Uncovering Secrets of the Maven Repository

Java Build Aspects

Maven, a Yiddish word meaning accumulator of knowledge, began as an attempt to simplify the build processes in the Jakarta Turbine project

Introduction

- *Maven* is a buildtool for *Java*-based software projects. The largest repository for these projects is *Maven Central*.
- The build system has evolved since its introduction in 2004.
- Version adoption, reproducibility and desirable build features are important metrics for maintainers in the software ecosystem.
- Maven Central provides an archive of releases suitable as dataset for research about the Java ecosystem.

Related Work

- Version adoption in dependencies [1].
- Library aging [2].
- Performance-related configuration smells in build cycles [3].

Research Questions

1. How common are Java modules (a language feature introduced in Java 9)?
2. What are the popular Java versions used?
3. How is the compiler configured?

Data Selection

- Artifacts hosted on Maven Central are publicly accessible.
- One unique version per package is randomly selected, giving a sample size of almost half a million packages.

Contact Info

G.J.T. Bot
g.j.t.bot@student.tudelft.nl

TU Delft

Supervisor: M. Keshani
Responsible Professor: Dr. S. Proksch

Research

Method

The set of available packages is retrieved from an index file. From this set, a subset is selected for analysis. For this research, one unique random version per package is selected.

These packages then get downloaded and passed to extractors, which fetch raw data from the downloaded artifacts. (Fig. 1)

For each research question the relevant data to fetch from artifacts are:

1. Does the archive contain *module-info.class* files?
2. Each *.class* (compiled Java code) file contains a version in its header. Optionally, the *META-INF/MANIFEST.MF* file may contain some Java version and multi-release data.
3. Every Maven artifact has a configuration file available called the *POM*. Part of the POM is the compiler configuration.

An additional program then further analyzes the raw data.

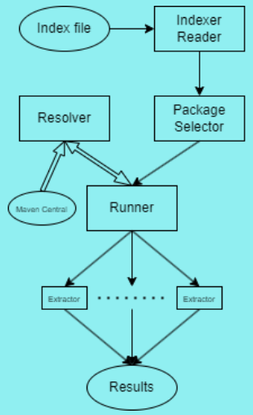

Figure 1. Application diagram.

Results

In total, 479915 packages were selected for analysis, of which 1.4% could not be resolved. 86.6% of the resolved packages have an archive artifact.

How common are Java modules?

- 1.7% of the archives use Java modules.
  - From these archives, 29.2% contain a majority of class files compiled in a version older than Java 9.

What is the Java version distribution?

- The most common Java version is Java SE 8, being the most common class version in 44.5% of the archives. In total, 51.2% of all artifacts use a *long-term support* (LTS) version. 17.9% does not contain class files. (Fig. 2)
- 4.0% of archives multiple different class versions.
- There are class files compiled using Java versions in early-access, and there are class files compiled using unsupported Java versions.
- Judging from the Java development kit (JDK) used to build artifacts, 37.26% build their project specifically compiling code to older Java versions.

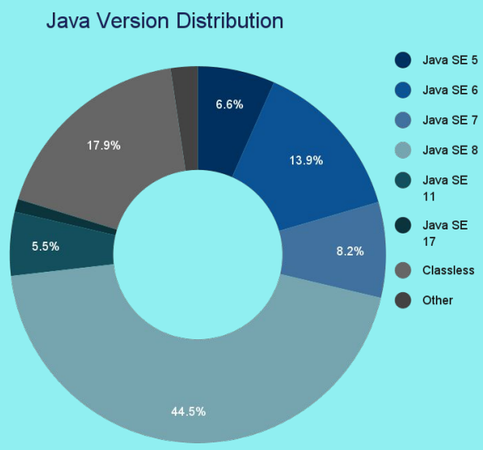

Figure 2. Java version distribution.

How is the compiler configured?

- 55.0% of all artifacts use the Apache Maven compiler plugin.
- Only 1.21% of configurations deviate from using a standard *javac* compiler. Since most other *Java virtual machine* (JVM) languages have a plugin to compile their code rather than using the Maven compiler plugin, these languages are rarely detected this way.
- 12.4% of configurations specify additional compiler arguments, with the most common compiler flags specified being debugging and verbosity settings such as -Xlint.
- In basically all configurations the source and/or target versions are specified. Only 3.92% of configurations specify neither. The most common version is Java SE 8.
- The vast majority of configurations do not specify a source encoding (74.5%), but if they do, most use *ASCII* (or extensions thereof). The most popular encoding used is *UTF-8*, making out 25.0% of the configurations.

Conclusion

1.
   a. Java modules are very uncommon, presumably because they are not required when developing Java software.
   b. Occasionally they are purposely included in projects using a Java version that does not support Java modules.
2.
   a. LTS versions are dominant. The oldest LTS version is the most common.
   b. Older versions are relatively common overall, but their presence is significantly declining in recent years.
   c. The presence of multiple class versions in a single archive implies the archive published contains repackaged code.
   d. Releases for unsupported Java versions are assumed to be the result of much legacy code still being used. The reason for releases using early-access Java builds is unknown.
3.
   a. Almost half the POMs rely on default configurations.
   b. The most important setting the release version.
   c. Encoding is often ignored.
   d. A significant chunk of projects improve their software's quality by utilizing more warnings and other feedback from both the compiler and linter.

Limitations

- Maven build cycles are inherently version-dependent.
- Maven can include both generated sources and third-party sources in the final artifact, which are not distinguished from the actual source.
- Predicting Maven build output is nontrivial and as such has been simplified for the purposes of this research.
- Non-Java and non-Maven artifacts exists in the Maven ecosystem.

Future Work

- Java version feature scope and release cycle.
- Quality of software published on public repositories.

References

[1] R. Kula, D. German, T. Ishio, and K. Inoue, "Trusting a library: A study of the latency to adopt the latest maven release," 2015
[2] R. G. Kula, D. M. German, T. Ishio, A. Ouni, andK. Inoue, "An exploratory study on library aging by monitoring client usage in a software ecosystem," 2017
[3] C. Zhang, B. Chen, J. Hu, X. Peng, and W. Zhao, "Buildsonic: Detecting and repairing performance-related configuration smells for continuous integration builds," 2023