Deep Dive Into Industry R Package Quality Assessment

useR! 2024 Salzburg

Szymon Maksymiuk
Lorenzo Braschi

Outline

A. Package Quality Assessment
   a. General Idea and key points
   b. Process overview

B. Open source contributions
   a. *covtracer*
   b. *checked*
   c. *rd2markdown*

Package Quality Assessment

Validation: what is is and why do we need it

Documenting package quality

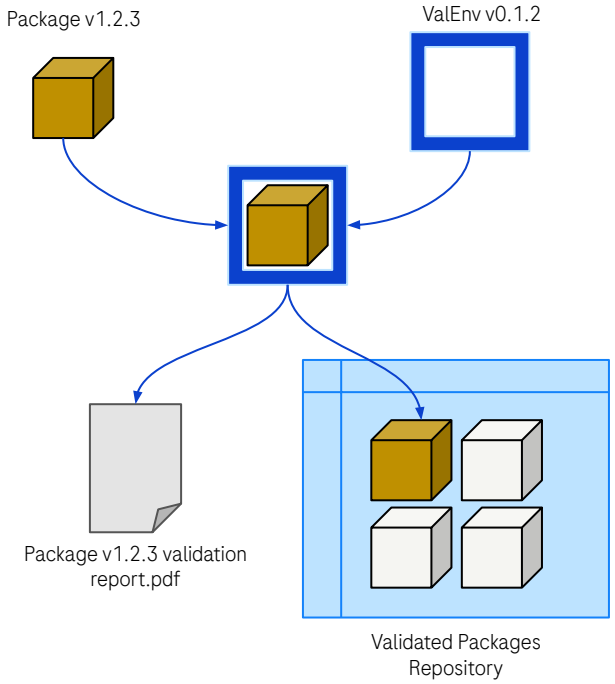
- Software used for submissions to Health Authorities needs to be properly documented.
- R packages are software components and as such they need to comply with regulations regarding computerised systems
- Validation centers around documenting that a package accurately and consistently meets its specifications
- As part of the R Validation Hub which promotes the development of resources for cross-industry use.

> "Validation: Establishing **documented evidence** which provides a **high degree of assurance** (**accuracy**) that a specific process **consistently** (**reproducibility**) produces a product meeting its **predetermined specifications** (**traceability**) and quality attributes."
>
> - FDA's Glossary of Computer System Software Development Terminology

Validation: what does it mean in practice?

Our approach to Validation

- Packages are considered for a specific **version**
- Packages must work in the context of a **validated environment**
- **Ideally**, each package passes a series of automated checks within that environment:
  - R CMD check
  - Unit Tests
  - Minimum **coverage**: 80%
  - Complete **traceability** (each exported function must be targeted by 1+ tests)
  - Reverse dependency checks
- Successful runs generate a PDF validation report
- The package is made available via our **validated packages repository**



Package quality checks

Theme	Description
Source Control	Reproducible source code ensured with git hash or a tar.gz checksum
Documentation	The package has clear ownership, documented as Authors & Maintainer fields in DESCRIPTION All exported objects are documented. These comprise our software requirements
R CMD check	The package passes R CMD check without ERRORs R CMD check WARNINGs or NOTEs can be remediated on a case basis
Testing	All evaluated unit tests succeed Code coverage is at or above 80%
Traceability	All exported functionality is evaluated by at least one unit test
Reverse dependency	Reverse dependencies pass R CMD check after installing the package Applicable only to package updates and systems dependent on our internal package repository

Addressing gaps in the validation process

Gaps are evaluated on a case-by-case basis.
Rationale to justify package gaps often falls in four major themes:

Consideration	Rationale
Complex Systems	When testing is minimal presumably because of system complexity, other indicators of quality may be more informative
Decision Impact	Packages that are unlikely to impact critical decision making requires less stringency
Adoption & Longevity	Wide adoption or historical stability may be good indicator of quality
Developer Trust	Packages may be developed by established members of the R community, trusted institutions or have corresponding peer-reviewed publications

Advantages of our approach

- ✓ Packages can be **independently validated for different versions** of R
- ✓ Packages are validated within (a copy of) the system, but they are **independent** of it
- ✓ Reproducibility managed through **repository snapshots** & image tags, not physical system immutability
- ✓ This allows a more **flexible approach** to validation, as packages are validated **individually**
- ✓ Packages can be **validated on demand**, without depending on long release cycles for validation
- ✓ This allows **reducing the time for validation** from a yearly cycle to a matter of a few days, minutes even for some fast packages.

Open source contributions

How Can We Contribute to Open Source?

- We want to **build consensus** on validation approaches industry wide (part of our efforts as members of the R Validation Hub).
- Making the approach open source **allows for contributions** from other players in the industry – and beyond.
- Broader user base provides new perspectives and **passive testing** of the package.
- We want to participate in and contribute to the **growing R ecosystem**

General overview

Roche open source packages for code validation

checked

- Orchestrate multiple R CMD checks runs managing their dependencies.
- Run reverse dependency check.
- Check all R packages in the given directory.

rd2markdown

- Convert arbitrary .Rd files into markdown files.
- Extract installed packages' documentation and convert it to markdown.
- Interact with raw Rd R objects.

covtracer

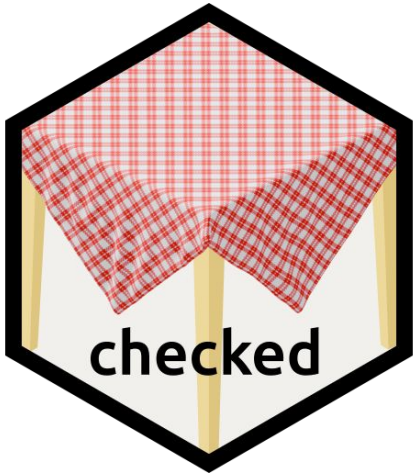
- Map tests to objects they test.
- Identify particular functions with 0% tests coverage.
- Identify functions tested only implicitly and not directly.

checked

Idea behind the package

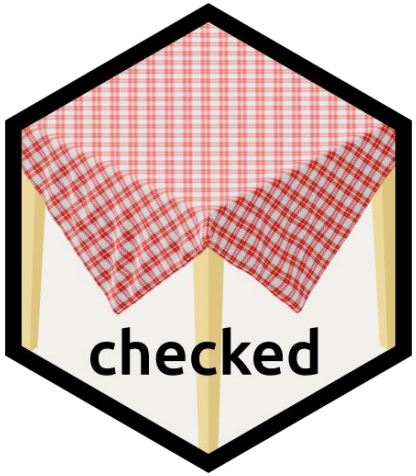
Regulatory compliant reverse dependency checks

- Maintenance of the regulatory-compliant validated packages repository requires the ability to **reliably and quickly perform reverse dependency checks**.
- In the R ecosystem, there are excellent tools that already address the reverse dependency check problem:
  - *revdepcheck,*
  - *tools::check_packages_in_dir(),*
  - *r-devel's recheck.*
- Each of these tools does it differently and thus **only partially fits all regulatory requirements** of existing tools, given that we decided to create the package to run **regulatory compliant reverse dependency checks**.



The checked package

R CMD check orchestrator



- A general R CMD check orchestrator.
- The package automatically manages dependencies and runs multiple checks as parallel subprocesses.
- All dependencies are installed in a dedicated library and reused for all check processes.
- The log for each subprocess process is stored to allow comfortable troubleshooting if any issue occurs.
- Most important data regarding the run are captured in the file system, meaning the analysis can be interrupted and restored later at any point.

Supported use-cases

```
check_reverse_dependencies(
  path = "~/Desktop/validation/code/DALEX/",
  n = 20,
  output = "~/Desktop/testing_building/reverse"
)
```

```
                  | S | OK  N  W  E |
testthat (v1.0.0) | ◑ | 47  1  0  0 |
   testthat (dev) | ◐ | 47  1  0  0 |
```

1: Sequential R CMD check [general]

- It can be used for specific packages or directories.
- Identifies a collective set of dependencies required to run all checks and schedules their installation.
- Tasks are performed prioritizing checks, as they take longer to run.
- It can use any number of parallel subprocesses.

2: Reverse dependency check

- A particular case of the general case - it runs a full reverse dependency check using the package's source (development) version.
- It can be run against any CRAN-like package repository.
- Compares R CMD check results for each reverse dependency when using the development and release version of the package, including potential issues.

Supported use cases - results

```
ceterisParibus package R CMD check diff
notes: OK
warnings: NEW POTENTIAL ISSUES [1]

@@ -1,4 +1,5 @@
 checking whether package ‘ceterisParibus’ can be installed ... WARNING
 Found the following significant warnings:
+  Warning: S3 method ‘yhat.randomForest’ was declared in NAMESPACE but not found
   Warning: package ‘ggplot2’ was built under R version 4.3.2
```

- The reverse dependency check API compares two R CMD check results - one using the development version of the package and the other one using the release.
- Whenever an issue is identified in the check using the development version of the package that is not present in the release, it is flagged and reported back.
- Contrary to existing checks, we check not only whether the issue has the same header but also whether the content is the same. If they are different, it's flagged as a potential issue.

covtracer

Idea behind the package

Mapping tests to specific files

- Most of the existing developer pipelines are running *testthat* and *covr* to support constant integration.
- Out of the box, they provide diagnostic data with little depth and limit it to metrics like total coverage or number of passed tests.
- The *covtracer* package stands out with its unique purpose of adding more depth to test data. It achieves this by connecting each test to associated objects.
- The package utilizes a Roche-contributed extension to the *covr* package, which maps tests to recorded coverage traces. Source references must also be kept when installing the package.

```
ttdf %>%
  filter(!doctype %in% c("data", "class")) %>%  # ignore objects without testable code
  select(test_name, file) %>%
  filter(!duplicated(.)) %>%
  arrange(file)
#>                                                    test_name                       file
#> 1            Example R6 Accumulator class methods are traced             Accumulator.Rd
#> 2         Example R6 Accumulator class constructor is traced             Accumulator.Rd
#> 3                                                       <NA>                   adder.Rd
#> 4            Example R6 Accumulator class methods are traced                   adder.Rd
#> 5               Calling a deeply nested series of functions.      complex_call_stack.Rd
#> 6             Calling a function halfway through call stack.  deeper_nested_function.Rd
#> 7               Calling a deeply nested series of functions.  deeper_nested_function.Rd
#> 8  hypotenuse is calculated correctly; with negative lengths              hypotenuse.Rd
#> 9                          hypotenuse is calculated correctly              hypotenuse.Rd
#> 10                  S4Example increment generic method works                increment.Rd
#> 11                              S4Example names method works  names-S4Example-method.Rd
#> 12                                                      <NA> names-S4Example2-method.Rd
#> 13              Calling a deeply nested series of functions.         nested_function.Rd
#> 14         Example R6 Person class public methods are traced                   Person.Rd
#> 15                                                      <NA>                    Rando.Rd
#> 16  Example R6 Rando class active field functions are traced                    Rando.Rd
#> 17                                                      <NA>              rd_sampler.Rd
#> 18            Calling a function halfway through call stack.      recursive_function.Rd
#> 19              Calling a deeply nested series of functions.      recursive_function.Rd
#> 20                                                      <NA>        reexport_example.Rd
#> 21                                                      <NA>               reexports.Rd
#> 22                 s3_example_func works using list dispatch         s3_example_func.Rd
#> 23              s3_example_func works using default dispatch         s3_example_func.Rd
#> 24                                                      <NA>                       <NA>
```

Supported use cases

Among many use cases, there are three we had mainly in mind when designing the package:

- **Generate Traceability Matrix** - Presents which requirements, in our case, R objects, are tested by which test and captures their description. It allows the identification of whether certain tests capture behaviours they were designed to capture.
- **Identify untested functions** - *covtracer* allows us to programmatically check whether there were functions not tested at all to ensure untested behaviours are immediately flagged.
- **Flag directly tested functions** - The package can differentiate which functions are tested directly by being called in the definition of the test and which were traced only by being called indirectly.

rd2markdown

Idea behind the package

Convert .Rd files to markdown

- Existing tools like *base* R (R CMD) or *tidyverse* (*pkgdown*) provide reliable ways to convert documentation into text or HTML files, **but not directly to markdown**.
- There are packages performing conversion to plain markdown, most notably **Rd2md**. However, they were underdeveloped at the time and could not fit our needs.
- The package provides a convenient, agnostic, fast method to **convert an arbitrary .rd file into a markdown**.
- It loads documentation as R objects and gradually **uses S3 dispatch to peel nested .Rd tags** - like peeling an onion!

How to use it?

There are two leading operation modes for the *rd2markdown* package, which might be applicable depending on the use case:

- render arbitrary .Rd files into markdown,
- extract specific topic from the installed package.

Users can extract specific topic parts of the documentation, such as the title, details, etc., and even supply custom Rd macros to further tailor the output.

At any point, it is also possible to interact with raw Rd R objects.

```
rd2markdown::rd2markdown(
  topic = "rnorm",
  package = "stats"
) |> writeLines("rnorm.md")

rd2markdown::rd2markdown(
  file = "./DALEX/man/explain.Rd"
) |> writeLines("explain.md")
```

rd2markdown in practice

DALEX - explain

Details

Please NOTE that the `model` is the only required argument. But some explanations may expect that other arguments will be provided too.

Returns

An object of the class `explainer`.

It's a list with the following fields:

- `model` the explained model.
- `data` the dataset used for training.
- `y` response for observations from `data`.
- `weights` sample weights for `data`. `NULL` if weights are not specified.
- `y_hat` calculated predictions.
- `residuals` calculated residuals.
- `predict_function` function that may be used for model predictions, shall return a single numerical value for each observation.
- `residual_function` function that returns residuals, shall return a single numerical value for each observation.
- `class` class/classes of a model.
- `label` label of explainer.
- `model_info` named list contating basic information about model, like package, version of package and type.

References

Explanatory Model Analysis. Explore, Explain and Examine Predictive Models. https://ema.drwhy.ai/

rd2markdown in practice

stats - rnorm

The Normal Distribution

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Arguments

- `x, q` : vector of quantiles.
- `p` : vector of probabilities.
- `n` : number of observations. If `length(n) > 1`, the length is taken to be the number required.
- `mean` : vector of means.
- `sd` : vector of standard deviations.
- `log, log.p` : logical; if TRUE, probabilities p are given as log(p).
- `lower.tail` : logical; if TRUE (default), probabilities are `P[X \le x]` otherwise, `P[X > x]`.

The team

The AutovalidatoRs

Coline Zeballos

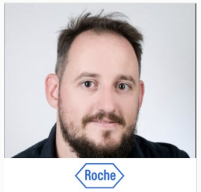
PDIX R Strategy Lead, Project Lead

Doug Kelkhoff

Development & Support Team

Szymon Maksymiuk

Development & Support Team



Lorenzo Braschi

Development & Support Team

Monika Washington

Validation Lead

Farzana Shaik

Technical Writer

Doing now what patients need next

Q&A

checked - https://github.com/maksymiuks/checked

rd2markdown - https://github.com/Genentech/rd2markdown

covtracer - https://github.com/Genentech/covtracer

Next talk by our team! →

Validation Hub

Regulatory Repositories

Coline Zeballos *Roche*

Yann Féat *mainanalytics*

AutovalidateR KPIs

2000+ validated packages in 18 months
new submissions & updates

5m end-to-end for “simple” packages
quick build time, ~~30 dependencies, no reverse dependencies

16hr for longest run
{testthat} upgrade with many reverse dependencies

~25 submissions initially rejected
mostly internal, and were later approved after improvements

4 rejections resulting in open source tests contributions
contributing tests to bring package up to quality expectations

4 support team open source contributions
contributing to open source code bases including issues to base R itself