5 years of internal software validation in Roche

useR! 2026 - Warsaw

Lorenzo Braschi
Szymon Maksymiuk

Introduction

Validation: What is it 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**

Origin Story

How it started

2015

R use in the industry is taking off

2017

PD deploys the “Biometrics Exploratory Environment” (BEE), a workbench for R.

2019

Analysts and statisticians begin using BEE “off-label” for clinical analyses

2020

Roche hires a vendor on yearly license for validation of R packages

2022

Launch of OCEAN, a unified platform for clinical data analyses

2023

End of license with the external vendor, validation *only* with internal team

2024

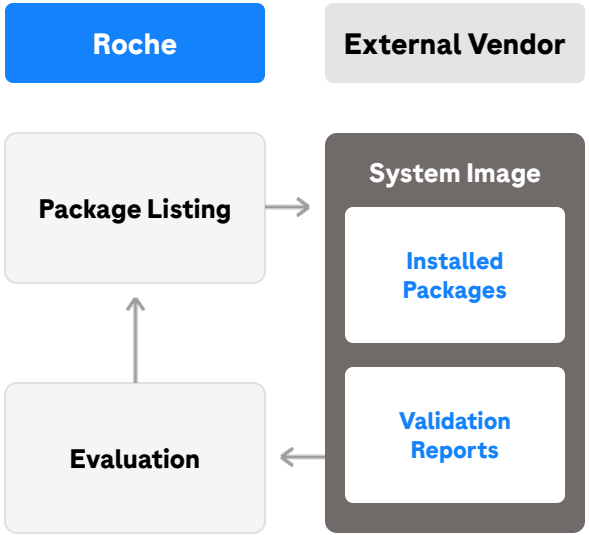
Roche’s First end-to-end R submission for a new drug application

2025

Migration of OCEAN complete

The Old Ways

Single piece validated environment



An external vendor provides Roche with a **validated system image** containing a **specific R version** and **validated packages** along with accompanying validation reports.

This image supports regulatory submission work and is **updated periodically** based on requested additions and version changes.

This approach worked for the most part but had key pain points:

01. **Lack of flexibility** to add new packages outside the development cycles.

02. **Slow feedback loop** for teams developing or merely requesting packages.

03. **Poor scalability** as adding new teams required baking hundreds of packages into the validated image.

A Better Way

Decoupling the image and the validated packages

01. The Validated Image

Work Environment

Contains only system libraries, a Posit Workbench, a single installation of R, and a few core libraries.

02. The Package Manager

Validated Package Repository

Contains a repository for each version of (validated) R, hosting a suite of independently validated packages.

Tailored & Flexible Fit

Dev and clinical teams can install just the specific packages they require for their analysis needs, avoiding bloated environments.

Agile Validation Cycles

Packages are validated independently of the system image. Validating on demand dramatically shortens cycle times and scales access across many more teams.

Guaranteed Reproducibility

The package manager supports snapshots, freezing the exact set of packages used for any specific regulatory analysis.

Solution

How did we understand business needs?

01. Outreach
We contacted teams doing regulatory submissions.

02. Collaboration
Worked closely with the validation leads looking for feedback.

03. Experience
Leveraged our research & package dev expertise.

EASY INSTALLS

"All my SAS macros are in a shared library, why is it so hard to get R packages?"

INTEGRATION TESTING

"My SAS code always works together. Will R packages break each other?"

RELIABLE UPDATES

"My SAS system has been consistent for years, why do I have to upgrade R packages so often?"

STRICT PKG CONTROL

"My SAS code always works the same... will my analysis change over time with constantly changing R packages?"

Design Priorities

01. Standards & Scope

Best Practices: Leverage industry-standard best practices for R package development.

Minimize Scope: Keep packages focused; those interfacing with peripheral systems should handle only the R language interface.

02. Automation & Gaps

Prefer Automation: Automate gating criteria to streamline package verification.

Human-in-the-Middle: Use structured human-in-the-middle intervention to reconcile gaps in automated checks.

03. Infrastructure

Reproducible Systems: Target ephemeral, reproducible environments with composable containers and package repositories.

Consistency: Avoid creating special cases for specific classes of packages to keep the pipeline clean.

04. Governance & Risk

Balanced Mitigation: Balance automated safety with manual escalation by end users and system administrators.

Transparent Review: Maintain an open, clear review process to encourage collaborative in-house development.

What is a package?

The Validation Principle

Validation state is not an inherent property of the package itself.

Instead, it is defined in advance as a strict, interdependent triple:

01. Package

The specific R package considered with its history, ownership and scientific scope

02. Version

The exact version of the package that has undergone formal testing.

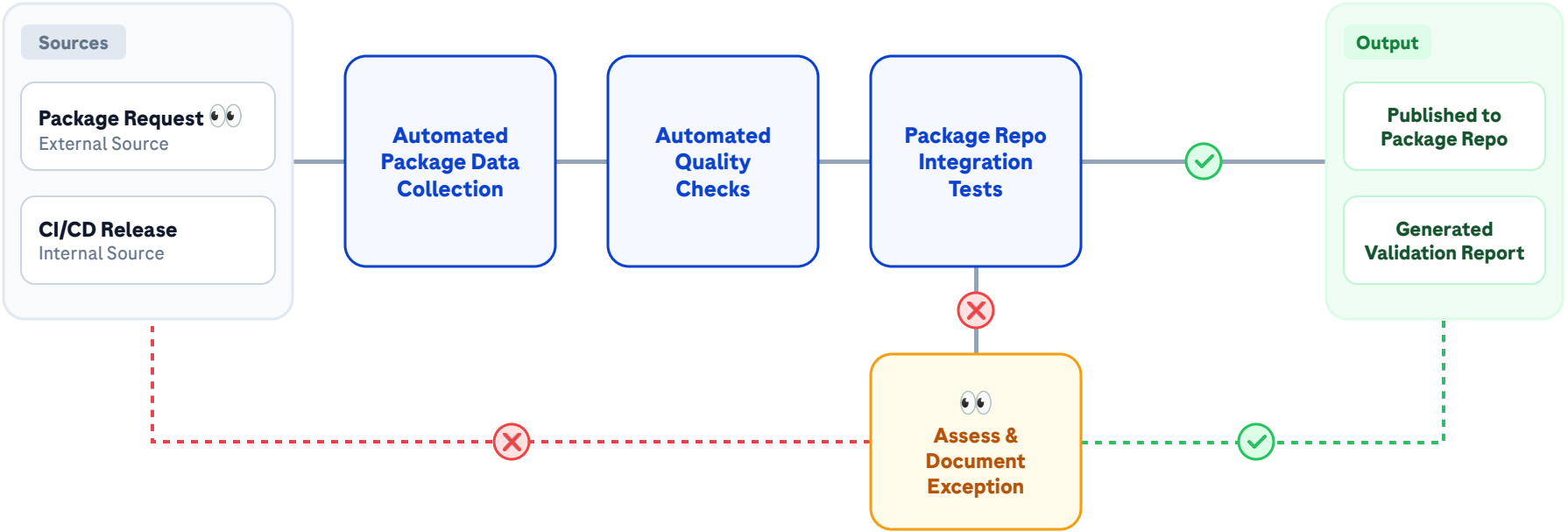
03. Environment

The execution environment, including operating system and package manager.

Strict Applicability: Packages are considered validated only when used in the precise version they were validated for, and within the specific target environment.

Scope of Environment: This validation boundary extends beyond the execution system itself to include the specific package manager associated with it.

Process overview



Graphical overview of the automated submission and gating process.
👀 **human-in-the-middle** components highlight manual review gating exception pathways.

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
Vulnerabilities	The package and system dependencies it declares does not have any major vuernelabilitis reported in the OSV
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 Both, total and R files only 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

Main challenges

Adoption & Compliance

The hardest part was not technical but organizational. It required significant effort to advertise, convince stakeholders, and demonstrate compliance with strict health authority requirements.

Continuous Evolution

The project constantly evolves. Over the past 2 years, nearly every aspect has been revisited, including:

- Changing isolation methods
- Adding vulnerability assessments
- Updating code coverage calculation and assessment
- Reworking reverse dependency checks and building completely new tool

Feedback Integration

We continuously gather and act on user feedback. This active loop drives modifications to requirements, final deliverables, and roles based on direct team suggestions.

Developer Pipelines

Continuous Integration

Allows running dry run validation as part of the package's CI/CD setup, catching issues early.

Automated Verification

Automates code validation to keep the codebase in a validation-ready state throughout development.

Enhanced Quality

Significantly increases quality across Roche, adopted even by teams without validation mandates.

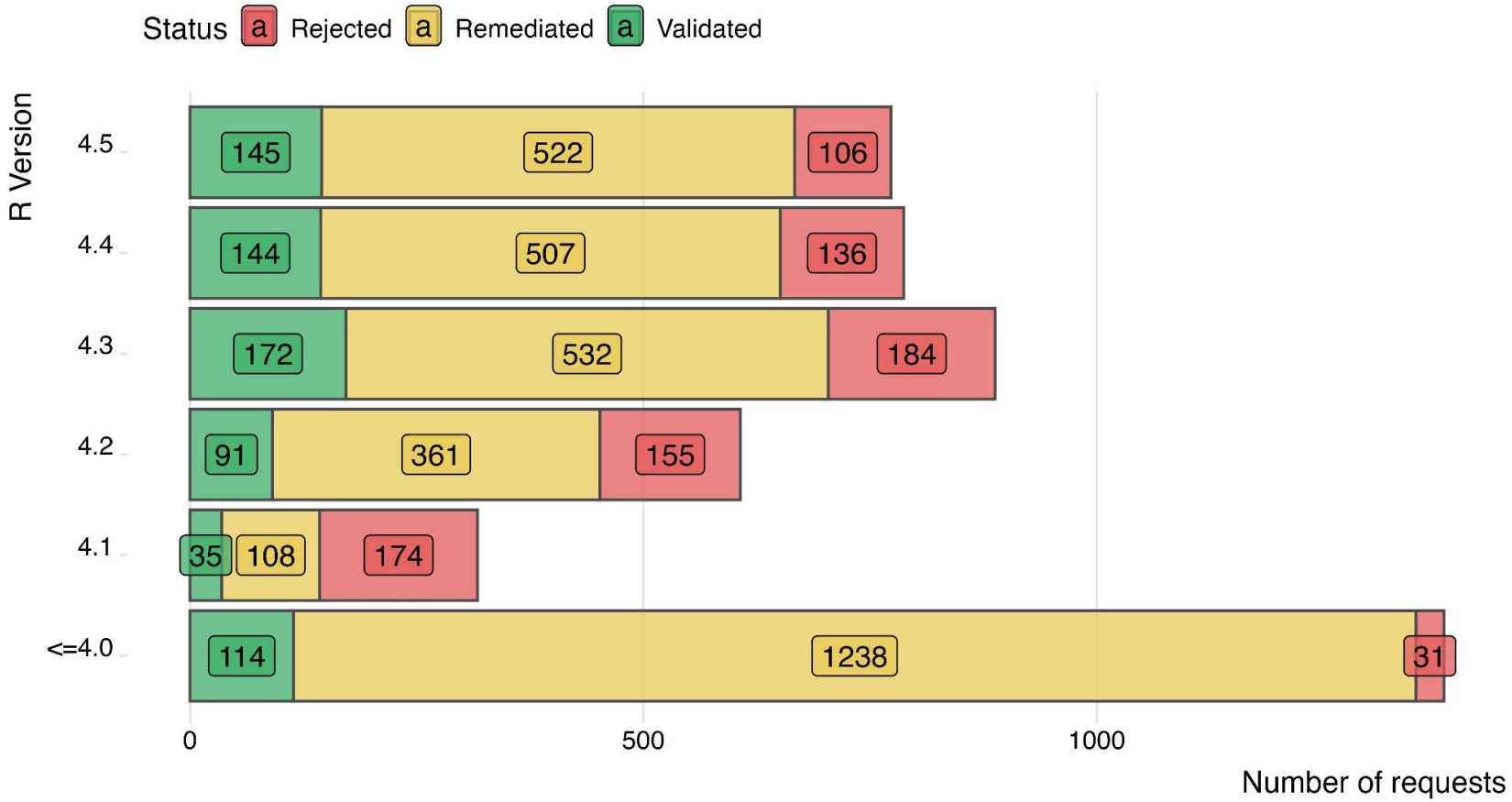
Accessible Compliance

Can be configured to automatically send actual validation requests for specific package versions.

Validation in numbers

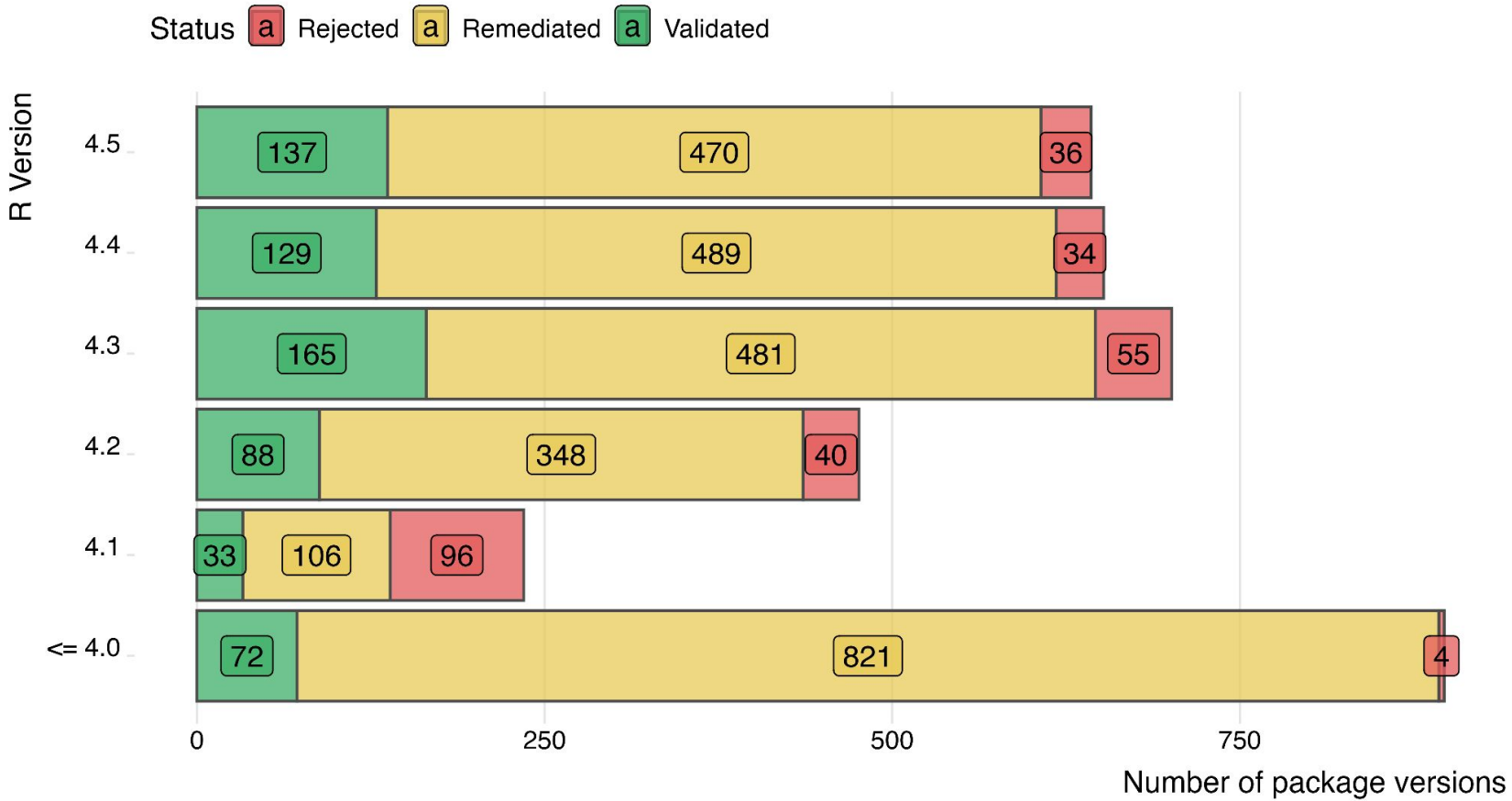
Number of requests per R version

Package requests since 2022



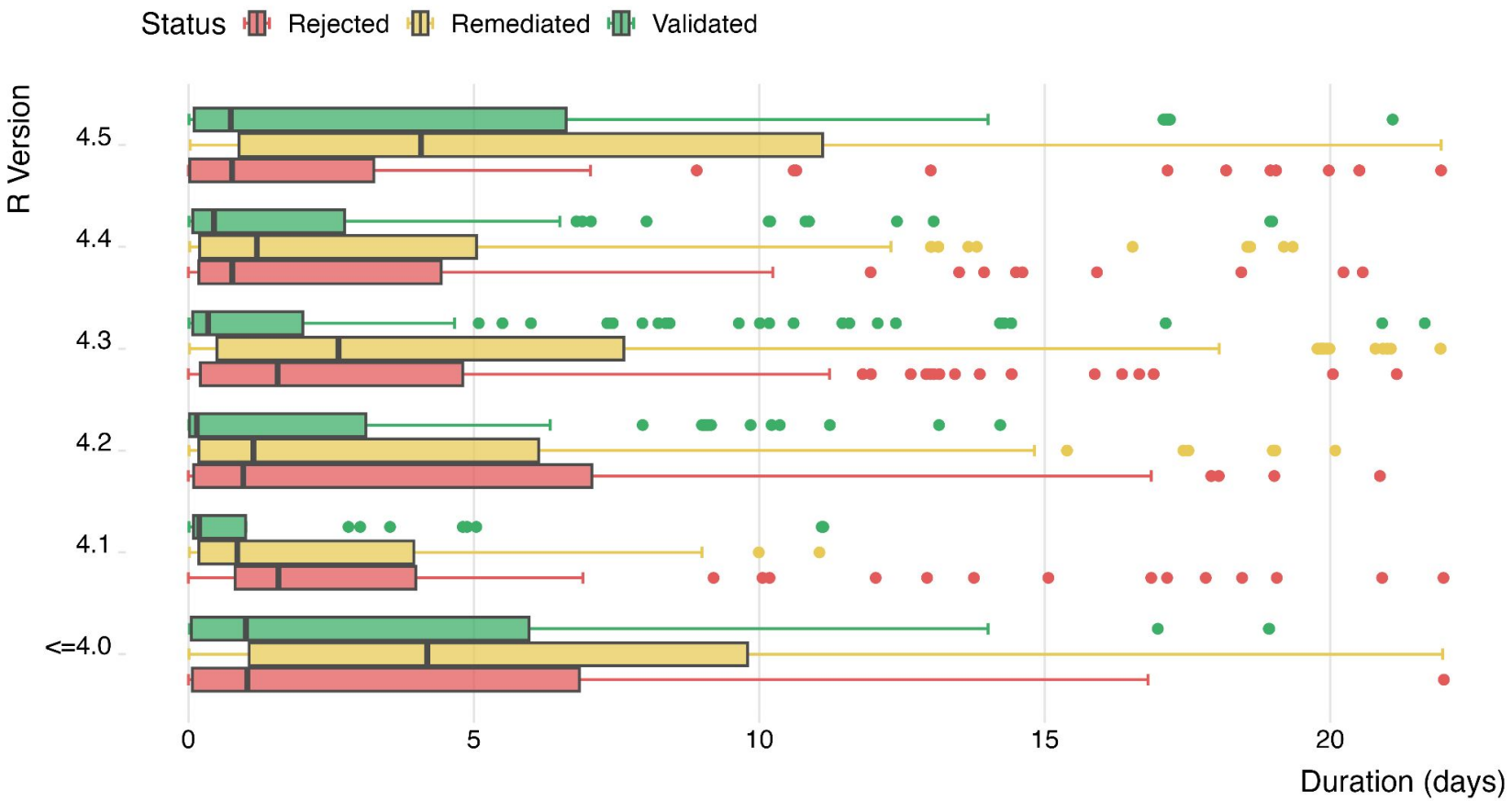
Number of package versions per R version

Specific package versions



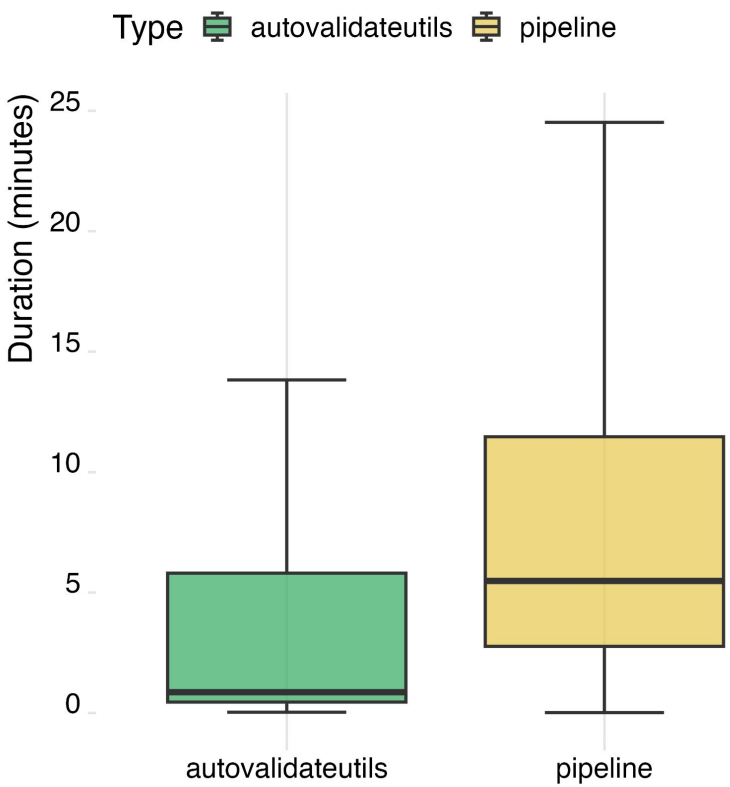
Duration of a request

How long does it take to close a request?



CI/CD Runtimes

Pipelines and main job durations



Outliers removed

Pipelines runtime statistics		
	autovalidateutils	pipeline
mean	16.928	20.945
median	0.863	5.500
q1	0.454	2.783
q3	5.804	11.533
min	0.031	0.017
max	2880.686	2885.700
sd	85.755	85.671

Validation in numbers

show me the data

5700+

automated requests processed in 4 years

new submissions & updates

1200+

packages validated in 4 months

“fastest” sprint

300

additional requests

legacy systems & one-off use cases

8m

end-to-end for “simple” packages

quick build times, no reverse dependencies

12hr

for longest run

{rlang} upgrade with many reverse dependencies

Doing now what patients need next