Easier Unit Tests and Better Examples with exampletestr and covr

by Rory Nolan and Sergi Padilla-Parra

Abstract In spite of the utility of unit tests, most R package developers do not write them. **exampletestr** makes it easier to *start* writing unit tests by creating shells/skeletons of unit tests based on the examples in the user's package documentation. When completed, these unit tests test whether said examples run *correctly*. By combining the functionality of **exampletestr** with that of **covr**, having ensured that their examples adequately demonstrate the features of their package, the developer can have much of the work of constructing a comprehensive set of unit tests done for them.

A survey of unit testing

Let us take a look at how many packages on CRAN employ unit tests by checking whether their source code contains a non-empty 'tests/' directory (using `https://rdrr.io`). We examine first all packages and then the subset that were authored or updated in 2016 or 2017. We will also compare the popularity of the testing frameworks **testthat** (Wickham, 2011), **RUnit** (Burger et al., 2015), **svUnit** (Grosjean, 2017) and **unitizer** (Gaslam, 2017) [code adapted from Nicholls (2015)].

```
required_pkgs <- c("tidyverse", "stringr", "lubridate", "rvest", "RCurl")
lapply(required_pkgs, library, character.only = TRUE)
check_for_tests <- function(pkg_name) {
  rdrr_address <- paste0("https://rdrr.io/cran/", pkg_name)
  if (url.exists(rdrr_address)) {
    rdrr_page_text <- rdrr_address %>% read_html %>% html_text
    trimmed_lines <- str_split(rdrr_page_text, "\n") %>% unlist %>% str_trim()
    trimmed_lines %>% str_detect("^tests/.+") %>% any
  } else {
    NA  # allow for not found
  }
}

download.file("http://cran.R-project.org/web/packages/packages.rds",
              "packages.rds", mode = "wb")
cran_packages_info <- readRDS("packages.rds")
cran_packages <- cran_packages_info[, "Package"]
post2015 <- ymd(cran_packages_info[, "Published"]) >= ymd("2016-01-01")
new_packages <- cran_packages[post2015]

tested_check <- cran_packages %>%
map_lgl(check_for_tests) %>%
set_names(cran_packages) %>%
na.omit  # allow for not found on rdrr
length(tested_check)
#> [1] 10464
sum(tested_check)
#> [1] 2622
new_packages <- intersect(new_packages,
                          names(tested_check))  # allow for not found on rdrr
length(new_packages)
#> [1] 6013
sum(tested_check[new_packages])
#> [1] 2024

unit_test_packages <- c("testthat", "RUnit", "svUnit", "unitizer")
reverse_deps <- tools:::package_dependencies(packages = unit_test_packages,
                          cran_packages_info, recursive=FALSE, reverse=TRUE,
                          which = c("Depends","Imports","LinkingTo", "Suggests"))
map_int(reverse_deps, length)
#> testthat    RUnit   svUnit unitizer
#>     1938      133       11        0
```

So we see that (at the time of writing) of the 10464 packages on CRAN found by https://rdrr.io, 25% (2622) are unit tested. Of the 6013 authored or updated since 1st January 2016, 34% (2024) are unit tested. We also see that **testthat** is the most popular testing framework, preferred in 93% of cases. Although these statistics suggest that developers' propensity to write tests is improving, it is still low. Perhaps this is because the task of unit testing an entire package from scratch can seem daunting. **exampletestr** makes it easier, providing a template of tests to be filled in. Those beginning to unit test no longer need to start with a blank page.

Unit test shells based on documentation examples

Most packages *do* contain examples in their documentation (indeed Bioconductor (Huber et al., 2015) requires documented functions to have at least one example). 'R CMD check' checks that these examples run without error, but cannot verify that they run *as intended*. Most package developers run their examples interactively at the command prompt to verify that they run correctly; this clearly suggests a corresponding unit test that automatically performs the same check. The **exampletestr** package (Nolan, 2017) does much of the work of writing *these* tests. It uses the **testthat** testing framework.

The best way to display the functionality of **exampletestr** is by example. Take the 'str_detect' function from the **stringr** package (Wickham, 2017). The man file 'str_detect.Rd' has the examples section:

```
\examples{
fruit <- c("apple", "banana", "pear", "pinapple")
str_detect(fruit, "a")
str_detect(fruit, "^a")
str_detect(fruit, "a$")
str_detect(fruit, "b")
str_detect(fruit, "[aeiou]")
# Also vectorised over pattern
str_detect("aecfg", letters)
}
```

The *test shell* that would be automatically created by **exampletestr** from these examples is:

```
test_that("str_detect works", {
  fruit <- c("apple", "banana", "pear", "pinapple")
  expect_equal(str_detect(fruit, "a"), )
  expect_equal(str_detect(fruit, "^a"), )
  expect_equal(str_detect(fruit, "a$"), )
  expect_equal(str_detect(fruit, "b"), )
  expect_equal(str_detect(fruit, "[aeiou]"), )
  expect_equal(str_detect("aecfg", letters), )
})
```

which can then be filled in by the package developer to give the complete and passing test:

```
test_that("str_detect works", {
  fruit <- c("apple", "banana", "pear", "pinapple")
  expect_equal(str_detect(fruit, "a"), rep(TRUE, 4))
  expect_equal(str_detect(fruit, "^a"), c(TRUE, rep(FALSE, 3)))
  expect_equal(str_detect(fruit, "a$"), c(FALSE, TRUE, FALSE, FALSE))
  expect_equal(str_detect(fruit, "b"), c(FALSE, TRUE, FALSE, FALSE))
  expect_equal(str_detect(fruit, "[aeiou]"), rep(TRUE, 4))
  expect_equal(str_detect("aecfg", letters),
               c(TRUE, FALSE, TRUE, FALSE, TRUE, TRUE, TRUE, rep(FALSE, 19)))
})
```

By running 'make_tests_shells_pkg()' in the root directory of a package, for each 'x.R' file in the 'R/' directory that has at least one function documented with an example, one gets a corresponding file 'test_x.R' in the 'tests/testthat/' directory of the package, containing these corresponding test shells to be filled in to create fully functional unit tests. For a complete overview of **exampletestr**'s capabilities, consult the package's manual and vignette.

The important point is that **exampletestr** takes care of much of the formulation of many unit tests for developers who have already created a package with adequate examples. This should encourage those who have never written unit tests to make a start and should also make unit testing easier for those who already do it.

A workflow for writing comprehensive examples and unit tests

The idea of basing unit tests around documentation examples suggests the use of **covr** (Hester, 2017) in the following way to ensure both that the examples in the documentation exhibit as much of the functionality of the package as possible and that the unit tests cover as much of the code as feasible:

1. Write comprehensive documentation examples for your package, using **covr**'s
   `'package_coverage(type = "examples") %>% shine()'`
   to ensure that all sections of code that the package user may find useful are demonstrated.
2. Write unit tests corresponding to *all* of your examples using **exampletestr**.
3. Complete the writing of unit tests, checking your code coverage with
   `'package_coverage(type = "tests") %>% shine()'`.

Using this workflow, the developer ensures that their *example coverage* (the portion of package features covered by documentation examples) is adequate, and simultaneously obtains a reduction in the work required to write comprehensive unit tests.

Good practice

One should not necessarily leave unit testing until last in the package creation workflow. However, that is already the case for the packages on CRAN that are not unit tested (a majority). **exampletestr** is the ideal companion to begin testing these packages.

exampletestr promotes a "one test per function" model. This will not always be ideal so take care to reorganise tests into a better structure when necessary. Consult Wickham (2015) for general advice on writing tests.

This package will not write a single *complete* expectation (`'expect_equal()'`, `'expect_true()'`, etc.) expression, nor should there be a routine that does so. Unit tests are meant to be automatic in the sense that they can be run automatically, however their creation is intended to involve a manual check. With **exampletestr**, this manual check is one's manual completion of the expectation expression.

Conclusion

Unit tests are crucial to ensuring that a package functions correctly, yet most developers do not write them. Most package developers do, however, write documentation examples. With **exampletestr**, documentation examples are easily transformed into unit tests; thereby promoting the inclusion of unit tests in new and existing packages.

Bibliography

M. Burger, K. Juenemann, and T. Koenig. *RUnit: R Unit Test Framework*, 2015. URL https://CRAN.R-project.org/package=RUnit. R package version 0.4.31. [p1]

B. Gaslam. *unitizer: Interactive R Unit Tests*, 2017. URL https://CRAN.R-project.org/package=unitizer. R package version 1.4.2. [p1]

P. Grosjean. *SciViews-R: A GUI API for R*. UMONS, MONS, Belgium, 2017. URL http://www.sciviews.org/SciViews-R. [p1]

J. Hester. *covr: Test Coverage for Packages*, 2017. URL https://CRAN.R-project.org/package=covr. R package version 2.2.2. [p3]

Huber, W., et al. Orchestrating high-throughput genomic analysis with Bioconductor. *Nature Methods*, 12(2):115–121, 2015. doi: 10.1038/nmeth.3252. [p2]

A. Nicholls. *Analyzing coverage of R unit tests in packages – the testCoverage package*, 2015. URL http://www.mango-solutions.com/wp/2014/11/analyzing-coverage-of-r-unit-tests-in-packages-the-testcoverage-package/. [p1]

R. Nolan. *exampletestr: Help for Writing Unit Tests Based on Function Examples*, 2017. URL https://CRAN.R-project.org/package=exampletestr. R package version 0.4.0. [p2]

H. Wickham. testthat: Get started with testing. *The R Journal*, 3:5–10, 2011. URL http://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf. [p1]

H. Wickham. *R Packages*. 2015. [p3]

H. Wickham. *stringr: Simple, Consistent Wrappers for Common String Operations*, 2017. URL https://CRAN.R-project.org/package=stringr. R package version 1.2.0. [p2]

Rory Nolan
Wellcome Trust Centre for Human Genetics, University of Oxford
OX3 7BN
United Kingdom
rnolan@well.ox.ac.uk

Sergi Padilla-Parra
Wellcome Trust Centre for Human Genetics and Division of Structural Biology, University of Oxford
OX3 7BN
United Kingdom
spadilla@well.ox.ac.uk