Paper TT12

# R Package Oriented Software Development Life Cycle in Regulated Clinical Trial Environments

Yalin Zhu, Rinki Jajoo, Clare Bai, Sarad Nepal, Daniel Woodie, Keaven Anderson, Yilong Zhang
Merck & Co., Inc., Kenilworth, NJ, USA

## ABSTRACT

R and its ecosystem have become popular for planning and analyzing data in different stages of drug development. There are several pioneer and ongoing efforts to guide the use of R and R packages in regulated clinical trial environments for the pharmaceutical industry. Within an organization, it is also critical to understand the process of developing and deploying internal R packages. In this paper, we proposed an R software development life cycle (R-SDLC) for R package development within an organization. The aim of the R-SDLC proposal is to provide guidance for internal analysis tool development and validation using R, build consistent programming style and principals, and comply with essential regulatory requirements. The proposed R-SDLC is a recommendation and only based on our experience. We hope the proposed R-SDLC will help to build industry guidance of using R and R packages within an organization.

## INTRODUCTION

Within an organization, people develop, maintain and share programming code. R and its ecosystem have become popular for planning and analyzing data in different stages of drug development. There are several pioneer and ongoing efforts to guide the use of R and R packages in regulated clinical trial environments for the pharmaceutical industry. For example, the R foundation provided a guidance for the use of R in regulated clinical trial environments (https://www.r-project.org/doc/R-FDA.pdf). The document provided a common foundation to address the good of practice (GxP) of using R for an organization to meet its own internal standard operating procedures (SOP), documentation requirements, and regulatory obligations.

There are many open source R packages that have been developed and widely used in clinical trial activities such as “survival”, “gsDesign”, and other R packages summarized in (https://cran.r-project.org/web/views/ClinicalTrials.html). Within an organization, internal R packages may be developed and applied for internal projects together with open source R packages. As indicated by the R foundation guidance document, *“There is a significant obligation on the part of the end-user’s organization to define, create, implement and enforce R installation, validation and utilization related Standard Operating Procedures (SOPs) within the end-user’s environment. These SOPs should define appropriate and reasonable quality control processes to manage end-user related risk within the applicable operating framework.”* One of the critical parts to fulfill the requirement of GxP from regulatory agencies is to understand the process of developing and deploying internal R packages.

In this paper, we suggested an R software development life cycle (R-SDLC) for internal R package development and deployment processes based on our experiences. The aim of the R-SDLC proposal is to provide guidance for internal R package development and validation and best programming practices to comply with essential regulatory requirements. The proposed R-SDLC is a recommendation and only based on our prior experience. We hope the proposed R-SDLC will aid in building industry guidance on developing internal R packages and provide a regulated infrastructure for use of R in an organization.

## R-SDLC OVERVIEW

An R Package is a ubiquitous way for developing, maintaining and sharing R code, with a standardized structure for including functions, documentation, and datasets. R-SDLC provides a streamlined framework

for defining, developing, testing, and deploying such an R package for regulatory use. Several powerful and open sourced R packages (e.g. devtools, usethis, roxygen2, covr, testthat, pkgdown, etc) simplified the development and maintenance of an R package.

Since an R package is a standardized structure to save R functions, documentations and sample data, the following standard R package folder structure can be used to keep all the required items:

- Package information: DESCRIPTION, README.md files
- R function (with specification): \R folder
- Documentation: \man folder
- Testing cases: \test folder
- Examples: \vignettes folder
- Change log: NEWS.md file

Developing and maintaining an internal R package within an organization requires collaboration from cross-functional areas that typically involve subject matter expert(s) (SME), statisticians, statistical programmers and R administrator(s), as shown in Figure 1. In general, each role takes its unique functionality and responsibility:

- The SME is the owner of the R-SDLC process. The SME develops and maintains the R-SDLC process documentation within an organization. The SME also helps to initiate workflow with input from statisticians and programmers. The SME also serves as the point of contact to communicate with the R administrator and IT department.
- Statisticians commonly provide proof of concept to construct the specification and validation plan. Statisticians may also develop R functions to specify the request and help review R functions developed by programmer as stakeholders.
- Statistical programmers are developers of internal R packages by preparing specifications, developing functions based on specifications, and validating functions based on a validation plan. Statistical programmers play an important role in ensuring compliance of the R-SDLC process while developing an internal R package.
- The R administrator maintains the internal R computing and development platform (e.g. Bitbucket and RStudio server pro, RStudio Package Manager etc.). The R administrator helps the SME to deploy, install and upgrade R packages regularly.

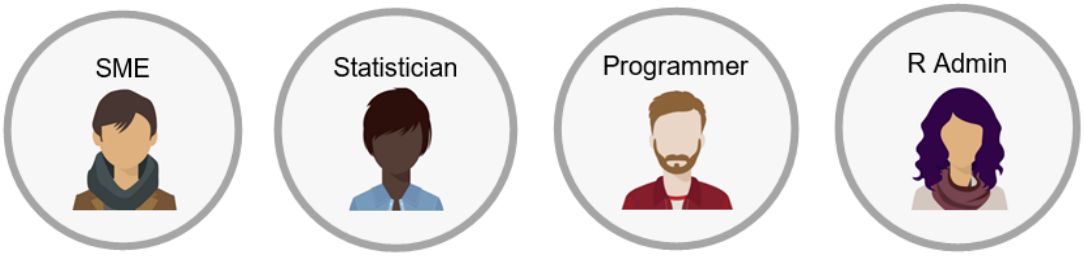


Figure 1. Functional area for internal R package development within an organization.

The R-SDLC can be defined in four stages:

- **Define**: gather all requirements for functions needed to generate an R package, such as document DESCRIPTION, README.rmd, NEWS.md, etc. R package and function validation is planned and documented.
- **Develop**: follow specifications to develop R functions.
- **Validation**: R functions from the development phase are tested according to the validation plan following the specification. Independent testing or double programming may be required for R functions within an R package.
- **Operation**: promote developed and validated R package to the production area of the computing platform. Change management/maintenance is done as needed to address new requirements or issues.

It is recommended to follow R-SDLC process to develop and deploy internal R packages to address software GxP requirements from regulatory agencies. It is worth noting that, the four stages make a cycle instead of a step-down procedure. For example, if statisticians and statistical programmers find any issues while developing and validating functions, they can return to the define stage to update the specification. A version control tool (e.g. git) shall be used throughout the R package development lifecycle. For facilitating the version control creation and updates, an issue report system (such as JIRA board) can be used. For the readers not familiar with Git, please refer to the "Happy Git with R" book (https://happygitwithr.com/)

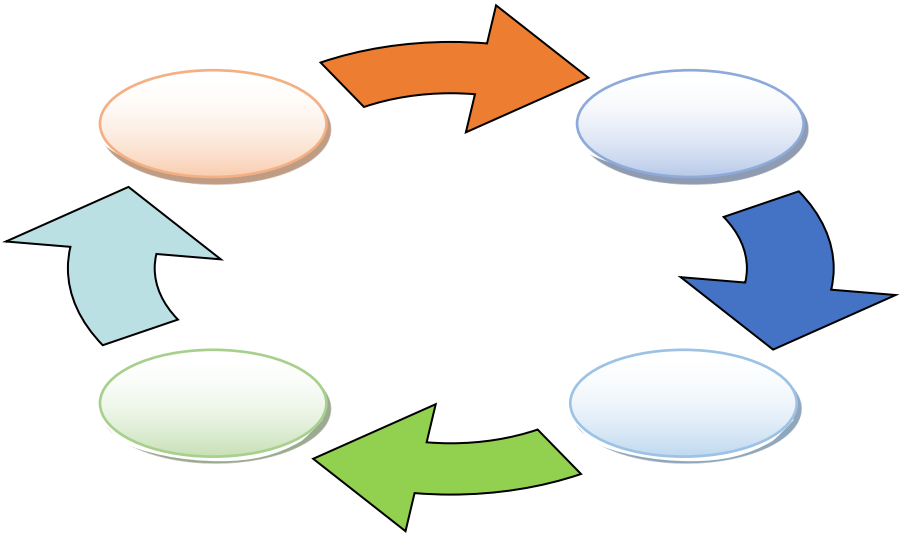

Figure 2. R-SDLC

Following the four R-SDLC stages, we propose to use a badge system to classify the status of an R package within the R-SDLC. This is inspired by the tidyverse lifecycle badges (https://www.tidyverse.org/lifecycle/). Figure 3 summarizes a recommended set of badges used in R-SDLC stages. Other badges may be generated as needed using the svg file generating tool (https://shields.io/). Figure 4 shows an example for displaying the badges in the front page of our standard package template website. Table 1 explains the definition of each badge.

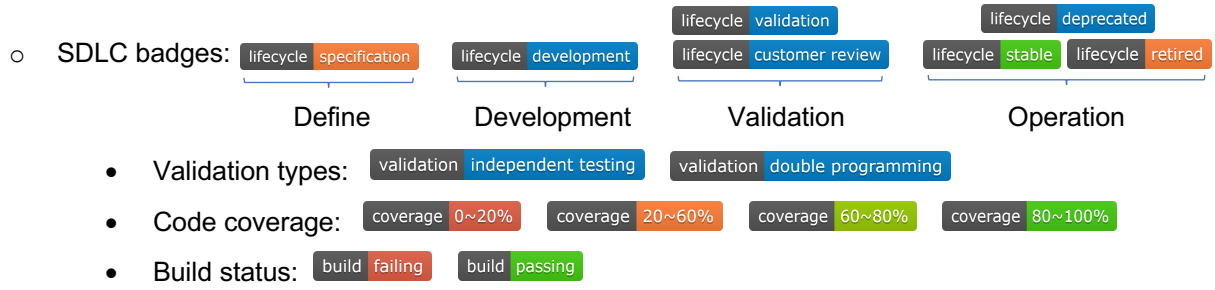


Figure 3. R-SDLC badge system.

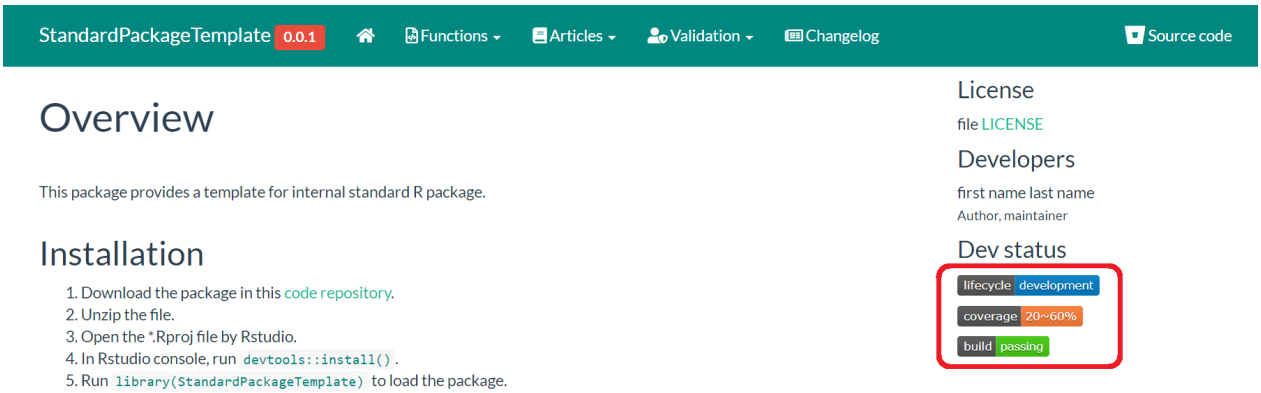


Figure 4. An example for badges displaying on package website.

Table 1. List of R-SDLC badges and definitions.

| R-SDLC Badge | Definition |
|---|---|
| Specification | Define a package and function's scope, business requirements, developing environment, etc. |
| Development | Develop functions following programming style and best practice, complete developer testing and build user manual. |
| Validation | Validate a developing function. |
| Independent Testing | Create testing programs to run the function and verify the results. |
| Double Programming | Re-create function and testing programs to reproduce all output from the function using the specs only, without looking at the function being validated. |
| Customer Review | Execute new functions and review output, usually acted by statistician and/or business end users. |
| Stable | Indicate an R package/function is ready to release in production at current version. |
| Deprecated | Indicate an R package/function will be retired and may be replaced by a new R package/function in near future. |
| Retired | Indicate an R package is no longer used. |
| Code Coverage | Measure source code test/validation completion degree |
| Build Status | Indicate whether the R package and corresponding functions pass build and check criteria or not. |

**DEFINE STAGE**

In the define stage, programmers and statisticians shall initiate or update the specification of an internal R package. The specification shall include a list of functions to be developed. For each function, the meaning of all function arguments should be provided. Step by step instructions for implementing each function is recommended to be provided. For an existing R package, it is necessary to determine if specification updates are required for functions that require updating.

The specification can be documented using the "roxygen2" package (https://cran.r-project.org/web/packages/roxygen2/vignettes/roxygen2.html). An example is provided in the Appendix B of this paper. Statisticians and programmers should determine the level of validation required for each function in an R package. The benefit of using "roxygen2" is that, all package development information including specification, user manual/documentation, R functions, unit testing, and examples relating to an R package will be stored in an R package. If a version control system is used, the SME can help statistical programmers to set up a repository to host source code of an internal R package.

An R package user manual (PDF version) can be automatically generated by "roxygen2" with specification information. Figure 5 shows an example of a specification section for a function to create a package badge in a PDF user manual.

**Specification**

Logic for creating `xxx_badge` function (describle your function's specification)

- define badge name and purpose
- customize badge color with corresponding stage/purpose
- link to corresponding hyperlink/url

The followings are some possible code:

- `library(StandardPackageTemplate)`
- `risk_badge("open")`
- `sdlc_badge("specification")`
- `codecov_badge(0)`
- `build_badge("passing")`

Retrieve code coverage percentage (to be updated in baamr_codecov_badge function):

- `x <-covr::package_coverage()`
- `covr::percent_coverage(x,by="file")`

Figure 5. An example of a specification section for a function to create a package badge in a PDF user manual.

Within an organization, the team can develop a standard package template to simplify the step of initiating an internal R package. By using the RStudio project template (https://rstudio.github.io/rstudio-extensions/rstudio_project_templates.html), developers are able to initiate an internal standard package with a template wizard with the help of the SME. An example from our standard package template is shown in Figure 6.

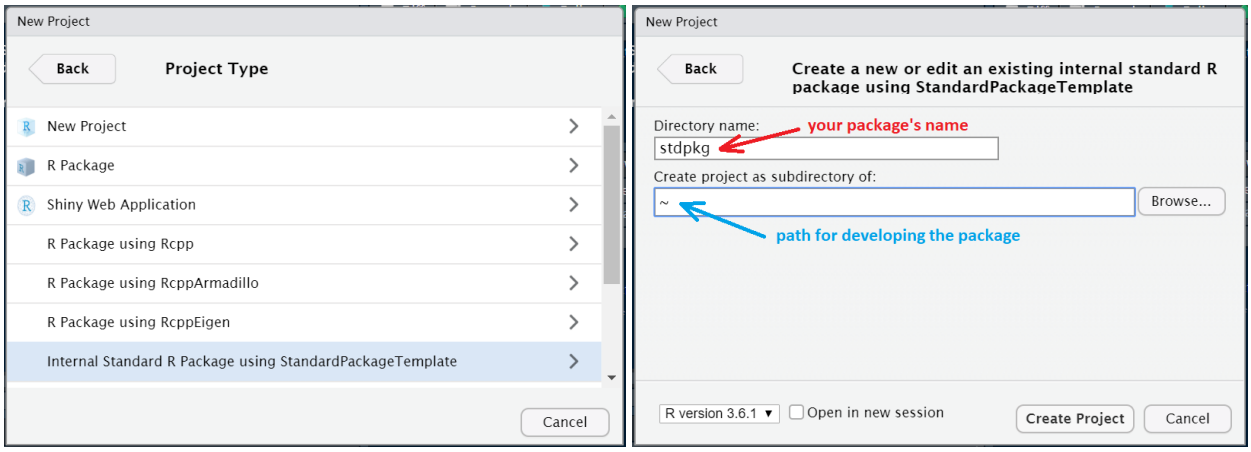


Figure 6. Project template wizard example for initiating an R package.

## DEVELOPMENT STAGE

While developing an internal R package, it is recommended to follow consistent programming practice and design guidance. For example, the Tidyverse group prepared the "Tidyverse Style Guide" (https://style.tidyverse.org/) and "Tidyverse Design Guide" (https://design.tidyverse.org/).

After developing the function, developers should follow a validation plan to start validating, testing functions and fixing errors and warnings produced by the function. Figure 7 shows the "plan and track" tool built in the standard package template website.

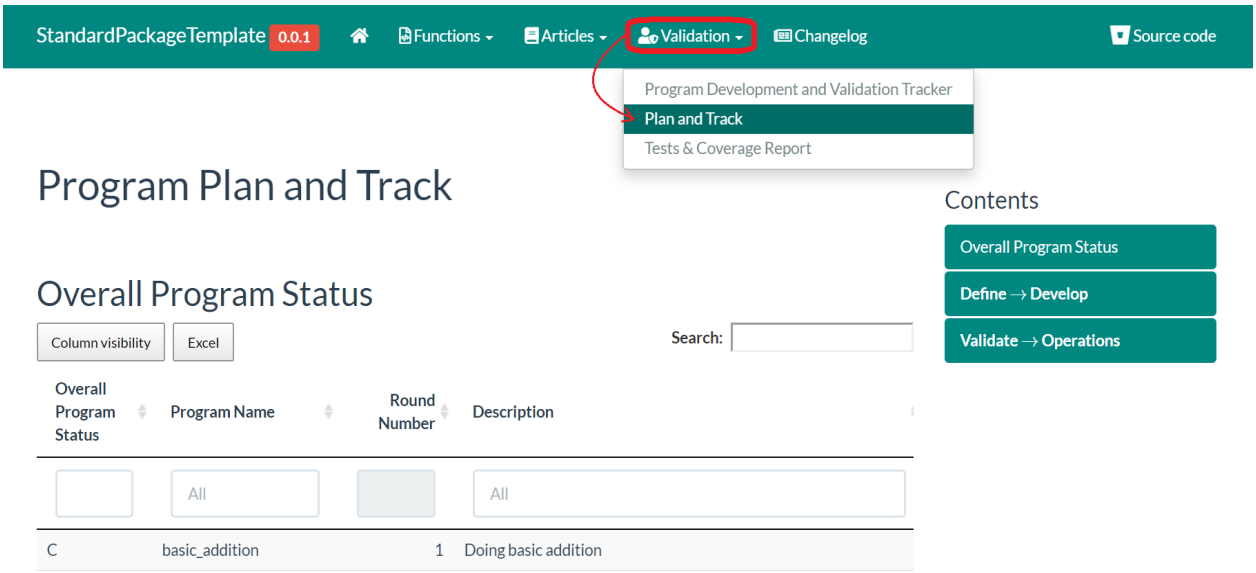


Figure 7. Plan and Track pages for R function's validation displayed on the package website.

A developer first needs to create an issue ticket and a new working branch for developer testing. Then the developer creates R programs in the test\testthat folder using file test-developer-testing-<filename>.R that matches the <filename> filename in the \R folder. "testthat" and "covr" packages can be used for developer testing. "testthat" aims to help the developer and tester illustrate test cases, testing expectations and catch any error, warning and note messages. "covr" can track test coverage for the developing R package and functions. It can also create a local code coverage report. Our standard package template is configured to also trigger automated builds and tests. As part of this, whenever a developer submits new code to a version control server (e.g. BitBucket), an automation server (e.g. Jenkins) builds and tests the code -- generating updated badges for build status and code coverage based on the outcomes. A general principle of testing R functions is discussed in the Chapter 10 of the "R Packages" book (https://r-pkgs.org/tests.html). In order to complete the developer testing, all existing and updated test cases should be passed with a build passing badge displayed in the corresponding branch. The new or updated function or package should pass all R package check list conditions with more than 80% (recommended) code coverage. The coverage 80% badge should be displayed in the corresponding branch.

The developer also needs to create or update the user manual. The general recommendation is to develop a user manual for each function unless it is an internal function within an R package. The SME determines if the user manual is necessary for each function. User manuals illustrating a function's input, output and examples can be documented with the "roxygen2" package. For more details on how to document an R function, please refer to Chapter 8 of the "R Packages" book (https://r-pkgs.org/man.html).

The final user documentation is automatically generated and saved in the \man folder of an R package by using "roxygen2". Both HTML and PDF user manuals can be displayed through an R package website using the "pkgdown" package. Figure 8 shows an example for viewing user manuals of the standard package template. If multiple functions are related and they can be summarized together as a case study, it is also strongly recommended to write an article and save it in the \vignette folder as a supplement tp a single R function's example in the user manual.

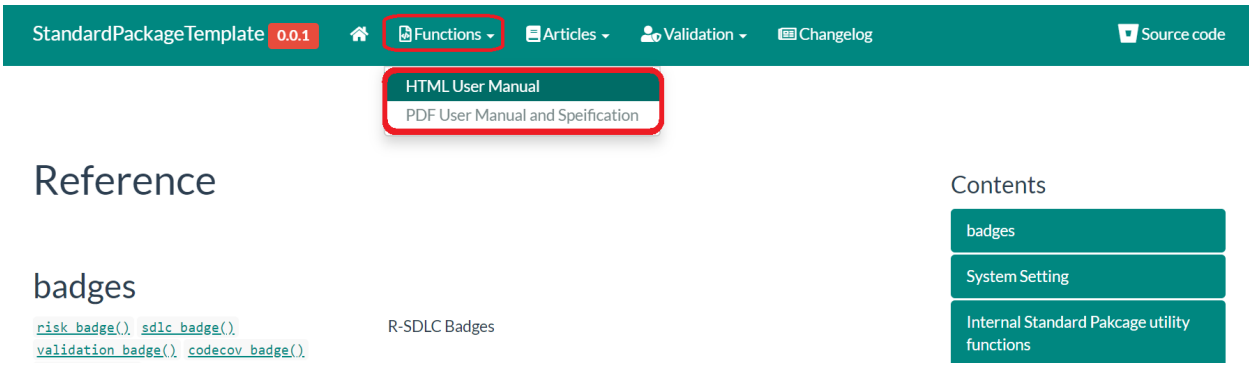


Figure 8. HTML and PDF user manual pages displayed on the package website.

While closing out the development, the SME needs to review the R package build history and code coverage report. After the review, the SME and Business Representatives need to approve the step of merging the working branch and the main branch of the R package repository. Then the SME can execute the merge step.

## VALIDATION STAGE

After functions of an R package are closed-out in the development stage, validation should be initiated by statistical programmers and/or statisticians as an independent tester and/or double programmer. For all new functions, either independent validation or double programming is required based on the validation plan. For function updates, the SME determines the level of further validation required (none, independent validation or/and double programming).

The validation stage ensures the accuracy and integrity of developed or updated R functions in the regulated environment and makes the R package ready for production. Similar as developer testing, independent tester or double programmer first creates a new working branch for validation. Then he or she creates R programs in the test\testthat folder using file test-independent-testing-<filename>.R or test-double-programming-<filename>.R that matches the <filename> filename in the R folder, runs the test functions and verifies the results.

Following the test principle to test all different cases, all existing and updated test cases should pass with a build passing badge displayed. The validator needs to review the program code for effective and complete commenting, efficient programming practices and to assure the development guidelines were followed. The new or updated function and test program should pass all R package check lists with more than 80% (recommended) code coverage, with a coverage 80% badge displayed in the corresponding branch. The package's code coverage report can be generated by the "covr" package, and the code coverage report can be displayed on the package website, as shown in Figure 9. Then the functions and package can be automatically tested as needed after test cases are developed.

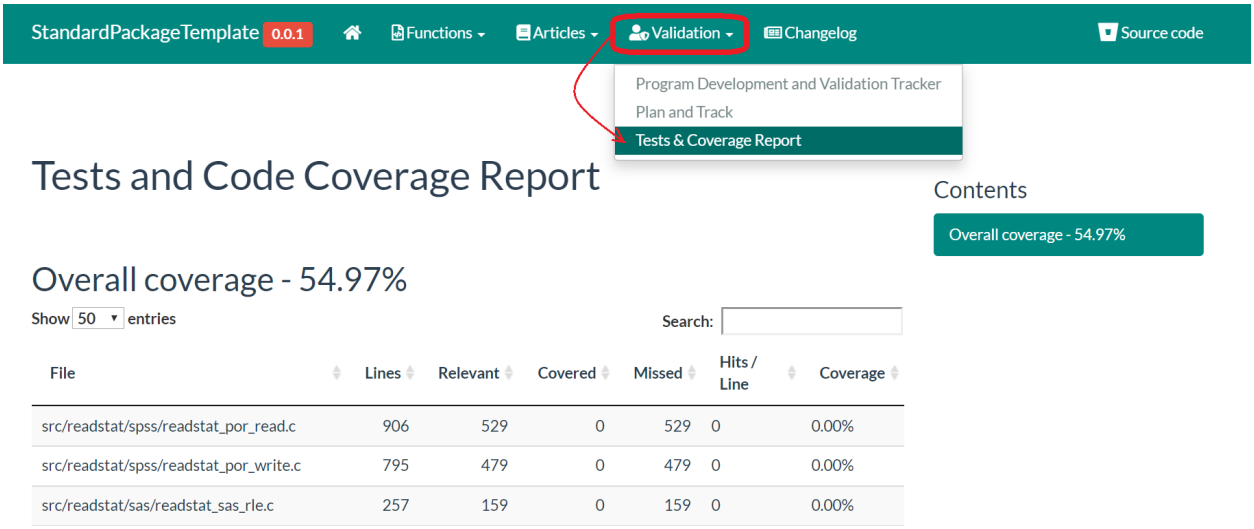


Figure 9. Code coverage report displayed on package website.

When closing out the validation stage, the independent tester and double programmer should update the R package validation plan to document all the test cases. If the user manual needs to be updated, they shall also review the user document to ensure it is clear and correct.

In general, customer review is encouraged for all new functions. Customer review is performed by a statistician (for functions implementing statistical methods) and business end users for functions not involving statistical methods.

## OPERATION

The operation stage ensures the validated R package is installed properly into the computing environment by the R administrator. Internal standard R packages should be installed in the organization's computing platform (such as RStudio Server) global library folders. The programmers do not require specs or any testing before using it. The R administrator should follow the organization's software release SOP. Before the package is moved to production, the SME should ensure the NEWS.md file is updated to document the main features added or updated, with a unique version number. Then it will be generated as "Changelog" within the package website, as shown in Figure 10.


Figure 10. Changelog addressing package feature implementation, displayed on the package website

If an R function or package needs to be deprecated (only function or argument level applied) or retired, the SME needs to add an entry in the latest announcement document located in the following folder, including: R function/argument name, date R function/argument will be deprecated or retired, reason for deprecating or retiring, workaround or alternate R function/argument.

After the deprecation is announced for a function/argument, the SME makes sure the deprecated function/argument can generate a warning message (i.e. “Warning: function/argument <fun> is deprecated; please use <subfun> instead.”). The SME should also update the R function in the master branch.

After the retirement is announced for an R package, the SME makes sure a zipped file of a repository clone exists and moves the retired R package repository to a specific retired package repository. After the retirement is announced for a function, the SME removes the R function in the master branch, initiates a commit to the master branch, and finally pushes the commit to the master branch on the version control platform.

## DISCUSSION AND CONCLUSION

In this paper, we introduce an R-SDLC process to guide internal standard R package development. The proposed guidance shall follow an organization’s internal SOP to ensure all development, testing and validation steps are designed for regulated clinical trial environment. Following the process, developed functions can be validated from different scenarios, and have clear specification and user-friendly documentation. Therefore, the statistical analysis using internal standard packages can be traceable and reproducible.

It is worth noting that a continuous integration or continuous deployment (CI/CD) technique is very helpful for automating and simplifying the R-SDLC process. After the define stage, the CI/CD can be initiated by the R administrator for automatic testing and deployment. Figure 11 shows an example of a CI/CD process with key components. As the figure shows, the CI/CD accompanies the development (“Develop” → “Build”), validation (“Test”) and operation (“Deploy” → “Use”) stages, and it can provide an end to end solution to be compliant with R-SDLC with help from the IT department.

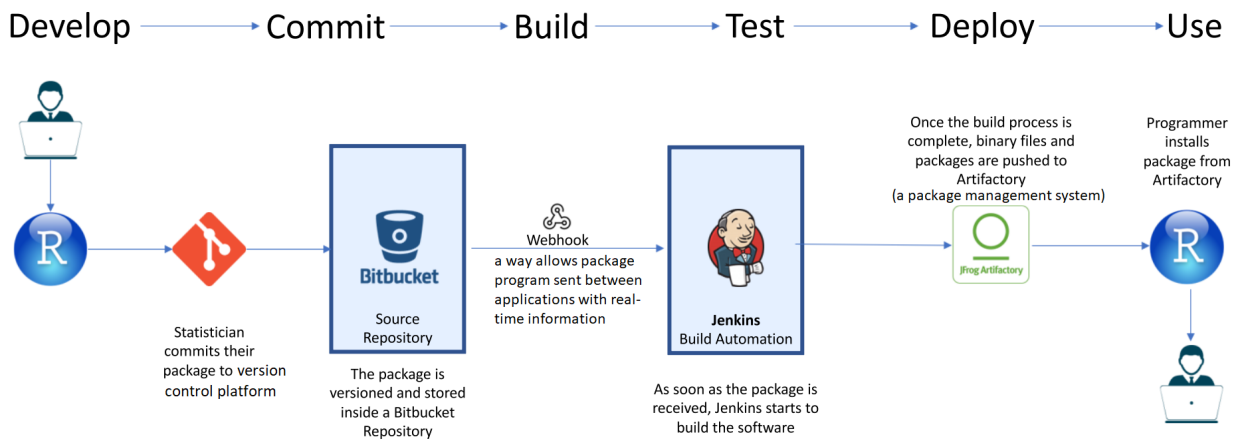


Figure 11. CI/CD Overview (Note: Bitbucket and Jenkins is just for illustrative purpose, other version control and CI/CD tools can be considered in the same framework).

In this paper, we propose a standard package template, which includes a required R package folder structure, and necessary customization satisfying the R-SDLC process. An integrated website can be built with several features such as an R-SDLC badge system, an HTML and PDF user manual, a vignettes article, a validation tracker, a code coverage report, a changelog, etc. It is recommended to use a web host server, such as RStudio connect, so that the package website can be deployed and available for developers and users within an organization.

## REFERENCES

The R Foundation for Statistical Computing (2018) R: Regulatory Compliance and Validation Issues A Guidance Document for the Use of R in Regulated Clinical Trial Environments
https://www.r-project.org/doc/R-FDA.pdf

Wickham, H. and Bryan, J. (2019). R packages: organize, test, document, and share your code. (2nd Ed.)
https://r-pkgs.org/

Bryan, J. and Hester, J. (2019) Happy Git and GitHub for the useR.
https://happygitwithr.com/

Wickham, H. (2019) The tidyverse style guide
http://style.tidyverse.org/

Tidyverse team. (2019) Tidyverse design guide
https://design.tidyverse.org/

## APPENDIX

A. Terminology used in this paper

| Term | Definition |
|---|---|
| Artifactory | Universal repository manager. R packages source code bundle can be hosted in Artifactory within an organization to consistently manage internal and external R packages. |
| Bitbucket | A web-based version control repository hosting service. An organization uses it for internal standard R packages development |
| CI/CD | Continuous integration / continuous deployment, a software development technique to ensure that all the components in a web service/server hosting software are working together harmoniously. |
| Code coverage | A metric to summary the percentage of the total line of source code is executed by testing or validation code. An organization uses it to record code coverage degree for functions of an R package. |
| Git | A version control system designed to handle everything from small to very large projects with speed and efficiency. An organization uses it for internal standard R package version control, create/edit/remove/merge branch, etc. |
| Branch | An independent working line of development for a repository, including project history. Before creating any new branches, developer automatically starts with the main branch (called “master branch”). |
| Jenkins | An automation server which enables developers within an organization to reliably build, test, and deploy their R packages. |
| Jira | A web-based tool to track issue and bugs related to software and mobile apps. It is also used for project management. An organization uses it to track internal standard R package issue and bugs, and initiate a branch for each R-SDLC stage. |
| Repository | A remote folder/directory to store an R package related files on version control platform. An organization uses it to remotely host and track a package code, metadata, configuration, etc. |
| User manual | A web-based (HTML) or PDF document attached with an R package, it documents package basic information, specification of functions (only applicable for PDF user manual), parameters of functions, parameters type, usage of functions and example code. |
| Vignette | A long-form web-based (HTML) or PDF document to provide a guidance of an R package. A vignette should divide functions into useful categories and demonstrate how to coordinate multiple functions to solve problems. |

B. Example Roxygen2 code to display Specification in PDF user manual only:

```
#' @section Specification:
#' \ifelse{latex}{
#'  \itemize{
#'   \item create \code{filename} function (describe your function's
specification)
#'  \itemize{
#'  \item define function name (filename) and purpose
#'  \item customize filename with corresponding input and output
#'  \item ink to corresponding hyperlink/url}}
#'
#'  The followings are some possible code: (optional)
#'  \itemize{
#'  \item \code{library(StandardPackageTemplate)}
#'  \item \code{filename("aaa")}
#'  \item \code{filename("bbb")}
#'  \item \code{filename("ccc")}
#'  }
#'
#' }{
#' The contents of this section are shown in PDF user manual only.
#' }
```


## ACKNOWLEGEMENTS

The authors would like to thank their management team for the support and review of this paper.


## CONTACT INFORMATION

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


Author Name: Yalin Zhu, Ph.D.
Company: Merck & Co., Inc.
Address: 126 E Lincoln Ave, Rahway, NJ 07065
Work Phone: 732-594-3192
Email: yalin.zhu@merck.com

Author Name: Yilong Zhang, Ph.D.
Company: Merck & Co., Inc.
Address: 126 E Lincoln Ave, Rahway, NJ 07065
Work Phone: 732-594-6196
Email: yilong.zhang@merck.com