PhUSE 2016

Paper AD12

# ValidR Enterprise: Developing an R Validation Framework

Andy Nicholls, Mango Solutions, Chippenham, UK

## ABSTRACT

In ValidR Mango have developed a build and test framework that allows for generating an R installation containing a set of candidate R packages for validation. This framework is ideal for a company where a single version of R is required. However, they may require multiple versions of R, with different sets of packages, for different groups. In addition, end users are increasingly developing their own R packages that they would like to be included in their validated environment for production usage. ValidR Enterprise addresses these points by supplying the build and test framework, but raises a series of new deployment challenges.

## INTRODUCTION

Mango has been supporting R since its inception in 2002. Today it is more than just a statistical tool and is being deployed far and wide for a variety of purposes. However, in the pharmaceutical industry SAS® remains the gold standard tool for statistical analysis and reporting. Part of the reason for this is the inconsistency in the standard of R's network of user-contributed packages that are generally accessed via the Central R Archive Network (CRAN). Coupled with varying release patterns for R packages, this presents a considerable challenge for any analysis team wishing to adopt R for use in a regulatory setting.

In 2014, in response to growing demand for R to be used in regulatory submissions, Mango created the first version of ValidR, a validated server installation of R to meet the strict regulations of the pharmaceutical industry. Subsequently, the challenge has been to find a balance between the need for a central, controlled, standardised environment and the traditional flexibility of the R language and its passionate user community. This challenge ultimately led to the creation of ValidR Enterprise, an R development and deployment platform, capable of producing both validated and non-validated R installations.

## WHAT IS VALIDATION?

In a pharmaceutical setting most companies base their validation efforts around the Food and Drug Administration's (FDA's) guidance document "Part 11: Electronic Records, Electronic Signatures" (Part 11 Electronic Records; Electronic Signatures, Revised April, 2015). The document was originally issued by the FDA in 1997 to provide regulatory requirements which act as recommendations for the processes and controls associated with electronic records and electronic signatures. However, despite the focus on records and signatures many of the themes discussed within the document are more widely applicable to general software development within the pharmaceutical industry. It is not the purpose of this paper to fully document a response to 21 CFR Part 11; however, it is worth noting that as part of the process of validating R Mango produced a complete response to the guideline.

Mango Solutions' adopts the following definition of validation, as published by the FDA (Food and Drug Administration): "Establishing documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its predetermined specifications and quality attributes" (FDA, August 1995). Further details and definitions are provided in documents such as "General Principles of Software Validation; Final Guidance for Industry and FDA Staff" (FDA, January, 2002) and "Guideline on General Principles of Process Validation" (FDA, May, 1987). These documents outline the FDA's interpretation of validation, and in particular highlight the differences between software validation and verification; the latter term typically refers specifically to the tests that are written in order to help validate the software. Also discussed are IQ/OQ/PQ activities, otherwise known as Installation Qualification, Operational Qualification and Performance Qualification respectively. These activities are explained in later sections of this paper.

### THE FDA'S DEFINITION OF VALIDATION

The core FDA statement upon which Mango bases its approach to software validation can be broken down as follows:

1. Establishing documented evidence
2. Providing a high degree of assurance
3. A specific process will consistently produce a product

4. Meeting predetermined specifications and quality attributes

In a practical sense this means that when building software, one must first start by defining and documenting the intended purpose of the software. Software is then designed, developed and tested against these original requirements. If further documented evidence of the software meeting these requirements can be provided, then it can be considered to be validated. It is actually a fairly straightforward process, however contention tends to lie around the second point, a "high degree of assurance". To provide this assurance the software must be tested. This is often referred to as "software verification". However, as the FDA state, "a developer cannot test forever, and it is hard to know how much evidence is enough" (FDA, January, 2002). Needless to say, software verification/testing is a key component of the validation process and is discussed later in this paper. As part of the software validation process software must be tested and retested for consistency.

#### IQ, OQ, PQ

The IQ/OQ/PQ activities are a key component of software validation. Each centres on establishing confidence that the software does what it is supposed to do. The aim of the Installation Qualification is to ensure that all components of the software are installed correctly. The Operational Qualification ensures that the software then functions as expected. The Performance Qualification seeks to ensure that the process is reproducible and performant. Software verification, whereby tests are written to ensure that the software produces the expected results, usually sits within the remit of Operational Qualification.

## VALIDATING R

The core (a.k.a. base) R product is constructed using software development best practices and the R Foundation have published their own response to the use of R as a validated tool, "R: Regulatory Compliance and Validation Issues, A Guidance Document for the Use of R in Regulated Clinical Trial Environments" (The R Foundation, December, 2014). However, the document relates to Base R and Recommended packages only. In other words, the document covers around 30 R packages, depending on the version of R. At the time of writing there are over 9,000 additional packages available from CRAN alone and there are no such guarantees that these packages meet their predetermined specifications.

Most public R packages are available via CRAN although many are only available via BioConductor, GitHub, or sometimes R-Forge. CRAN enforces a minimum set of standards via mechanisms such as the "R CMD check". These checks essentially ensure that a package builds "correctly" on various operating systems and that any dependencies on other R packages or external tools have been correctly specified. However, there are no checks to ensure that code that has been uploaded to CRAN does what it claims to do. Additionally, if a package author writes unit tests and stores them in the appropriate directory, then these are executed as part of the CRAN checks and must pass for the package to be added to the repository. Any examples in the package documentation must also run successfully. But there is no requirement for an author to actually write tests or examples. At the time of writing less than 17% of the packages on CRAN use one of the three recognised unit testing frameworks in R: testthat, RUnit and svUnit. Even if one of these frameworks is used there is no guarantee that the level of testing sufficiently covers the package risk or that the tests are even appropriate in the first place.

When there is such a low level of formal verification of R packages, one would be hard pressed to claim that there is a *high degree of assurance* that any particular combination of R packages meets its predetermined specifications and quality attributes. ValidR, and the associated package validation process aims to provide the necessary assurance.

## VALIDR PROCESS

An overview of the process is provided in the following subsections. Each element had a significant bearing on the design of ValidR Enterprise. The process was developed by Mango to ensure that each test could be traced back to a package requirement and that the tests developed are appropriate given the package complexity and risk. It should be noted that the overview here focusses on the technical aspects of the process in relation to ValidR and ValidR Enterprise. Mango has also developed guidance with respect to the review and approval of the validation stages but this is not covered here.

#### GATHERING REQUIREMENTS

Normally, the first step in building a validated piece of software is to define its requirements. This presents quite a challenge to anyone wishing to validate R and/or R packages since they have already been written! Package content and purpose can vary greatly and it takes a large amount of expertise, and patience, to be able to adequately define the scope and purpose of an unknown R package.

A package DESCRIPTION file provides some high level requirements for a package, but this must be broken down into testable chunks. The better R packages tend to provide further supporting information in the form of a package

vignette, website or blog posts. Once the package functionality is understood the requirements should be formally defined and prioritised. Mango implements the MoSCoW methodology of requirement prioritisation when building software and this same methodology is used to prioritise R package requirements.

In order to ensure that the process is fully documented the requirements themselves must then be stored centrally. This could be in a document or spreadsheet, or they could be captured via a bug tracking or project management tool such as JIRA or Mantis. In the first two iterations of ValidR Mango used JIRA to capture requirements. In the most recent release of ValidR a custom-built Shiny application was developed for this purpose. The important thing is that requirements are captured, stored and versioned. If capturing requirements in a document or spreadsheet, then a tool such as SharePoint or Mango's ModSpace repository would suffice.

### RISK ASSESSMENT

Many of the more popular R packages have been widely tested by an active and vocal R community who will connect directly with a package maintainer to discuss bugs, bug fixes and missing functionality. Clearly a package that has been produced by a commercial company, has many unit tests, has been available for several years, has undergone many updates and releases and that has been downloaded many times, presents less risk than a brand new package produced by an unknown author. These considerations, and many more, help a package assessor to assign a level of risk. Later in the validation process, this risk is used to determine an appropriate level of "remediation", in other words the amount of testing that is required to be comfortable that the package is functioning as expected.

For each of the packages that Mango validates, metrics are collected via a custom built R package called packageMetrics. The packageMetrics package includes a primary function which, when supplied with the name of an R package, collects information about that package from the freely available RStudio® download logs. These logs correspond to the default CRAN mirror in the hugely popular RStudio IDE and are hence considered to be a very representative sample of the R community. Table 1 provides an example of some metrics that were gathered for the dplyr package for ValidR 3.2.5.

*Table 1 Package Metrics for the dplyr Package, ValidR 3.2.5*

| **Description** | **Guideline Risk** | **Result** |
|---|---|---|
| Number of times = is used for assignment | Low | 3 |
| Author Test Coverage | Low | 59.48 |
| Cyclomatic Complexity | Low | 1.91 |
| Num of Dependencies | Medium | 18 |
| Number of Reverse-Dependencies | Low | 277 |
| Number of Downloads | Low | 1412359 |
| Average number of code lines per function | Low | 7.16 |
| Date of First Release | Low | 2014-01-16 |
| Number of library and require calls | High | 17 |
| Number of code lines longer than 80 characters | Medium | 121 |
| Number of lines of compiled code | Low | 4515 |
| Number of lines of R code | Medium | 9016 |
| Date of Last Release | Low | 2015-09-01 |
| Number of attach and detach calls | Low | 0 |
| Number of times T/F is used instead of TRUE/FALSE | Low | 0 |
| Updates During the Last 6 Months | Medium | 0 |
| Whether the package is on GitHub | Low | 1 |
| Number of sapply calls | Medium | 7 |
| Number of trailing semicolons in the code | Low | 2 |
| 1:length(vec) is avoided | Low | 0 |
| Number of setwd calls | Low | 0 |
| Number of vignettes | Low | 8 |

High level package information including the package metrics shown in Table 1 are presented to package assessors via a Shiny application. This application, from here on in referred to as the Validation Application, provides links to relevant resources to enable the assessor to gather any additional information required when assessing an R package. A screenshot of the Validation Application can be seen in Figure 1. Once the assessor has gathered sufficient information about a package he or she writes up their assessment within the application and clicks a button to signify that they have completed their review. This unlocks a review tab within which a reviewer can make their own comments and log that their review is complete.

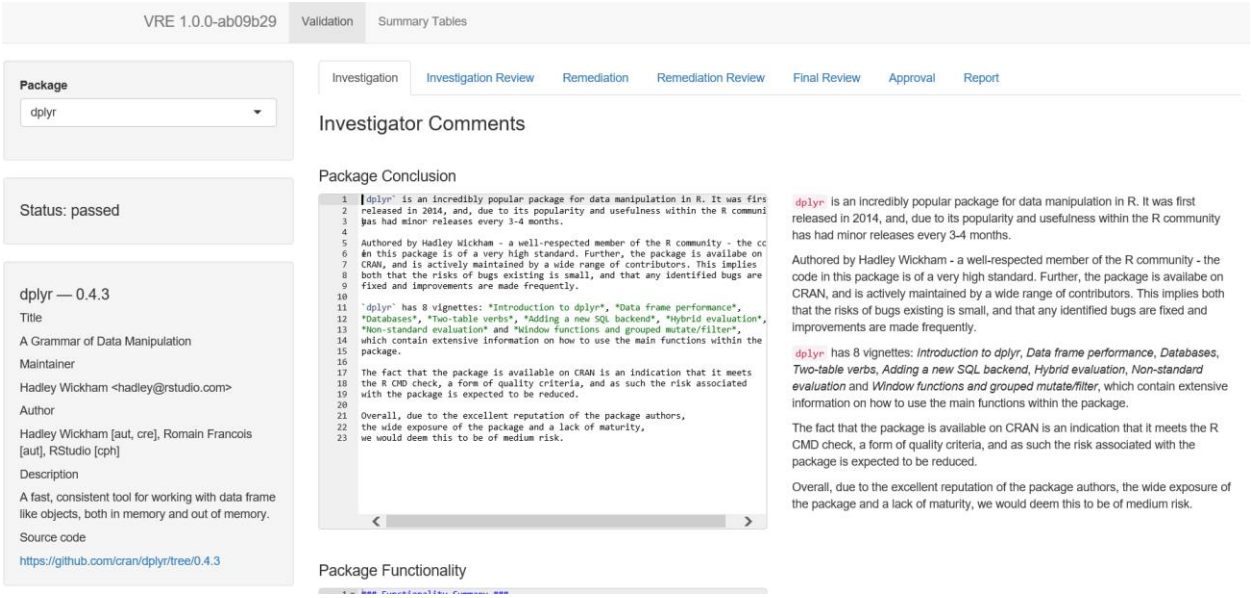



*Figure 1 Mango Solutions' Validation Application*

Tests are developed in the remediation phases of the process. Summaries of the test results are presented back to the user once this phase has been completed. Consultants reviewing the whole process can follow the complete validation journey via the application. The application itself has also been validated having been developed using the same best practices discussed in this paper.

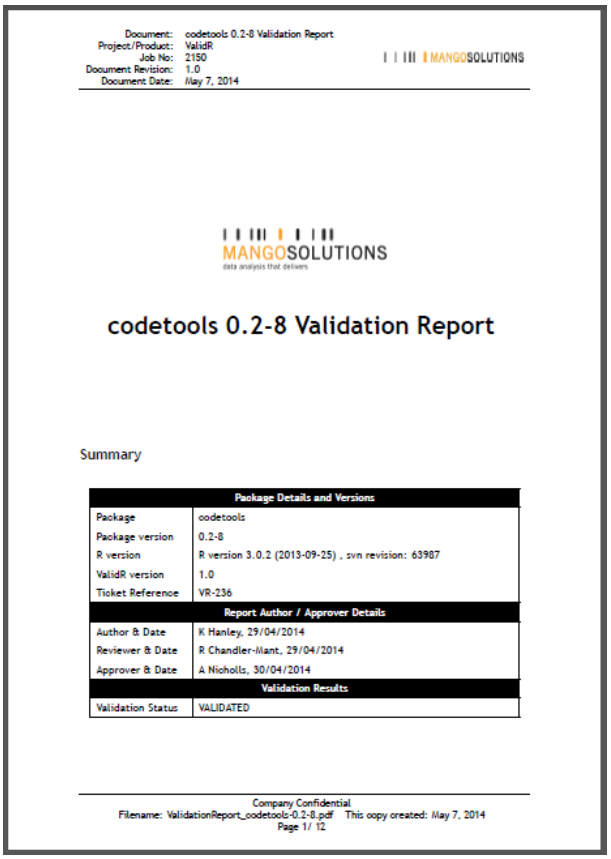
## DOCUMENTATION

In order to provide the necessary documented evidence of the factors considered in a risk assessment, a report is generated. In previous iterations of ValidR, this was a semi-manual process. The package build process collected package metrics and other package information and generated a LaTeX document from a ".Rnw" file via R's Sweave function. For ValidR 3.2.5 this process has been fully automated and the reports are now generated directly from the Validation Application. An example of a report from ValidR 1.0 is shown in Figure 2.

## TESTING

The key to ensuring that a package behaves as expected is to write unit tests that can be executed in any environment to ensure consistency of the code base. The level of testing required in order to provide the necessary assurances around the package functionality is determined by the package requirements and in the risk assessment.

In order to ensure that the packages themselves remain untouched, Mango developed a test framework in which each R package in the installation is associated with a test or "validation" package containing additional tests. These packages follow the naming conventions of the packages that they are testing with the addition of a "validation_" prefix. Later, upon installation, these package pairs are loaded and the tests within the validation package are automatically executed via a test harness.



Document: codetools 0.2-8 Validation Report
Project/Product: ValidR
Job No: 2150
Document Revision: 1.0
Document Date: May 7, 2014

MANGOSOLUTIONS
data analysis that delivers

codetools 0.2-8 Validation Report

Summary

| Package Details and Versions | |
|---|---|
| Package | codetools |
| Package version | 0.2-8 |
| R version | R version 3.0.2 (2013-09-25) , svn revision: 63987 |
| ValidR version | 1.0 |
| Ticket Reference | VR-236 |
| **Report Author / Approver Details** | |
| Author & Date | K Hanley, 29/04/2014 |
| Reviewer & Date | R Chandler-Mant, 29/04/2014 |
| Approver & Date | A Nicholls, 30/04/2014 |
| **Validation Results** | |
| Validation Status | VALIDATED |

Company Confidential
Filename: ValidationReport_codetools-0.2-8.pdf This copy created: May 7, 2014
Page 1/ 12

*Figure 2 Front page of a validation report from ValidR 1.0*

Package tests are written using the "testthat" framework for R package testing. The following example demonstrates the console output for a passing and for a failing test.

```
> library(testthat)
> myPi <- 3.14159
>
> # Passing test
> expect_equal(myPi, pi, tolerance = 0.00001)
>
> # Failing test
> expect_equal(myPi, pi)
Error: `myPi` not equal to lazyLoadDBfetch(c(483539L, 39L), datafile, compressed,
envhook).
1/1 mismatches
[1] 3.14 - 3.14 == -2.65e-06
```

In addition to providing a number of useful utility functions to check that the output is as expected, the package provides a useful test framework from which tests can be linked back to package requirements. In the current ValidR iteration package requirements are captured in the Validation Application. Once these requirements have been approved, a script is activated which generates template test scripts and tags the scripts (using the "context" feature of testthat) with a unique requirement in order to provide traceability of tests. Later in the process the same mechanism is used to construct the ValidR traceability matrix.

The following example is taken from a test script for the digest package. The text within the 'context' function call relates to a high level package requirement, while the text within the 'test_that' call provides lower level information about the test.

```
context('[critical]VR-480 digest: Must create hash function digests of R objects')
```

```
library(digest)

tmp <- digest(list(LETTERS, data.frame(a=letters[1:5], b=matrix(1:10,ncol=2))))

test_that('Can create hash for list object', {
  expect_identical(tmp,"8b23d6c96da88f525c0a4a4909cce01f")
})

# etc ...
```

In order to provide traceability and keep track of the test development Mango stores all packages and their corresponding “validation_” counterparts in a version controlled repository, currently a centralised Subversion® repository. The tests developed in this phase of the process are later executed as part of the Operational and Performance Qualification activities.

## VALIDR ENTERPRISE

The output of the validation process is a number of unit tests stored in validation packages. ValidR Enterprise is the name given to the infrastructure that stores, maintains and deploys ValidR builds. In order to ensure that a ValidR build is constructed correctly, a system of RPMs is used (the MSI mechanism is used for Windows). Each RPM contains information about the package’s dependencies (both internal and external to R). The RPM mechanism itself controls the installation order and ensures that dependencies are resolved. A ValidR release is essentially a collection of RPMs that are collated using a master RPM.

The generation of package RPMs is orchestrated by a Continuous Integration Server, Jenkins. Mango created two primary Jenkins “jobs” to facilitate the addition of new packages to the system and the deployment of ValidR builds. An overview of the complete system can be seen in Figure 3. Further details are provided in the following subsections.

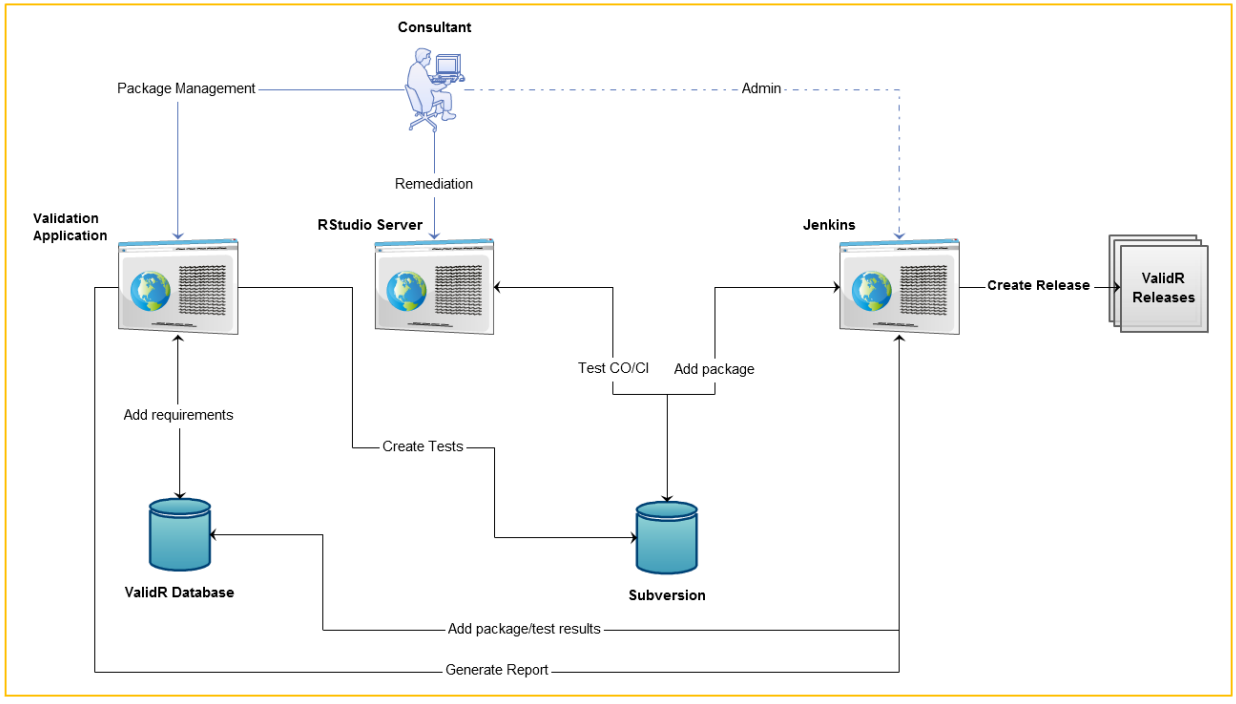



*Figure 3 ValidR 3.2.5 Architecture Diagram*

### ADDING PACKAGES

In order to add new packages to the system the administrator simply provides a comma separated list of package names to a pre-existing Jenkins job and sets this running. The purpose of the job is to spawn a unique Jenkins job for each package that in turn manages the series of activities that enable the validation and eventual deployment of

the package within an R build. At a high level these activities are:

1. Retrieve the package from the appropriate location and create a directory in the Subversion repository to store package source
2. Run packageMetrics and store the results in the ValidR database
3. Expose the package to consultants via an RStudio Server instance

RESOLVING DEPENDENCIES

When a new package is requested the Jenkins job searches the CRAN and BioConductor repositories for the package. The continuous evolution of these repositories initially presented a technical challenge to ensure that all packages included in ValidR are compatible with each other. At any point in time all packages available from CRAN must be compatible with each other, in other words all dependencies must resolve so that if "PackageA" requires a particular version of "PackageB" then that version is also on CRAN. However, a package author can update their package at any time and, as shown Figure 4, next week's version of PackageB might not be compatible with today's version of PackageA.

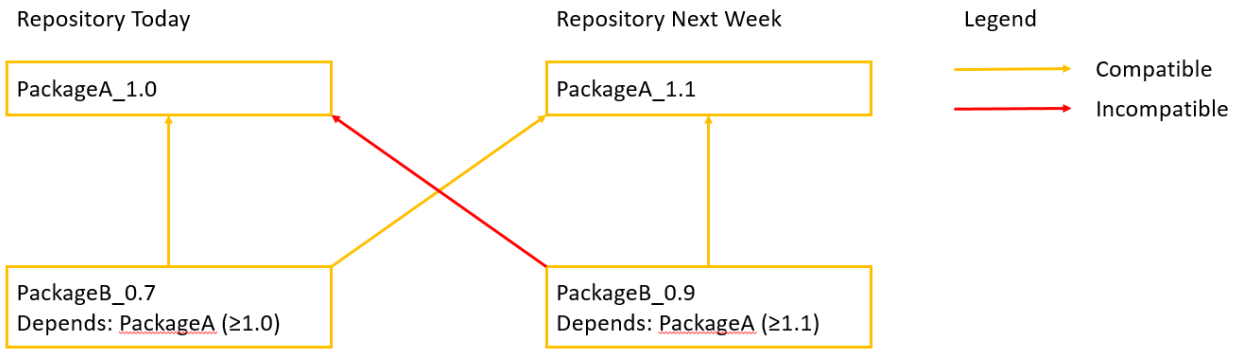



*Figure 4 Example of package dependency complications*

Compatibility is further complicated by the BioConductor repository which follows its own unique release cycle. It is therefore vitally important that a snapshot of the CRAN and BioConductor repositories is taken and that these snapshots are aligned. Today Mango takes snapshots using Microsoft's MRAN tool (see Figure 5). MRAN collects daily snapshots of the CRAN repository dating back to September 2014. The specific CRAN snapshot is chosen to match a particular BioConductor release, ensure complete compatibility between the two repositories.


*Figure 5 An MRAN Snapshot - https://mran.microsoft.com/timemachine/*

If the package is not available on either of these repositories, then a URL and specific version number can be provided to the Jenkins build to ensure that the correct version is added. However, it is then the responsibility of the administrator to resolve any dependency issues.

Packages are stored in a Subversion repository using a customised storage structure. This structure allows for validation packages, containing the ValidR test suite for the package, to be stored alongside the primary package of interest. The Jenkins job responsible for creating a package RPM understands this structure and bundles up the original and validation packages accordingly.

### VALIDR DATABASE

The ValidR database is not available in today's ValidR Enterpise product release, but the intention is to expose it for future releases. Once the source code for the package has been successfully located the packageMetrics R package automatically collates the relevant meta-information for the package, as described in a previous section. This information is then stored in a database which the Validation Application exposes to the appropriate consultant. Additional information resulting from the validation process, such as the assessor's conclusion on package risk, is later added to the database.

### RSTUDIO SERVER

One of the primary aims of ValidR is to ensure that everyone is using the exact same combination of packages in production/submission work. In developing ValidR Mango were faced with the same challenge. A single RStudio Server instance is used for the development of unit tests. This ensures that, even whilst packages are still being added to the ValidR remit, all consultants are writing tests against the exact same versions of R packages. Version control features within RStudio facilitate the commits of new tests to the Subversion repository that underpins ValidR.

### CONTINUOUS INTEGRATION

In addition to performing the set-up activities that are triggered when a new package is requested, Jenkins continuously polls the Subversion Repository to detect changes. Once package remediation has begun and tests start being committed to the repository, Jenkins finds these tests and builds the relevant RPM containing the original package source and the corresponding validation package. These RPMs are deployed nightly to a test server to ensure that the end-to-end process is functioning as expected.

### DEPLOYMENT

Mango created a specific Jenkins deployment job in order to create new ValidR deployments. Within the configuration for the build the administrator can either specify individual R packages or recognised package "modules" containing collections of R packages. A new deployment job is then created automatically which retrieves the necessary RPMs for the deployment and builds the master RPM. The master RPM is then executed nightly as part of the continuous integration described in the previous section.

### DELIVERY

Once a package has been assessed and been through the subsequent remediation and review phases, some might consider the package to have been validated. To complete the process however, the package must be installed on the target machine, the tests must be executed and the whole process documented. In other words, the build cannot be considered a validated build until it has successfully passed through IQ, OQ and PQ phases.

The IQ phase begins by triggering the master RPM. This in turn executes the package RPMs. Once the installation is complete it is automatically checked to ensure that everything that *should* be installed is installed in the correct location and that there are no unexpected side-effects of the installation. For example, it checks that no unexpected files have been created. The installation process is then automatically documented in an IQ report containing an overall PASS/FAIL for the stage and some high level details of the installation verification.

Following the completion of the IQ phase, the ValidR build script proceeds to the OQ phase. During this phase all of the package validation tests are executed. As with the IQ, an OQ document is automatically generated upon completion of the process. The final document contains an overall PASS/FAIL, test summaries for all packages, details of any test failures, and a full appendix of test results.

The final delivery phase is the PQ. The tests that form the OQ are executed a second time to ensure that the results are consistent and that the execution is timely. Once again the PQ report is generated automatically.

## VALIDR ENTERPRISE AS A PRODUCT

The primary aim of ValidR is to provide a stable R environment to ensure consistency and reproducibility and to meet regulatory requirements. There were three additional drivers behind the ValidR Enterprise product:

1. The need for a company to develop their own internal R packages
2. The ever evolving R landscape and the need to add additional R packages
3. A need to create custom R builds for different departments within an organisation

The first of the three drivers is actually now possible via ValidR. The modular approach introduced in version 3.2.5 enables additional packages to be added to an existing build, however the client would still be reliant upon Mango to add the packages. The second and third drivers tend to go hand in hand. The majority of R users within an organisation should be content with an annual R build containing the latest upgrades and improvements that the R

community has to offer. However, those responsible for driving innovation often require a separate environment within which to develop and test new R packages. Typically, this build contains a mixture of validated and non-validated R packages and resides on its own independent server.

### ENTERPRISE ARCHITECTURE

In order to meet the drivers outlined in the introduction to this section Mango's internal validation framework was engineered such that the key components could be decoupled from Mango systems and delivered as a standalone system. Engineering such a system was not straightforward, but as can be seen in Figure 6 the ValidR Enterprise product contains all of the necessary components for customers to select packages and create their own custom R builds. The key components that enable an R build to be created and deployed were considered to be:

1. Package upload mechanism
2. Dependency checking
3. Storage of package source code
4. Automated build and deployment process, including IQ, OQ and PQ report generation

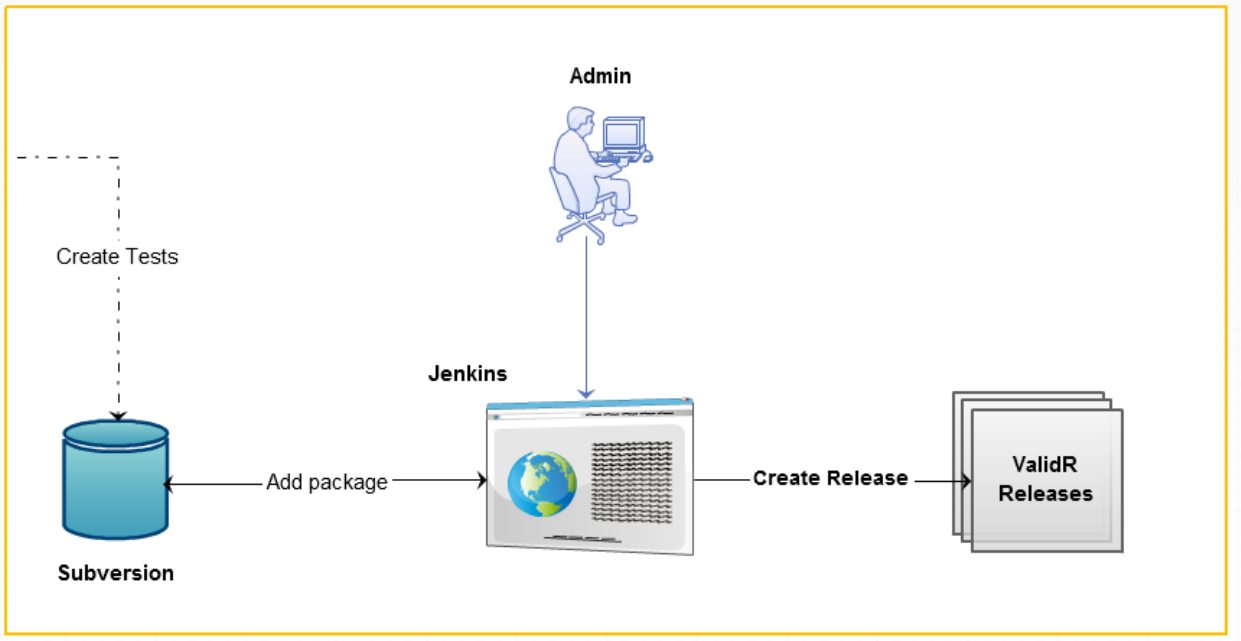



*Figure 6 The ValidR Enterprise Product Architecture*

The R builds created by ValidR Enterprise may be "validated" R builds (i.e. consisting only of validated R packages) or non-validated, experimental builds. This enables customers to create custom R builds to suit individual team, or even user, needs.

In addition to the build framework, ValidR Enterprise also provides the means for customers to validate packages themselves. For the build to be considered a validated build they simply need to upload the associated tests to the Subversion repository and the build system does the rest.

### A VALID SESSION?

Like R itself ValidR can be accessed via the command line, as part of a wider analytical system or interactively via RStudio Server. For each use case Mango has constructed a Session Tracking Application based on R's own "sessionInfo" function. The application tracks the packages that are loaded within the session and determines whether they are validated R packages or not. This information is stored in a log file, along with other environment information such as the date, username and working directory.

In addition to providing traceability with respect to R packages used within the session, the metrics collected by the Session Tracking Application are collated by Mango on a periodic basis and are then fed into the product roadmap. This helps ensure that the set of validated R packages remains relevant to different user groups around the world.

## FUTURE ENHANCEMENTS

The current version of ValidR Enterprise provides both Mango and Mango's customers with the means of creating bespoke R installations to meet the varying needs of the end-user. In order to continue to meet the needs of both the end-user and the IT teams that support them, the product must continue to evolve. On the immediate roadmap for the product are an enhanced Session Tracker and a product expansion to include the Validation Application.

### VALIDR SESSION TRACKER

Mango is currently developing the logging mechanism that underpins the Session Tracking Application. The aim is to combine information from the tracker with standard server usage metrics in order to provide IT teams with more detailed usage metrics. This will be particularly useful for those using ValidR interactively as the Session Tracking Application must currently be started manually. An IT team will thus be able to use the enhanced application to determine whether a particular user initiated the application or not. Further, since this information will be live, the application could potentially be configured to email or otherwise alert the user to the fact that they have not initiated the tracker.

### VALIDATION APPLICATION

As part of a major revision to the validation process used for ValidR 3.2.5, Mango developed a Shiny application to assist with the creation of validation reports. At the time of writing the intention is to develop the application further and include it in future ValidR Enterprise releases. In order to facilitate deployment and to ensure that the application is enterprise-ready, third party tools such as LaTeX and JIRA must first be decoupled from the application so that it may be used as a stand-alone tool. In the case of LaTeX, rmarkdown presents a viable, if slightly less flexible, alternative.

In ValidR today JIRA is used as a means of capturing the key decisions on the project. It is thus the primary tool that Mango uses in audits to demonstrate the traceability of decision-making. However, JIRA is a commercial product that Mango customers may not wish to purchase. The aim is therefore to replicate features such as the JIRA comment facility within the ValidR application.

### ADDING PACKAGES AND CREATING BUILDS

In the current iteration of ValidR Enterprise, in order to add a package or create a build an administrator must add a list of packages as a comma separated list. Similarly, when creating builds, the administrator must manually provide the required packages as a comma separated list. Today these features are only used periodically by an administrator. However, Mango has spoken with groups that would prefer to be able to self-serve. Rather than expose the full capabilities of a tool like Jenkins to a much wider group of users, the idea of building specialist applications that sit on top of Jenkins is currently being discussed. In particular, for the creation of custom builds a user would be able to view all available packages, split by whether they have been through a validation process or not, and pick and choose packages or bundles of packages for their build.

## A GOOD NEWS STORY

In addition to the tools discussed in this paper, ValidR has spawned many further subprojects which are now available as open source R packages. These include "visualTest", an R package to facilitate the comparison of graphics files; "testCoverage" for assessing how much of a package code base is covered by unit tests (subsequently superseded by "covr"); and latterly the "goodpractice" package which seeks to quantify the quality of an R package. In addition, where the validation process has identified bugs in packages, these have been flagged to the author, and in some cases Mango has even provided the necessary fix. In the case of the "SASxport" package, Mango eventually took ownership of the package and made it available to the community after an issue with the package caused it to be dropped from CRAN.

## CONCLUSION

The validation of an open source language like R is not straightforward. It has taken three major iterations and a large amount of effort on the part of Mango Solutions' R Consultancy team to get ValidR and ValidR Enterprise to where they are today. ValidR Enterprise takes ValidR to the next level by offering a validated development environment for R that can be used to generate validated or non-validated R distributions within an organisation.

## REFERENCES

**FDA. January, 2002.** *General Principles of Software Validation; Final Guidance for Industry and FDA Staff.* January, 2002.

**—. August 1995.** *Glossary of Computerized System and Software Development Terminology.* August 1995.

**—. May, 1987.** *Guideline on General Principles of Process Validation.* May, 1987.

*Part 11 Electronic Records; Electronic Signatures.* **FDA. Revised April, 2015.** Code of Federal Regulations, Revised

April, 2015, Vols. Title 21, Volume 1.
**The R Foundation. December, 2014.** *R: Regulatory Compliance and Validation Issues, A Guidance Document for the Use of R in Regulated Clinical Trial Environments.* December, 2014.

## CONTACT INFORMATION

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

Andy Nicholls
Mango Solutions
2 Methuen Park
Chippenham
Wiltshire
SN15 1AH
Work Phone: 01249 705450
Email: anicholls@mango-solutions.com
Web: www.mango-solutions.com
Twitter: @MangoTheCat