Paper RM05

Scalable R Packages Qualification Using a CI/CD Approach

Jon Harmon, Atorus Research, USA
Natalia Andriychuk, Gilead Sciences, USA
Spencer Childress, Gilead Sciences, USA
Laura Maxwell, Atorus Research, USA
Zelos Zhu, Atorus Research, USA
Jeremy Wildfire, Gilead Sciences, USA

ABSTRACT

As the pharmaceutical industry increasingly adopts R, ensuring software quality and compliance becomes critical. This presentation describes the {qcthat} R package (https://gilead-biostats.github.io/qcthat/), a lightweight qualification framework that extends R package development and GitHub best practices with custom GitHub actions that document package quality with a focus on documenting GxP compliance. The {qcthat} R package was developed in support of the {gsm} family of R packages (https://gilead-biostats.github.io/gsm.core/), which supports Risk-Based Quality Management in clinical trials.

The {qcthat} process documents package requirements using GitHub issues, which are then linked to tests implemented with standard tools such as the {testthat} and {shinytest2} R packages, and ran using GitHub actions. {qcthat} then provides custom reports that link requirements and test results. The result is a scalable, low-overhead approach to software quality that's reproducible across platforms and adaptable to any R package created using best practices in R and GitHub.

INTRODUCTION

R package qualification is necessary in regulated environments, but it can be challenging. Regulators require evidence that software performs as intended. Qualification is often a manual, retrospective burden, which can delay releases and increase risk. The standard R package lifecycle generates evidence but lacks traceability. Modern R packages use {testthat} for robust automated testing. Continuous integration / continuous deployment (CI/CD) pipelines include automated checks every time a package is updated. Standard {testthat} CI/CD scripts provide the evidence of implementation, but the evidence is out of context; a passing test suite doesn't explain which requirements were met. {qcthat} provides a straightforward syntax to connect issues to tests for continuous qualification. {qcthat} transforms qualification from a retrospective burden into a seamless byproduct of development. Qualification reports automatically appear in Pull Requests and GitHub Releases. Packages are qualified continuously with every change.

SAMPLE REPORT

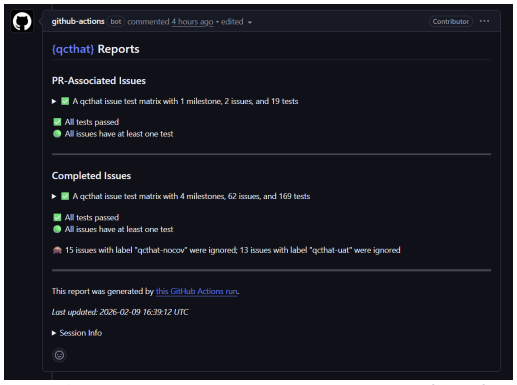

Figure 1. This report was automatically attached to a {qcthat} pull request on GitHub by a GitHub Action provided by {qcthat}.

Figure 2. An expansion of the "Completed Issues" section of the report shown in Figure 1.

REQUIREMENTS

To use {qcthat} with your R package, you must:

1. **Use Git and GitHub to manage the source code of your R package.** If you do not already do so, we recommend using the {usethis} R package (https://usethis.r-lib.org) to make the process more straightforward. Specifically, `usethis::use_git()` to activate git version control for your package, and usethis::use_github() to connect your package to a GitHub.com repository. See the usethis setup article (https://usethis.r-lib.org/articles/usethis-setup.html) for more details.
2. **Use GitHub issues to track requirements for your R package.** If you do not already do so, you will need to create at least one issue describing the functionality of your package. {qcthat} is agnostic to the exact nature of these issues.
3. **Use {testthat} to test your R package.** The {testthat} R package (https://testthat.r-lib.org/) is the industry standard for confirming that your package functions how you believe it functions, and continues to do so after you make changes. You can activate {testthat} testing with `usethis::use_testthat()`.

INSTALLATION

You can install {qcthat} from GitHub with:

```
install.packages("pak")
pak::pak("Gilead-BioStats/qcthat")
```

Each release of {qcthat} (https://github.com/Gilead-BioStats/qcthat/releases) includes detailed quality reports embedded in the release description[1], providing evidence for both the issues addressed in that particular release ("Milestone") and all issues addressed by the package ("Completed Issues").

ACTIVATION

To activate {qcthat} for your package with the recommended settings, run `qcthat::use_qcthat()`. This will do the following:

1. Add a GitHub Actions workflow at `.github/workflows/qcthat.yaml` in the active project. This action attaches {qcthat} reports to GitHub pull requests and releases, and manages user-acceptance tests.
2. Add {qcthat} labels to the associated GitHub repository. These include "qcthat-nocov" (to indicate issues that should not have associated tests), and "qcthat-uat" to indicate issues created by {qcthat} for user-acceptance testing. More labels may be added in the future.

As we add additional functionality to the {qcthat} package, we may include additional features in `use_qcthat()`.

IMPLEMENTATION

To associate a test with a particular GitHub issue, add "`(#ISSUE)`" to the `desc` field of the `test_that()` call. For example, the test below is tagged to issue 162 (https://github.com/Gilead-BioStats/qcthat/issues/162, emphasis added):

```
test_that("GuessIssueNumber returns NULL for bad extracted issue number (#162)", {
  expect_null(GuessIssueNumber(list(issue = list(number = "a"))))
})
```

Tests can be tagged to multiple issues, separated by commas. For example, this test is tagged to both issue 84 (https://github.com/Gilead-BioStats/qcthat/issues/84) and issue 132 (https://github.com/Gilead-BioStats/qcthat/issues/132):

```
test_that("GetGHAPRNumber returns NULL for bad extracted PR number (#84, #163)", {
  expect_null(GetGHAPRNumber(list(pull_request = list(number = "a"))))
})
```

Issues and their tagged tests can be viewed in reports.

REPORTS

The primary outputs of {qcthat} are "issue-test matrices," data frames which link together {testthat} test results (the test description and whether it passed, failed, produced warnings, or was skipped) and GitHub data (issue type, number, title, and milestone, and whether the issue is open or closed). These data frames print as reports in an easy-to-read format, with issues grouped by milestone (if available), and tests nested under their linked issues.

An overall report for a package can be generated with the `QCPackage()` function. For example, this is a subset of the report generated by `QCPackage()` applied to {qcthat} itself:

```
├─🟦─Milestone: v1.0.0 (21 issues, 51 tests)
│ ├─☑─Requirement 68: PR/Branch Report
│ │ ├─✅─Action_QCPRIssues targets the expected action (#55, #68)
│ │ ├─✅─Action_QCMilestone targets the expected action (#88, #68)
│ │ ├─✅─QCMergeGH filters to merge-associated issues (#68, #84)
│ │ ├─✅─QCMergeLocal filters to ref-specific issues (#68, #84)
│ │ ├─✅─QCPR filters to PR-related issues (#68, #84)
│ │ └─✅─QCMilestones reports on specific milestones (#88, #68)
<… additional features removed …>
# Issue state: 📥 = open, ☑ = closed (completed), ⛔ = closed (won't fix)
# Test disposition: ✅ = passed, ❌ = failed, 🚫 = skipped

✅ All tests passed
⭕ 36 issues lack tests
⛓️‍💥 15 tests are not linked to any issue
🙈 26 issues with label "qcthat-nocov" were ignored; 16 issues with label "qcthat-uat" were ignored
```

Since the issue-test matrices are data frames, they can be filtered to display meaningful subsets of data. Convenience wrappers for common subsets are available, such as:

- `QCCompletedIssues()` for issues that have been closed as completed on GitHub.
- `QCMilestones()` for issues within a particular milestone or milestones.
- `QCPR()` for issues tagged to the current pull request on GitHub.

With the default configuration, all three reports are automatically added in a comment on pull requests on GitHub (with each report collapsed into an expandable summary). The "Completed Issues" and "Milestone" reports are also added to the descriptions of GitHub releases (again collapsed into an expandable summary per report).

CONCLUSION

The {qcthat} R package allows R package developers to seamlessly integrate qualification information directly into R package development workflows. It associates qualification information with releases directly and automatically, making it much easier to provide evidence that packages meet qualification requirements.

Visit the {qcthat} site at https://gilead-biostats.github.io/qcthat/ for up-to-date information about this package and system.

Commented [NA1]: I'm thinking we could add a short sentence, something like "Once the issues are tagged in the tests, both are ready to be used in the reports." or something along those lines to connect the usage portion with the report portion. Might be helpful for new users.

Commented [J(1R2]: I'm working on wording. Technically you can (and likely should) generate a report before you have a single tagged issue. I did for both qcthat and gsm.kri to see the issues and set a baseline of what's left to tag.

Commented [J(1R3]: Good enough IMO.

FOOTNOTES

1. For release v0.2.0 and v1.0.0, the report was attached to the release as a txt file.

RECOMMENDED READING

Bryan, Jennifer, *et al. Happy Git and GitHub for the useR*. https://happygitwithr.com/, 2016-2025.
Wickham, Hadley, and Jennifer Bryan. *R Packages (2e)*. https://r-pkgs.org/. O'Reilly, 2023.

CONTACT INFORMATION

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

Author Name: Jon Harmon
Company: Atorus Research
Email: jon.harmon@atorusresearch.com
Website: https://www.atorusresearch.com/