Universidad Tecnológica de Tijuana

TSU en Tecnologías de la Información Área Desarrollo de Software Multiplataforma

Nombre:

García Arreola Howard Isaí

Subject:

Desarrollo movil Integral

Actividad:

Strategy versioning

Grupo: 10-B

Profesor: Ray Brunett

Parra Galaviz

Fecha de Realización:

January 7th, 2025

Versioning Strategies

Versioning strategies are methods used to manage and track changes in software over time. They help developers and users understand the evolution of a software product and ensure compatibility between different versions.

Semantic Versioning

<major>.<minor>.<patch>[.<build number>] (1.20.11, 1.20.11.403)

1. Major: increasing the major version usually breaks compatibility, allowing developers to remove the deprecated API or rework the existing ones. Users know about it and do not expect a smooth update.

2. Minor: version increment implies adding new functionality without breaking compatibility.

3. Patch: also known as bugfix version that includes fixing security vulnerabilities, etc.

4. Build number: optionally, the build number can be additionally added.

Calendar Versioning

<year>.<month>.<patch> (2022.07.01)
<year>.<minor>.<patch> (2022.1, 2022.1.1)

Calendar Versioning is used when you don’t care about compatibility. For such cases, using SemVer leads to having the first digit `1` forever.

This versioning is especially great for on-premise software (most meaningful for external users).

Sprint-based Versioning

<symbol><sprint number>[.<patch>] (M27, M27.1)

When the development process is based on Scrum sprints, and the release is rolled out following the sprint results, then a reasonable solution is to version the product based on the sprint number. The first character here helps you not to confuse this versioning strategy with all the others. If a hotfix is needed for the latest release, a patch suffix is added.

In this approach, all numbers are clear for the team, and they do not tell the clients anything. But usually, users are not interested in the version in web applications.

Selection

I would personally use the **Semantic Versioning**. I feel it has the simplest concept of them all, thus, its easier to follow and understand and identify every stage of your software releases. Thanks to the rigid structure it utilizes its harder for any errors to occure when createing the versioning.
Personally, I had never thought each number has its own meaning. I thought each value was open to interpretation but naturally using numbers as an increasing and more developed version. This being said, SemVer sounds the easier but strong approach to take.

References:

Kozhenkov, A. (2022, January 3). Application Versioning Strategies - Javarevisited - medium. *Medium*. https://medium.com/javarevisited/application-versioning-strategies-de353a84faaa

Preston-Werner, T. (n.d.). *Semantic Versioning 2.0.0*. Semantic Versioning. https://semver.org/

API Versioning Strategies: Best Practices Guide. (2024, September 22). https://daily.dev/blog/api-versioning-strategies-best-practices-guide