Motivating Version Range Adoption in Maven Through Quantified Trust

Gijs Hoedemaker

Supervisor: Cathrine Paulsen, Responsible Researcher: Sebastian Proksch

TU Delft

1. Introduction

- Developers are more and more reliant on external libraries to use as building blocks for their projects.
- Dependency managers such as npm for Node.js, Cargo for Rust, or Maven for Java facilitate developers with automated dependency version resolution.
- Improper dependency management practices result in the continued use of vulnerable dependency versions despite fixes being promptly available.
- These issues arise in part because developers are unaware of these vulnerabilities, as well as security not being a high priority because of the introduced effort caused by backward incompatible changes.
- The Semantic Versioning policy aims to inform which kinds of updates were performed on a dependency.

MAJOR.MINOR.PATCH

- Version ranges, in combination with SemVer, can be used to automatically adopt dependency updates.
- Failure in complying with the semver policy results in developers being hesitant to use version ranges in order to avoid updates with backward incompatible changes.
- A study by Zhang et. al found that out of 82 million dependency relationships, only 1.02% used semantic version ranges, while 98.94% of these ranges helped resolve patched versions of once-vulnerable dependencies. These numbers show that version ranges are heavily underavlued.

2. Research Gap

- Version ranges are especially underutilized in Maven compared to other dependency managers.
- The notion of trust has been shown to be an important factor for version range adoption, but it is unclear what defines the trustworthiness of a dependency.
- OpenSSF's Scorecard provides quantitative metrics on a dependency's security hygiene and release frequency also contributes to dependency stability. These and other factors can be combined to provide a concrete notion of trust.

3. Research Questions

This project aims to answer the following research questions:

- **What is the impact of using fixed versions as opposed to version ranges?** Motivation: by investigating a large set of open-source projects for the amount of declared dependency versions which are outdated but can be safely updated without compatibility issues, we can provide a representative picture of the underutilisation of version ranges in Maven projects and motivate developers, as well as provide further motivation for our second research question.
- **Which dependencies can be trusted to adhere to semantic versioning?** Motivation: other than popularity and community opinion, dependencies have few indicators that suggest whether or not they comply with SemVer policies. By compiling a Trust Score metric based on several key heuristics of a dependency, developers may be persuaded to use version ranges for dependencies with a high score.

4. Methodology

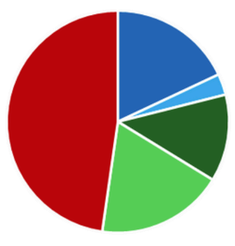
- A selection of GitHub projects was made based on a careful selection of parameters:
  1. **No toy projects.** Projects with fewer than 100 commits and 1000 LOC were filtered out to exclude toy projects with few or no dependencies.
  2. **Actively maintained.** Only projects that saw any form of activity after January 1, 2025 were selected to increase the likelihood of a dependency management policy.
  3. **Mature.** Dependency management is not a priority in the early stages of a project, so selected projects were required to be over a year old.
- For each project and its modules, each dependency was extracted along with the declared version.
- For each unique dependency, a list of all releases was queried from the Maven repository.
- To answer RQ1, each extracted declaration was compared using an API comparison tool, *japicmp*, to see how often it could be updated without breaking changes being introduced, and whether it could be updated to the latest release with the same major version number.
- To answer RQ2, we developed a command-line tool, TeSTer, which analyses a dependency to generate a Trust Score based on the following metrics:
  1. **OpenSSF Scorecard Score.** This is an assessment of a library repository based on a number of heuristics to indicate the security hygiene of a project. Some important factors include code reviews, the existence of a security policy, and the use of dependency update tools.
  2. **Release frequency**. A low MTTU indicates active maintenance, suggesting that potential vulnerabilities get fixed promptly after discovery.
  3. **Automated analysis of semantic compatibility.** This is an important factor in motivating developers to use version ranges, as low semantic compatibility introduces a large maintenance overhead. Providing insight on the amount of backward-compatible minor and patch versionscan provide a lot of information on the trustworthiness of a dependency.

5. Results

Impact of fixed versions (RQ1)

- Analysis for RQ1 shows that 52.2% of declared dependencies are outdated and are safely able to be updated at least once. Many dependencies even have multiple newer versions available, causing developers to miss out on multiple bug-fix or security updates.
- Further analysis shows that 85.47% of patch versions and 40,68% of minor versions can even be updated to their latest release.

Dependencies with newer compatible releases



Trust Score (RQ2)

- Results of TeSTer's analysis of the most popular Maven dependencies is shown in the table. It can be seen that all factors are considered and low metrics result in low Trust Scores.

GroupId	ArtifactId	Scorecard Score	Minor Score	Patch Score	Release Frequency	Trust Score
org.junit.jupiter	junit-jupiter-api	8.0	44.8%	76.8%	37 days	74.1
org.jetbrains.kotlin	kotlin-stdlib	3.5	12.5%	77.0%	18 dayas	53.0
org.slf4j	slf4j-api	4.7	41.67%	85.23%	64 days	62.6
com.google.guava	guava	8.8	0.00%	17.07%	36 days	53.1
org.mockito	mockito-core	7.5	53.7%	77.7%	16 days	75.6

6. Discussion

- Results for RQ1 indicate that 62% of minor versions can be safely updated, and 48% of patch versions can be safely updated. This is lower than expected, considering previous work showing that a third of minor updates and a quarter of patch updates introduce breaking changes.
- Only around a quarter of outdated dependencies can be safely updated to the latest version without conflicts. This is lower than expected, but should still provide some motivation for developers to put more thought into their declaration strategies.
- TeSTer is currently presented as a proof of concept and more research could be done into additional factors that determine the trustworthiness of a dependency, resulting in a more grounded Trust Score.
- TeSTer can be made more accessible by repurposing it as a Maven plugin, GitHUB CI tool, or an IDE plugin providing real-time suggestions to convert fixed versions into version ranges.

7. Conclusions

- Dependency managers can greatly aid developers with dependency declaration as projects grow larger in size.
- Version ranges help reduce maintenance overhead for developers by automatically adopting new releases.
- Releases breaking SemVer conventions dissuade developers from employing version ranges, undermining their value.
- To motivate developers to consider making the transition to using version ranges, we performed quantitative empirical analysis on a large set of open-source projects.
- We showed that more than half of outdated dependencies can be safely updated to a newer compatible version, and a quarter can be safely updated to the latest available release.
- We developed TeSTer as a conceptual command-line tool which analysis a dependency for key security and compatibility heuristics, generating a Trust Score which allows developers to make conscious, risk-aware decisions when incorporating a dependency into their project.

References

[1] T. Mens A. Decan. What do package dependencies tell us about semantic versioning? In *IEEE Transactions on Software Engineering*, volume 47.6, pages 1226–1240, 2019.

[2] Jens Dietrich, David Pearce, Jacob Stringer, Amjed Tahir, and Kelly Blincoe. Dependency versioning in the wild. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 349–359, 2019. doi: 10.1109/MSR.2019.00061.

[3] Sascha Fahl Yasemin Acar Erik Dell, Sven Bugiel and Michael Backes. Keep me updated: An empirical study of third-party library updatability on android. In *CCS*, pages 2187–2200, 2017.

[4] F. Massacci I. Pashchenko, D. Vu. A qualitative study of dependency management and its security implications. In *2020 ACM SIGSAC conference on computer and communications security*, pages 1513–1531, 2020.

[5] Raula Gaikovina Kula, Daniel M. German, Takashi Ishio, and Katsuro Inoue. Trusting a library: A study of the latency to adopt the latest maven release. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 520–524, 2015. doi: 10.1109/SANER.2015.7081869.

[6] J. Visser S. Raemakers, A. van Deursen. Semantic versioning and impact of breaking changes in the maven repository. In *Journal of Systems and Software*, volume 129, pages 140–158, 2017.

[7] Scorecard. Accelerating openssf adoption: Unlocking scorecard insights with a centralized dashboard. https://openssf.org/blog/2025/01/22/accelerating-openssf-adoption-unlocking-scorecard-insights-with-a-centralized-dashboard/, 2025.