Paper SM11

Leveraging GitHub Actions and Continuous Integration in R Package Qualification

Matt Roumaya, Atorus Research, Philadelphia, PA, United States
Mike Stackhouse, Atorus Research, Chapel Hill, NC, United States
Jeremy Wildfire, Gilead Sciences, Durham, NC, United States

ABSTRACT

The open-source Good Statistical Monitoring, or {gsm}, R package is built to support Risk-Based Quality Management in active clinical trials. To support this GxP use case, a robust qualification framework is included in the package. The {gsm} qualification framework builds upon established best practices by leveraging a set of machine-readable specifications and test cases that are evaluated and summarized in a concise qualification report. This process is highly automated using continuous integration, and leverages existing R tools such as *testthat, R Markdown,* and *pkgdown*, as well as custom GitHub Actions that are tied to each package release. This paper offers a detailed exploration of the framework's architecture and processes, spotlighting its pivotal role in enhancing the stability and operational utility of the {gsm} R package.

DEFINITIONS

Qualification: Qualification ensures that the package is functioning as intended and that core functions execute as expected on a system-wide scale.
SDLC: Software development lifecycle; the process of planning, creating, testing, and deploying software.
RBQM: Risk-Based Quality Management; a framework for managing the overall quality of a clinical trial[2].
CI/CD: Continuous Integration and Continuous Delivery/Deployment, or continuous software development.

INTRODUCTION

Open-source software qualification is a complex topic, and best practices are evolving as organizations increasingly adopt open-source solutions. When discussing qualification in this paper, we are referring to the practice of scrutinizing a codebase's functionality, ideally with an independent reviewer, and documenting the review process, outcomes, and making all materials available for end-users to inspect.

{gsm} follows an internal qualification approach, which includes a suite of qualification tests that are routinely run as part of the software development lifecycle (SDLC), both as part of the development team's workflow and as a Continuous Integration (CI) / Continuous Deployment (CD) pipeline. As part of the release process, supplemental qualification documentation and reports are created and attached to the release as artifacts and made available on GitHub. These qualification materials provide additional transparency into the quality of {gsm} and allow for inspection by users of the package and the open-source community.

In addition to robust quality checks, CI/CD workflows are also run on multiple operating systems, as well as on multiple versions of R, which ensures that {gsm} runs correctly on current, legacy, and development versions of computing environments that end users might be using. Implementing this framework requires a low overhead cost, as all of the tools needed are open-source, and the setup is straightforward. This paper will provide a detailed explanation of {gsm}'s qualification framework and outline future development plans for spinning off the framework into a reusable lightweight qualification package .

{GSM} QUALIFICATION WORKFLOW

Qualification for {gsm} is done to ensure that the package is functioning as intended and that core functions execute as expected on a system-wide scale. While over 1,300+ unit tests exist to test the code, qualification testing is implemented as a distinctly separate facet that tests that the expected behaviors are happening correctly and are producing the expected outputs.

Qualification is done using a set of machine-readable documents and associated functions to create a strong documentation structure and a cohesive qualification report. This qualification process allows for new functionality to be reviewed for quality and documented as soon as it is introduced to the code base. Qualification tests are designed to provide developers with a repeatable process that is easy to follow and allows for modification of existing qualification tests if needed.

In practice, the process of qualification uses *specifications* and *test cases* which are then compiled into a *qualification report*.

SPECIFICATIONS

Specifications capture the most important use cases for a given function. Each qualified function must have at least one specification, and each specification must have at least one associated test case. Multiple specifications can exist for a function, and multiple test cases can be attributed to a specification.

Each specification should include the following components:

- **Spec ID:** A unique ID that is used to trace a specification throughout the qualification report and documentation.
- **Description:** Defines the use case for the specification. E.g., *Given appropriate input data, an Adverse Event Assessment can be done using the Normal Approximation Method*.
- **Risk:** A value of *Low, Medium,* or *High*, corresponding to the risk associated with the specification failing.
- **Impact:** A value of *Low, Medium,* or *High*, corresponding to the severity of the impact associated with the specification failing.
- **Associated Test IDs:** Test IDs of test cases that are associated with the specification, and collectively serve to confirm that the specification description has been qualified.

See Figure 1, below, for an example of specifications.

TEST CASES

Test cases translate specifications into programmatic qualification tests. These test cases are written using the {testthat} R package, which seamlessly integrates into most R package software development workflows and is the most popular unit testing package for R[4].

Multiple test cases are usually connected to a single specification. This is because each test case tends to qualify a specific aspect of a function's potential use. For example, a specification stating that "*Given appropriate input data, an Adverse Event Assessment can be done using the Normal Approximation Method"* may require two individual test cases that address the most common grouping variables (e.g. Site and Country).

See Figure 1, below, for an example of test cases. See here for a closer look at a test case as it is written in {gsm}.

Figure 1

Specifications

Spec ID	Spec Description	Risk	Impact	Associated Test IDs
S1_1	Given appropriate input data, an Adverse Event Assessment can be done using the Poisson method.	High	High	T1_1, T1_2, T1_3, T1_10
S1_2	Given appropriate input data, an Adverse Event Assessment can be done using the Identity method.	High	High	T1_4, T1_5, T1_6, T1_11

Test Cases

Function	Spec ID	Test ID	Test Description	Test Result
AE_Assess	S1_1, S1_4, S1_5, S1_6	T1_1	Given an appropriate subset of Adverse Event data, the assessment function correctly performs an Adverse Event Assessment grouped by the Site variable using the Poisson method and correctly assigns Flag variable values when given a custom threshold.	Pass
AE_Assess	S1_1, S1_4, S1_5, S1_6	T1_2	Given an appropriate subset of Adverse Event data, the assessment function correctly performs an Adverse Event Assessment grouped by the Study variable using the Poisson method and correctly assigns Flag variable values.	Pass

Test Results

Function Name	Number of Tests	Number Passed	Number Failed	Number Skipped
AE Assess	12	12	0	0

QUALIFICATION REPORT

The Qualification Report is generated to provide a summary of the package's quality control process and outputs. The report is attached to each release and included in the {pkgdown} site and showcases the qualification status of all

assessments within {gsm}. It presents an overview of qualification test results, unit test coverage, the qualification testing environment's `sessionInfo()`, and a package list with risk scores. Additionally, it summarizes all Pull Requests since the last release, offering details such as title, branches, links, requester, reviewers, date requested, and status.

This comprehensive approach ensures the transparency and reliability of {gsm} with each new release.
The report lives as a Qualification vignette on GitHub, and is also rendered as an equivalent PDF document. The PDF document is attached to each release as an artifact, which means that each release includes its own qualification report that can be referenced at any point in the future. See {gsm}'s releases page on GitHub to download the qualification report in PDF format.

QUALIFICATION DEVELOPER

In addition to having a testing framework that is separate and distinct from unit tests, qualified functionality in {gsm} is designed, coded, and confirmed by a *qualification developer* who is intentionally separate from the core package development team to reduce bias and ensure an objective approach to qualification. In practice, this means that an R programmer with domain knowledge will review the specifications and test cases with the development team and independently "double-program" the tests using {testthat} as described above.

IMPLEMENTATION DETAILS

FILE STRUCTURE

The quality framework described above is implemented by adding the following files to the standard R Package framework:

- Specifications (`inst/qualification/qualification_specs.csv`; `{gsm} example`) – Specifications are saved as a `.csv` under `/inst`. See figure 1 for examples.
- Tests (`tests/testqualification/qualification/test_qual_{TEST_ID}.R`; `{gsm} example`) – Tests are saved in a custom location under `tests/`. Qualification tests use a single `test_that()` function call per file. The qualification tests are run via `testthat::test_dir(here:here("tests", …)`.
- Qualification Report Template (`vignettes/articles/Qualification.Rmd`, `{gsm} example`) – The Qualification report is implemented as a custom Vignette.
- CI/CD via Github Actions
    - Actions to run Qualification Checks (`.github/workflows`, `{gsm} example`) – triggers execution of qualification tests on pull requests to the `dev` and `main` branche.
    - Action to generate Qualification Report (`.github/workflows`, `{gsm} example`) - triggers creation of qualification report on pull requests to the `main` branch, as well as whenever a release is published. This builds the qualification report as an HTML vignette page, as well as a PDF report that is attached to each new release.

DEVELOPMENT WORKFLOW

We describe the development workflow and release process for {gsm} in this vignette. When the package is updated, the specifications and tests should be updated to document the quality of all new and existing functionality. Otherwise, little maintenance is required. Qualification tests and reporting are run automatically via continuous integration using GitHub Actions.

CI/CD

GitHub Actions is a CI/CD platform that allows software development teams to create pipelines for automating standard processes that happen when creating new code and software products[3]. The *actions* are commonly referred to as *workflows*, and are typically triggered when some change has been made in a GitHub repository.

YAML files are used as guidelines for GitHub Actions, detailing step-by-step instructions on how to setup a computing environment and then perform computing actions within that environment. In other words, YAML files serve as configuration files that define the tasks to be performed in a CI/CD pipeline.

In a standard GitHub repository structure, all YAML files that define GitHub Actions workflows reside within a designated folder named `.github`. This is all that is needed for GitHub Actions to be setup in a repository; GitHub automatically detects this folder and attempts to run all YAML files found within `.github/workflows`.

The development checks are run on a more limited system scope (e.g., only on Ubuntu) to speed up development time, and the main checks include Ubuntu as well as Windows and MacOS. These files run the qualification testing suite and will alert the developer if a single test fails.

FUTURE DEVELOPMENT

Development of {gsm} has been active for over two years in its current repository, and the qualification framework outlined in this paper has successfully achieved what the developers intended. We plan to create a lightweight R package that can be used as a utility to scaffold a similar framework for other developers who wish to implement a qualification process or framework.

CONCLUSION

The {gsm} package utilizes a lightweight qualification framework featuring machine-readable specifications and test cases that are evaluated and summarized in a comprehensive Qualification Report. The qualification process allows for independent review and documentation of new functionality as it is introduced to the codebase. The CI/CD workflows, run on multiple operating systems and R versions, ensure compatibility across diverse computing environments, and fit in nicely with an Agile development team.

The Qualification Report, generated as an R Markdown document, documents the quality of the functions that make up {gsm}, and is attached to each release, providing insights into qualification test results, unit test coverage, testing environment details, and an overview of Pull Requests. The development team's intentional separation of the qualification developer from the core development team ensures an objective and unbiased approach to qualification.

Looking ahead, the development team is planning to create a lightweight R package to serve as a utility for developers wishing to implement a similar qualification process or framework. By sharing their experience and best practices, the {gsm} development team aims to contribute to the broader open-source Pharma community's efforts in enhancing software quality.

REFERENCES

1. https://en.wikipedia.org/wiki/Good_clinical_practice
2. https://cyntegrity.com/whats-the-difference-between-rbqm-and-qrm-in-clinical-trials/
3. https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions
4. Wickham H (2011). “testthat: Get Started with Testing.” The R Journal, 3, 5–10. https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf

ACKNOWLEDGMENTS

Many thanks to the contributors to {gsm}.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Matt Roumaya
Company: Atorus Research
Email: matt.roumaya@atorusresearch.com
Website: https://www.atorusresearch.com/

Author Name: Mike Stackhouse
Company: Atorus Research
Email: mike.stackhouse@atorusresearch.com
Website: https://www.atorusresearch.com/

Author Name: Jeremy Wildfire
Company: Gilead Sciences
Email: jeremy.wildfire@gilead.com
Website: https://www.gilead.com/