

How CI/CD Enhances the Development of R Packages in the Pharmaverse

PhUSE 2023, US Connect

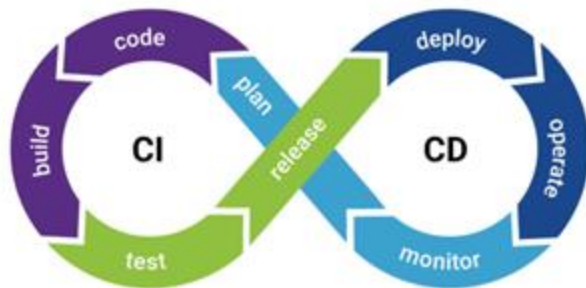
March 7th, 2023

Ben Straub (GSK) & Dinakar Kulkarni (Roche)

What is CI/CD?

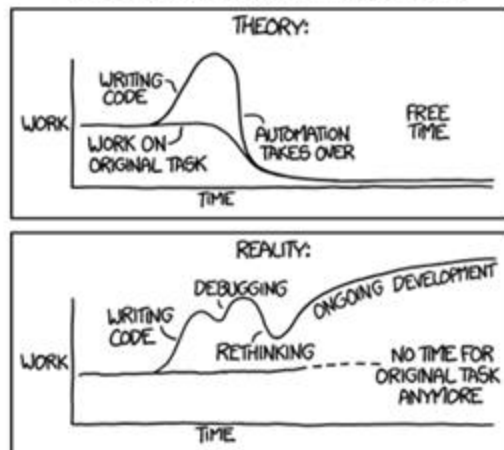
- Continuous Integration (CI): Frequent merging of several small changes into a main branch
- Continuous Delivery (CD): Repeatable deployment process when deciding to deploy

CI/CD bridges the gaps between development and operation activities and teams by **enforcing automation** in building, testing and deployment of applications. CI/CD services compile the incremental code changes made by developers, then link and package them into software deliverables



Does it help?

"I SPEND A LOT OF TIME ON THIS TASK.
I SHOULD WRITE A PROGRAM AUTOMATING IT!"



...Yes! Yes, it does!!

How does CI/CD help R packages?

- Catch issues (bugs) early on
- User base on multiple OSes and multiple R versions
- Faster turnaround on Code Review
- Multiple Contributors on your R Package
- Enforce style conventions and preferences
- Measure test coverage for new code
- Keep docs up-to-date
- And we can just keep going!

We covered a lot of custom CI/CD actions for R packages in the R/Pharma Workshop in 2022: [Intro to CI/CD for R Packages](#) 

Two Case Studies



Case Study - admiral

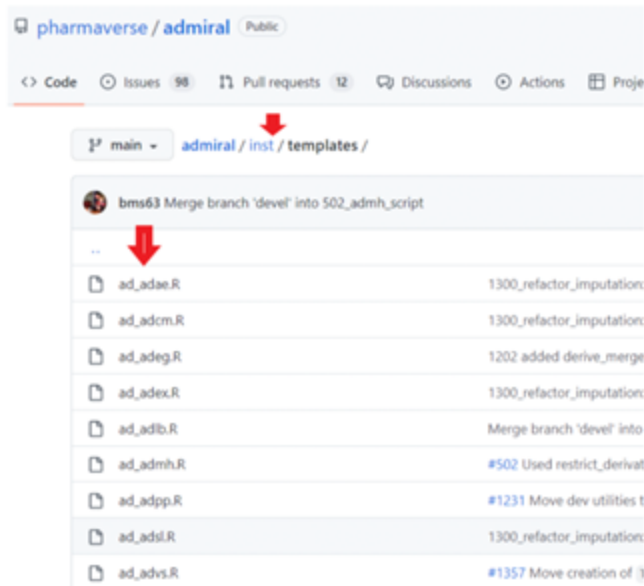
About admiral

- Provide an open source, modularized toolbox that enables the pharmaceutical programming community to develop ADaM datasets in R.
- ADaM is one of the required standards for data submission to FDA (U.S.) and PMDA (Japan) for clinical trials
- Links
 - [CDISC](#)
 - <https://github.com/pharmaverse/admiral>
- **Issue 1:** Checking ADaM Template code
- **Issue 2:** Common CI/CD workflows for the **admiral** family of packages



Issue 1 - How to Check our Template Code

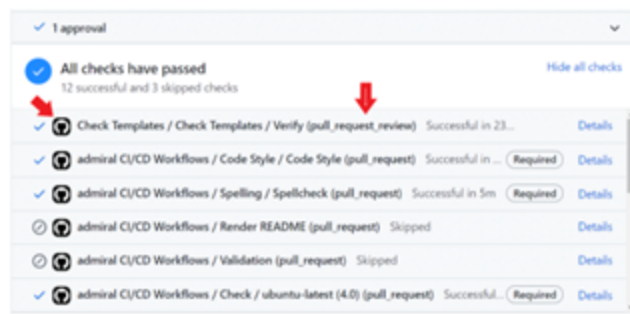
- Create a reference files to build common ADaM datasets that shows users how to implement our functions
- Way less text than a Vignette - Code is ready to go and build a dataset
- Where we store this code is not checked by R-CMD
- How to ensure code stays up to date with deprecated functions or unforeseen bugs get in from functions working together?
- CI/CD for the win!



Solution 1 - CI/CD for Templates

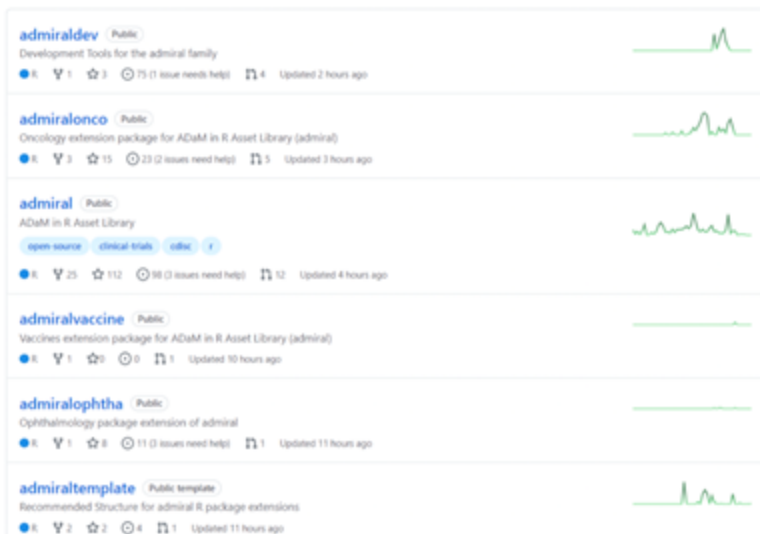
- Dedicated CI/CD workflow that executes the Template code
- Once a Code Review is completed the **Check Template** Workflow is executed
- If any errors or warnings are detected the CI/CD check fails and the contributor must fix the error or warning.

.github/workflows/check-templates.yml



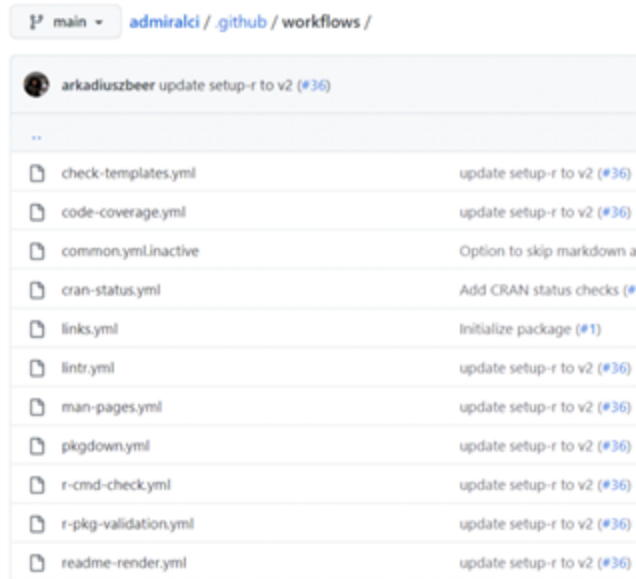
Issue 2 - admiral upstream and downstream dependencies

- As you can imagine there can be a lot of different types of ADaMs!
- Extension packages focus on specific disease areas like oncology
- The **admiral** family has a package for developers, template R package repo and dummy data
- Eek!! How to keep this all in line!



Solution 2 - Common CI/CD workflows for admiral upstream and downstream dependencies

- Using `{admiralci}`, we have a common set of CI/CD workflows
- Developers moving between packages are familiar with these workflows
- Common documentation between packages for CI/CD workflows - easy to maintain and provide to new contributors



The screenshot shows the GitHub interface for the `admiralci` repository, specifically the `workflows` directory. The breadcrumb navigation at the top reads `main > admiralci / github / workflows /`. Below this, the repository name `arkadiuszbeer` and the workflow name `update setup-r to v2 (#36)` are displayed. A list of workflow files is shown, each with a file icon, the filename, and a link to the workflow details.

	<code>check-templates.yml</code>	update setup-r to v2 (#36)
	<code>code-coverage.yml</code>	update setup-r to v2 (#36)
	<code>common.yml</code>	Inactive
	<code>cran-status.yml</code>	Add CRAN status checks (#
	<code>links.yml</code>	Initialize package (#1)
	<code>lintr.yml</code>	update setup-r to v2 (#36)
	<code>man-pages.yml</code>	update setup-r to v2 (#36)
	<code>pkgdown.yml</code>	update setup-r to v2 (#36)
	<code>r-cmd-check.yml</code>	update setup-r to v2 (#36)
	<code>r-pkg-validation.yml</code>	update setup-r to v2 (#36)
	<code>readme-render.yml</code>	update setup-r to v2 (#36)

Case Study - NEST

About NEST

- A collection of R packages for creating TLGs/TFLs and exploratory clinical trials data visualization
- **tern** and **rtables** for creating TLGs
- **teal** for creating exploratory web applications for analyzing clinical trial data
- Links
 - [rtables](#)
 - [tern](#)
 - [teal](#)



Use Case 1 - Integration Testing

- An in-development package must be tested against the latest versions of upstream dependencies
- Monorepo emulation via a git branch naming strategy is achieved by using
 - the [staged.dependencies R package](#) 
 - and the [staged.dependencies GitHub Action](#) 
- Testing as a cohort can be done at any stage (eg. development, pre-release, release)



Use Case 2 - Web Application Testing & Deployment

- Analysts create Shiny web apps via the **teal** framework for analyzing data
- Apps are tested via a CI pipeline that uses the **shinytest2** [R](#) package
- Apps deployed to an Posit Connect Server instance via a CD pipeline
 - With the help of the **rsconnect** [R](#) package and **connectapi** [R](#) packages

The screenshot displays the shinytest2 web interface. On the left, a preview of a Shiny app is shown with the text "What is your name?" and a text input field, followed by a "Greet" button. On the right, the "expectations" section for the test is configured. It includes two checkboxes: "Expect Shiny values" and "Expect screenshot", both of which are checked. Below these is a red error message: "At least one expectation must be made". The "Code" section contains a single line of R code: `1: appTest_window_size(width = 411, height = 749)`. At the bottom, there is a "Save" section with a "Test name" field containing "simple-app" and a "Random seed" field with the value "(None)". At the very bottom, there are two buttons: a red "Exit" button and a green "Save test and exit" button.

Use Case 3 - Validating R Packages

- R packages are validated by an internal validation team that uses CI/CD pipelines to automatically
 - accept new package submissions via a form
 - running tests against the new package to ensure package integrity
 - enforcing criteria to ensure that the package meets regulatory requirements
- Also validated externally via an open source project called [thevalidatorR](#) 

Validation Report

tsal.code (v0.2.0)

Server: <https://github.com> Repository: [insightsonengineering/tsal.code](#)

Reference: [refs/tags/v0.2.0](#)

Commit SHA: [6d2083bed8c6d4d2191b0ff477e40325e2de](#)

Fri Oct 14 01:33:23 AM 2022

Contents

1	Context	1
2	Installation environment and package	1
2.1	System Info	1
2.2	Package installed	2
2.3	R Session Info	2
3	Metric based risk assessment	3
4	Installation documentation	3
4.1	R CMD check	3
4.2	Testing Coverage	5
4.3	Traceability	5

1 Context

This report was generated via the GitHub action [insightsonengineering/validateR](#) (gh-action ID: [insightsonengineering_validateR](#)). It produces automated documentation of the installation of this package on an open source R environment, focusing on:

- Installation environment description
- Testing coverage
- Traceability matrix of specifications (documented behaviours) and testing
- Risk assessment benchmarks

This report is fully automated, as is limited to assess whether unit tests and documentation are present and can execute without error. An assessment would be required that the tests and documentation are meaningful. Validation aims to be system independent as the underlying workflow is based on the "composite" type of GitHub Action.

Additional Materials

- Further Reading
 - [GitHub Actions](#)
 - [GitLab CI](#)
- Advanced Examples
 - [r-lib/actions](#)
 - [{admiralci}](#)
 - [Docker](#)
- Presentation built with [Quarto](#)
- [R/Pharma 2022 CI/CD Workshop](#)
- [This Presentation](#)