Anthony Sigler
Semester Project

Building Projects and Managing Dependencies within Maven

I did not know what build tools or dependencies were before taking this course. After researching the list of options presented for the projects on the first day of class, the project that appealed to me the most involved making a custom build tool modeled off the three predominant tools in Java ecosystem: Ant, Maven, and Gradle. I did not ultimately, produce my own build tool. It is still a project I am interested in picking up on one day. However, in researching the features of Apache Maven in particular, I am now more interested in projects that evaluate how best to use the existing tools out there in order to increase productivity. These tools were designed with large projects, often worked on by many developers, in mind. As someone who's only worked on small projects on my own, I'm fascinated to peek into that professional world and the kinds of problems teams of developers can solve using dependency management tools such as Maven.

Maven is project management tool used primarily (though not exclusively) for Java programs. It can be used to build projects and manage dependencies. It can also be used for testing, documentation, and site generation. It shines in large projects with many dependencies. According to a study done in 2018, Maven had 60% of the market share for Java applications, with 20% from Gradle and 10% from Ant.[1]

It helps at this point to elucidate what is meant by managing dependencies and why this process is important to developers. Dependencies are external packages required to run a program. If A "depends on" some function in library B to work, then B is considered a

[1] Simon Maple, Andrew Binstock. "JVM Ecosystem report 2018 – About your Tools." *Java Magazine, JVM Survey Report.* 2018. https://snyk.io/blog/jvm-ecosystem-report-2018-tools/

dependency of A. This gives rise to what is known as transitive dependencies—in this case, where B may have its own dependencies, or A may inherit dependencies from its parents. In large project, the list of transitive dependencies is potentially very deep, including hundreds of dependencies. This naturally gives rise to potential for error, and "this situation gets even worse … with huge code bases and fast-paced continuous integration pipelines."[2] Other studies have noted that on average 12% of development effort is not spent on developing software but on maintaining build scripts.[3]

Maven handles dependency management in what is termed the Project Object Model, or POM. The .pom file is an XML file that contains all the metadata necessary to build a project from source code, run the unit tests, and package the project into a suitable format. It is essentially a cookbook for the project, and its presence is an essential part of any Maven project. Maven will read the POM to execute a specific task or goal. The POM file keeps track of dependencies and their version numbers. POMs can inherit from other POMs. This becomes an extremely useful feature as any dependencies will also be inherited via the POM files.

If Maven needs to find an artifact specified in a POM file, it will look first in the local repository. The local repository is a hidden folder created in the user's home directory upon installation of Maven. If not found there, it will search the Maven Central Repository. Thirdly, Maven can be configured to look for dependencies in private repositories. After the artifact is

[2] Maudoux, Guillaume, and Kim Mens. "Correct, efficient, and tailored: The future of build systems." IEEE Software 35.2 (2018): 32-37. https://dial.uclouvain.be/pr/boreal/object/boreal%3A189586/datastream/PDF_01/view
[3] Kumfert, Gary, and Tom Epperly. Software in the DOE: The Hidden Overhead of''The Build''. No. UCRL-ID-147343. Lawrence Livermore National Lab., CA (US), 2002. https://grosskurth.ca/bib/2002/kumfert.pdf

found at Maven Central or elsewhere, it will be downloaded to the local repository, essentially caching the artifact there.

The Maven Central Repository is in my opinion one of the cooler features of Maven. It is a large hub of commonly used libraries maintained by the Maven community, currently containing upwards of 6.7 million indexed jars. The number of contributions to the central repository has increased exponentially in the last decade.

After Maven has downloaded a jar into the local repository from Maven Central, it will use that file from then on, essentially freezing the version number of the dependency. Should the third party that published the library to Maven Central put out a security update or bug fix, the project would by default continue to use the previously downloaded file. However, Maven dependencies can be declared as SNAPSHOTs, meaning that Maven will check to see if there is an updated version of the jar on Maven Central, and if so will download and use the newest version. By default, Maven checks once a day, although this setting can be configured.

It is interesting to note here that some researchers point out that developers are often reluctant to update their dependencies, as deprecated methods that are left out of newer versions of a library may break the build.[4] This certainly has security implications when relevant updates patching security flaws are ignored. However, in other cases using an older version of an external library has no correlation to increased security risk, such as if the library in question

[4] Raemaekers, Steven, Arie van Deursen, and Joost Visser. "Semantic versioning and impact of breaking changes in the Maven repository." Journal of Systems and Software 129 (2017): 140-158. https://research.tudelft.nl/files/19482719/TUD_SERG_2016_011_cc.pdf

is only used for test files.[5] So, the useful ability to control the version of a dependency is facilitated by Maven.

If multiple versions of a dependency are encountered, Maven determines what version to use through dependency mediation. In this, the nearest definition encountered in the dependency tree is used. For example, let us say that A has a dependency B. B has a dependency on D version 1.5. Now, what if A also had a dependency on D, but version 2.0? Version 2.0 would be used, because it is closer to the root of the dependency tree. Maven also provides options to exclude dependencies, or make them optional, which can be used to make packages smaller.

It would be too time-consuming to build a large project in its entirety every time a developer needed to make a change. The solution to this is a process known as incremental building: after a file changes, the build system identifies and executes only those build operations whose result has been invalidated by the change.[6] Incremental builds are supported through Maven build lifecycles.

A lifecycle is a pre-defined group of build steps called phases. Each phase can be bound to one or more plugin goal. The actual "work" is done by the Maven plugins themselves. Lifecycles and phases provide the framework to call plugin goals in sequence. Goals can be

---

[5] Pashchenko, Ivan, Duc-Ly Vu, and Fabio Massacci. "A qualitative study of dependency management and its security implications." Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. 2020. https://www.researchgate.net/profile/Duc_Ly_Vu/publication/343021994_A_Qualitative_Study_of_Dependency_Management_and_Its_Security_Implications/links/5f699b89299bf1b53ee9950d/A-Qualitative-Study-of-Dependency-Management-and-Its-Security-Implications.pdf

[6] Erdweg, Sebastian, Moritz Lichter, and Manuel Weiel. "A sound and optimal incremental build system with dynamic dependencies." ACM Sigplan Notices 50.10 (2015): 89-106. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.725.6063&rep=rep1&type=pdf

executed from the command line with the syntax `mvn [plugin-name]:[goal-name]` and. For example, `mvn compiler:compile` will compile a Java project with the maven-compiler-plugin's compile-goal. One can configure tasks to be executed by binding them to the goals of a plugin. One can also create their own custom plugins or download popular ones from Maven Central such as ones used to execute Python scripts in Maven projects. Adding plugins (or dependencies for that matter) to a project is simple; they are included in the POM along with the necessary metadata such as the coordinates, which include the groupID, artifact's name, and its version number.

Maven's approach emphasizes conventions over configuration, which reduces the overall setup time of a project and helps it get off its feet more quickly. An example of this is the standard directory structure, which was adopted by other tools such as Gradle. Some consider XML dated and the fact the POM file is an XML document as a disadvantage. The argument for Maven is that the convention provided by the XML schema simplifies most build processes. Where additional flexibility is needed, developers may opt for Gradle. Gradle uses a Groovy DSL instead of an XML document as its buildfile, which can offer greater build flexibility.[7] It seems up for debate how rarely the additional flexibility is needed; with the customizability of plugins and their ability to be combined with other lifecycle goals, there are still many tasks Maven can accomplish.

In the future I'd like to research how build tools can be used to reinforce test-driven development. I'm interested in test-driven development as a paradigm. Maven comes with

[7] Ikkink, Hubert Klein. Gradle Dependency Management. Packt Publishing Ltd, 2015. http://orkish5.tplinkdns.com:8080/books/Java/Gradle%20Dependency%20Management.pdf

plugins for unit testing. I wonder how those could be customized to maximize code writing productivity.

Although it seems clear to me that Maven has many strengths, my goal is not to advertise for Maven; rather, I look at what I've learned about Maven as a means to expose some of the problems that arise when building and maintaining large projects. Outside of the Java world there are many build tools that have sprung up with their respective languages. For the most part, these tools offer similar features and functionality. The goal is to continue to learn about the full functionality of these tools and how to use them to optimize projects of my own, or be prepared if I need to use these tools when working on a project with others.