Paper OS3652

Posit Team: Metadata Driven Risk Assessment & Surfacing Validated R Packages

Aaron Jacobs, Posit PBC, Toronto, Canada
Michael Mayer, Posit PBC, Basel, Switzerland

ABSTRACT
Many companies now make increasing use of open-source languages like R and the Posit Team software suite for late-stage clinical development. The Computerized Systems where these late-stage workloads run—such as Statistical Compute Environments (SCE)—are governed by GxP principles that put very strict controls on their use, and as a consequence there is now heightened scrutiny on the software testing and validation of R and R packages.

In this paper we outline a technical approach to conducting risk-based validation of infrastructure that includes R and a library of R packages using some of the unique features of Posit Team.

INTRODUCTION
Open-source languages—R and its packages in particular—have become a very popular alternative to proprietary software systems for late-stage clinical development at pharmaceutical companies. But because there is by definition no "vendor" to formally attest the quality of these open-source packages, companies typically require some level of additional scrutiny for them above and beyond the limited (and inconsistent) quality criteria of the existing public repositories (i.e. CRAN, Bioconductor, and r-universe).

Moreover, the number of available R packages from public repositories is very large (20k+ on CRAN, 3k+ on Bioconductor, etc.), and while no company can expect to use all of them, it is not uncommon to be assessing 500-1500 public R packages, plus internally-developed ones.

At this scale, detailed testing of each selected R package is practically impossible. Instead, companies typically adopt **risk-based validation**, where the risk of each package is assessed beforehand using various metrics and the result is used to inform the testing strategy.

Many large pharmaceutical companies today (including Novartis, Merck, Roche, and Merck/Serono) have invested significant resources to build out internal processes that implement risk-based validation of R and R packages. Atorus Research have also released a commercial product called OpenVal to help companies validate packages, and previously Mango Solutions (now part of Ascent) had a similar but now-discontinued product called ValidR.

Lastly, there have been various cross-company efforts to create open-source tools for package validation. Prominent examples include the riskmetric package and R Validation Hub. Unfortunately, while the principles of risk-based validation are very similar across the industry, the details matter, and so far no open-source effort has succeeded in providing a turn-key solution.

SUMMARY OF A RISK-BASED VALIDATION PROCESS
A risk-based validation approach typically plays out as follows:

1. Create a **validation strategy** that describes a reproducible process for installing and testing R and R packages**.** This document also includes a description of how risk is assessed how the assessed risk is used to inform testing, and also documents the testing approach. Last but not least it also highlights any risks that are not covered by this testing approach. (From a compliance perspective, it is better to be aware of and document a risk you are accepting rather than omit it.) The specific metrics used to assess risk, how to weigh them, and how exactly risk affects the required testing is typically the most controversial and subjective part of the validation strategy and consequently the most likely to differ between companies. Because we want to be agnostic, the remainder of this paper simply treats the chosen risk metrics as given.
2. Select a version of R and a library of R packages**.** Importantly, these package are chosen based on end user requirements, not on the basis of their perceived risk (which mostly informs subsequent testing). Only very rarely are packages deemed too risky for inclusion.
3. Devise a **validation plan** that operationalizes the validation strategy for this specific version of R and library of R packages, describing an end-to-end process for how R and each package should be installed and tested. Installation and testing is typically done via so-called Installation Qualification (IQ), Operational Qualification (OQ) and Performance Qualification (PQ) scripts.
4. Execute the validation plan and collate the results (including any deviations) in a **validation report**.

The steps above omit many details, of course (like updating the Configuration Master File or Trace Matrix, the approval process, etc.) in order to keep the focus on the technical aspects of the validation/testing process.

Note also that steps 2-4 can happen more iteratively in practice as additional packages are added to the library, which may entail revisiting the validation plan and regenerating the validation report.

POSIT TEAM FOR RISK-BASED VALIDATION

Posit Team is the software suite of Posit Workbench, Posit Connect and Posit Package Manager. All of them can be used independently, but as shown in Figure 1, they also complement one another when used in concert. As expected for enterprise software, they can be configured to run in a wide variety of on-premises and cloud-based environments, and are in use in many SCEs already.

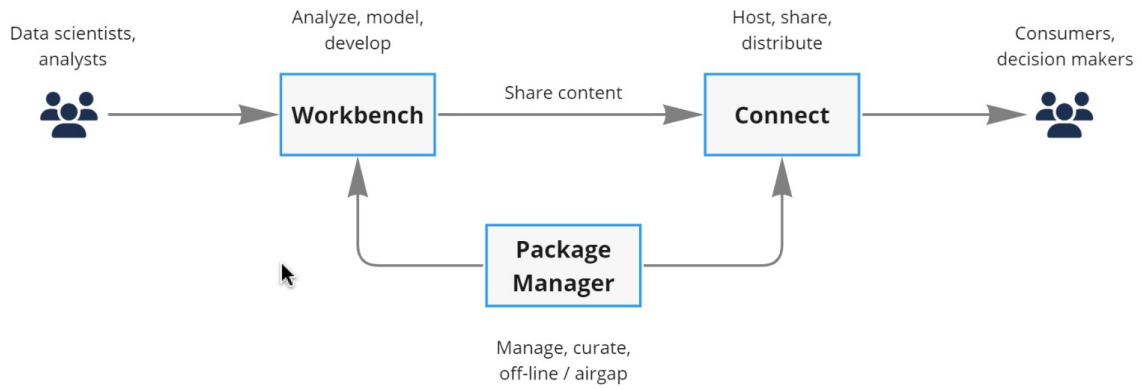

Figure 1: Posit Team Overview

To help improve the story for R in late-stage clinical development, Posit Team has grown or is growing a number of major new product features. In the worked example below, the focus is on one of these features—custom metadata support in Posit Package Manager—but we also make use of longstanding strengths of Posit Connect and Workbench in the process.

Custom metadata in Package Manager is hierarchical and can be attached on a per-repository, per-package, or even per-package version basis. This maps particularly well to the high-level validation process outlined in the previous section, because we can identify attributes that fit this hierarchical model well: validation strategies, plans, and reports are repository-level attributes, many attractive risk metrics (author reputation, software development lifecycle indicators, etc.) are package-level attributes, and the output of individual IQ/OQ/PQ runs are by definition scoped to a specific package version. Package Manager also natively understands metadata that contains a link, making it well-suited to serve as a portal to internal documentation (see Figure 2 for an example).

WORKED EXAMPLE

What follows is a worked example of implementing risk-based validation of R packages for GxP work using open-source tools and Posit Team. Our overall goal is to move from a validation strategy, validation plan, and set of desired R packages to a self-contained “golden image” that can be used for GxP work. A couple of elements are worth highlighting from the outset:

- Although we hope this can serve as a template for others, we’ve made some simplifying assumptions and do not expect our code to be adopted wholesale in a real-world implementation.
- We store all of the metadata created or consumed by our pipeline in Posit Package Manager so that it can be consumed by automated tools and made more accessible to stakeholders via the UI.
- We make use of Posit Connect to host the canonical validation strategy, validation plan, and validation report, as well as hundreds of intermediate documents for IQ/OQ/PQ test output that are deployed there automatically during the validation pipeline. In principle, any document management system could be employed here, but Connect’s first-class support for Rmarkdown and Quarto make it very appealing for this workflow, and we rely heavily on its API-driven publishing support.
- We rely on the open-source R package pak to make bulk package installation faster and more reproducible. pak significantly improves the parallelism of package installs, detects and installs missing system dependencies automatically, and supports generating build logs we can use for IQ and lock files that we can consume in a validation report. It is the workhorse to our approach for generating golden images.

- We use the open-source riskmetric package as an illustrative example of how automated risk assessment for packages *might* work at a company.

All of the associated code is available on GitHub at https://github.com/sol-eng/package-testing.

Our pipeline works as follows:

1. We use pak to compute a complete set of package and system dependencies from a list of "approved" packages or from a snapshot date.
2. We build a Docker image from this list that installs all packages to a site library (again using pak). This is the golden image.
   3.
4. We consume this image and for each R package installed in step 2,
   - We install any additional dependencies needed to run R CMD check on the package (i.e. items from Suggests) temporarily.
   - We run R CMD check and capture the output.
   - We upload the package binary to Package Manager (which supports custom package binaries[1]) for archival purposes.
     -
   - W calculate a risk score using the riskmetric package.
   - We generate and render a Quarto document with the package description, risk score, and IQ/OQ/PQ[2] results and upload it to Posit Connect.
   - We add the risk score and a link to this Quarto document as package metadata on Package Manager.
5. We generate and render a validation report that summarizes the pak lock file and the test results for each package and uploaded it to Posit Connect.
6. We then add or update the link to this report as repository metadata on Package Manager, along with links to the validation strategy and validation plan.

Once the pipeline has completed, any package that has been tested/validated will surface metadata in the Package Manager instance. An example of that is show in Figure 2.

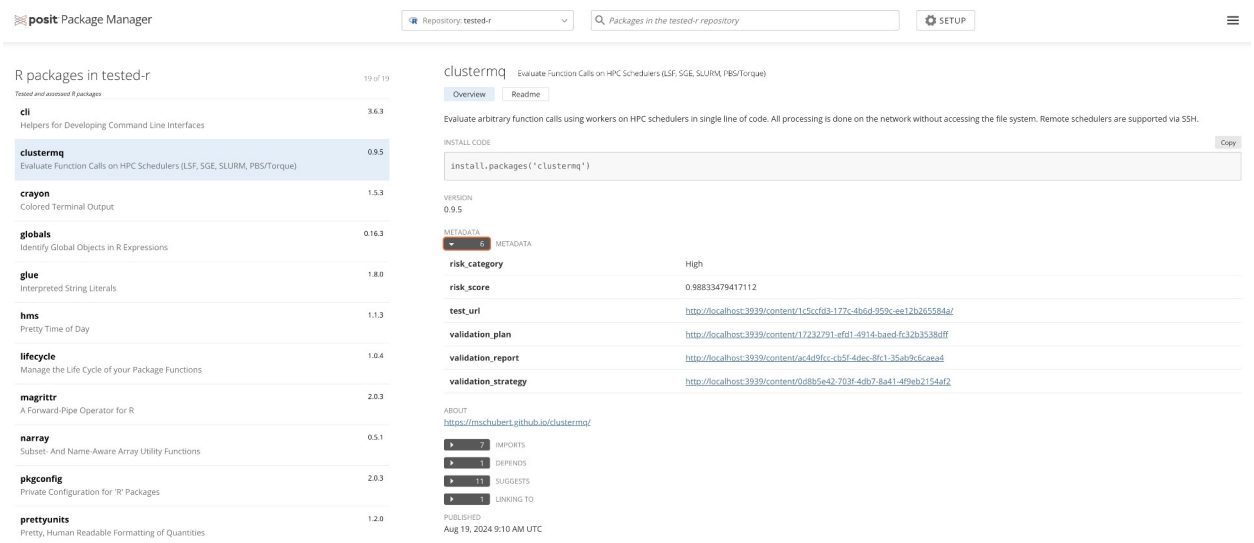

Figure 2: Example for Metadata in Package Manager UI

FURTHER REFINEMENTS

A more robust and realistic iteration on the pipeline above could include:

- Generate a more detailed analysis and summary of IQ/OQ/PQ outputs for each package.

[1] https://docs.posit.co/rspm/admin/appendix/configuration/#Binaries.Distributions

[2] For simplicity reasons we use the build logs from pak as IQ, the output of library(packageName)as OQ and the R CMD check output as PQ.

- Instead of just relying on R CMD check for PQ testing, employ a testing strategy that actually utilizes the risk score.
- Build Singularity/Apptainer images instead of (or in addition to) Docker images.
- Use the generated binaries on future runs to make rebuilding images (e.g. with security fixes to the base operating system) faster and more deterministic.
- Rather than starting from a static list of packages, query Package Manager's API to find all packages with certain metadata—perhaps uploaded from external sources via other automated systems.
- Automatically deploy the resulting images to the SCE, perhaps by making them visible to a Posit Workbench instance.

Figure 3: Schematic overview on a typical repository on package manager. Areas where metadata can be applied are marked with a "M".

ADDITIONAL NOTES

The process described above gives all the deployed R packages the stamp of being ready to use for GxP work almost in a blank check manner. But strictly speaking, in a GxP/CSV world, only certain well-defined use cases (driven by user requirements) are allowed to run on such a system. It is an open question whether the generic testing approach we and others employ (whether relying on `R CMD check` or doing more with risk metrics does not matter) really minimizes risk significantly. Instead, a set of targeted PQ scripts with typical data and algorithms can likely produce more meaningful results. Those PQ scripts ought to be linked back to user requirements via the so-called trace matrix, making the R and R package validation much more integrated into the overall CSV process. But `R CMD check` in this context does not hurt and definitely helps suss out some basic issues (for example, historically, `R CMD check` would fail if the BLAS/LAPACK integration would not produce reproducible results[3]).

Some observers also argue that testing should happen on a per-function basis, and only functions should be called validated. Though in this case one would need a technique to hide untested functions from end users of the package.

Lastly, some folks may not warm to the idea of validating/testing packages in this way at all. They simply provide a qualified infrastructure where they document the actual installation of R and the R packages in a fully traceable and reproducible manner but leave it up to the consuming Statistical Programmer or Statistician to not only document the

[3] For example in the case of Intel MKL that used highly optimised BLAS/LAPACK routines and different code branches for each supported CPU chipset generation can lead to numerically different results when run on different hardware. The introduction of MKL_CBWR however mitigates this risk.

R packages and versions used by their analysis but also provide reasoning why they think that their chosen packages provide reasonable results. Such an approach considers R packages part of the code, and thus discussed and tested and documented together with other components of a Statistical Analysis Report.

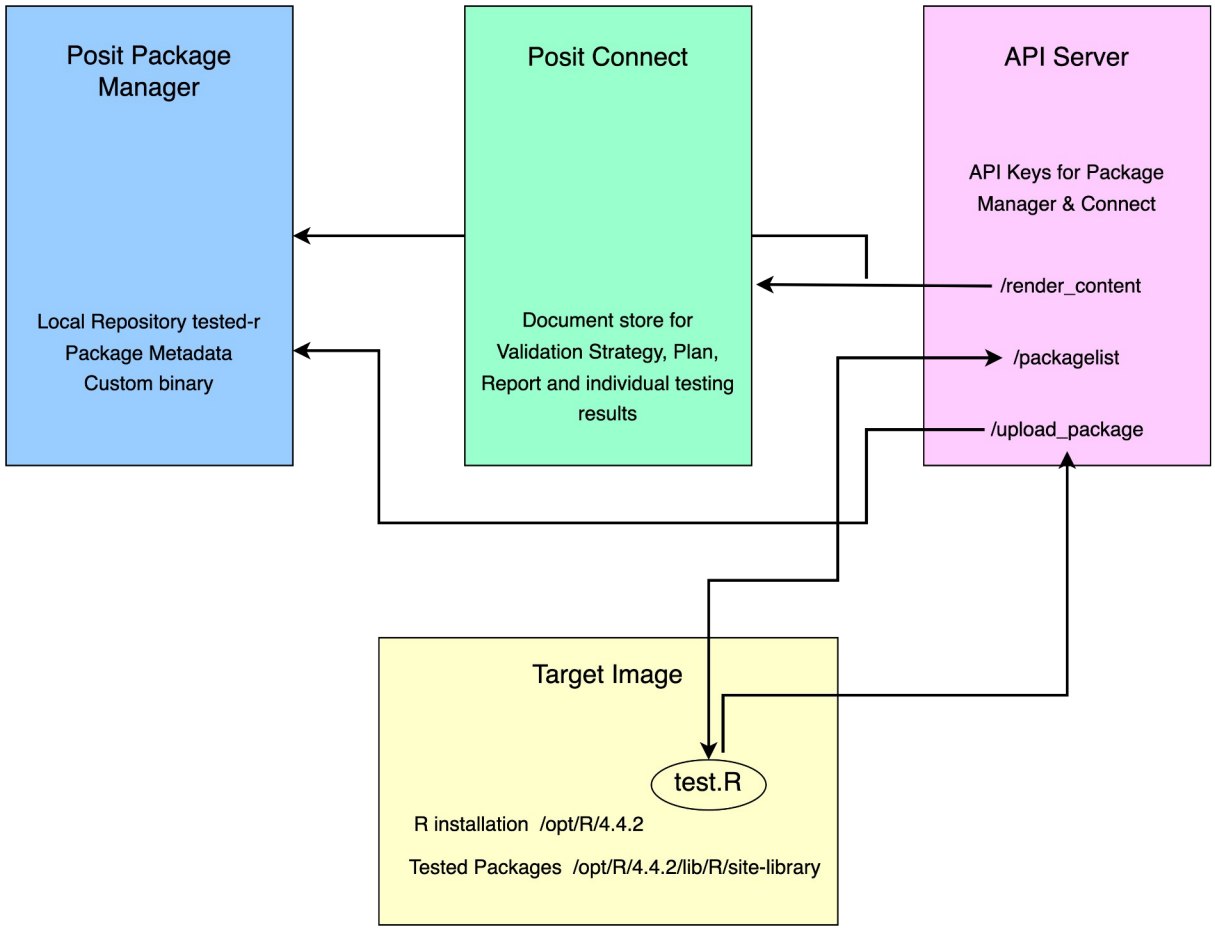

Figure 4: High-Level Architecture/Process diagram of Testing infrastructure

CONCLUSION

In this paper we discuss an approach to risk-based validation of R and R packages using the Posit Team software suite, including a concrete pipeline that implements it. We highlight the newly-released support for custom metadata in Posit Package Manager in particular. Finally, we discuss potential improvements to this pipeline and highlight various alternatives, both procedural and to the general approach.

CONTACT INFORMATION

Author Name: Michael Mayer
Company: Posit PBC
Email: michael.mayer@posit.co
Website: https://posit.co