Paper OS03

R Package Validation Maintenance
We Didn't Know What We Didn't Know

Mark Sellors, Atorus Research Inc, Stroud, UK
Ashley Tarasiewicz, Atorus Research Inc, Philadelphia, USA

ABSTRACT
As R gains acceptance in clinical programming many companies have begun to tackle the task of R package validation. While there's a lot of information available for the initial validation of R packages, there's not a ton of information about the change management process required for long term maintenance. In this paper we'll explore the successes, challenges, and lessons learned of maintaining validated packages in the ever-evolving open-source landscape.

INTRODUCTION
Our intention with this paper is share some insights that we've learned since we created our OpenVal product in the hopes that it might help some of you and your organizations in your own validation journeys.

First up though, we need to level set quickly for those readers who aren't already familiar with the idea of validation itself.

WHAT IS R PACKAGE VALIDATION?
In many cases in the past, proprietary programming languages were used in clinical submissions and other related biostatistics areas. However, in recent years there has been increased interest in the adoption of open-source technologies for these tasks.

When using a proprietary language there is a single source of truth, the vendor, for the language's "correctness". With the increasing adoption of open-source tooling organizations are using multiple "products" (both the language and the add-on packages) from multiple sources (open-source developers) all potentially working in slightly different ways and with different standards. Open-source tooling, by its very nature, varies a lot in terms of the quality of code, the amount of testing performed by the authors and the level of rigor applied to the development process.

Of course, while there is a lot more variation in the open-source community, we can at least see the code, which allows us to inspect it for ourselves and make judgements on it's suitability for a particular task. This is not the case for proprietary code and all too often vendors must be taken at their word with precious little supporting evidence available.

Validation is about confidence. You need to be confident that the tools you use will work the same way again and again. When running on different computers, or several years from now, or in another research environment where results must be replicated.

We must be confident that the results will be accurate, and the processes used to obtain them repeatable. Science is rigorous and, when working on parts of that scientific research that just happen to be performed computationally -- like a clinical submission -- we must be too.

Programming can often feel like a solo pursuit. Just you, your programming knowledge and your wits against your data and the problem you're trying to solve, or insight you're trying to extract. It's a somewhat romantic and naïve vision though.

The reality is, even when programming alone, it's much more of a careful balancing act and a team activity whether you realize it or not. Instead of trusted team members whose work you're already familiar with, the other people in your team are strangers and you have no real idea of whether they're fit for the task at hand.

When you use any programming language or package – open source or commercial – you're inviting the developers of those tools into your work and trusting that they're up to the task.

The great thing about open source is that we can generally find out who those other programmers are, so we're not flying blind. We can see their work, make judgements about its efficacy and fitness for purpose and then decide how to proceed from there.

We validate to gain confidence in these shadow contributions to our work. We document that validation so that it can be shared with QA and as part of any audit, to prove that we did what we could to ensure these contributions met our expectations in terms of their methodological correctness.

INITIAL VALIDATION

So, what does our initial validation look like?

This is a challenge that has been discussed elsewhere at length, but in short it means setting up some sort of test framework and acceptance criteria for the packages you wish to validate, running them through that process and, assuming things pass, calling what comes out of the other end, "validated".

You must develop and adopt a reliable and repeatable means for accepting or rejecting a package into your validated set. This is generally done in conjunction with your organizations Quality team, to ensure that your validation approach meets the requirements of the wider business. It's about bringing some rigor to the process of choosing which packages to use.

In short, choose a methodology, define your validation criteria, test the packages meet the criteria, document. There are obviously a lot of nuances to this, but with very broad strokes, this is the process.

LONG-TERM MAINTENANCE

Once your initial validation is complete, you're into the long-term maintenance phase, which is where many validation projects fail.

Now that the march of the package validation has begun, it will never stop. The package updates will keep coming. R will get updated. Operating systems get updated. New packages are released. Old packages are archived. Users will want new and improved tools. sIt's a never-ending stream of change and we've found a lot of organisations unprepared for the ongoing work of maintaining a validated package set.

Organisations take on the challenge of providing a validated set of packages without realising that it's a continually evolving landscape that they must now continually adapt to and maintain. It's not a project you do for a few months and then it's over. It's a project that will run for as long as you're using open-source tools for GxP work.

Whilst you can hopefully use your validated packages indefinitely, life in the R world never stands still. You'll have to contend with updates and bug fixes, new packages, changes or maintainers and so on and these changes require a robust change management process to handle them appropriately and consistently within the context of your validated set.

You need to be able to respond to this changing environment in your next release.

At this point you've only just begun what will become an ongoing process with no end. The field of data science and the tools we use doesn't stand still and neither will you. Once you've completed your first validation, you must move your attention to the second.

However, the process of creating a second release can be quite different to the first.

CHANGE MANAGEMENT FOR VALIDATED R PACKAGES?

Now that your first release is complete, we turn our attention to all subsequence releases.

We need robust protocols in place to handle the full package life-cycle within your validated set.

How do we provide the sort of rigor we developed in the initial validation but applied to the ongoing maintenance of the set?

For example, for a package version update the risk level and testing need to be reevaluated to ensure our additional validation remains sufficient. There are ways to see the scope of this, like package documentation, naming convention of package to indicate major/minor/patch etc, tools like diffify and so on. However, good testing means reevaluating the package as a whole each time its updated to make sure *that version* of the package is tested appropriately.

A package might get removed from CRAN which makes it riskier depending on the reason it was removed.

Sometimes packages have a higher download score (more downloads within a year) and then they could potentially be less risky.

When R itself is updated you'll need to make sure the packages work with the version of R you're using and continually reevaluate as you adopt new R versions. Some packages rely on certain R versions and this has certainly broken our testing in the past.

If your organization adopts a new operating system, you'll need to check your validation will work in the same way on that. You might also find yourself supporting multiple-operating systems with your validation suite.

Table 1. Package updates/changes to OpenVal releases over time.

Release	Description	Released	Packages				Snapshot Date	R
			Total	New	Updated	Removed		
2023.09	Sept 2023 Major	2023-10-11	263	N/A	N/A	0	2023-07-31	4.2.1
2023.09.01	Dec 2023 Minor	2023-12-29	316	53	0	0	2023-07-31	4.2.1
2023.09.02	Jan 2024 Minor	2024-01-31	316	0	0	0	2023-07-31	4.2.1
2024.03	Mar 2024 Major	2024-03-29	355	39	117	0	2023-11-20	4.3.2
2024.03.00.01	Mar 2024 Patch	2024-05-06	355	0	0	0	2023-11-20	4.3.2
2024.03.01	June 2024 Minor	2024-07-01	391	36	1	0	2023-11-20	4.3.2
2024.09.00	Sept 2024 Major	2024-10-02	407	17	217	1	2024-06-22	4.4.1
2024.09.01	Dec 2024 Minor	Under active development					2024-06-22	4.4.1

The above table details the level of change that has occurred within the OpenVal validated R project at Atorus Research since the project's inception. As you can see, many new packages have been added, others updated and most recently, one has been removed all over the course of a single year.

You'll also need to consider how your end users might be impacted by changes to your validated set. For example, there exists the potential to introduce a regression to company specific displays, so care must be taken that the validation effort does not have a surprise impact to the business and tasks that should be reproducible are not.

How should you communicate changes and updates to the wider business? What level do programmers and statisticians need to know about function specific changes and what should they investigate on their own? For OpenVal, we provide the information on which packages are new, had version changes, removed etc. as part of the release notes.

How often will you update? Opensource evolves quickly, updating less than once a year might mean there's way better versions of packages out there. Updating too frequently may be hard for project teams. In the case of OpenVal we update the snapshot twice a year with the idea that customers can ignore versions if that's too often for them.

CONCLUSIONS:

Maintenance of package validation is a huge time commitment. Those attempting to put such a process in place need to fully understand the magnitude of the task they're taking on, but with good planning and sufficient resourcing, it is possible to be successful.

The backdrop to this is a shared programmatic and human one. You can write programs to evaluate the risk, and use tools to compare package release notes, even package code, but the most important element is the human element -- someone who opens the package code, looks at the code and the unit tests and decides if the changed functionality of the package needs a different testing approach for validation. More tests, different tests, etc.

Programmatic approaches can tell you the scope, but humans are the ones who need to verify the scope and do the work to meet that scope.

You'll also need to understand the importance of layers of validation. For example, you must be set up to make sure you're not relying on one individual to make these testing decisions in isolation. One person shouldn't solely decide if what changed in a package requires new testing because validation isn't as clear cut as that. Consider using at least one tester, and at least one independent reviewer, or perhaps a senior reviewer.

Say a release includes a total of 50 packages. One approach could be that testers and reviewers see a subset of the packages, because they are given assignments of, for example, testing 10, reviewing 10 (that they didn't test). Then a senior reviewer takes a high-level look at all 50 so they can see if similar decisions were made across the board. This adds lots of opinions (which are good for validation) and the senior reviewer adds cohesion to something that could otherwise be highly variable.

Only with a proper and full understanding of the challenges of producing and *maintaining* a validated package set can an organization be successful in the long-term.

RECOMMENDED READING

R Validation Hub: https://www.pharmar.org
OpenVal product information: https://www.atorusresearch.com/openval/
Diffify: https://diffify.com

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Mark Sellors
Company: Atorus Research
Email: mark.sellors@atorusresearch.com
Website: https://atorusresearch.com