IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 1, October 2025
Impact Factor: 7.67

Systematization of Versioning and Release Management Approaches in Open-Source Projects Based on Ruby

Topalidi Anna
Specialist Degree
Moscow State University of Geodesy and Cartography, Moscow, Russia

Abstract: *This article examines the system of version and release management in open-source projects developed in the Ruby programming language. Special attention is given to the application of the Semantic Versioning standard, Git and GitHub Actions tools for release process automation, as well as the role of conventions and changelog formats in ensuring transparency and development stability. The structure and practices of dependency maintenance using tools such as Bundler and Dependabot are studied, along with mechanisms for compatibility control and project stability during version updates. It explores the impact of automation and standardization on project maintenance quality and their significance for reliability and predictability within the Ruby open-source ecosystem.*

Keywords: Ruby, open source, version management, Semantic Versioning, release, GitHub Actions, release cycle, Bundler.

I. INTRODUCTION

Versioning and release management takes a special place in the software development process, especially for continuous open-source projects. Ruby's rich set of libraries and frameworks supported by community contributions, the issues of standardizing versioning policies and managing dependencies are especially pronounced. The performance of these processes directly affects the stability and compatibility of software applications that rely on third-party packages.

Contemporary open-source projects involving Ruby contain a broad range of tools and practices, from the embrace of Semantic Versioning standards and Git-driven development pipelines to the use of release automation tools such as GitHub Actions and Dependabot. Yet, though there are many, there isn't one universally accepted norm for structuring them in the industry. This irregularity leads to significant inconsistency in implementation, even in cases where there are complete documents available. Providing an assessment of current practices makes it easier to determine best-practice models and prevalent trends in the field. The goal of this research is to provide a structured overview of versioning and release management approaches in open-source projects based on Ruby, with a focus on the application of Semantic Versioning, automation tools, and mechanisms for managing and updating dependencies.

II. MAIN PART. APPROACHES TO VERSION MANAGEMENT IN OPEN-SOURCE RUBY PROJECTS: STANDARDS, PRACTICES, AND AUTOMATION

Version management is a major part of the software development lifecycle and is particularly paramount in open-source projects, which often involve dozens or even hundreds of contributors. Within the Ruby ecosystem, a set of conventions and practices has emerged that promotes predictable releases, backward compatibility, and reliable integration with other components.

One of the foundational standards adopted by the vast majority of Ruby projects is Semantic Versioning (SemVer). According to this specification, software versions follow the format MAJOR.MINOR.PATCH, where a change in the major version number indicates breaking changes to the API, a minor version increment reflects the addition of backward-compatible functionality, and a patch version is used for backward-compatible bug fixes. SemVer enables

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 1, October 2025
Impact Factor: 7.67

both developers and users to clearly interpret the nature of changes introduced in each release and to make informed decisions about dependency updates [1].

In the Ruby ecosystem, versioning practices are largely standardized and centralized. Each gem includes a version.rb file, where the current version of the library is explicitly defined. When a new version is released, this file is updated either manually or with the help of dedicated tools. Additionally, it is customary to maintain a changelog as part of the release process, documenting key modifications. However, the structure and level of detail in these changelogs can vary significantly across projects. Some projects follow the guidelines of keepachangelog.com, which recommends categorizing changes by type, such as Added, Changed, or Fixed, to ensure consistency and clarity [2]. A representative example of versioning discipline and structured changelog usage is provided by Capistrano, a widely adopted Ruby-based deployment automation tool [3]. Its release records demonstrate the application of Semantic Versioning principles and categorized change types (table 1).

TABLE I: RECENT CAPISTRANO RELEASES

Version	Release date	Type of changes	Description
3.19.2	2024-11-07	Fixed	Fixed `uninitialized constant Capistrano::SCM::Git::StringIO` error.
3.19.1	2024-07-02	Fixed	Addressed issue causing `REVISION_TIME` to hang during deployment.
3.19.0	2024-06-17	Added / changed	Introduced `REVISION_TIME` based on Git commit timestamp; improvements in test suite and CI infrastructure.
3.18.1	2024-03-10	Fixed / docs	Documentation updates; enhanced support for Ruby 3.3; improvements in CI pipeline.
3.18.0	2023-10-18	Changed	Removed legacy `webpacker` paths from `linked_dirs`; ensured compatibility with Ruby 3.3; migrated CI to GitHub Actions.

Version control systems, particularly Git, play a crucial role in maintaining the stability and traceability of versioning processes. Git tags corresponding to release versions serve as definitive markers of the codebase at the moment of release. In many Ruby projects, tags are created in the format vX.Y.Z, aligned precisely with the version defined in version.rb. Various branching and release strategies are employed, including Git Flow, trunk-based development, and dedicated release branches. Among Ruby projects, lightweight release models are commonly favored, characterized by frequent merges into the main branch (main or master) followed by the creation of version tags to formalize the release.

Among the automation tools available to developers, CI/CD platforms play a particularly important role, with GitHub Actions being the most widely adopted solution in open-source Ruby projects. This platform enables developers to define a series of steps required for preparing and publishing a release, including running tests, performing code style checks, building the gem, and publishing it to RubyGems. In most cases, the workflow is configured to trigger automatically upon pushing a version tag. While many projects use standardized workflows for building and releasing RubyGems, others follow customized release processes tailored to their specific distribution models.

Some Ruby projects, such as ruby/setup-ruby, use custom workflows that trigger manual releases via GitHub Actions [4]. These workflows often rely on internal Ruby scripts for release automation (e.g., release.rb) and update versioned branches (e.g., v1) instead of publishing RubyGems packages.

In some projects, automation extends further to include version number updates. Developers often rely on external utilities or custom scripts that generate version numbers and changelogs automatically, based on modifications recorded in CHANGELOG.md files and Git commit history, often using conventions such as Conventional Commits. These practices are becoming increasingly common, particularly in actively maintained libraries, where maintaining release discipline is essential for long-term sustainability.

Therefore, version management in open-source Ruby projects is a sophisticated but structured activity that embraces both manual rules and automatic tools, as well as proven development practices. Through using the principles of Semantic Versioning, the utilization of Git as the underlying system for tracking changes, and the adoption of

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 1, October 2025
Impact Factor: 7.67

continuous integration frameworks to enable the release process to be automated, the Ruby community has developed a flexible and robust framework. This model has proven effective across a wide range of projects, from small utility libraries to full-scale frameworks such as Rails.

III. DEPENDENCY MANAGEMENT AND THE RELEASE CYCLE: TOOLS AND PRACTICES FOR ENSURING SUSTAINABILITY AND RELEVANCE IN OPEN-SOURCE RUBY PROJECTS

In open-source Ruby projects, sustainability and maintainability are closely tied to effective dependency management and the organization of a well-structured release cycle. Ensuring long-term project stability requires not only thoughtful versioning but also a systematic approach to handling external libraries on which the software relies [5]. Errors or vulnerabilities in third-party components can affect hundreds of dependent projects, making processes such as updates, compatibility verification, and version releases essential elements of the development architecture. The overall release workflow in Ruby-based open-source projects can be illustrated as a structured pipeline, from code readiness to artifact publication. The figure 1 was adapted based on the official documentation for GitHub Actions and typical release pipelines found in Ruby open-source projects, including GitHub Docs: About workflows [6].

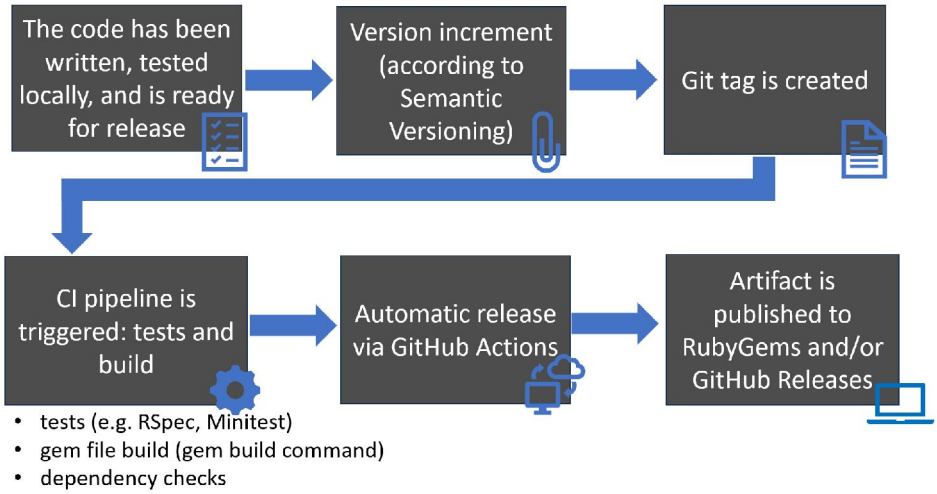

Fig. 1 A typical release workflow in open-source Ruby projects integrating versioning, CI/CD automation, and artifact publication using GitHub Actions

Ruby projects traditionally rely on a dependency management system based on the Gemfile and Gemfile.lock files, which are integral parts of the Bundler ecosystem. The Gemfile specifies the desired libraries and version constraints, while Gemfile.lock captures the exact versions installed in the current environment [7]. This approach supports reproducible builds, which is particularly important in team-based workflows and continuous integration environments. However, the locking of dependencies at specific versions introduces the need for regular updates, as newer releases may include both functional enhancements and critical security patches.

Automated tools are widely employed for systematic dependency updates and the monitoring of potential vulnerabilities. One of the most commonly used solutions is GitHub Dependabot, which is integrated into the GitHub infrastructure and automatically detects outdated libraries. It generates pull requests with proposed updates, including references to changelogs and alerts about potential incompatibilities. This functionality is particularly valuable given the frequent changes in the Ruby ecosystem and the regular release cadence of the language core and major frameworks. Some projects supplement Dependabot with custom update policies—for example, allowing automatic updates for minor and patch versions contingent on successful test runs, while requiring manual review for major version upgrades [8].

Automation also plays a pivotal role in the release cycle itself. Once dependency updates are approved, projects typically proceed through testing, build, and deployment phases, which are often fully automated using CI/CD

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 1, October 2025
Impact Factor: 7.67

platforms. Tools such as GitHub Actions, CircleCI, and others allow the release process to be defined as a structured pipeline, from executing test suites to building and publishing artifacts [9]. Projects that follow a consistent release schedule often document upcoming release dates or adopt calendar-based versioning in conjunction with Semantic Versioning. This practice enhances predictability and simplifies upgrade planning for downstream dependencies.

Compatibility management plays a critical role in maintaining the stability of Ruby-based open-source projects. Given the high degree of interdependence among Ruby libraries, it is essential to explicitly define the minimum and maximum supported versions of dependencies. Failure to enforce these boundaries can result in runtime errors and integration failures in downstream projects. To ensure backward compatibility, some developers implement additional test suites against earlier versions of their API or conduct integration testing within a CI matrix that spans multiple versions of Ruby and Rails. While such practices require additional effort, they significantly enhance the reliability of the software and foster trust within the community.

Another key practice is the maintenance of clear and accessible changelogs, which facilitate version transitions and support risk assessment during upgrades. Developers often complement these changelogs with detailed release notes that document minimum required dependency versions, configuration changes, migration steps, and upgrade instructions. This is particularly important for large-scale libraries such as Devise, Puma, or Dry-rb, where modifications to a single module may have wide-ranging effects across the broader ecosystem.

The long-term survival of open-source Ruby projects depends heavily on the complexity of update management policies and the openness of the release process. Under the rapidly evolving environment with diverse library background, the adoption of automatic update systems, careful analysis of interoperability, and the availability of information about the releases have become integral parts of good maintenance practices.

IV. CONCLUSION

In the open-source Ruby community, version and release management are an extremely well-organized and sophisticated system that combines established standards. These include Semantic Versioning, with flexible automation practices using tools such as Git, CI/CD pipelines, and platforms like GitHub Actions. Adhering to consistent standards and having a culture that emphasizes stable versioning, Ruby developer communities ensure reliability, compatibility, and a very high degree of confidence in the libraries they maintain.

At the same time, the handling of dependencies, periodic updates of components, complete changelogs, and clearly documented releasing procedures have become essential parts of an effective release cycle. Integration with tools like Bundler, Dependabot, and automated testing matrices allows Ruby projects to rapidly respond to changes in the programming environment and its supporting frameworks, lowering the risk and ensuring long-term maintainability. While the complexities inherent in the open-source model keep increasing, the system provided by the above practices becomes the basis for the stability and longevity of the applications created with them.

REFERENCES

[1]. L. Carvalho, J.C. Seco, Deep semantic versioning for evolution and variability, InProceedings of the 23rd International Symposium on Principles and Practice of Declarative Programming, 2021, pp. 1-13.

[2]. Main page / Keep a Changelog // URL: https://keepachangelog.com/en/1.1.0/ (date of application: 12.08.2025).

[3]. Capistrano / GitHub // URL: https://github.com/capistrano/capistrano/releases (date of application: 13.08.2025).

[4]. Release.yml / GitHub // URL: https://github.com/ruby/setup-ruby/blob/master/.github/workflows/release.yml (date of application: 15.08.2025).

[5]. E. Ponomarev, Practical application of ai and NLP in android mobile applications, Naukosphere, 2024, no. 12(1), pp. 22-27.

[6]. About workflows / GitHubs Docs // URL: https://docs.github.com/en/actions/writing-workflows/about-workflows (date of application: 19.08.2025).

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 1, October 2025
Impact Factor: 7.67

[7]. C. DiLeo, P. Cooper, C. DiLeo, P. Cooper, Projects and Libraries. Beginning Ruby 3: From Beginner to Pro, 2021, pp. 197-213.

[8]. H. Rebatchi, T.F. Bissyandé, N. Moha, Dependabot and security pull requests: large empirical study, Empirical Software Engineering, 2024, Vol. 29(5), no. 128.

[9]. S. Jackson, Setting Up GitHub Actions, InAccelerating Unity Through Automation: Power Up Your Unity Workflow by Offloading Intensive Tasks. Berkeley, CA: Apress, 2023, pp. 245-297