Testing for fun and profit

Rich FitzJohn
MRC Centre for Global Disease Analysis
Imperial College London
The take-home: If testing feels like a chore, change how you do it
Testing in <1 minute
add <- function(a, b) {
    a + b
}

expect_equal(add(1, 2), 3)
How to draw an Owl.

“A fun and creative guide for beginners”

Fig 1. Draw two circles
Fig 2. Draw the rest of the damn Owl
Only a few commands to learn
Only a few commands to learn
What sorts of test?
https://www.contino.io/insights/the-testing-pyramid
This still does not sound fun
Why test?
Better workflows
Mocking
Things that are hard
But why test?
code that is easy to test is easy to understand and easy to reuse and easy to replace
data <- read.csv("input.csv")
mymodel <- function(a, b) {
  for (i in unique(data$group)) {
    fit <- long_running_fit(a, b, data)
    plot(y ~ x, data)
    if (fit$pvalue < 0.05) {
      lines(fit)
    }
  }
}
saveRDS(fit, "c:/myfiles/fit.rds")
lapply(fit, coef)
data <- read.csv("input.csv")

mymodel <- function(a, b) {
  for (i in unique(data$group)) {
    fit <- long_running_fit(a, b, data)
    plot(y ~ x, data)
    if (fit$pvalue < 0.05) {
      lines(fit)
    }
  }
}

saveRDS(fit, "c:/myfiles/fit.rds")
lapply(fit, coef)
}
```r
data <- read.csv("input.csv")
mymodel <- function(a, b) {
  for (i in unique(data$group)) {
    fit <- long_running_fit(a, b, data)
    plot(y ~ x, data)
    if (fit$pvalue < 0.05) {
      lines(fit)
    }
  }
}
saveRDS(fit, "c:/myfiles/fit.rds")
lapply(fit, coef)
}```
data <- read.csv("input.csv")
mymodel <- function(a, b) {
  for (i in unique(data$group)) {
    fit <- long_running_fit(a, b, data)
    plot(y ~ x, data)
    if (fit$pvalue < 0.05) {
      lines(fit)
    }
  }
}
saveRDS(fit, "c:/myfiles/fit.rds")
lapply(fit, coef)
data <- read.csv("input.csv")
mymodel <- function(a, b) {
  for (i in unique(data$group)) {
    fit <- long_running_fit(a, b, data)
    plot(y ~ x, data)
    if (fit$pvalue < 0.05) {
      lines(fit)
    }
  }
}
saveRDS(fit, "c:/myfiles/fit.rds")
lapply(fit, coef)
function(...) {
    if (...) {
        if (...) {
            a1
        } else if (...) {
            a2
        } else {
            a3
        }
    } else if (...) {
        b
    } else {
        c
    }
}
The tool shapes the hand.
How to incorporate testing into your workflows
1. Write a bunch of code
2. Play around with it interactively
3. Get your colleagues to try it out
4. Publish a number of papers
5. Realise that bugs have crept in
6. Start writing tests
First attempt at RUnit based unit tests

```r
# Test case code, based on Emma's tests.
test.geosse <- function()
{
  tree <- read.tree(text="((((0:0.4610876,1:0.3077172,2:0.2310825,3:0.3210825:0.075092):0.154159):0.425922,(4:0.244819,5
  ):0.128952:0.325922:0.154159):0.425922,((6:0.4610876,7:0.2310825,8:0.3210825:0.075092):0.154159):0.425922,(9:0.244819,10
  )):0.128952:0.325922:0.154159):0.425922,((11:0.4610876,12:0.2310825,13,14:0.3210825:0.075092):0.154159):0.425922,(15:0.244819,16
  )):0.128952:0.325922:0.154159):0.425922,((17:0.4610876,18:0.2310825,19:0.3210825:0.075092):0.154159):0.425922,(20:0.244819,21
  )):0.128952:0.325922:0.154159):0.425922,((22:0.4610876,23:0.2310825,24:0.3210825:0.075092):0.154159):0.425922,(25:0.244819,26
  )):0.128952:0.325922:0.154159):0.425922,((27:0.4610876,28:0.2310825,29:0.3210825:0.075092):0.154159):0.425922,(30:0.244819,31
  )):0.128952:0.325922:0.154159):0.425922,((32:0.4610876,33:0.2310825,34:0.3210825:0.075092):0.154159):0.425922,(35:0.244819,36
  )):0.128952:0.325922:0.154159):0.425922)

states <- c(0, 1, 0, 2, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0
names(states) <- c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14")

starting.point.geosse(tree)

# 1.5788510 1.5788510 1.5788510 0.0000000 0.0000000 0.3157702 0.3157702

pars <- c(0.9, 0.8, 0.1, 0.2, 0.3, 0.5, 0.6)

lnL.7par <- make.geosse(tree, states)

names(pars) <- argnames(lnL.7par)

checkEquals(lnL.7par(pars), -23.71574, tolerance=1e-7)

checkEquals(lnL.7par(pars, condition.surv=TRUE), -24.10259,

........ tolerance=1.06e-7)
```

github.com/richfitz/diversitree/commit/10e69a
Install, once more, the correct version of g++
richfitz committed on 14 May 2015

Attempt to fix both C++11 and Fortran installation on travis
richfitz committed on 14 May 2015

Commits on May 13, 2015

Attempt to fix gfortran installation
richfitz committed on 13 May 2015

Use g++ 4.8 not 4.9
richfitz committed on 13 May 2015

Attempt to get fortran installed properly
richfitz committed on 13 May 2015

Install newer gcc on travis
richfitz committed on 13 May 2015

Remove util_lang_range.h in the hope that allows compilation
richfitz committed on 13 May 2015

Select C++11 compiler in the correct way
richfitz committed on 13 May 2015

Make early termination warning optional in grow_plant_to_size
richfitz committed on 13 May 2015

Install RcppR6 on travis
richfitz committed on 13 May 2015

Add travis
richfitz committed on 13 May 2015

github.com/traitecoevo/plant/commits/d5aebd
1. Write a bunch of code
2. Play around with it interactively
3. Get your colleagues to try it out
4. Publish a number of papers
5. Realise that bugs have crept in
6. Start writing tests
testing
Turn your experimentation into tests
Turn your user requirements into tests
Turn your bug reports into tests
1. Create a trivial skeleton
2. Write a function and test it
3. Set up covr/codecov
4. Create a branch
5. Create a Minimal Viable Product (with tests)
6. Create a Pull Request
7. Justify all your coverage gaps
8. GOTO 4
The Beyoncé Rule

If you liked it, you should have put a test on it

increment.com/testing/testing-as-communication/
```r
1  function(a, b) {
2    if (is.null(a)) b else a
3  }
```

```r
1  context("util")
2
3  test_that("null-or-value works", {
4    expect_equal(1 || NULL, 1)
5    expect_equal(1 || 2, 1)
6    expect_equal(NULL || NULL, NULL)
7    expect_equal(NULL || 2, 2)
8  })
```
There's heaps more to the added here, though the upstream docs don't make it super easy. I am not 100% sure about the name (always feels a bit just going with the same name as the upstream package) but am coming up short on alternatives (my 2nd choice was "traduire", which at least has a good R sound to it).

The big drama with packaging this was getting v8 to see Promise - turns out that's only in recent V8 and Debian ships with an ancient version.
Codecov Report

!! No coverage uploaded for pull request base (master@e5f578c). Click here to learn what that means.

The diff coverage is 100%.

<table>
<thead>
<tr>
<th>Coverage Diff</th>
<th>master</th>
<th>#1</th>
<th>+/-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>?</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Files</td>
<td>?</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Lines</td>
<td>?</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>Branches</td>
<td>?</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Hits</td>
<td>?</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>Misses</td>
<td>?</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Partial</td>
<td>?</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

github.com/reside-ic/traduire/pull/1
Codecov Report

Merging #7 into master will not change coverage. The diff coverage is 100%.

@@ Coverage Diff @@
## master #7 +/- ##

<table>
<thead>
<tr>
<th>Coverage</th>
<th>100%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Lines</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>+ Hits</td>
<td>100</td>
<td>101</td>
</tr>
</tbody>
</table>

Impacted Files Coverage Δ

<table>
<thead>
<tr>
<th>File</th>
<th>Coverage Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/translator.R</td>
<td>100% &lt;100% (ω)</td>
</tr>
</tbody>
</table>
AUTOMATE

ALL THE THINGS!
100% coverage is only the beginning
Mocking
relink <- function(from, to) {
  backup <- paste0(from, "\.bak")
  fs::file_move(from, backup)
  withCallingHandlers(
    fs::link_create(to, from, FALSE),
    error = function(e) fs::file_move(backup, from))
  fs::file_delete(backup)
}

test_that("relink error handling", {
  ...

  mockery::stub(relink, "fs::link_create",
    function(...) stop("Some error linking"))
  info <- fs::file_info(c(from, to))$inode
  expect_error(relink(from, to), "Some error linking")
  expect_true(all(fs::file_info(c(from, to))$inode == info))
})
System-specific behaviour
Long running processes
Sensitive data
Interactive user input
Awkward global state

blog.r-hub.io/2019/10/29/mocking/
The hard basket randomness
The hard basket

shiny.rstudio.com/articles/shinytest.html
shiny.rstudio.com/articles/integration-testing.html
github.com/reside-ic/shiny-selenium
The hard basket

data & analyses

assertive, assertr, ensurer, assertthat, checkmate, tester, validator
github.com/frictionlessdata/goodtables-py
github.com/vimc/orderly
github.com/vimc/dettl
The hard basket

long running tests

https://github.com/vimc/montagu-ci
https://github.com/features/actions
If testing feels like a chore
change how you do it
change why you do it