PharmaSUG 2023 - Paper SI-212

R Package Qualification: Automation and Documentation in a Regulated Environment

Paul Bernecki, MSD, Zurich, Switzerland
Nicole Jones, Merck & Co., Inc., Rahway, NJ, USA
Uday Preetham Palukuru, Merck & Co., Inc., Rahway, NJ, USA
Abhilash Vasu Chimbirithy, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

In the recent years, there is an increasing trend in using open-source software, such as R, in clinical trial analysis and reporting (A&R). The qualification of an R package is a critical step in ensuring its quality and its compliance within the highly regulated clinical trial environment. Given the sheer number of R packages, automating qualification process is necessary to ensure consistent quality and increased efficiency.

Aspects of our company's R package qualification process were presented in PharmaSUG 2022. The current paper focuses on some updates made since then. Firstly, an internal R package mkqualify streamlines the qualification process further while incorporating inputs from human reviewers. This package generates most of the documentation used to support the qualification. Secondly, the open source riskmetric package generates the risk scores associated with an R package.

Communicating the qualification process also plays an important part in ensuring user compliance and in increasing user trust in the qualification process. We discuss internally developed Shiny applications, which include real time results of qualification, and automation of user qualification requests. Additionally, we discuss various processes on how to communicate with users about R package qualification including training and blog posts. The elements of this qualification process are continuously evolving to ensure adherence to best practices followed industry wide.

INTRODUCTION

The use of compliant and qualified software is a requirement when operating in a highly regulated environment such as clinical trial development. Open-source software such as R pose a unique challenge to organizations in ensuring compliance. The recent growth in the adoption of R for clinical trial analysis and reporting (A&R) makes it a necessity to ensure the open-source software is qualified. The qualification process ensures that the R packages used are accurate, traceable, and reproducible.

Our organization's efforts towards the R package qualification process were presented recently in PharmaSUG 2022 [1]. The end-to-end process presented in the paper offers guidelines on qualifying and installing internal and external R packages in a regulated R environment. We adopted a risk-based approach to qualify open-source software based on published sources of R package qualification processes including the R Validation Hub white paper [2,3]. Our strategy required clear definition of validation based on R-FDA document [4] and setting the goal of creating documentation that describes the qualification details of R packages based on predefined criteria. The risk-based strategy qualifies both internal and external packages based on the type of A&R deliverables generated. Table 1 provides the common types of A&R deliverables and the associated risk levels (Open, Moderate, Low) used within our organization.

Type of Deliverables	Examples	R Package Risk Expectation
Electronic Common Technical Document (eCTD) shared externally	Clinical Study Report (CSR) Submission package Drug labeling Agency request	Low
Non-eCTD	Data monitoring committee	Moderate or Low

Internal (Outside Department)	Manuscript and publication (using clinical data) Internal committee review or presentation	
Exploration analysis/Within Department	Data exploration Data quality checks Exploratory analysis	Open, Moderate, or Low

Table 1. Examples of A&R deliverables and respective risk categories of the R packages.

The internally developed R packages follow established Standard Operating Procedures (SOP) based on a properly defined software development lifecycle (SDLC).
To qualify external R packages into each risk category, we established 5 criteria:

- c1: Package is developed and maintained by a trusted vendor or organization.
- c2: Package is user-facing with sufficient SDLC evidence equivalent to internal SDLC requirement.
- c3: Package is not user-facing and all packages dependent on this R package are qualified.
- c4: Package is user-facing with additional internal work to complete necessary steps following internal SDLC requirements.
- c5: Package maintained by a trusted person or organization.

A low risk R package needs to meet at least one of the first 4 criteria(c1-c4) and one criterion is sufficient to qualify. A moderate risk R package can be qualified by any of the five criteria (c1-c5). An open-risk R package is any R package available on CRAN, Bioconductor or other repositories not qualified under either low risk or moderate risk. To establish trusted vendors/organizations for c1 criterion, the SDLC documentation provided by the vendors/organization is carefully reviewed and periodic audits of the SDLC documentation are undertaken by our organization. The term “user-facing” used in c2, c3 and c4 criteria indicates that a user interacts directly with the package during an R session. The non-user-facing R package is defined as a package that runs in the background as sub-routines during an R session.

To manage a reproducible R environment our organization also adopted the shared baseline strategy [5]. This shared baseline strategy allows users to easily share and re-run work in an environment and allows users to access the same set of installed packages using a common library. The centralized library used within our organization is termed the global R library – refer to our previous paper [1] for more details. The global R library has a nested structure as shown in Figure 1. The low risk packages are also part of moderate risk and open risk directories, similarly the moderate risk packages are part of the open risk libraries.

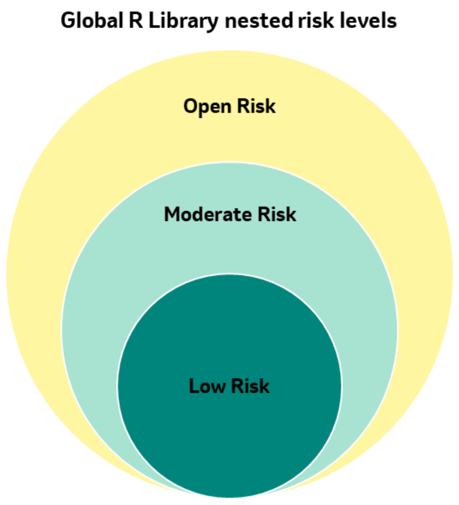


Figure 1. Global R Library nested directories for low, moderate, and open risk levels.

The generation and maintenance of documentation is another critical step ensuring a complaint and regulated clinical trial data analysis environment. The qualification strategy employed at our organization also emphasizes the maintenance of qualification documents to support both internal audits and inspections by regulatory agencies. The summary and individual package level documentation for low risk and moderate risk R packages qualified for a particular global R library update are generated and maintained on our computing platform. The summary level document lists all packages in the respective risk category along with metadata information such as R version, source repository, etc. The individual package level document contains details regarding the package qualification including package information, dependencies, code coverage, qualification details such as criteria fulfilled and details of qualification criteria.

In this paper, we focus on updates made to the R package qualification process employed in our organization since our previous publication in May 2022. An internal R package mkqualify was developed to streamline the qualification process by automating parts of testing and document generation, while incorporating inputs from human reviewers. The open source riskmetric package was also integrated into the qualification process to generate risk scores associated with an R package. We provide an overview of internally developed R Shiny based applications that are used to communicate real-time results of qualification as well as automate user qualification requests. We also discuss complementary processes that employ blog posts and training to better communicate to users about the R package qualification.

AUTOMATION OF QUALIFICATION PROCESS

In order to automate some of the steps in the qualification process, an internally developed R package mkqualify is used. The main goal of this package is to provide tools for evaluating R packages and generate documentation to support package qualification. The current functions in mkqualify provide a variety of tools to assess package quality and validate the package output. These functions can be split into two categories:

- Functions to create libraries: These functions help the user create various risk libraries as well as check the installation status of the libraries
- Functions to generate qualification documents: These functions help the user in generating qualification documents for each package by providing details about package including description, dependencies, qualification criteria and code coverage

The current workflow for the package qualification is as follows:

1. Create initial request form (packages and their versions to install) and define risk level for each of them. If package is low/moderate risk, then user needs to enter pre-specified criteria(c1-c5), that will be used for qualifying. Depending on use case, package might meet several criteria.
2. Commence dry-run installation:
    a. Each installation is done on a test server, before deploying it to production. This ensures that all low or moderate packages are installed and working without issues. To accomplish this, we use built-in R CMD functionality, that allows installation in a batch and at the same time creates one installation log. Depending on source of packages, log’s length can vary from 130,000 lines (if installing pre-compiled binaries) to over 230,000 lines (in case of compiling source code).
    b. After installation, logchecker() function is run, that scrapes the log for error messages. For easier viewing and storage, enhanced log report is saved along with original log, as an R data (.rda) file.
3. Run vignettes files for generating qualification documents. Each package listed in the initial request form will have corresponding documentation created. Elements of a qualification document include risk categories, package code coverage, scraped NEWS/NEWS.md file, and risk metric scores generated by the riskmetric package.
4. Initial qualification review team classifies a package as statistical/non-statistical in nature. If package is identified as statistical in nature, additional review and documentation is required.
5. After review has been completed, final request form file is created, which is used to request formal installation.

Once the dry-run installation is complete and qualification documents are created, documentation for each package is reviewed. The initial request form is revised for a variety of reasons including:

- Installation errors
- Package incompatibility with R version.
- R package not meeting the qualification criteria (For example: risk is too high, provided documentation is insufficient)

After this step, the final request form is generated for submission to IT team. Compared to previous version, changes were made to reflect the possibility of updating the request form after generating the qualification documents. After SME review a package might not meet the risk level requirements, resulting in an update to the documented risk level. The updated qualification workflow is shown in Figure 2.

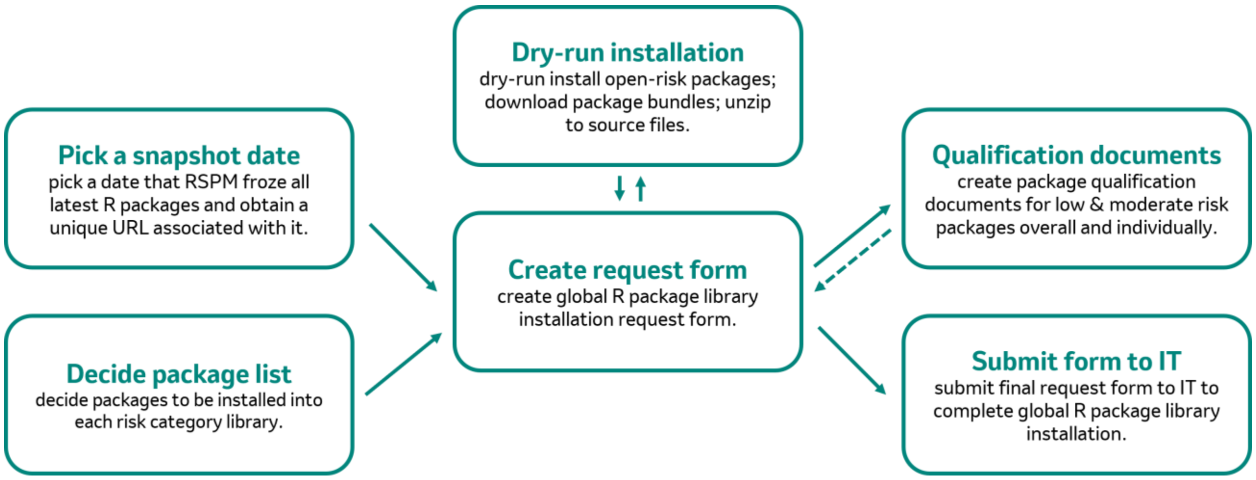

Dry-run installation
dry-run install open-risk packages; download package bundles; unzip to source files.
Pick a snapshot date
pick a date that RSPM froze all latest R packages and obtain a unique URL associated with it.
Qualification documents
create package qualification documents for low & moderate risk packages overall and individually.
Create request form
create global R package library installation request form.
Decide package list
decide packages to be installed into each risk category library.
Submit form to IT
submit final request form to IT to complete global R package library installation.

Figure 2. Updated R package qualification workflow.

In addition to the information already mentioned in previous paper [1], individual package risk scores calculated using riskmetric package were integrated into the workflow. riskmetric is an open-source package, that uses various metadata information of a package, and assesses final "risk score" of a package. The final riskmetric score might vary between 0 and 1 with lower score indicating lower risk. Currently, the qualification review team takes the riskmetric score into consideration when qualifying a package. However, it should be noted that the score represents a collection of experimental metrics and individual organization could have additional workflows for determining the risk of an R package.

The code coverage of a package can be used as supplementary information when assessing the quality. It indicates the percentage of code used in the package is being covered by tests. The interpretation of code coverage is inverse to that of riskmetric score, higher values indicate higher code coverage. All unit tests are executed for packages included in the global library to ensure the package behaves as expected in our internal environment. The source files of each package are searched to identify if test cases were exported. If a test folder is present, the tests are executed using the below line of code:

```
coverage <- try(covr::package_coverage(source[i]))
```

Wrapping the covr::package_coverage() function call in the function try() prevents failing tests from halting the execution of downstream tests. If an error is encountered during the execution of these test cases, the result is reported and the error message is captured in the exported log. Otherwise, if all unit tests pass, the total package code coverage is reported. For user-facing packages that have no tests or failing tests, test cases will be developed in-house to validate the packages for intended use. The below code is used to generate these outputs:

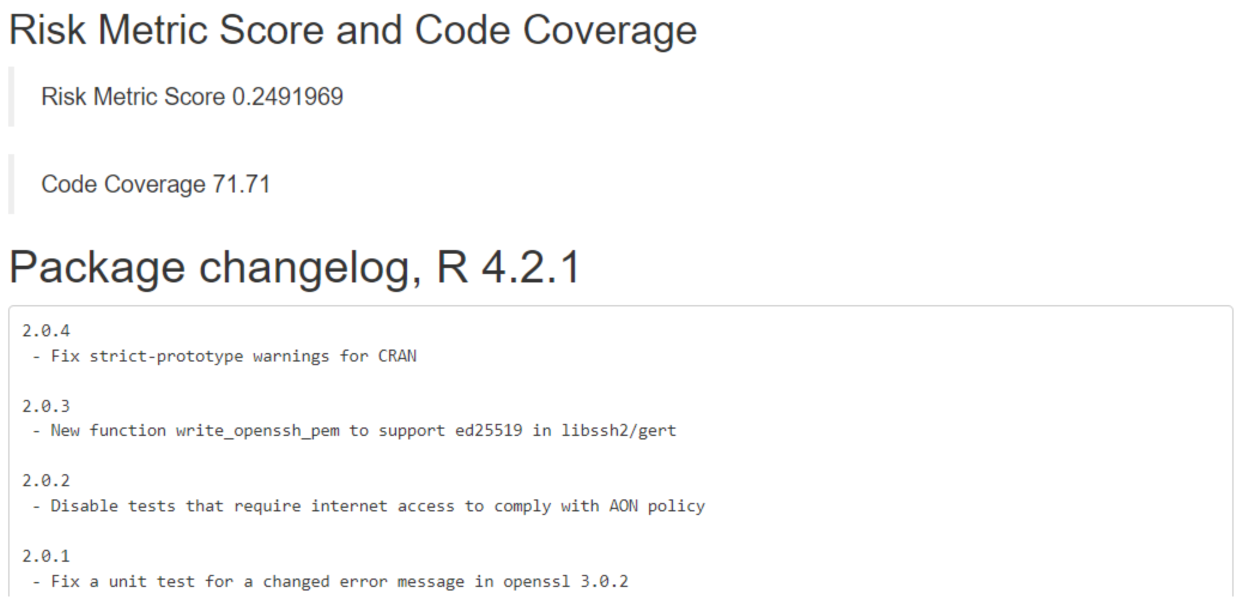
```
   if(class(coverage) == "try-error") {
     pkgcoverage[i, 1] <- "Test Failure"
     pkgcoverage[i, 2] <- gsub(paste0(path$source, "/"), "", source[i])
     next
   } else {
     covge <- coverage_to_list(coverage)
     pkgcoverage[i, 1] <- covge$totalcoverage
     pkgcoverage[i, 2] <- gsub(paste0(path$source, "/"), "", source[i])
   }
```

For qualification purposes, the higher the code coverage, the lower the risk level of the package. However, practice has shown, that relying strictly on test/code coverage might lead to incorrect interpretation of quality. There is always a possibility that the code might be impossible to test in its entirety. Another thing worth mentioning is that code coverage does not test the algorithm or logic being implemented within a source code.

Due to the aforementioned factors, the qualification review team should always be prepared to write additional tests for certain methods/functionality that are implemented within a package to ensure accuracy. This enables validation of the package internally and upskilling the team at the same time. Another element used by reviewers is reviewing the package changelog. The changelog is usually found within the installed package directory in NEWS.md file. In some cases, the NEWS.md file might not be found, this would require manual navigation to package source repository (CRAN or GitHub) and check for the updates. We include the changelog in report, only if one of the following criteria is met:

- Package has not yet been qualified
- Package has an updated risk level
- Package has been updated (compared to previous snapshot)

A document with implemented risk metric score, code coverage and scraped changelog files can be summarized and presented as a vignette HTML output as seen in Figure 3.



Risk Metric Score and Code Coverage

Risk Metric Score 0.2491969

Code Coverage 71.71

Package changelog, R 4.2.1

```
2.0.4
 - Fix strict-prototype warnings for CRAN

2.0.3
 - New function write_openssh_pem to support ed25519 in libssh2/gert

2.0.2
 - Disable tests that require internet access to comply with AON policy

2.0.1
 - Fix a unit test for a changed error message in openssl 3.0.2
```

Figure 3. Snippet of qualification document for package openssl.

To generate the document, R Markdown built-in “params” parameter and apply function were used for looping through a list of qualified package names. To avoid printing pound signs, a combination of comment=NA parameter and cat() function was used in the R Markdown chunk responsible for printing the NEWS/NEWS.md. The code chunk is shown below:

~~~
```{r, comment=NA}
if (length(changelog$package) > 0) {
  cat(unlist(changelog$news_file), sep = "\n")
}
```
~~~

To avoid incorrect formatting and ensure NEWS/NEWS.md fits the report's layout, the results='asis' and echo=FALSE options were used in a separate chunk as shown below:

~~~
```{r, results='asis', echo=FALSE}
if (length(changelog$package) == 0) {
  cat(paste("> NEWS/NEWS.md file was not found for package", changelog$package))
}
```
~~~

OPEN-SOURCE RISKMETRIC INTEGRATION

The riskmetric package provides a framework for retrieving package metadata, assessing package metrics, and summarizing the risk that the package might not provide accurate results [6]. Integration of the riskmetric scores into the package qualification workflow allows for supplemental information to be provided to the qualification reviewers. The integration is achieved through a workflow that begins with obtaining the package metadata. Next, a series of 15 assessment functions from the riskmetric package are called to assess the extracted metadata against different risk criteria. These assessment functions are named such that they have the prefix of "assess_". A score is then generated for each assessment function using the pkg_score() function. This scoring provides meaning to the data returned by the assessment functions. Finally, the scores are aggregated into an overall risk score where a high score indicates higher risk. Table 2 shows the different assessment functions and their purpose.

Index	Assess Function	Value Returned	Purpose
1	assess_news_current()	Logical	Contains a logical vector indicating if NEWS file is up-to-date
2	assess_has_vignettes()	Number of discovered vignette files	Assesses if the package has vignettes
3	assess_has_bug_reports_url()	URL	To return the URL for the bug report
4	assess_last_30_bugs_status()	TRUE/FALSE	Logical vector indicating if last 30 bug reports were closed
5	assess_license()	String	Returns the string indicating the license under which the package is released
6	assess_export_help()	TRUE/FALSE vector	Logical vector indicating if each namespace export has documentation
7	assess_downloads_1yr()	Number of downloads	Numeric value indicating the volume of downloads
8	assess_has_website()	Character vector	A character vector of website URLs associated with the package
9	assess_r_cmd_check()	Numeric	A tally of errors, warnings and notes from running R CMD check locally

10	assess_remote_checks()	Numeric	A tally of R CMD check results run on different OS flavors
11	assess_has_maintainer()	Character Vector	Gets the names of maintainers associated with the package
12	assess_exported_namespace()	List	A list of functions and objects exported by a package
13	assess_has_news()	Numeric	Numeric value for number of NEWS files discovered
14	assess_has_source_control()	URL	Source control URLs associated with the package
15	assess_covr_coverage()	List	A list containing the filecoverage (a vector for coverage of each file) and totalcoverage (a single value for overall test coverage)

Table 2. Assess Functions used in riskmetric package. The functions not used within our organization's qualification workflow are greyed out.

When integrating the riskmetric package we exclude, assess_r_cmd_check(), assess_remote_checks(), and assess_covr_coverage() as these assessments returned NA values. The remaining 12 assessment functions are used to generate a risk score. As mentioned above, there are three basic steps in the workflow of the riskmetric package:

1. Obtain package information using pkg_ref();
2. Assess the package against validation criteria using pkg_assess();
3. Score the assessment criteria using pkg_score()

By default, pkg_assess() will try to assess the package using all 15 assessments in Table 2. The list of assessments applied to a package can be obtained by using the assessments argument in pkg_assess(). Additionally, some of the assessments will return an error and pkg_score() will not be able to properly handle the error. The error_handler argument allows end users to define a function to handle the case when an assessment produces an error. Within our workflow, this error was generated when assessing last 30 bugs resolution status. If a package did not have a bug report Uniform Resource Locator (URL), then pkg_assess() would throw an error. The result returned after running pkg_score() is a list containing values from 0 to 1 for each assessment applied where 0 is a failed assessment and 1 is a perfect assessment. So, if 5 assessments were applied to the package, the returned list would contain 5 values. The code below shows the custom error handler we defined and the metrics function.

```
error_handle <- function(x) {
  if (class(x)[1] == "pkg_metric_error") {
    return(0)
  }
}

list_of_assessments <- riskmetric::all_assessments()


metrics1 <- function(x) {
  hold <- riskmetric::pkg_ref(x) %>%
    pkg_assess(assessments = list_of_assessments[c(-5, -12, -9, -10, -15)]) %>%
    pkg_score(error_handler = error_handle)
}
```

```
   metrics_1 <- lapply(low1$package, metrics1)
```

In the code above, two assessment functions are excluded in addition to the three assessment functions already excluded from the workflow. These assessment functions were for assessing the presence of the license (Index 5) and for assessing the number of exported namespaces (Index 12). These two assessments need special processing because pkg_score() returns a value of NA despite pkg_assess() returning a meaningful value.

The assess license function returns a character value indicating the type of license being used for the specified package. If no value is found, an empty string is returned. A modified function was developed to test if a value is returned by the assessment function is valid. The assessment is then scored, to ensure the score is of the same format as the previous scores. The NA value is then overwritten with a value of 1. If no license is found, the assessment is scored and then given a value of 0. The final returned value is a list containing the result of the license assessment function.

```
   metrics2 <- function(x) {
     score <- riskmetric::pkg_ref(x) %>%
       pkg_assess(assessments = list_of_assessments[c(5)])
     if (length(score[[1]])) {
       score <- score %>% pkg_score()
       score[[1]] <- 1
     } else {
       score <- score %>% pkg_score()
       score[[1]] <- 0
     }
     return(score)
   }

   metrics_2 <- lapply(low1$package, metrics2)
```

If functions or objects are found, after running pkg_score() the total number is returned. So, if a package has 100 exported functions or objects, a value of 100 is returned after using pkg_score(). If no functions or objects are exported, the value of NA is returned. In our current algorithm, if any functions or objects are found, a value of 1 is assigned otherwise a value of 0 is assigned for this assessment. In the future, a more complex algorithm can be designed that may include the number of vignettes compared to the number of functions. This algorithm can check whether at least one vignette exists for all the exported functions. This could be used as an additional indicator of a well-documented package.

```
   metrics3 <- function(x) {
     score <- riskmetric::pkg_ref(x) %>%
       pkg_assess(assessments = list_of_assessments[c(12)]) %>%
       pkg_score()
     if (is.na(score[[1]])) {
       score[[1]] <- 0
     } else {
       score[[1]] <- 1
     }
     return(score)
   }

   metrics_3 <- lapply(low1$package, metrics3)
```

The three lists are then combined into one large list containing one list per package with the results of all 12 assessments. This is currently done using the Map function:

```
listf <- Map(c, metrics_1, metrics_2)

listfinal <- Map(c, listf, metrics_3)
```

These scores were then weighted and totaled using the function below. Given that the values returned by pkg_score() are in the format of the list, it is important to first “unlist” the results before aggregating the scores.

```
score <- function(x) {
  weights <- c(.083, .084, .083, .083, .084, .084, .084, .083, .083, .083)
  1 – sum(as.numeric(unlist(x)) * weights, na.rm = T)
}

risk_metrics <- lapply(listfinal, score)
```

The final result is a data frame containing a value ranging from 0 to 1 for each package where a lower score indicates a more reliable package. In our current state, all assessments are considered of equal weight where some of the assessments are arbitrarily assigned a value of 0.084 instead of 0.083 to allow for all the weights to add to 1. Although we are assigning equal weights, not all of the assessments have the same level of significance regarding the quality of a package. A widely used package that is well documented but has many unclosed bug reports, might not be as reliable as another widely used package with less documentation but more frequently closed bug reports. Continued discussions are needed to define our final internal weights applied for these assessments.

AUTOMATED GENERATION OF QUALIFICATION DOCUMENTS

For package review and audit purposes, we create a set of supporting documentation that is later stored on the organization’s computing platform. As a standard approach, we use a combination of R scripts and R Markdown to automate most of the tasks. All of the documents are generated automatically, with little input from the user, such as the parameters for working directory and R version. The rest of the tasks are performed in batch run, by running vignettes. The set of documents are listed below:

- Qualification-review document: Purpose of this document is to summarize package qualification results and compare them with previous snapshot. This helps Subject Matter Expert (SME) to spot differences between package qualification categories in different snapshots.
- Package Qualification Tracker document: The purpose of this document is to track the status of the qualification for a given snapshot. A number of the details for each package are automatically generated including snapshot date used for the global library, previous package risk category, R version under which the package is qualified, the date that the qualification documents were generated, criteria used for qualification and general package information. Names of reviewers, requestors, re-qualification reasons, exceptions and status need to be updated manually. This document is used to track review activities according to our internal SOPs and may not be applicable for all organizations.
- Document with an overview for low and moderate risk packages: Each package at low or moderate risk will have an overview document that contains general package information such as link to CRAN site, authors, change log, code coverage and riskmetric scores. They also contain associated risk categories, package dependency and criteria within the present snapshot that were used for qualification purposes.
- Document with package types: Depending on a package type (statistical/non-statistical), additional measures are undertaken for package qualification. We differentiate the two package types using the following criteria:
  - Non-statistical packages do not require additional statistical validation. Packages used for:
    - Graphing (E.g., visualization functions of ggplot2)

        - Software engineering/utility (E.g., devtools)
        - Data manipulation (E.g., tidyverse)
        - Documentation and reporting (E.g., r2rtf, shiny, knitr)
    - Statistical packages require additional statistical review to be qualified as low or moderate risk. A statistician performs a thorough review and provides justification on whether the package is suitable to be used for a specific purpose. Once the review is complete, it is documented in a Microsoft Word document (.docx) and is stored on the computing platform.

- Final request form: CSV file, that contains final list of packages and their corresponding versions.

- Dry-run installation log: After each installation, a copy of log is saved in the production area. Apart from install.packages() messages generated by the installation code, this document also contains values of environmental and session variables which are helpful for solving potential IT-related issues. The environmental and session variable values are obtained by using the Sys.getenv() and sessionInfo() functions.

- Log checker results: During installation an .Rout file is generated that contains printed messages from execution of install.packages(). Due to the fact that length of file varies from a few hundred to few thousands of lines, a logchecker() function is used to scan the file for potential issues. Regular expressions are used to search for error and warning messages, which are stored in a list (See

- **APPENDIX**). R Markdown is then used to generate and save a custom HTML document that contains the results from logchecker(). This HTML file is then reviewed by a programmer and a SME. If any low/moderate risk package fails installation, this is escalated to IT team.

After each formal dry-run installation, process SMEs commence the post-installation review. This step is needed to ensure the installation was completed successfully and all requested packages have been deployed.

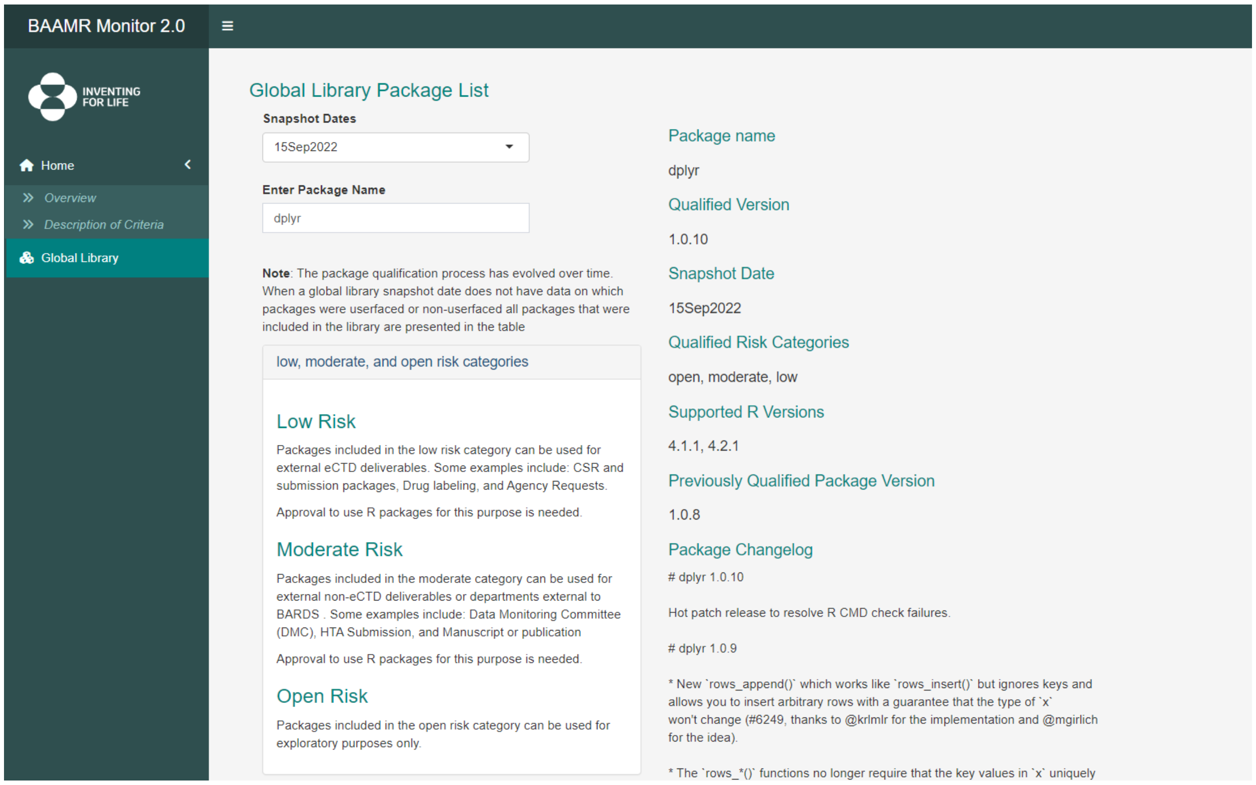
INFORMATION DISSEMINATION & UPSKILLING

The results of the qualification process are an important source of information to end users of an R package as it enables the users to confidently use the package in the appropriate risk categories to generate requested A&R deliverables. Thus, the dissemination of the qualification process results plays a vital role in promoting the use of R or other open-source software in clinical trials A&R.
To efficiently distribute the qualification process and global R library update results, a workflow was developed for information dissemination. First the qualification SME team sends out an email to the user distribution list making them aware of the changes in the global r library snapshot. Next, the qualification SMEs also pen a blog post within the internal R portal making users aware of the changes in the global R library and directing them to the internally developed R Shiny application 'baamr-monitor'. The combination of emails, R Shiny application and blog posts help with efficient dissemination of qualification related to end users as well as maintaining compliance with established SOP. We also have several internal training modules developed to ensure that users have sufficient knowledge related to package qualification and implementation. These aspects will be discussed here.

SHINY APPLICATION TO MONITOR QUALIFICATION METRICS

The 'baamr-monitor' application allows users to find details and metrics regarding the qualification status of individual packages. The global R library updates are often released in the middle of development lifecycle and may impact functionality. 'baamr-monitor' allows users to investigate previous global library snapshots and compare to the current global library to determine what has changed with respect to their needs. A screenshot of the information displayed when users search a given snapshot for a specific package is shown Figure 4.

The application also provides a number of metrics for the current global library. These metrics include package count within each risk category, distribution of criteria usage, and a description of each criterion. The combination of emails, R Shiny application and blog posts help with efficient dissemination of qualification related to end users as well as maintaining compliance with established SOP.



Global Library Package List

Snapshot Dates: 15Sep2022

Enter Package Name: dplyr

Note: The package qualification process has evolved over time. When a global library snapshot date does not have data on which packages were userfaced or non-userfaced all packages that were included in the library are presented in the table

low, moderate, and open risk categories

Low Risk

Packages included in the low risk category can be used for external eCTD deliverables. Some examples include: CSR and submission packages, Drug labeling, and Agency Requests.

Approval to use R packages for this purpose is needed.

Moderate Risk

Packages included in the moderate category can be used for external non-eCTD deliverables or departments external to BARDS . Some examples include: Data Monitoring Committee (DMC), HTA Submission, and Manuscript or publication

Approval to use R packages for this purpose is needed.

Open Risk

Packages included in the open risk category can be used for exploratory purposes only.

Package name

dplyr

Qualified Version

1.0.10

Snapshot Date

15Sep2022

Qualified Risk Categories

open, moderate, low

Supported R Versions

4.1.1, 4.2.1

Previously Qualified Package Version

1.0.8

Package Changelog

```
# dplyr 1.0.10

Hot patch release to resolve R CMD check failures.

# dplyr 1.0.9

* New `rows_append()` which works like `rows_insert()` but ignores keys and
allows you to insert arbitrary rows with a guarantee that the type of `x`
won't change (#6249, thanks to @krlmlr for the implementation and @mgirlich
for the idea).

* The `rows_*()` functions no longer require that the key values in `x` uniquely
```

Figure 4 The 'baamr-monitor' application showing the details and metrics of a global library update.

TRAINING AND UPSKILLING

A clinical project may contain different types of deliverables, therefore multiple project-specific R packages could be required to address analyses needs in a clinical project when packages in global R library doesn't meet the needs. A project lead works with a project-specific R Package SME to determine the risk level of a new external package or package update being used within the clinical project. It is important for the study team to understand the process to determine the risk levels of the packages that will be used to support their deliverables. If risk level of a package in the global library needs to be reassigned, then the programmer must understand the available resources to help achieve the package qualification. In our organization, we have created several methods such as development of internal R trainings, creation of SAS/R governance committee team, etc. ensuring that each member has the knowledge related to package qualification and implementation.

R e-Learning training for Statistics and Statistical programming groups provides details to utilize R platform and complete A&R work for project-specific package. The training is organized as one set of slides per topic covered in Analysis and Reporting (A&R) workflow process using R. This training helps members become familiar with key A&R SDLC components by using examples within R environment. At the end of each topic knowledge checks are included to gauge the trainee's level of understanding.

SAS/R governance committee was established within our organization to review and approve R usage requests from a study team. It is designed to help staff understand when it is appropriate to use R and when it is not. This ensures process compliance, traceability, and reproducibility for A&R deliverables using R. Project teams can benefit from SAS/R governance team by review of pre-approved R usage. It also reduces the burden to qualify external R packages and connects the study team with R SME for technical support.

PLANNED FUTURE ENHANCEMENTS

While working with external R packages, it was noticed that there are different ways for an end user to submit a request for package qualification. Most frequently used were:

- Send email to SME mailing list or individual SME
- Submit JIRA ticket
- Write request directly to a tracker (Excel spreadsheet)

Data submitted through these different channels requires additional effort to review and manually consolidate into one data source which is prone to errors. Also, the request source may be difficult to trace, and the submitted data may not be sufficient or partly incorrect. In the long run, these issues result in qualification process being tedious and inefficient.

To avoid errors, data duplication and lack of traceability, we are working on creating a one-stop solution for requesting a package qualification. The solution is to have end users submit these requests through an application that is easy to maintain and flexible. Most programmers and statisticians are already aware of process used for submitting IT-related issues. We plan to take a similar approach and implement it with R Shiny technological stack. More details about the proposed Shiny application will be discussed in this section.

PROPOSED SHINY APPLICATION TO PROCESS QUALIFICATION REQUESTS

The application not only simplifies the package request process (for both SMEs and package users), but also makes the process more transparent and Quality Assurance (QA)/audit ready. The use of R Shiny technological stack leads to easier application maintenance and integration into already existing R production space available on the organization’s servers.

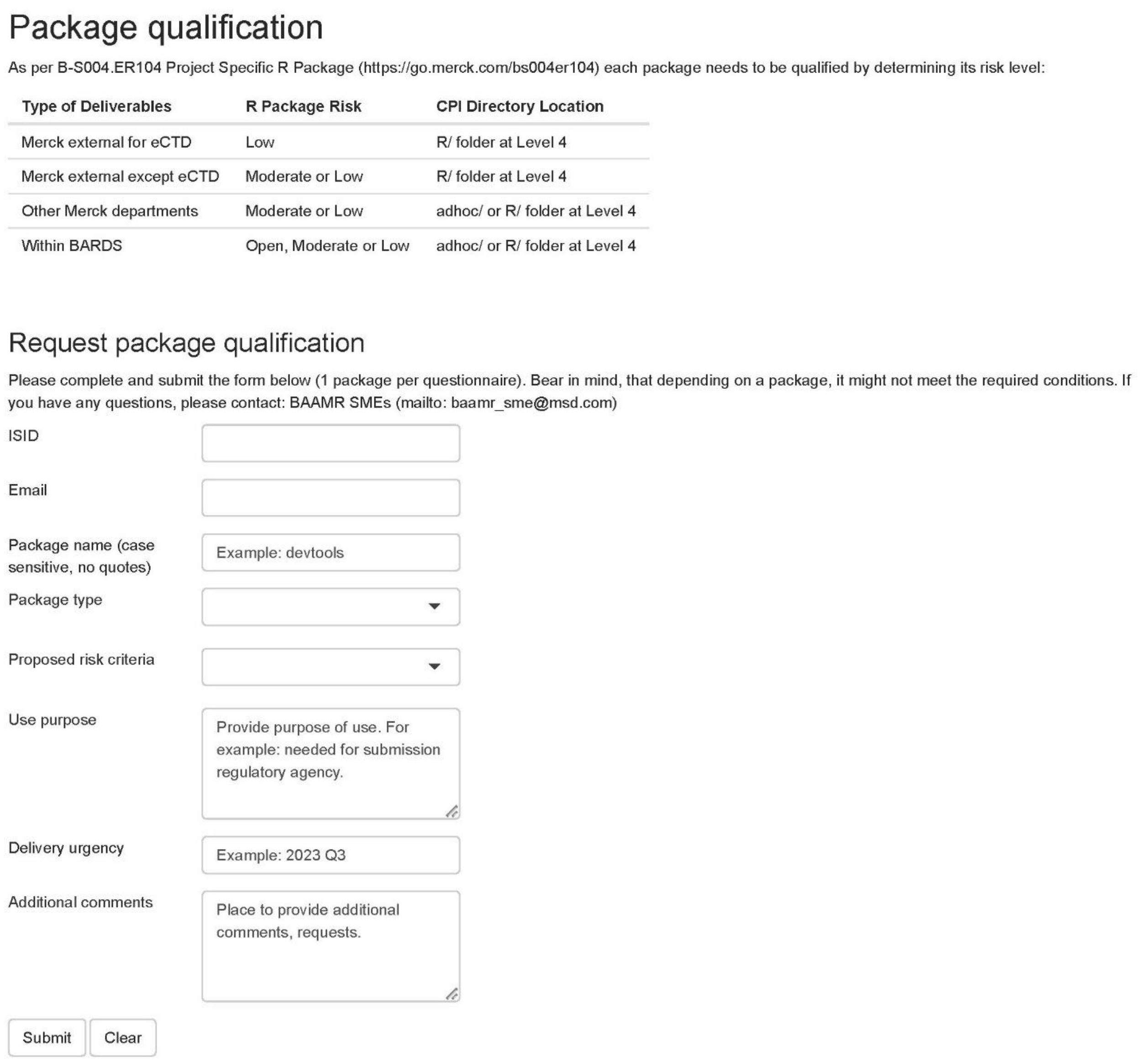
The application contains a simple user interface, with a brief introduction and description of risk categories. For each field (except for “Additional comments”) there is input validation implemented. To avoid duplicating the qualification process, a solution was designed to check if a package has already been qualified for a proposed risk criterion. Once entered data is validated and submitted, it is transferred to a separate data frame. At the same time, generic email is sent to team responsible for package qualification. If the data is complete and there are no additional comments or requests, SME initiates a formal package qualification process. Once the qualification process is complete, the data frame is updated, and requester is notified using provided email address.

Additional steps are needed when a statistical package is requested for qualification. Due to the nature of the statistical package type, the qualification process is longer and requires a few additional steps to formally qualify into anan appropriate risk level. The pros and cons of using the Shiny application in qualification process are listed in Table 3. The user interface of the proposed R Shiny application used for requesting qualification is shown in **Error! Reference source not found.**.

Pros	Cons
• One stop solution for requesting creation of package qualification (everything is implemented within R eco-system) • Reduced manual work in consolidating data sources • Fully traceable • Easier to maintain and automate downstream qualification process • Automatically create qualification document tracker • Minimum user input required	• Extra steps needed for setting up email account used for sending messages to package qualification team (after pressing "Submit" button) • Additional time required to develop and test R Shiny app • Separate data frame is needed to track requests

• Use can be extended to other open-source programming languages such as Python	

Table 3. The pros and cons of R Shiny application usage in qualification process.



Package qualification

As per B-S004.ER104 Project Specific R Package (https://go.merck.com/bs004er104) each package needs to be qualified by determining its risk level:

Type of Deliverables	R Package Risk	CPI Directory Location
Merck external for eCTD	Low	R/ folder at Level 4
Merck external except eCTD	Moderate or Low	R/ folder at Level 4
Other Merck departments	Moderate or Low	adhoc/ or R/ folder at Level 4
Within BARDS	Open, Moderate or Low	adhoc/ or R/ folder at Level 4

Request package qualification

Please complete and submit the form below (1 package per questionnaire). Bear in mind, that depending on a package, it might not meet the required conditions. If you have any questions, please contact: BAAMR SMEs (mailto: baamr_sme@msd.com)

ISID	
Email	
Package name (case sensitive, no quotes)	Example: devtools
Package type	
Proposed risk criteria	
Use purpose	Provide purpose of use. For example: needed for submission regulatory agency.
Delivery urgency	Example: 2023 Q3
Additional comments	Place to provide additional comments, requests.

Figure 5 Proposed R Shiny application used for processing qualification requests

CONCLUSION

The process of qualification of open-source software is continuously evolving to ensure adherence to best practices followed industry-wide and to align with internal organizational requirements. We anticipate that our qualification process for R packages will require updates to further improve efficiency and reliability. One of the updates planned is to set up gating criteria for risk scores generated by the riskmetric package. The gating criteria would help in enhancing the risk-based strategy by associating package risk levels with risk scores.

The human reviewers play an important role in the qualification process by providing valuable feedback regarding the quality of the package being qualified as well as reviewing the compliance of the generated qualification documents. The assessment is subjective and there is no standard guidance available for reviewers to reference when assessing a package for qualification into a given risk level. We hope to establish a standard guidance document for internal reviewers of package qualification with detailed steps for reviewing a package for quality and compliance.

Further automation of the qualification process is also possible by automating the processing of new requests raised by users regarding qualification of R packages. We are focusing on accomplishing this aspect by integrating the “requests app” into the qualification process. Once request form is submitted by user, qualification process could be initiated automatically. After the qualification process is completed, system would notify SMEs about generated documents/data. This would enable faster and efficient processing of new requests as well as help with adherence to established SOP by automatic generation of request related documentation.

REFERENCES

[1] Jane Liao, Fansen Kong, Yilong Zhang “External R Package Qualification Process in Regulated Environment”. 2022. https://www.lexjansen.com/pharmasug/2022/SI/PharmaSUG-2022-SI-057.pdf

[2] Andy Nicholls, Paulo R. Bargo, John Sims on behalf of the R Validation Hub "A risk-based approach for assessing r package accuracy within a validated infrastructure" January 23, 2020. Available at https://www.pharmar.org/white-paper/

[3] Yalin Zhu, Rinki Jajoo, Clare Bai, Sarad Nepal, Daniel Woodie, Keaven Anderson, Yilong Zhang “R Package Oriented Software Development Life Cycle in Regulated Clinical Trial Environments” 2020. https://www.lexjansen.com/phuse-us/2020/tt/TT12.pdf

[4] The R Foundation for Statistical Computing c/o Institute for Statistics and Mathematics “R: Regulatory Compliance and Validation Issues, A Guidance Document for the Use of R in Regulated Clinical Trial Environments” October 18, 2021. https://www.r-project.org/doc/R-FDA.pdf

[5] Posit Solutions – “Shared Baselines”. https://solutions.posit.co/envs-pkgs/environments/shared/

[6] Juliane Manitz, Douglas Kelkhoff, Eli Miller, Yilong Zhang “Introduction to the R Package riskmetric” June 09, 2020. https://www.pharmar.org/blog/2020/06/09/2020-06-02-riskmetric-intro-jun-2020/

ACKNOWLEDGMENTS

The authors would like to thank BAAMR and management teams from Merck & Co., Inc., Rahway, NJ, USA, for their advice on this paper/presentation.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Corresponding Author:
Uday Preetham Palukuru, Ph.D.
Merck & Co., Inc., Rahway, NJ, USA
E-mail: preetham.palukuru@merck.com

Paul Bernecki
MSD, The Circle 66, CH-8058, Zurich Airport, Switzerland
E-mail: paul.bernecki@msd.com

Nicole Jones
Merck & Co., Inc., Rahway, NJ, USA
E-mail: nicole.jones3@merck.com

Abhilash Vasu Chimbirithy
Merck & Co., Inc., Rahway, NJ, USA
E-mail: chimbirithy_abhilash@merck.com

APPENDIX

The task of extracting R generated errors strings using regex is split into three parts. below:

- In the first part "error_strings" vector is defined, that contains the errors patterns as shown below:

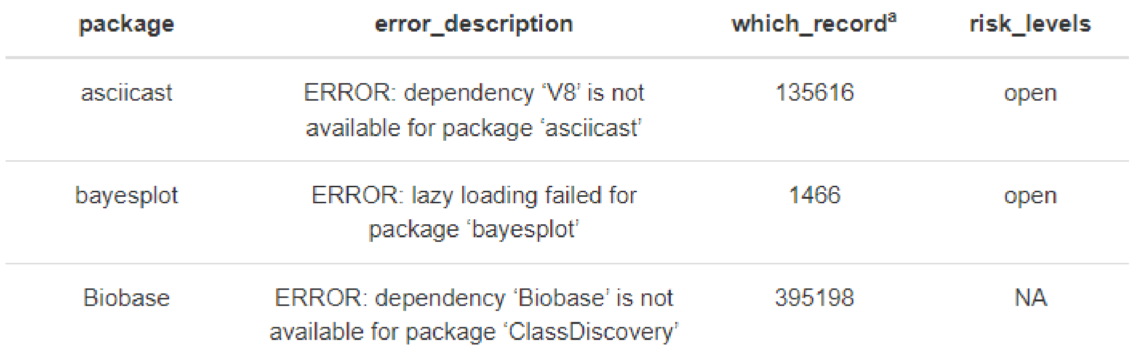
```
error_strings <- c("configuration failed for package",
                   "lazy loading failed",
                   "compilation failed",
                   "is not available for this version of R",
                   "is not available for package")
```

- In the second part the installation log is imported and a combination of lapply() and grep() functions are used to find all records that contain any of error_strings elements.
- In the third part all the error containing log entries are looped through and package names are extracted. Since R uses special non-ASCII characters to enclose package name (for example openssl), a special pattern is defined that can be used for extracting enclosed name as shown below:

  ```
  pattern <- "ERROR:.*\u2018 *(.*?) *\u2019.*"
  ```

  In this case, left and right single quotation marks are being encoded in Unicode, by using u2018 and u2019 representation, with a backslash as escape character. It is suggested to use this method to avoid R CMD warnings, as those checks are initiated to find non-ASCII characters.

At the end of this process, package name is extracted along with error message and a line number. This data is then appended to a list data structure for storage. By adding internal risk levels, the findings can be presented in HTML format by using R Markdown vignettes. The example of such report can be seen below:



package	error_description	which_record[a]	risk_levels
asciicast	ERROR: dependency 'V8' is not available for package 'asciicast'	135616	open
bayesplot	ERROR: lazy loading failed for package 'bayesplot'	1466	open
Biobase	ERROR: dependency 'Biobase' is not available for package 'ClassDiscovery'	395198	NA

The report helps in review of the errors that occurred during dry-run installation. The "which_record" column is helpful while navigating the installation log file and programmatically extracting the snippet containing the error.