A Formal Framework for Measuring Technical Lag in Component Repositories

Ahmed Zerouali
ahmed.zerouali@umons.ac.be
University of Mons

***Abstract*—Reusable Open Source Software (OSS) components for major programming languages are available in package repositories. Developers rely on package management tools to automate deployments, specifying which package releases satisfy the needs of their applications. However, these specifications may lead to deploying package releases that are outdated or otherwise undesirable because they do not include bug fixes, security fixes, or new functionality. In contrast, automatically updating to a more recent release may introduce incompatibility issues. To capture this delicate balance, we formalise a generic framework of *technical lag*, a concept that quantifies to which extent a deployed collection of components is outdated with respect to the *ideal* deployment. The framework can be used to assess and reduce the outdatedness, vulnerability and bugginess of software deployments, software projects, software containers and reusable software libraries. We argue that such a metric is very relevant for assessing the health of software (eco)systems, and should be used.**

***Index Terms*—Empirical analysis, technical lag, software repositories**

Extended Abstract

Software components are being created and reused on a regular basis. Over the past years, depending on external software components has become a common software development practice, especially in the free, Open Source community. This practice can lead to a significant gain in productivity, due to the ability to reuse complex functionality, rather than implementing it from scratch.

Because these components are usually evolving to avoid becoming obsolete, many versions of them are being created and distributed via online package managers and repositories everyday (e.g., *npm, Maven, Debian*, etc). Usually, new versions include new features, changed requirements, improved performance, fixed bugs, etc. In general, these changes are seen as a good sign of a well maintained software component. On the other hand, major changes may require breaking changes.

While the availability and abundance of reusable components facilitates building software, it can also cause problems in maintenance and evolution. For example, a recent version of an application may be outdated although not because of its own code, but due to depending on components that were not updated to their latest versions. If this happens, there is a higher risk of having bugs and security issues that may have been already fixed in newly released versions. On the other hand, updating to more recent releases of reusable components is not *for free*, since it might lead to a risk of facing backward incompatible changes, which cause conflict and problems for developers using these components. In many cases, these problems may eventually lead to ripples through software ecosystems.

For individual developers, there is a balance between benefits (e.g., new functionality, bug and vulnerability fixes, etc) and cost of updating a dependency (e.g., risk of having breaking changes). To represent this balance, we introduce the *technical lag* concept as a measurement to capture the difference between the reusable software component version that we *want* to update to and the deployed version of the same software component that we rely on. The concept of technical lag aims to quantify to which extent a deployed collection of components is outdated with respect to an ideal deployment. How to interpret this “ideal” and the “outdatedness” w.r.t. this “ideal” is highly context-specific. Depending on the needs and goals of a specific project or a maintainer, the focus may be on functionality, security, stability or even other factors. The “components” under consideration could be individual software packages, third-party libraries, component dependencies, or software containers bundling collections of components.

This extended abstract is a summary of a chain of research studies including different quantitative analyses [1]–[3], a qualitative analysis with software practitioners [4] and an implementation of the technical lag metric in an actual tool [5].

References

[1] Ahmed Zerouali, Tom Mens, Jesus Gonzalez-Barahona, Alexandre Decan, Eleni Constantinou, and Gregorio Robles. A formal framework for measuring technical lag in component repositories — and its application to npm. *Journal of Software: Evolution and Process*, page e2157, 2019.

[2] Ahmed Zerouali, Tom Mens, Gregorio Robles, and Jesus M Gonzalez-Barahona. On the relation between outdated docker containers, severity vulnerabilities, and bugs. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 491–501. IEEE, 2019.

[3] Ahmed Zerouali, Valerio Cosentino, Tom Mens, Gregorio Robles, and Jesus M Gonzalez-Barahona. On the impact of outdated and vulnerable javascript packages in docker images. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 619–623. IEEE, 2019.

[4] Ahmed Zerouali. *A Measurement Framework for Analyzing Technical Lag in Open-Source Software Ecosystems*. PhD thesis, University of Mons - Belgium, 2019.

[5] Ahmed Zerouali, Valerio Cosentino, Gregorio Robles, Jesus M Gonzalez-Barahona, and Tom Mens. A tool to analyze packages in software containers. In *Mining Software Repositories 2019 (MSR)*, 2019.