Engineering a Reliable R Package for Regulatory Use Using “rpact” as an Example

useR! 2024, Salzburg, Austria

Friedrich Pahlke and Gernot Wassmer
RPACT

July 10, 2024

Motivation: Why validate an R package?

Why validate an R package?

In many industries, it is necessary to ensure that the software does what it is supposed to do, e.g.:

- **Finance**: Trading systems, payment platforms, risk management
- **Automotive**: Autonomous driving, engine control, safety systems
- **Manufacturing**: Automation, production monitoring, quality control
- **Pharmaceuticals and Medical Devices**: FDA/EMA regulations, trial design and analysis, diagnostic software

… R is used in more and more industries.

Why validate an R package?

Validation guidelines in various industries:

- **GxP**[1]: “Good Practice” guidelines

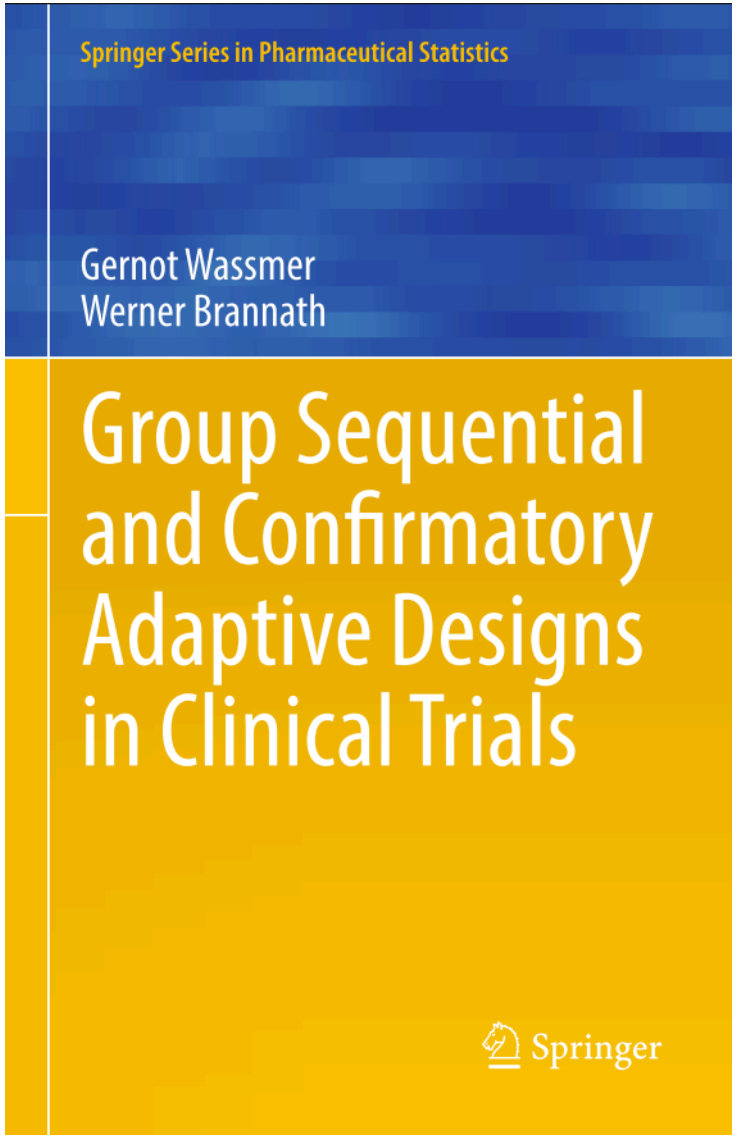
Examples:

- **GMP**: Good Manufacturing Practice
- **GLP**: Good Laboratory Practice
- **GCP**: Good Clinical Practice

Why validate an R package?

Example today:

- Regulatory use in pharmaceutical[1] industry
- Software validation is essential for the approval of new drugs and treatments
- Major risk of faulty software: Incorrect decisions about the safety and efficacy of drugs or treatments



The “rpact” Example: Engineering a Reliable R Package for Regulatory Use

- 2017: Idea

  …

- 2024: “rpact” and RPACT are trusted and widely accepted

What does GxP-compliant validation mean for R packages?

- Show that the R package is “reliable”[1]
- This can be demonstrated through traditional formal validation, as shown later with “rpact”
- Or through a less formal “Risk Assessment” approach, as promoted by the R Validation Hub (pharmar.org/risk)

Status 2024: RPACT in Numbers

- 1,500,000 € sponsorship money
- 62,000 CRAN downloads total; 1,600[1] downloads per month (June 2024)
- 36,593 lines of code
- 34,427 unit tests covering 29,701 lines of code (see codecov.io)
- 20,000 hours of work
- 98 separate source code files: Modular software design
- **29 vignettes** based on Quarto and published on rpact.org/vignettes
- 28 releases on CRAN since 2018

- **20 SLA sponsors**: “The RPACT User Group”
- **8 websites**: CRAN, rpact.com, rpact.org, GitHub, GitHub Pages, OpenSci R-unisverse, METACRAN, Codecov
- 4 CI/CD pipelines on GitHub to automate checks, tests, code coverage calculation, and GitHub pages creation
- 3 imported dependencies: knitr, Rcpp, and R6
- 3 suggested dependencies: ggplot2, testthat, and rmarkdown
- 2 main developers; 3 contributors; feedback and feature requests from many different users and companies
- 2 rpact Shiny apps: rpact Shiny app and RPACT Cloud
- 1 developer platform: GitHub is used for issues, comments, feature requests

Project Challenges 2017

Challenge 1: Funding

- Crowdfunding
  $\longrightarrow$ 10 pharma companies and CROs agreed to sponsor the project
- Service Level Agreement
  $\longrightarrow$ Together we developed a simple contract: software support and training; no software development!
- RPACT was founded as a GbR $\longrightarrow$ Easiest and fastest solution for freelancers

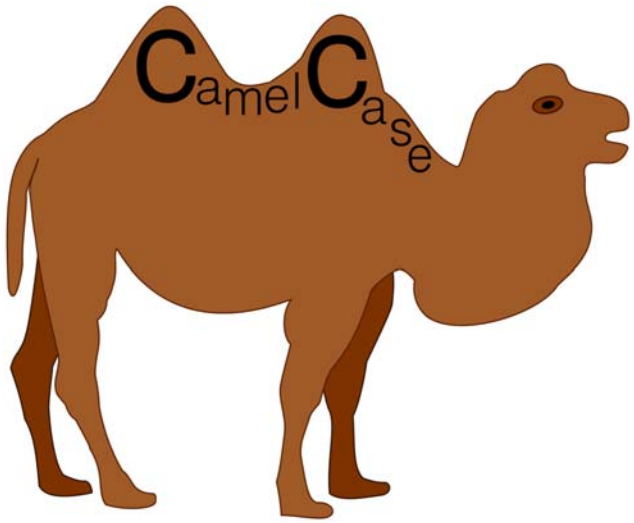
Current "RPACT User Group": Boehringer Ingelheim, Metronomia Clinical Research, F. Hoffmann-La Roche, Dr. Willmar Schwabe, Bayer, Merck, AbbVie, Dr. Falk Pharma, Klifo, FGK Clinical Research, UCB, GKM, Parexel, Nestlé, Janssen (Johnson & Johnson), Novartis, PPD (Thermo Fisher Scientific), Sanofi, Pfizer, Gilead

Challenge 2: Realization with a small team

- Collaboration, project management, version control, bug tracking, feature requests
  $\longrightarrow$ GitHub
- Integrated development environment (IDE)
  $\longrightarrow$ Eclipse + StatET, RStudio IDE
- Clean code rules for R packages
  $\longrightarrow$ We developed own guidelines inspired by Java and widely accepted clean code rules (see Robert C. Martin (2009) Clean Code)
- Test, validation, and release
  $\longrightarrow$ Automation wherever possible

Challenge 3: Sustainable user concept

- Usability (easy to learn and use; high user acceptance):
  - Many default arguments $\longrightarrow$ Getting started is much easier
  - Support of R generics $\longrightarrow$ `print()`, `plot()`, `summary()`, `names()`, ...
  - Inline help and documentation $\longrightarrow$ roxygen2
  - Vignettes $\longrightarrow$ Practical examples and tutorials
- Consistency:
  - Lower camel case names
  - Clear output format: structured and meaningful



Challenge 4: Validation Concept

- Formal validation inspired by "GAMP[1] 5" principles
- As few dependencies as possible because we cannot validate other R packages
- We assume that base/core R is validated/reliable (see R-FDA.pdf)
- High test coverage[2]:
  Usage of covr and codecov.io

- Comprehensive validation documentation
- Automation of recurring validation processes/activities
  - Utility package `rpact.validator`
  - CI/CD pipeline on GitHub to automate checks, tests, code coverage calculation, and GitHub pages creation
- Template-based unit tests: Automatic generation of
  - testthat test cases
  - test plans and references to functional specifications
  - test protocols directly linking to individual test cases

Basic Idea of Template-Based Unit Testing

- **Step 1**: Compare software results manually, e.g., with simulation results and results from the literature and/or other programs
  $\rightarrow$ Reference point is correct and trustworthy
- **Step 2**: Fix the validated state, i.e., generate unit tests which test the software systematically and reproducibly
  $\rightarrow$ Further development and refactoring do not cause undetected side effects

Advantages of the Template-Based Approach

1. **Automation and Consistency**:
    - **Uniform Test Structure**: Improving maintainability and readability
    - **Automated Test Generation**: Reduces manual effort and minimizes errors
2. **Flexibility and Extensibility**:
    - **Easy Switch of Test Packages**: Replace `testthat` by another test framework
    - **Extensibility**: Add new tests for all functions and results in one code location

3. **Traceability and Documentation**:
    - **Granular and Traceable Tests**: Each test case is clearly defined and traceable, making debugging easier.
    - **Documentation**: References to functional specifications and software design specifications can be defined in the templates
4. **Efficiency Improvement**:
    - **Time Savings**: Test templates can be reused to generate tests automatically
    - **Scalability**: Tests can easily be scaled to new functions and modules, increasing test coverage

Conclusion

- Validation of an R package is challenging, time consuming, and expensive
- Applying good software engineering rules is essential to achieve high user acceptance, especially in a GxP regulated environment
- The template-based unit testing approach offers a structured, flexible, and efficient method for software quality assurance
- The combination of manual validation and automated verification ensures that the software remains stable and reliable, even as it is further developed or refactored
- In small teams, a high level of automation right from the start makes life much easier

Further Information

- Example of a template-based unit test definition: rpact-com.github.io/user2024-presentation
- Visit our website: rpact.com
- Ask a question or connect: Friedrich Pahlke

Thank you!