation tool in recent times. Future exposure should critically look at the break down of existing plugins in Gradle because this is leading to unexpected changes in build operation causing a lot of restrictions in usage of functionalities and can work on the complex dependency trees in Maven to minimize the disputes within dependencies.

ACKNOWLEDGEMENTS

The author would like to thank the organization L&T Technology Services(LTTS) for giving the opportunity to work on Build Tools. Thanks to the immense resources online on Maven, Gradle and Bazel that helped to get deeper understanding.

REFERENCES

[1] Kaiyun Wang, Greg Tener, Vijay Gullapalli, Xin Huang, Ahmed Gad, Daniel Rall, “Scalable Build Service System with Smart Scheduling Device”, International Symposium on Software Testing and Analysis (ISSTA), July 18-20,2020.

[2] Mubarak Albarka Umar, Zhanfang Chen, “A Study of Automated Software Testing: Automation Tools and Framework”, International Journal of Computer Science Engineering, December 2019, Volume 8.

[3] Adrian Paschke, “OntoMaven API4KB-A Maven-based API for Knowledge Bases”, International Workshop on Semantic Web Applications and Tools for Life Sciences, December 2013.

[4] Adrian Paschke, Ontomaven: “Maven-based ontology development and management of distributed ontology repositories.” In 9th International Workshop on Semantic Web Enabled Software Engineering (SWESE2013). CEUR workshop proceedings, 2013.

[5] Wilfried Elmenreich, Philipp Moll, Sebastian Theuermann, Mathias Lux, “Making simulation results reproducible— Survey, guidelines, and examples based on Gradle and Docker”, Peerj Computer science,2019.

AUTHOR

Mridula Prakash has over 13 years' experience in the electronics industry, largely dedicated to embedded software. She is a senior executive with extensive experience in architecting and building embedded products in the Industrial Products domain. She is an active member in embedded system community and has been working on various technologies of microprocessors and microcontrollers, including x86, PIC, AVR, MIPS, PowerPC and ARM, developing firmware and low-level software in C/C++ on Linux, Android, FreeRTOS and many other kernel and operating systems.

In her current role as Specialist – Embedded Architect at L&T Technology Services (LTTS), she is responsible to understand major trends in the embedded sector and help in the implementation of embedded design software and modernization of legacy system.