Unit testing

(With a dash of API design)

Jan 2019

Charlotte Wickham
@cvwickham
cwickham@gmail.com
cwick.co.nz

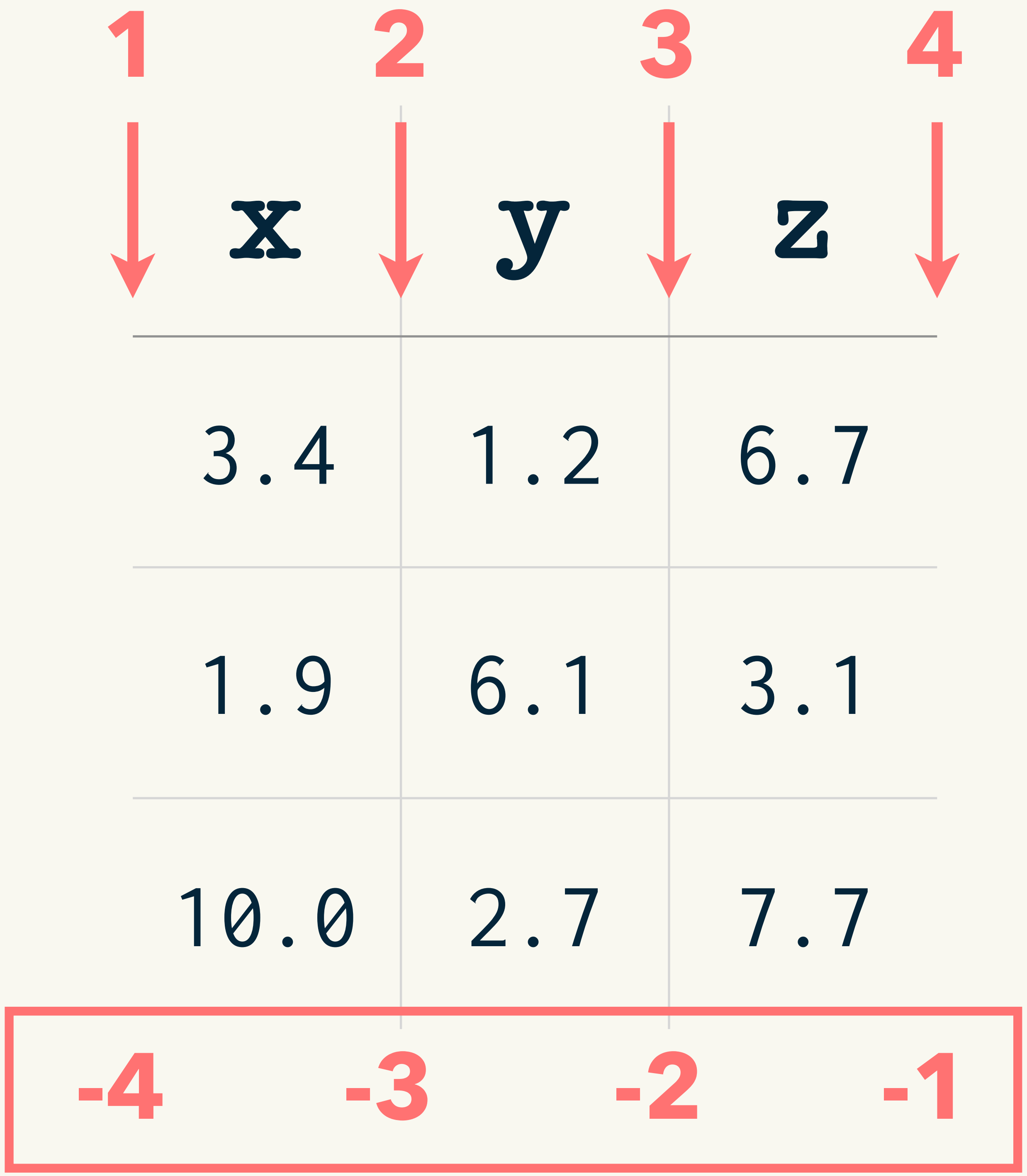
Adapted from *Tidy Tools* by Hadley Wickham

Motivation

Let’s add a column to a data frame

```
# Goal:
# Write a function that allows us to add a
# new column to a data frame at a specified
# position.

add_col(df, "name", value, where = 1)
add_col(df, "name", value, where = 2)

# Start simple and try out as we go
```

where =



x	y	z
3.4	1.2	6.7
1.9	6.1	3.1
10.0	2.7	7.7

Would be nice to have; but we won’t implement today

Start with insert_into()

Works like cbind() but can insert anywhere

df1

a	b	c
3	4	5

df2

X	Y
1	2

```
insert_into(df1, df2,
   where = 1)
```

X	Y	a	b	c
1	2	3	4	5

```
insert_into(df1, df2,
   where = 2)
```

a	X	Y	b	c
3	1	2	4	5

Add the columns of df2 to df1 at position where

Your turn

What goes in ...?

```
# Hint: cbind() will be useful
# Add the columns of df2 to df1 at position where
insert_into <- function(x, y, where = 1) {
  if (where == 1) { # first col
    ...
  } else if (where > ncol(x)) { # last col
    ...
  } else {
    ...
  }
}
```

My first attempt

```r
insert_into <- function(x, y, where = 1) {
  if (where == 1) {
    cbind(x, y)
  } else if (where > ncol(x)) {
    cbind(y, x)
  } else {
    cbind(x[1:where], y, x[where:nrow(x)])
  }
}
```

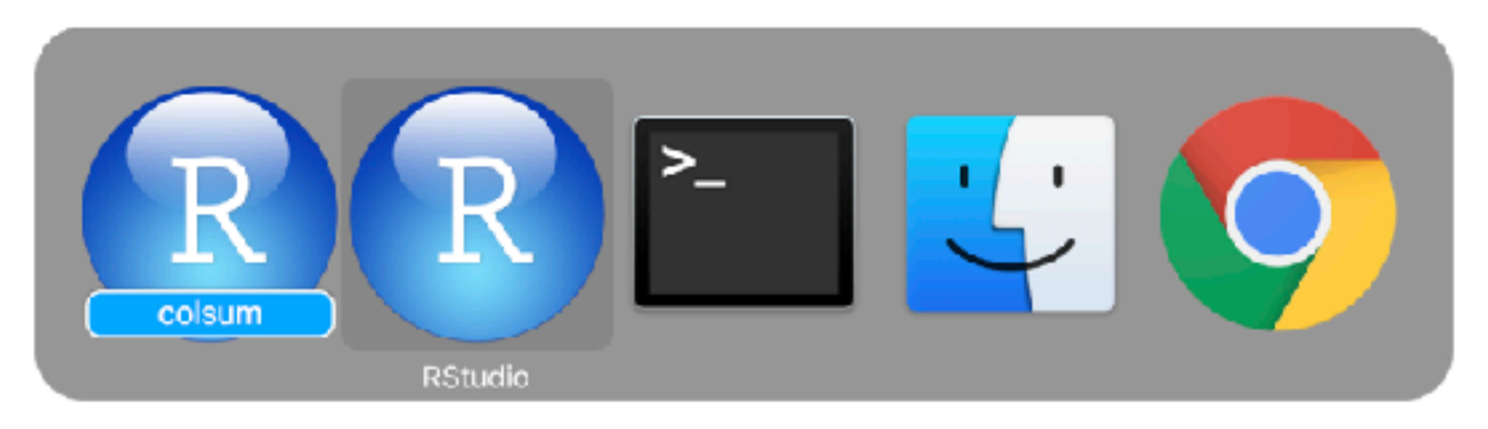
Actually correct

```
insert_into <- function(x, y, where = 1) {
  if (where == 1) {
    cbind(y, x)
  } else if (where > ncol(x)) {
    cbind(x, y)
  } else {
    lhs <- 1:(where - 1)
    cbind(x[lhs], y, x[-lhs])
  }
}
```

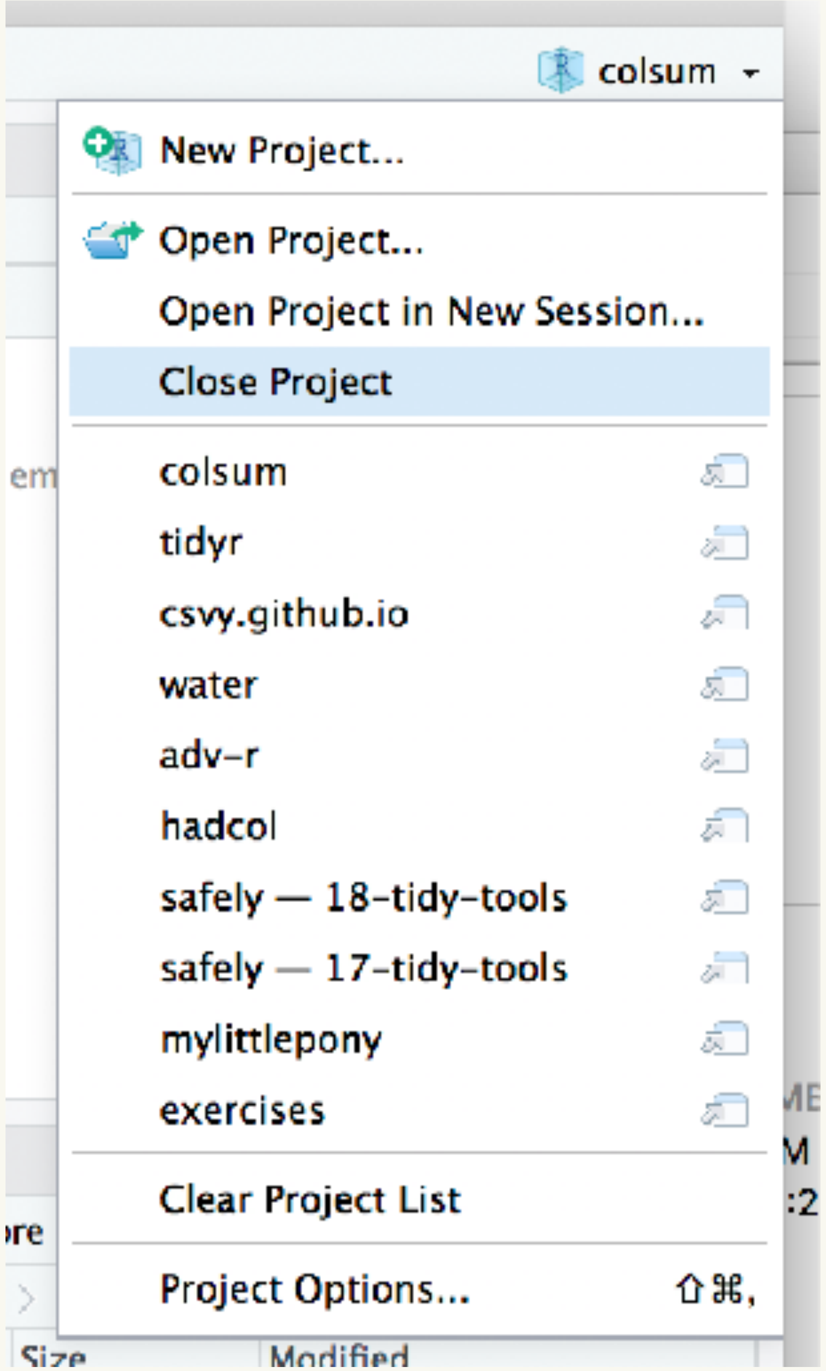
How did I write that code?

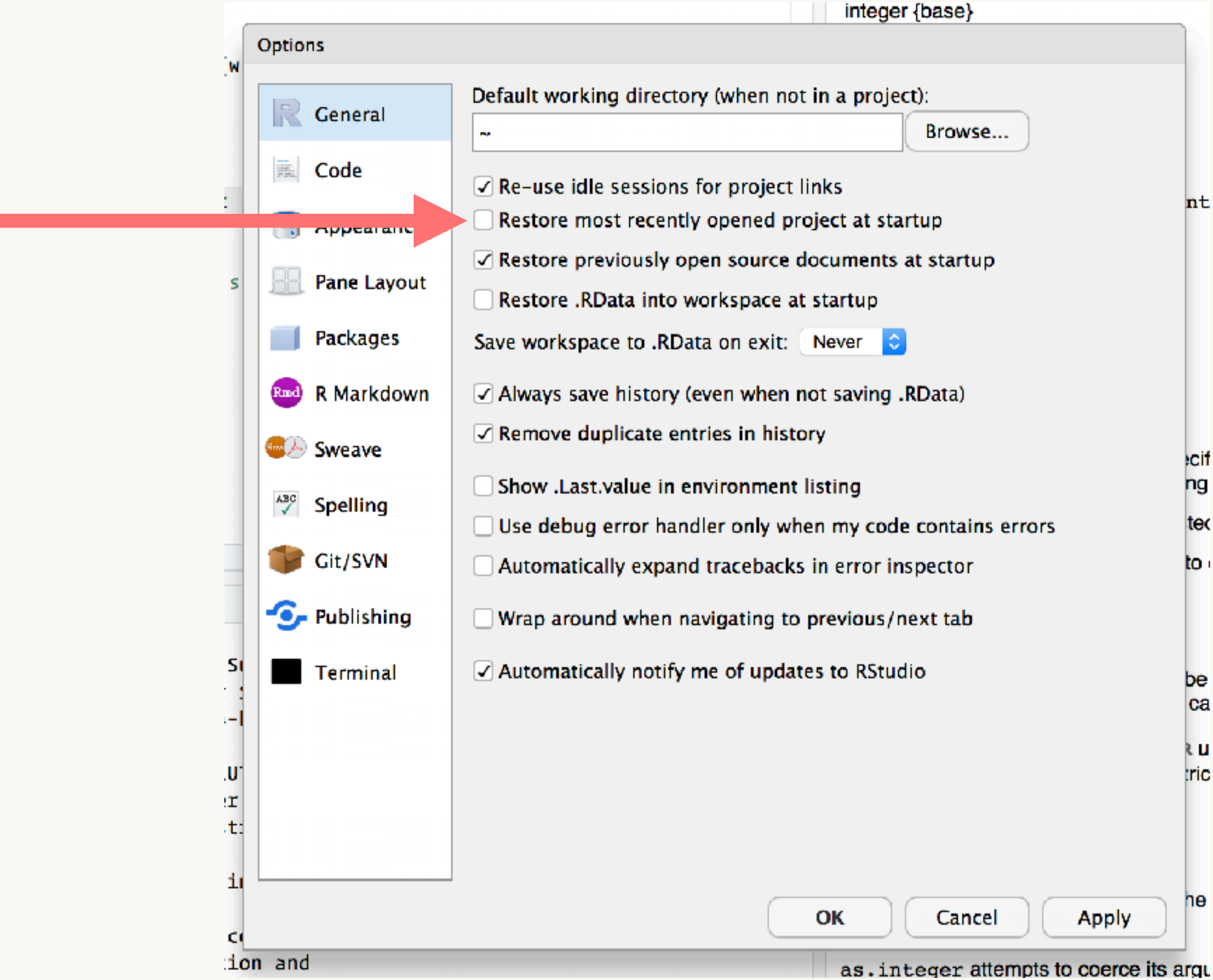
```r
# Some simple inputs
df1 <- data.frame(a = 3, b = 4, c = 5)
df2 <- data.frame(X = 1, Y = 2)

# Then each time I tweaked it, I re-ran
# these cases
insert_into(df1, df2, where = 1)
insert_into(df1, df2, where = 2)
insert_into(df1, df2, where = 3)
insert_into(df1, df2, where = 4)
```

Where did I write that code?



As well as RStudios associated with a project, you also get one associated with no project





Default working directory (when not in a project):

~

☑ Re-use idle sessions for project links

□ Restore most recently opened project at startup

☑ Restore previously open source documents at startup

□ Restore .RData into workspace at startup

Save workspace to .RData on exit: Never

☑ Always save history (even when not saving .RData)

☑ Remove duplicate entries in history

□ Show .Last.value in environment listing

□ Use debug error handler only when my code contains errors

□ Automatically expand tracebacks in error inspector

□ Wrap around when navigating to previous/next tab

☑ Automatically notify me of updates to RStudio

Two challenges

Cmd + Enter is error prone

Looking at the outputs of each run is tedious

We need a new workflow!

Cmd + Enter is error prone

Put code in R/ and use devtools::**load_all()**

Looking at the outputs of each run is tedious

Write unit tests and use devtools::**test()**

Testing workflow

http://r-pkgs.had.co.nz/tests.html

First, create a package

```
usethis::create_package("~/Desktop/hadcol")
usethis::use_r("insert_into")
```

```
insert_into <- function(x, y, where = 1) {
  if (where == 1) {
    cbind(y, x)
  } else if (where > ncol(x)) {
    cbind(x, y)
  } else {
    lhs <- 1:(where - 1)
    cbind(x[lhs], y, x[-lhs])
  }
}
```

this gets
copy + pasted
into
insert_into.R

Then, set up testing infrastructure

Set up testthat infrastructure

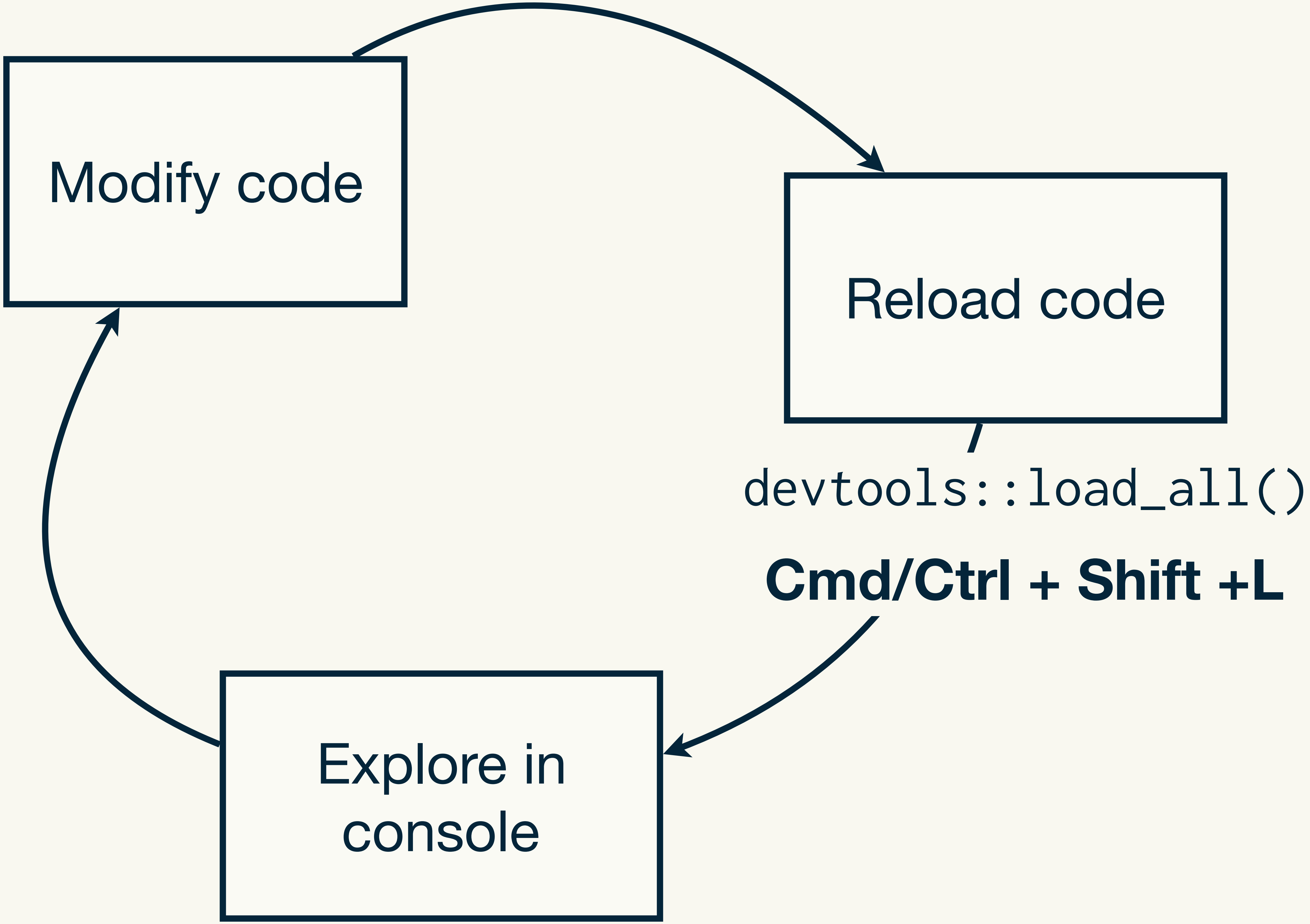
```
usethis::use_test()
✓ Adding 'testthat' to Suggests field
✓ Creating 'tests/testthat/'
✓ Writing 'tests/testthat.R'
✓ Writing 'tests/testthat/test-insert_into.R'
● Modify 'tests/testthat/test-insert_into.R'
```

Create test file matching script

```
devtools::test()
# Or Command + Shift + T
```

Run tests

So far we've done this:



Modify code

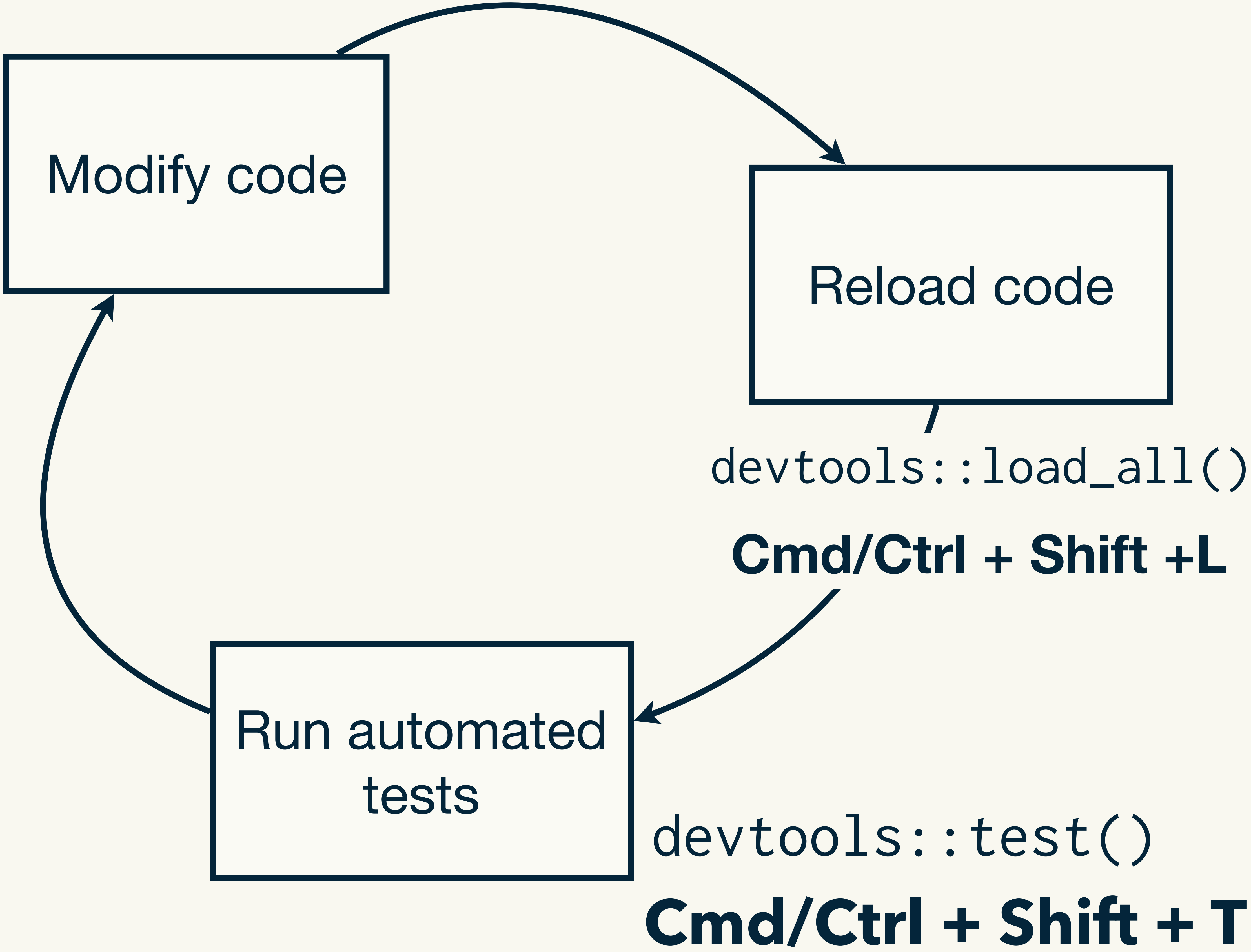
Reload code

`devtools::load_all()`

Cmd/Ctrl + Shift +L

Explore in console

Testthat gives a new workflow



Modify code

Reload code

`devtools::load_all()`

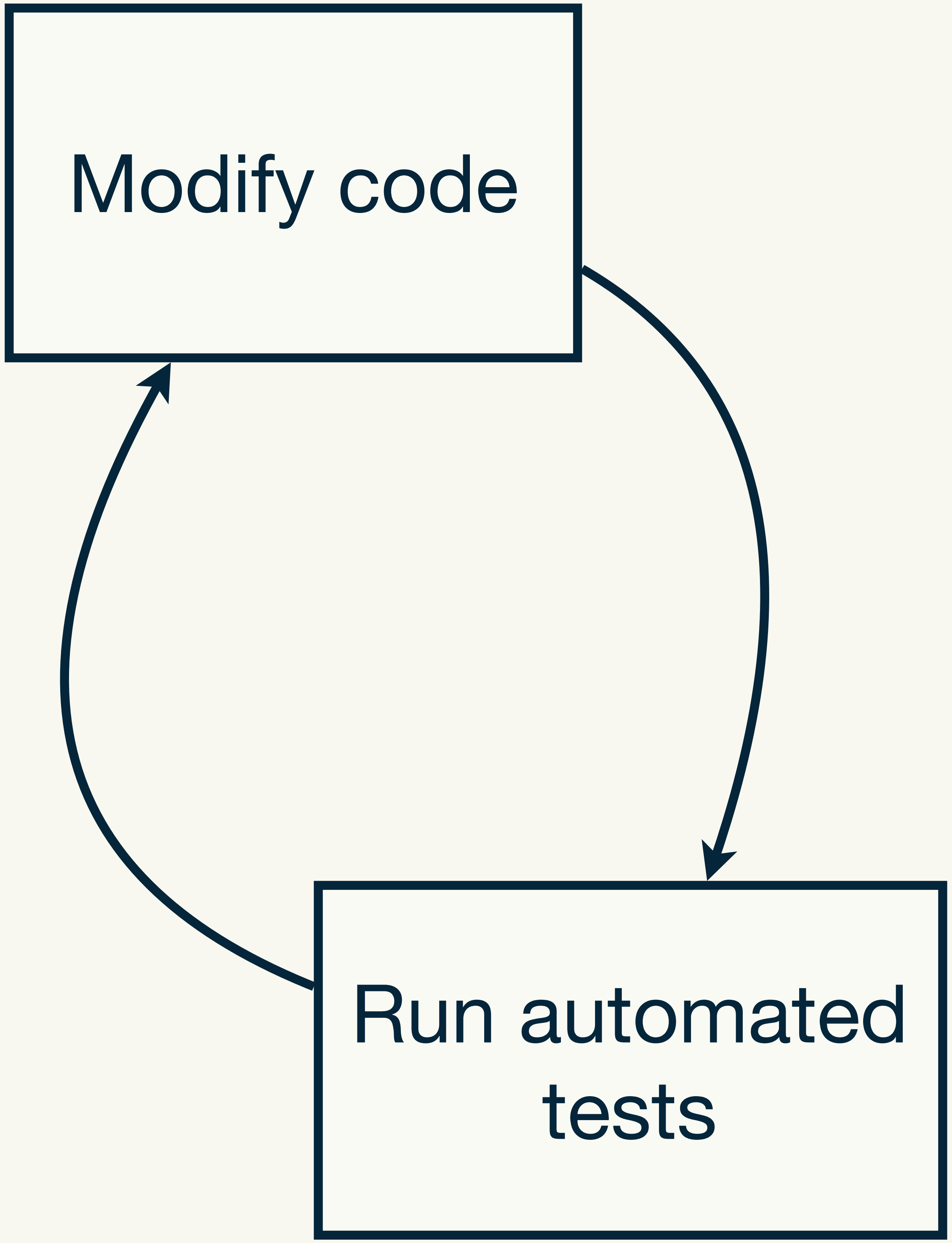
Cmd/Ctrl + Shift +L

Run automated tests

`devtools::test()`

Cmd/Ctrl + Shift + T

But why reload the code?



`devtools::test()`

Cmd/Ctrl + Shift + T

Key idea of unit testing is to automate!

Helper function to reduce duplication

```
at_pos <- function(i) {
  insert_into(df1, df2, where = i)
}

expect_named(at_pos(1), c("X", "Y", "a", "b", "c"))
expect_named(at_pos(2), c("a", "X", "Y", "b", "c"))
expect_named(at_pos(3), c("a", "b", "X", "Y", "c"))
expect_named(at_pos(4), c("a", "b", "c", "X", "Y"))
```

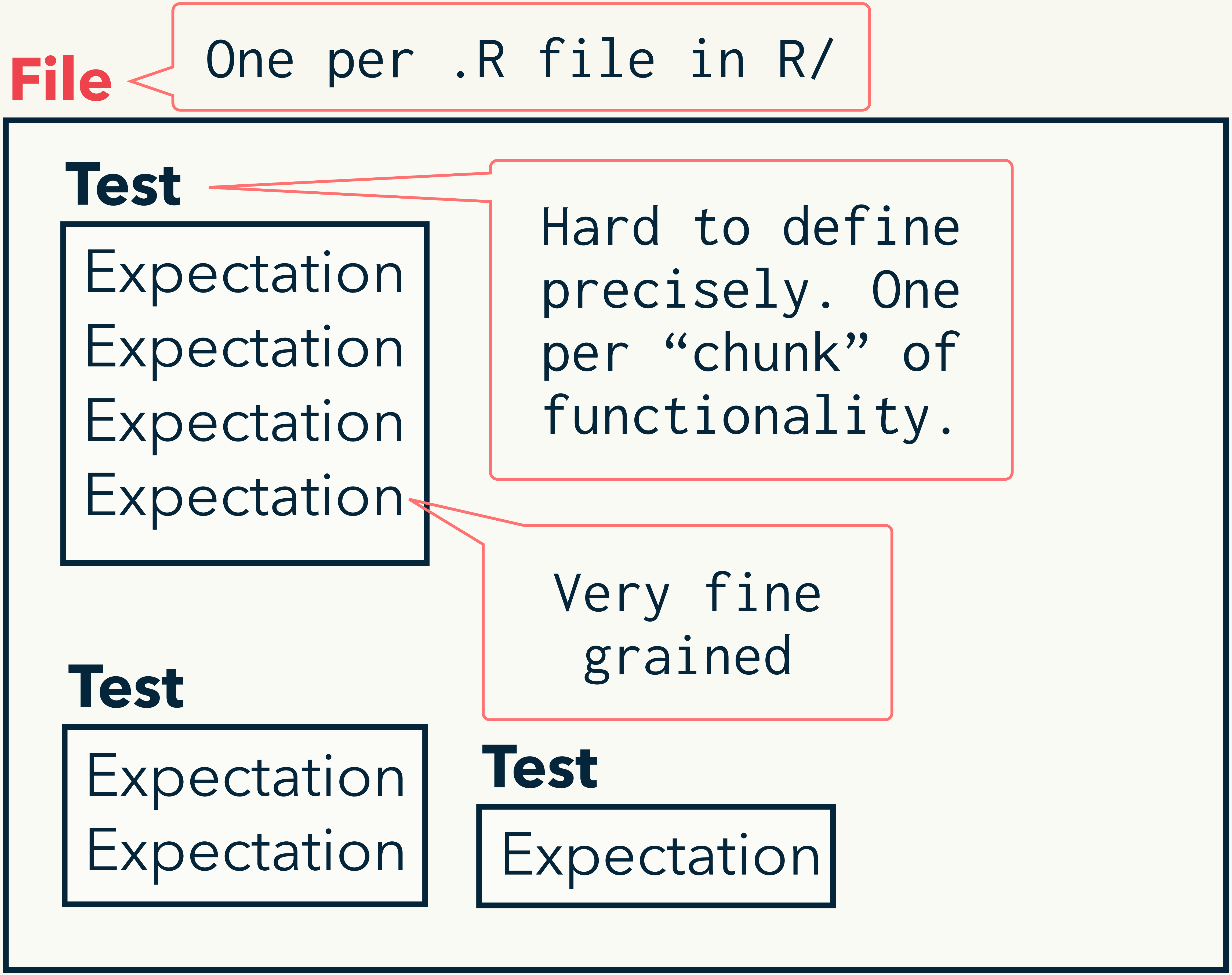
Describes an expected property of the output

This automation must follow conventions

Tests for R/insert_into.R

```
# In tests/testthat/test-insert_into.R
test_that("can add column at any position", {
  df1 <- data.frame(a = 3, b = 4, c = 5)
  df2 <- data.frame(X = 1, Y = 2)
  at_pos <- function(i) {
    insert_into(df1, df2, where = i)
  }

  expect_named(at_pos(1), c("X", "Y", "a", "b", "c"))
  expect_named(at_pos(2), c("a", "X", "Y", "b", "c"))
  expect_named(at_pos(3), c("a", "b", "X", "Y", "c"))
  expect_named(at_pos(4), c("a", "b", "c", "X", "Y"))
})
```

Tests are organised in three layers



File

One per .R file in R/

Test

Hard to define precisely. One per “chunk” of functionality.

Expectation

Expectation

Expectation

Expectation

Very fine grained

Test

Expectation

Expectation

Test

Expectation

Practice the workflow

```
usethis::create_package("~/Desktop/hadcol")

usethis::use_r("insert_into")
# Copy insert_into() from next slide
# Check all is ok with load_all()

usethis::use_test()
# Copy expectations from next next slide
# Run tests with keyboard shortcut
# Confirm that if you break insert_into() the
# tests fail.
```

insert_into()

```r
# In R/insert_into.R
insert_into <- function(x, y, where = 1) {
  if (where == 1) {
    cbind(y, x)
  } else if (where > ncol(x)) {
    cbind(x, y)
  } else {
    lhs <- 1:(where - 1)
    cbind(x[lhs], y, x[-lhs])
  }
}
```

Expectations

```r
# In tests/testthat/test-insert_into.R
context("test-insert_into")

test_that("can add column at any position", {
  df1 <- data.frame(a = 3, b = 4, c = 5)
  df2 <- data.frame(X = 1, Y = 2)
  at_pos <- function(i) {
    insert_into(df1, df2, where = i)
  }

  expect_named(at_pos(1), c("X", "Y", "a", "b", "c"))
  expect_named(at_pos(2), c("a", "X", "Y", "b", "c"))
  expect_named(at_pos(3), c("a", "b", "X", "Y", "c"))
  expect_named(at_pos(4), c("a", "b", "c", "X", "Y"))
})
```

You should now be in freshly created

`[hadcol]`

(Download also has more complete `hadcol-test` if you get stuck)

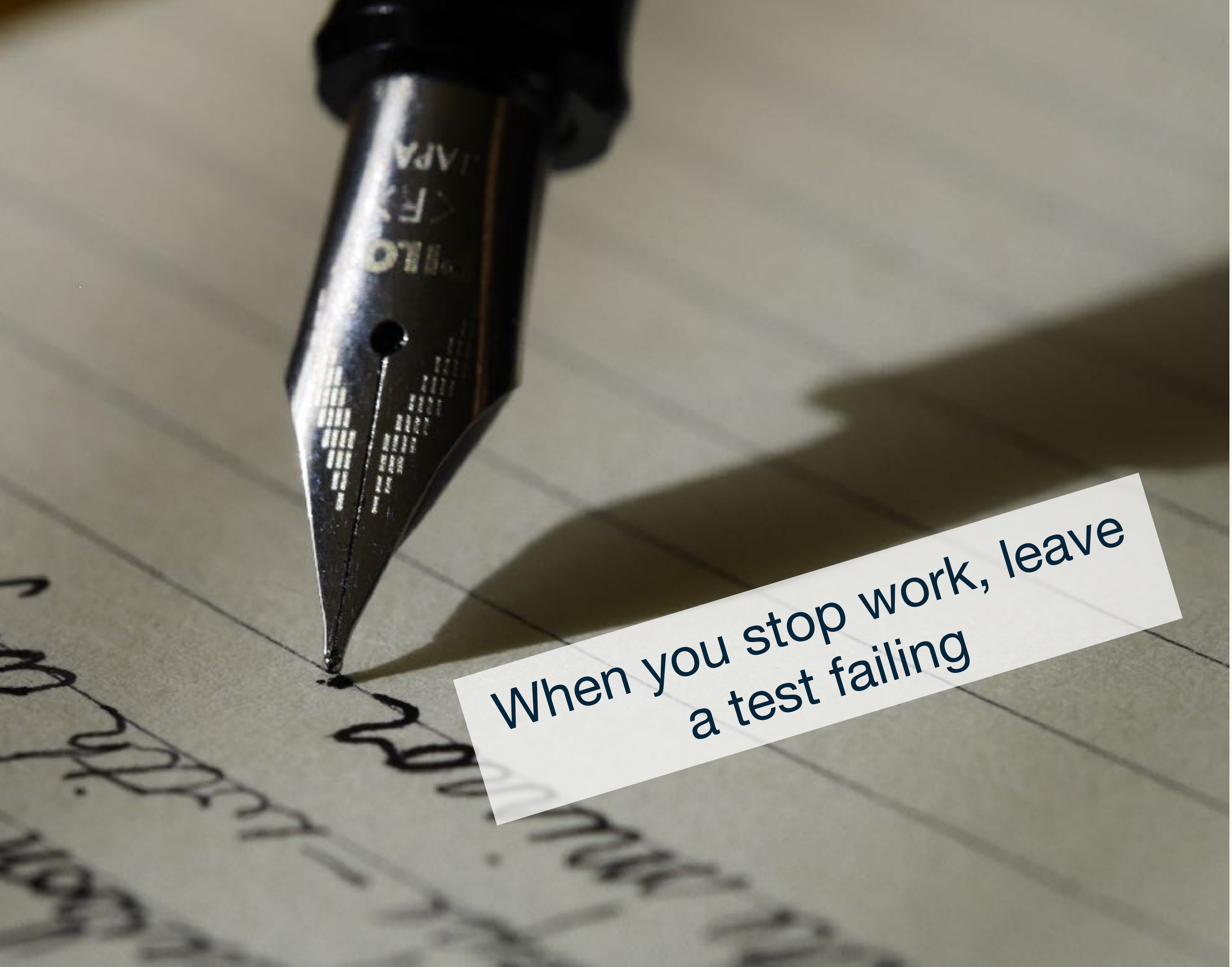
Other advantages

Writing tests improves your API

https://unsplash.com/photos/TL_5Vv1IM-uA

Improve readability or performance without changing behaviour.

https://unsplash.com/photos/qYRMinOIAUF



When you stop work, leave a test failing

https://unsplash.com/photos/y02jEX_B0O0

add_col

Next challenge is to implement add_col()

```
df <- data.frame(x = 1)

add_col(df, "y", 2, where = 1)
add_col(df, "y", 2, where = 2)
add_col(df, "x", 2)
```

Two expectations cover 80% of cases

```
expect_equal(obj, exp)
expect_error(code, regexp)

# You'll learn others throughout the course.
# Complete list at
# http://testthat.r-lib.org/reference
```

Make these tests pass

```r
usethis::use_test("add_col")
# Copy this test:
test_that("where controls position", {
  df <- data.frame(x = 1)

  expect_equal(
    add_col(df, "y", 2, where = 1),
    data.frame(y = 2, x = 1)
  )
  expect_equal(
    add_col(df, "y", 2, where = 2),
    data.frame(x = 1, y = 2)
  )
})
# Run tests with keyboard shortcut
# Some hints on next slide
```

Hint: getting started

```r
usethis::use_r("add_col")

# In R/add_col.R
# Start by establishing basic form of the
# function and setting up the test case.
add_col <- function(x, name, value, where) {

}

# Make sure that you can Cmd + Shift + T
# and get two test failures before you
# continue

# More hints on the next slide
```

Hint: add_col()

```
# You'll need to use insert_into()

# insert_into() takes two data frames and
# you have a data frame and a vector

# setNames() lets you change the names of
# data frame
```

My solution

```
# Lives in R/add_col.R
add_col <- function(x, name, value, where) {
  df <- setNames(data.frame(value), name)
  insert_into(x, df, where = where)
}
```

Make this test pass

```
# add me to test-add_col.R
test_that("can replace columns", {
  df <- data.frame(x = 1)

  expect_equal(
    add_col(df, "x", 2, where = 2),
    data.frame(x = 2)
  )
})
```

My solution

```r
add_col <- function(x, name, value, where) {
  if (name %in% names(x)) {
    x[[name]] <- value
    x
  } else {
    df <- setNames(data.frame(value), name)
    insert_into(x, df, where = where)
  }
}
```

Make this test pass

```r
# add me to test-add_col.R
test_that("default where is far right", {
  df <- data.frame(x = 1)

  expect_equal(
    add_col(df, "y", 2),
    data.frame(x = 1, y = 2)
  )
})
```

1 2 3 4

x	y	z
3.4	1.2	6.7
1.9	6.1	3.1
10.0	2.7	7.7

My solution

```
add_col <- function(x, name, value,
                    where = ncol(x) + 1) {
  if (name %in% names(x)) {
    x[[name]] <- value
    x
  } else {
    df <- setNames(data.frame(value), name)
    insert_into(x, df, where = where)
  }
}
```

Can we use add_col() to **remove** columns?

```
df <- data.frame(x = 1, y = 2)

expect_equal(
  add_col(df, "x", NULL)
  data.frame(y = 2)
)

# Should we? If not, what should add_col()
# do when value is NULL? Would a separate
# remove_col() be a good idea?
```

What if columns are unequal lengths?

```r
# What should happen here?

df <- data.frame(x = 1:4)
add_col(df, "y", 1:2)

# Should it recycle silently?
# Recycle with a warning?
# Throw an error?
```

Can we use add_col() to **move** columns?

```
df <- data.frame(x = 1, y = 2)

expect_equal(
  add_col(df, "x", 1, where = 2)
  data.frame(y = 2, x = 2)
)

# Should we?
# Would move_col() be better?
```

How should we name this collection of functions?

```
# Prefix?
add_col()
move_col()
remove_col()

# Suffix?
col_add()
col_remove()
col_move()
```

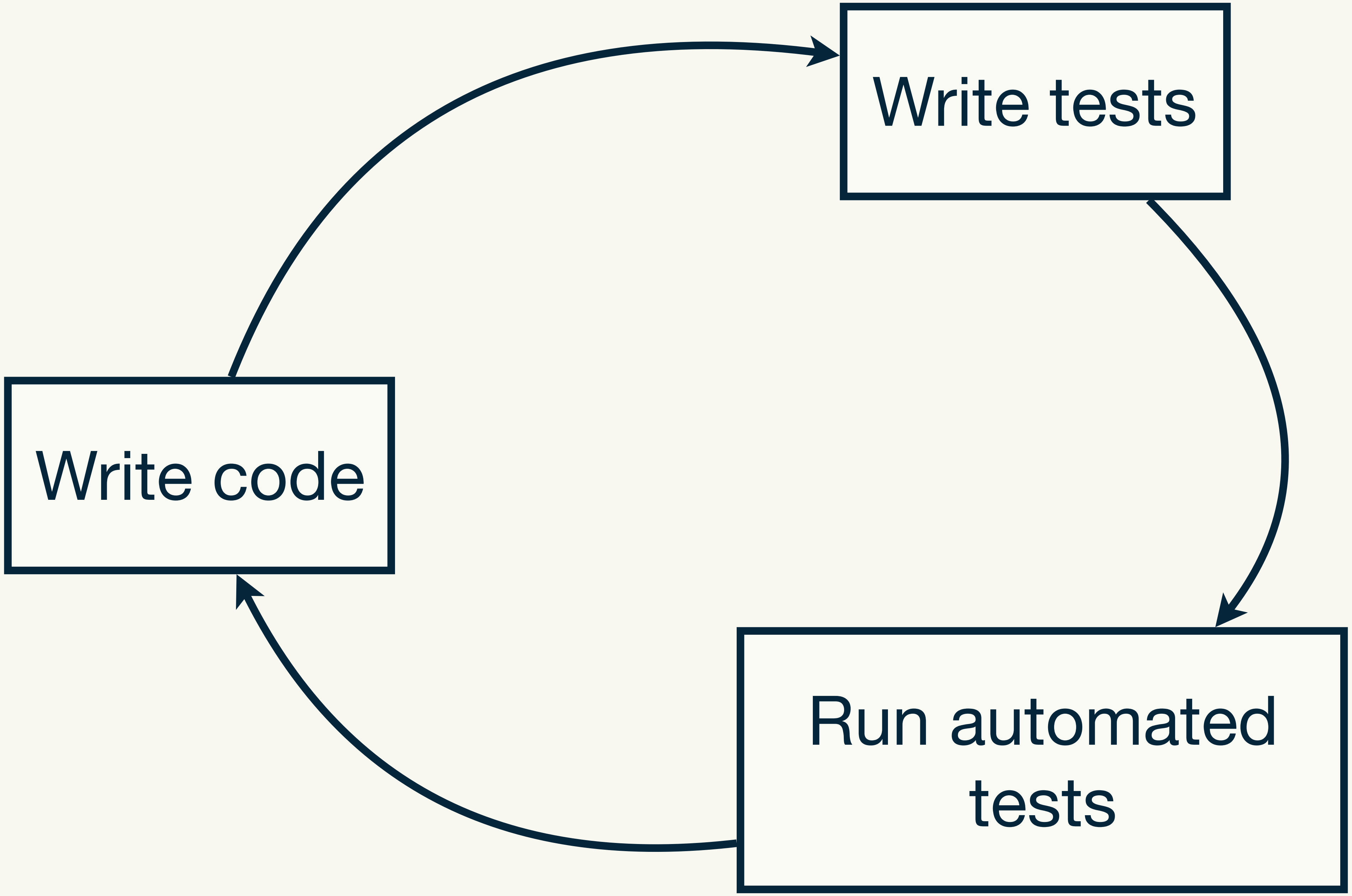
What about bad inputs?

```r
# We need to test for errors too

df1 <- data.frame(a = 3, b = 4, c = 5)
df2 <- data.frame(X = 1, Y = 2)

insert_into(df1, df2, where = 0)
insert_into(df1, df2, where = NA)
insert_into(df1, df2, where = 1:10)
insert_into(df1, df2, where = "a")
```

We'll return to this tomorrow…

Test driven development

So far we've written code, then tests

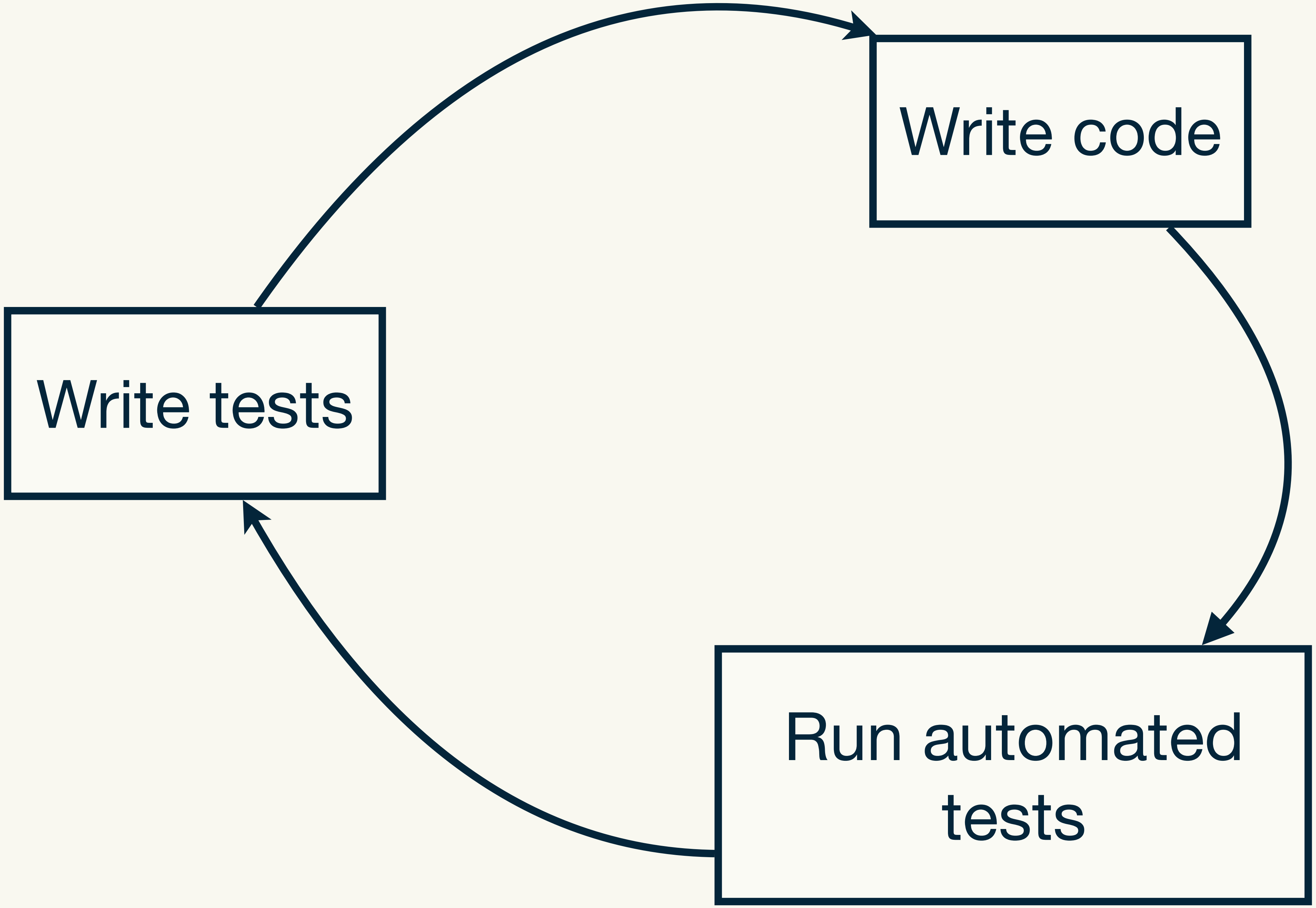


Write tests

Run automated tests

Write code

What happens if we write the tests first?



Test driven development

Test coverage

Test coverage shows you what you’ve tested

```
devtools::test_coverage()
```

Adapted from *Tidy Tools* by Hadley Wickham