Testing R Packages

A Short Overview and Introduction to `tinytest`

Dirk Eddelbuettel

BioConductor Developers Forum

25 Feb 2021

https://dirk.eddelbuettel.com/papers/bioc_testing_feb2021.pdf

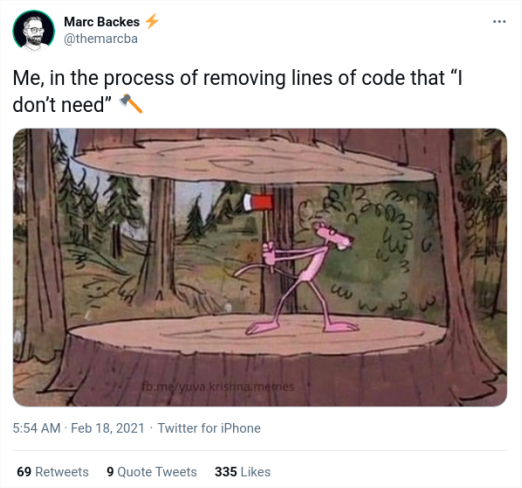
ILLINOIS

Outline

Points Covered

- What is testing? What is a unit test?
- "WDRCS" aka *What Does R Core Say*?
- Brief Survey of Approaches and Packages
- Short Intro to `tinytest` (incl. Conversion from `RUnit`)

What Is Testing? Why Do We Care?



Source: https://twitter.com/themarcba/status/1362369937688453120

Informally speaking

Testing may help with …

- simple syntactical errors
- or logic / semantic blunders
- or toolchain changes (!!)
- or thinkos

A Simple Definition

"Validating That Invariants Are Just That"

- We sometimes have a maintained set of assumptions
- We probably do not need to check (system library) functions like `sqrt(4)`
  - because implicitly we trust the system we are on already did that for us
  - but maybe 'trust and verify' beats 'trust and pray' ?
- We probably do want to check anything computationally 'sensitive'
  - and that could be as simple as an (iterative) line search algorithm
  - which may have multiple steps and a convergence criterion
  - so test on that 'new and shiny piece of metal' (Apple M1 anyone?)

"WDRCS" aka *What Does R Core Say ?*

Very Little! Zero mention of 'unit test' or the testing packages (but see below)

One paragraph in *WRE 1.1.5 Package Subdirectories* (using the current r-devel)

> *Subdirectory* `tests` *is for additional package-specific test code, similar to the specific tests that come with the R distribution. Test code can either be provided directly in a* `.R` *(or* `.r` *as from R 3.4.0) file, or via a* `.Rin` *file containing code which in turn creates the corresponding* `.R` *file (e.g., by collecting all function objects in the package and then calling them with the strangest arguments). The results of running a* `.R` *file are written to a* `.Rout` *file. If there is a corresponding[^21]* `.Rout.save` *file, these two are compared, with differences being reported but not causing an error. [Two more sentence omitted.]*

Plus one outburst about one less-than-perfectly-successful release of one testing package.

Minimal Setup

- Directory `tests` can have scripts to be executed (and we come back to this later)
- Output of each script `foo.R` will be collected in `foo.Rout`
- If present, a file `foo.Rout.save` will be used for (clever enough) line-by-line comparison (skipping version numbers from the R session etc)
- This is … somewhat rustic but surprisingly robust … and used by the packages that come with R (and the ‘Recommended’ packages)
- Else one can also ‘just loop over a large set of assertions’ (and `data.table` does just that, and well, with a helper function)

Enter Unit Test Packages

Unit Test Packages

Contestants

- `RUnit`: "R Unit Test Framework"
  N=120, released June 2006, quirky setup and output, S4-based, robust
  also used by several metric tons of BioConductor packages
- `testthat`: "Unit Testing for R"
  N=5626, released November 2009, widely used, many extensions
- `unittest`: "TAP-Compliant Unit Testing"
  N=3, released August 2014, used only by its authors
- `tinytest`: "Lightweight and Feature Complete Unit Testing Framework"
  N=135, released April 2019, new, simple, zero dependencies, rising in popularity
  currently used by four BioConductor packages

Counts from 19 Feb 2021 using `str(tools::package_dependencies(pkgs, reverse=TRUE, db=tools::CRAN_package_db(), which="most"))`

TINYTEST

Highlights

- Simple, fast, easy, zero dependencies
- Files can be run as scripts via `Rscript` (and/or my `r` from `littler`)
- Files install along with the package (just like `RUnit`)
- Tests can be run
  - per file
  - per test directory
  - per (installed) package
  - in a build / load / test cycle
  - both serially and in parallel
- Side effects are monitored (e.g. environment variables)

Adding tinytest

Three Simple Things

- Add a `Suggests: tinytest` to `DESCRIPTION`
- Invoke `tinytest` *conditionally* from `tests/tinytest.R` *e.g.*, via

```
if (requireNamespace("tinytest", quietly=TRUE))
    tinytest::test_package("testpkg")
```

- Add your tests in files `inst/tinytest/test_*.R`

And a helper function `setup_tinytest()` does just that.

Conversion from RUnit

- Pretty straightforward, have done did it for well over a dozen packages
- Mostly search and replace
  - as the 'triplet' <argA, argB, msg> may become <argA, argB>
  - `checkEquals`, `checkTrue`, ... become `expect_equal`, `expect_true`
  - many available `expect_*` predicates
- Files become standard R scripts, no extra requirements

Conversion from RUnit: Real (yet randomly chosen) Rcpp S4 example

```
## some function conditional wrapping this ...
    test.S4.dotdataslot <- function(){
        setClass( "Foo", contains = "character", representation( x = "numeric" ) )
        foo <- S4_dotdata( new( "Foo", "bla", x = 10 ) )
        checkEquals( as.character( foo) , "foooo" )
    }
```

is now

```
#    test.S4.dotdataslot <- function(){
setClass( "Foo", contains = "character", representation( x = "numeric" ) )
foo <- S4_dotdata( new( "Foo", "bla", x = 10 ) )
expect_equal( as.character( foo) , "foooo" )
```

tinytest: Another (Real) Conversion Example from RUnit to tinytest

```
.setUp <- RcppArmadillo:::unit_test_setup("rng.cpp")

test.randu.seed <- function() {
    set.seed(123)
    a <- randu(10)
    set.seed(123)
    b <- randu(10)
    checkEquals(a, b, msg="randu seeding")
}

test.randi.seed <- function() {
    set.seed(123)
    a <- randi(10)
    set.seed(123)
    b <- randi(10)
    checkEquals(a, b, msg="randi seeding")
}
```

```
library(RcppArmadillo)

Rcpp::sourceCpp("cpp/rng.cpp")

#test.randu.seed <- function() {
set.seed(123)
a <- randu(10)
set.seed(123)
b <- randu(10)
expect_equal(a, b)#, msg="randu seeding")

#test.randi.seed <- function() {
set.seed(123)
a <- randi(10)
set.seed(123)
b <- randi(10)
expect_equal(a, b)#, msg="randi seeding")
```

tinytest: Comparing Startup Files

RUnit

```
## parts of the remaining unconverted `tests/doRUnit.R`
stopifnot(require(RUnit, quietly=TRUE))
## if an option is set, we run tests. otherwise we don't.
## recall that we DO need a working Bloomberg connection...
if (getOption("blpUnitTests", FALSE)) {
    ## load the package
    stopifnot(require(Rblpapi, quietly=TRUE))
    ## without this, we get (or used to get) unit test failures
    Sys.setenv("R_TESTS"="")
    Sys.setenv("RunRblpapiUnitTests" = "yes")
    ## Define tests
    testSuite <- defineTestSuite(name="Rblpapi Unit Tests",
                                 dirs=system.file("unitTests",
                                                  package = "Rblpapi"),
                                 testFuncRegexp = "^[Tt]est.+")
    tests <- runTestSuite(testSuite)          # Run tests
    printTextProtocol(tests)                  # Print results
    ## Return success or failure to R CMD CHECK
    if (getErrors(tests)$nFail > 0) stop("TEST FAILED!")
    if (getErrors(tests)$nErr > 0) stop("TEST HAD ERRORS!")
    if (getErrors(tests)$nTestFunc < 1) stop("NO TEST FUNCTIONS RUN!")
}
```

tinytest

```
if (requireNamespace("tinytest", quietly=TRUE))
    tinytest::test_package("testpkg")
```

Conditioning tinytest: Real Examples

As **tinytest** files are just R files, *any* R expression works. Actual Examples follow:

```
if (!requireNamespace("Matrix", quietly=TRUE))
    exit_file("Need the 'Matrix' package")
if (packageVersion("Matrix") < "1.3.0")
    exit_file("Old 'Matrix' package?")
```

```
if (tiledb_version(TRUE) < "2.2.0")
    exit_file("Needs TileDB 2.2.* or later")
```

```
if (Sys.getenv("RunAllRcppTests") != "yes")
    exit_file("Set 'RunAllRcppTests' to 'yes' to run.")
```

Benefits I have seen

- Simpler use: For Rcpp we had to ensure paths in order to `sourceCpp()` C++ components of tests – this all just works now
- Simpler scripts: I can treat the files as scripts and 'just run them' (and for that I often add `library(tinytest); library(thisPackage)`)
- Run directory, run file, run package
  - which I find myself doing *all time time* during development
- Lightweight, zero depends
  - which you think may not matter til it does...
- Nicely concise and pleasant summary

One More Things: Extensible

- Michel Lang was first to plug-in (100+ ?) assertions from this `checkmate` package
- For student grading in the (awesome, automated) PrairieLearn backend at Illinois, we realized that “unit testing == submission grading” so we wrote a setup where I exam or quiz or home work questions are ... validated with unit test predicates and summaries
- For incorrect answers, we realized we could marry `tinytest` with `diffobj` (a nice object difference visualizer) and created `ttdo` (on CRAN too)
- (Those two are joint with Alton Barbehenn who is an awesome TA)

tinytest Documentation

Using tinytest

Mark van der Loo

December 17, 2020 | Package version 1.2.4

Contents

Reading guide

Readers of this document are expected to know how to write R functions and have a basic understanding of a package source directory structure.

Tinytest by example

Mark van der Loo

December 17, 2020 | Package version 1.2.4

Contents

Introduction

This document provides a number of real-life examples on how **tinytest** is used by other packages. The examples aim to illustrate the purpose of testing functions and serve as a complement to the technical documentation and the 'using `tinytest`' vignette. There is a section for each function. Each section starts with a short example that demonstrates the core purpose of the function. Next, one or more examples from packages that are published on CRAN are shown and explained.

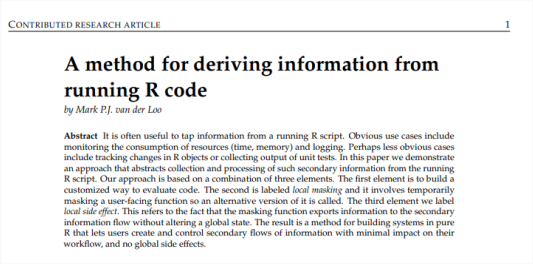
Sometimes a few lines of code were modified or deleted for brevity. This is indicated with comment between square brackets, e.g.

```
## [this is an extra comment, only for this vignette]
```

This document is probably not interesting to read front-to-back. It is more aimed to browse once in a while to get an idea on how **tinytest** can be used in practice.

Package authors are invited to contribute new use cases so new users can learn from them. Please contact the author of this package either by email or via the github repository.

The Key Idea Behind the Implementation



CONTRIBUTED RESEARCH ARTICLE 1

A method for deriving information from running R code

by Mark P.J. van der Loo

Abstract It is often useful to tap information from a running R script. Obvious use cases include monitoring the consumption of resources (time, memory) and logging. Perhaps less obvious cases include tracking changes in R objects or collecting output of unit tests. In this paper we demonstrate an approach that abstracts collection and processing of such secondary information from the running R script. Our approach is based on a combination of three elements. The first element is to build a customized way to evaluate code. The second is labeled *local masking* and it involves temporarily masking a user-facing function so an alternative version of it is called. The third element we label *local side effect*. This refers to the fact that the masking function exports information to the secondary information flow without altering a global state. The result is a method for building systems in pure R that lets users create and control secondary flows of information with minimal impact on their workflow, and no global side effects.

At arXiv:2002:07472

and forthcoming, R Journal

Caveat

What Could Still Go Wrong?

Source:
https://twitter.com/laurieontech/status/1355600163226660872

Limitations

- Tests cannot guarantee correctness
- We must remain awake, alert, attentive
- So still no free lunch…
- Yet testing arguably helps a lot

Things Not Covered Today

- Code Coverage
  - Useful metric I was at first somewhat sceptical about
  - And which I clearly game at times too (hello, my friend `#nocov`)
  - But which has helped me find a bug or two
- The other two test runner packages on CRAN
  - Mostly similar rather than different
- Testing and Continuous Integration
- Test Aggregation (TAP Protocol)
  - Most have exporters so if you're into this you know where to look

Things Covered Today

- R has basic support for testing, offers a hook
- RUnit was first widely package: quite useful but quirky
- tinytest is the newest entry
  - simple, clean, efficient, lightweight
  - serial or parallel use
  - works with source and installed package
  - works directly with files
  - extensible

Yep!

Image source https://www.howtogeek.com/403438/do-the-people-you-follow-on-social-media-spark-joy/

Thanks!

Thank you!

slides	https://dirk.eddelbuettel.com/presentations/
web	https://dirk.eddelbuettel.com/
mail	dirk@eddelbuettel.com
github	@eddelbuettel
twitter	@eddelbuettel