UTT

UNIVERSIDAD TECNOLÓGICA DE TIJUANA

GOBIERNO DE BAJA CALIFORNIA

TOPIC:

Strategy Versioning

STUDENT:

Maciel Leyva Ariana Lizeth

GROUP:

10A

SUBJECT:

Comprehensive Mobile Development

TEACHER:

Ray Brunett Parra Galaviz

Tijuana, Baja California, January 8th of 2025

“Strategy Versioning”

What is software release versioning?

Software release versioning is the act of bundling or managing multiple product releases under one giant umbrella. Where developers once released monolithic apps and projects, now with release versioning, we lean into a more modular form of software delivery. This modular approach brings many benefits, including accelerating the rate at which developers innovate and ship new things.

Software release versioning is a critical aspect of software engineering and product management. It serves as a bridge between the technical and business sides of product development. It provides a structured approach to releasing, tracking, and managing software product versions.

Why versioning is needed?

Versioning is a crucial aspect of software engineering that helps manage changes, track releases, and facilitate collaboration among developers. It solves the following tasks of software development and maintenance:

1. Tracking Changes: Versioning allows developers to track changes made to the software over time. It helps identify what features, enhancements, or bug fixes have been added in each release. This is crucial for maintaining a historical record of the software’s evolution and understanding the state of the codebase at different points in time. Moreover, it allows users to choose the most useful state (for example, with some functionality or compatible with a specific OS).

2. Collaboration: Versioning facilitates collaboration among developers working on the same project. It allows multiple developers to work concurrently on different features or bug fixes by providing a way to merge and synchronize their changes. It helps avoid conflicts and ensures that everyone is working on the latest version of the code.

3. Code Stability: By following versioning practices, developers can designate certain versions as stable or production-ready. This provides a clear distinction between stable releases and ongoing development work. It allows users and stakeholders to rely on specific versions of the software for their production environments, ensuring stability and minimizing the risk of introducing new bugs or breaking changes.

4. Dependency Management: Many projects rely on external libraries, frameworks, or components. By specifying compatible version ranges or using dependency lock files, developers can ensure that the software works correctly with the required dependencies. Versioning helps maintain consistency and avoids compatibility issues.

5. Bug Fixing and Maintenance: Versioning enables efficient bug fixing and maintenance. By identifying the version in which a bug was introduced, developers can trace it back to its source and apply the necessary fixes. Additionally, versioning helps prioritize bug fixes and allocate resources to address issues based on their impact on different versions.

Versioning systems and software version numbering

In the realm of software development, two popular versioning systems stand out: Semantic Versioning (SemVer) and Date-Based Versioning. While SemVer organizes versions into a structured format of major, minor, and patch updates to reflect the scope and nature of changes, Date-Based Versioning uses the calendar date or release date as a straightforward indicator of the software's latest iteration, providing a clear timeline of updates.

Semantic Versioning

Semantic Versioning, or SemVer, is like a finely tuned roadmap for developers and users alike, laying out the journey of a software product through numbers. The version number, broken down into major, minor, and patch segments (e.g., 2.3.1), tells a story. A major update (2.0.0) signals significant, possibly compatibility-breaking changes, a minor update (0.1.0) introduces new features that play nicely with the existing setup, and a patch (0.0.1) smooths out the bumps, fixing bugs without stirring the pot. It's a system that values precision and preparation, helping everyone brace for impact or look forward to enhancements.

A project that uses semantic versioning announces a major version number (major), a minor version number (minor), and a patch number (patch) for each release. The version string 1.2.3 indicates major version 1, minor version 2, and fix 3. Major is Incremented when incompatible API changes are made, the minor version when you add backward-compatible functionality, and the patch version for backward-compatible bug fixes.

Date-Based Versioning

On the other hand, Date-Based Versioning, or CalVer takes a more chronological approach, marking versions with the release date in various formats, like YYYY.MM.DD (e.g., 2024.03.20). It's akin to flipping through a diary, where each entry is a snapshot of the software on a specific day. This method is simple and transparent, offering an intuitive sense of a version's recency and development pace. It's especially favored in environments where frequent, incremental updates are the norm, providing a clear, linear progression of the software's evolution.

Version Control Systems (VCS)

VCS allows you to manage your codebase, i.e. track changes, create branches, and revert to previous versions when needed. It also enables collaboration and concurrent development. One of the mostly widely used VCS is Git. If you use Git, there is a branching model called Git Flow. It defines specific branch names and their purposes, helping teams collaborate effectively and maintain a well-organized versioning system. The combination of Git Flow and Semantic Versioning can provide a powerful framework for versioning and release management: SemVer can be applied to label different releases or versions in Git Flow. When a new release is ready, the version number is incremented according to the SemVer rules, indicating the nature of the changes included in that release.

Essential release versioning best practices

Navigating the intricate world of software releases demands a solid framework of best practices to ensure clarity, efficiency, and reliability. Here are some best practices to help you implement the best software release versioning practices within your organization.

Choose a clear and consistent versioning scheme

Adopting a straightforward and consistent versioning scheme, like Semantic Versioning or Date-Based Versioning, sets the stage for predictable and understandable releases. This clarity helps developers and users anticipate each update's impact.

Use version control systems

Leveraging version control systems such as Git (open-source) or GitHub (paid service) is non-negotiable for managing changes and collaborating smoothly. These tools provide a historical record of modifications, facilitating rollback and auditing processes.

Document your versioning policy

Clear documentation of your versioning policy is crucial. It outlines the rationale behind version numbers and ensures all team members are aligned, reducing misunderstandings and inconsistencies.

Automate versioning wherever possible

Automation of the versioning process minimizes human error and streamlines releases. Utilize CI/CD pipelines to integrate versioning into your development workflow, enhancing efficiency and reliability.

Communicate effectively about releases

Transparent communication with stakeholders about what each release entails promotes trust and manages expectations. Clear release notes and change logs are invaluable tools in this regard.

Use versioning for configuration management

Applying versioning principles to configuration management helps track the evolution of system settings and infrastructure, which is crucial for maintaining consistency and troubleshooting in complex environments.

Regularly review and update your versioning practices

The tech landscape is ever-evolving, and so should your versioning practices. Regular reviews ensure your methods stay relevant, effective, and in tune with the latest industry standards and needs.

Leverage feature flags for controlled rollouts

Incorporating feature flags into your release strategy offers a dynamic layer of control over how new features are rolled out and tested. Feature flagging and software release versioning are complementary practices in modern development.

Sources

LaunchDarkly. (2025, January 7th). *Software Release Versioning - Guide and Best Practices | LaunchDarkly*. LaunchDarkly. Retrieved January 8th, 2025, of https://launchdarkly.com/blog/software-release-versioning/

Thales Group. (s. f.). *Software Versioning Basics | What is Software Versioning?* Retrieved January 8th, 2025, of https://cpl.thalesgroup.com/software-monetization/software-versioning-basics

Team, Q. (2024, November 20th). *Best Practices of Versioning in Software Engineering*. Qodo. Retrieved January 8th, 2025, of https://www.qodo.ai/blog/best-practices-of-versioning-in-software-engineering/

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