Unit testing

(With a dash of API design)

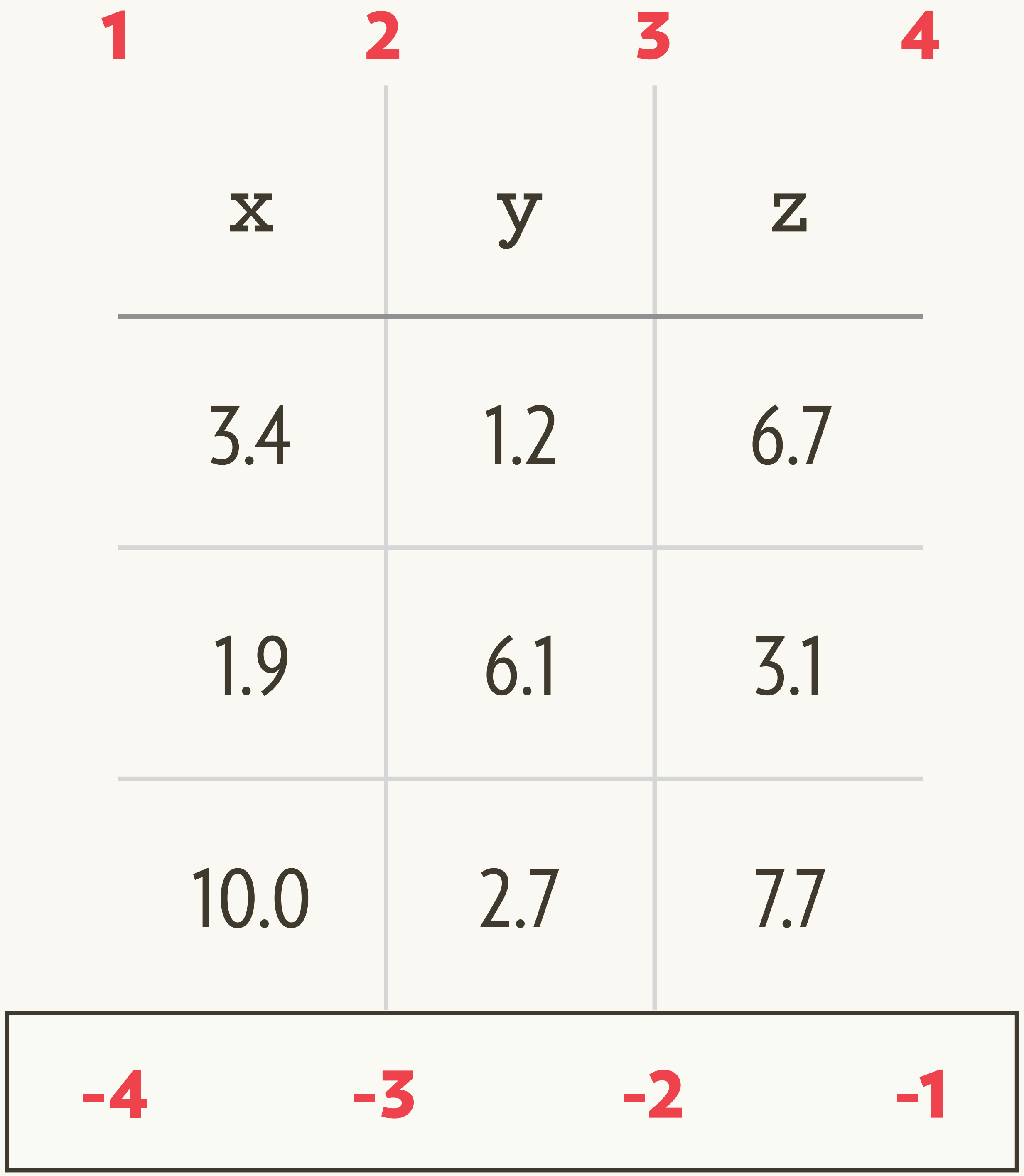
May 2018

Hadley Wickham
@hadleywickham
Chief Scientist, **RStudio**

Motivation

Let's add a column to a data frame

```
# 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
```



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

Your turn

```r
# A useful building block is add_cols() -
# works like cbind() but can insert anywhere

add_cols <- function(x, y, where = 1) {
  if (where == 1) { # first col
    ...
  } else if (where > ncol(x)) { # last col
    ...
  } else {
    ...
  }
}
```

My first attempt

```
add_cols <- 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)])
  }
}
```

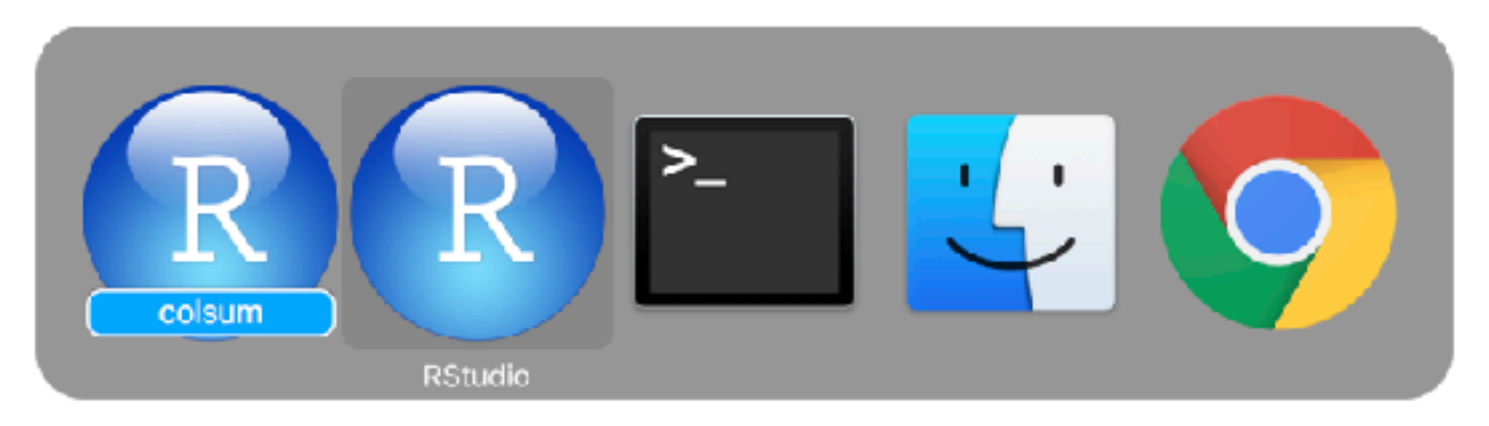
Actually correct

```
add_cols <- 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])
  }
}
```

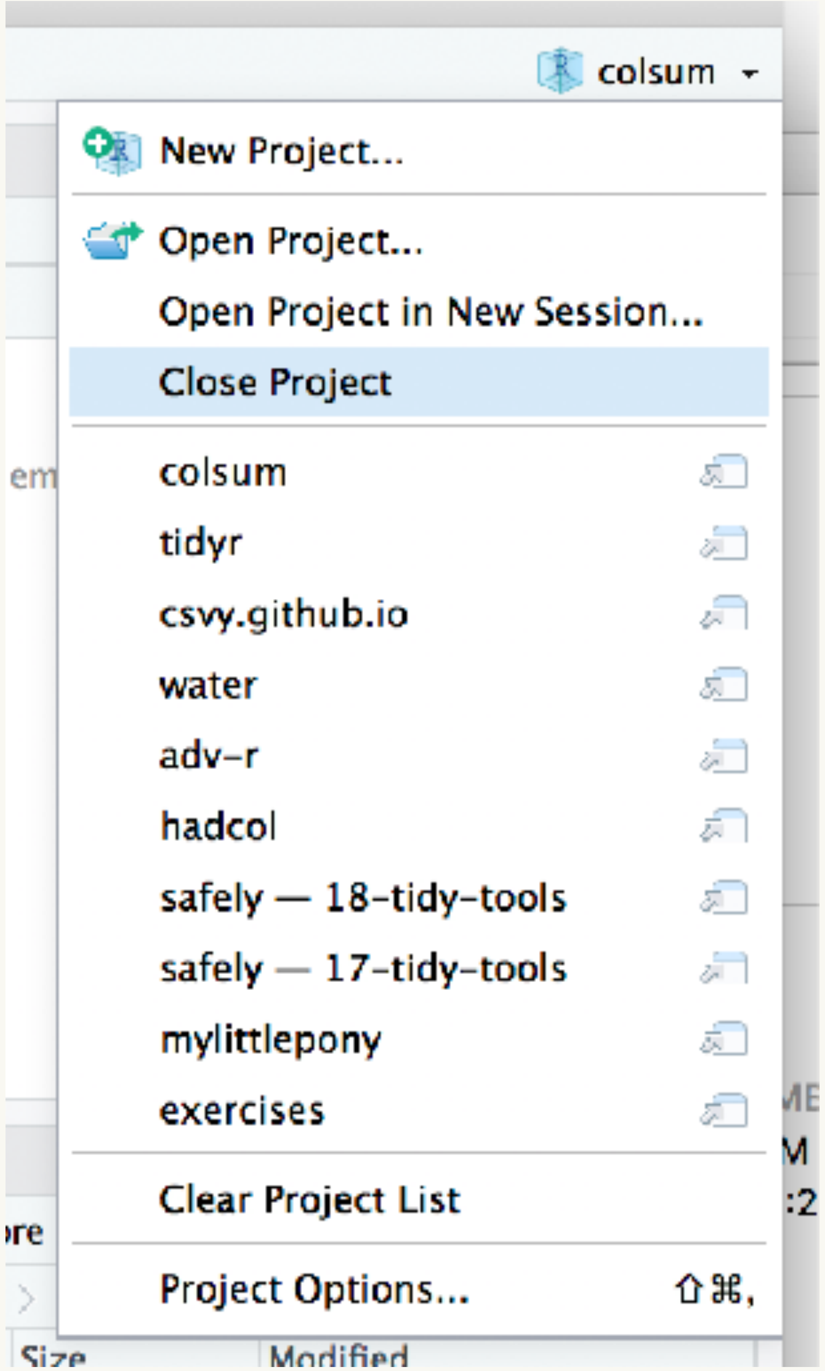
How did I write that code?

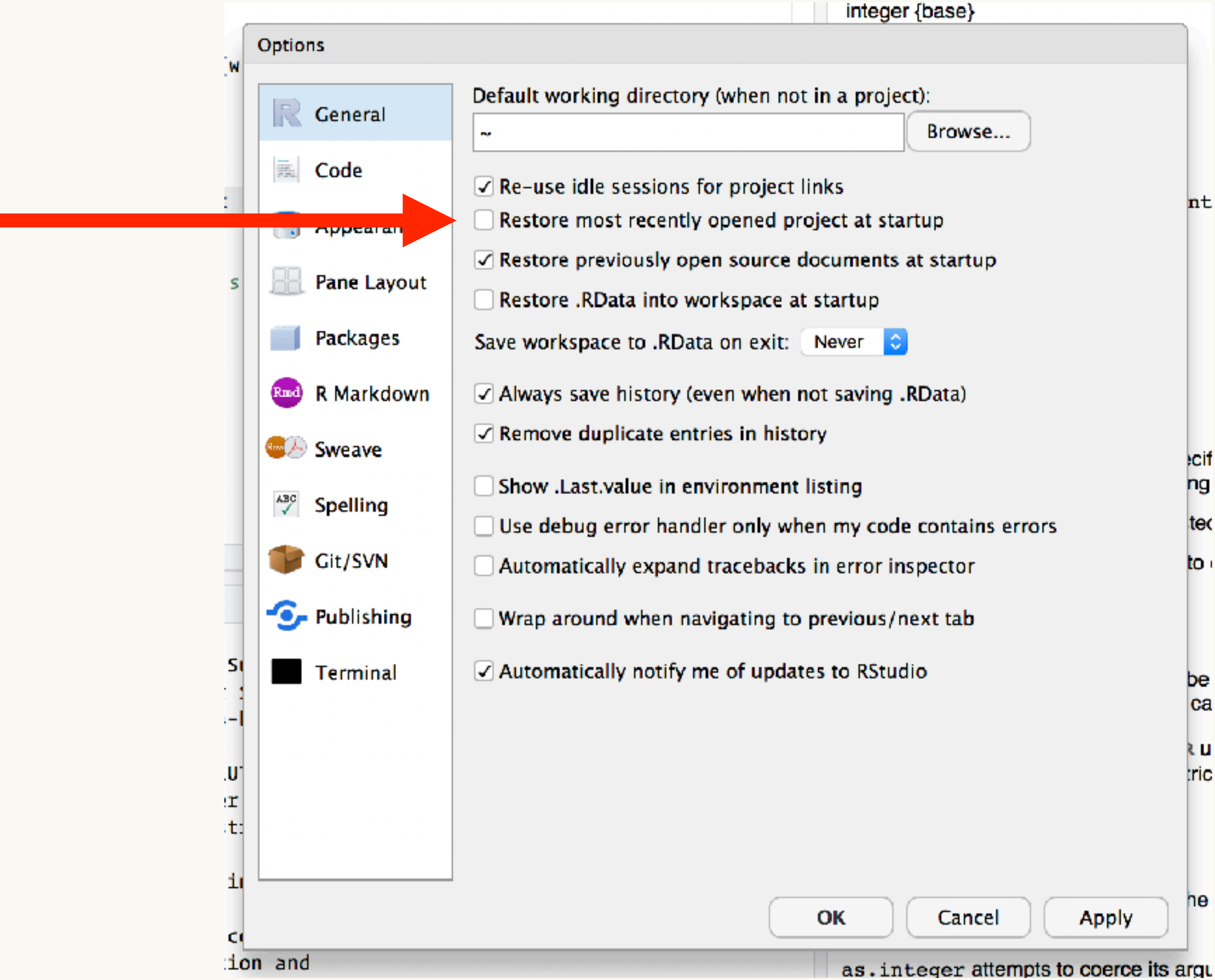
```r
# Some simple inputs
df1 <- data.frame(a = 3, b = 4, c = 5)
df2 <- data.frame(X = 1, Y = 2)

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

```r
usethis::create_package("~/desktop/hadcol")
usethis::use_r("add_cols")
```

```r
add_cols <- 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])
  }
}
```

Even more convenient with some conventions

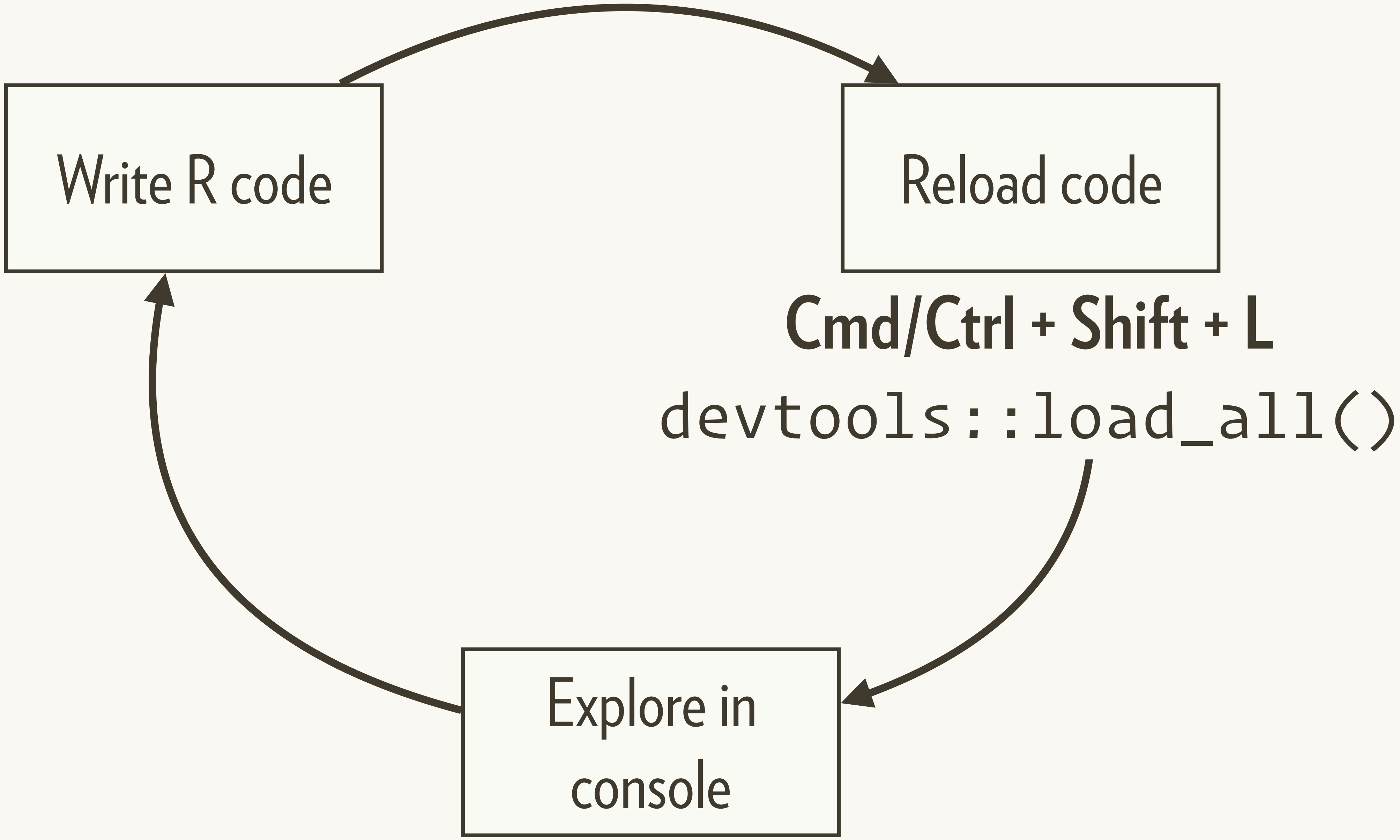
Set up testthat infrastructure

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

Create test file matching script

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

So far we've done this:



Write R code

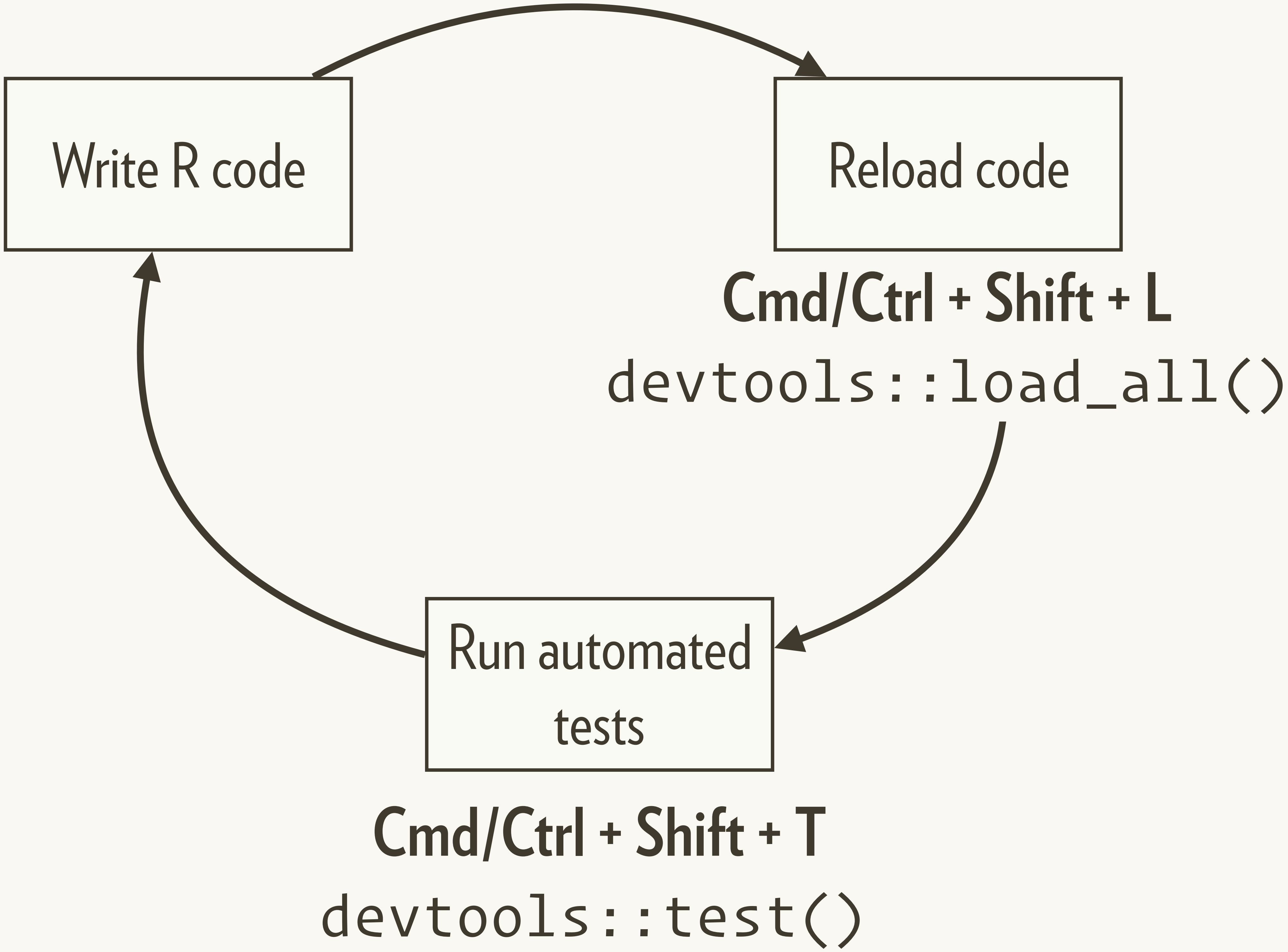
Reload code

Cmd/Ctrl + Shift + L

`devtools::load_all()`

Explore in console

Testthat gives a new workflow



Write R code

Reload code

Cmd/Ctrl + Shift + L

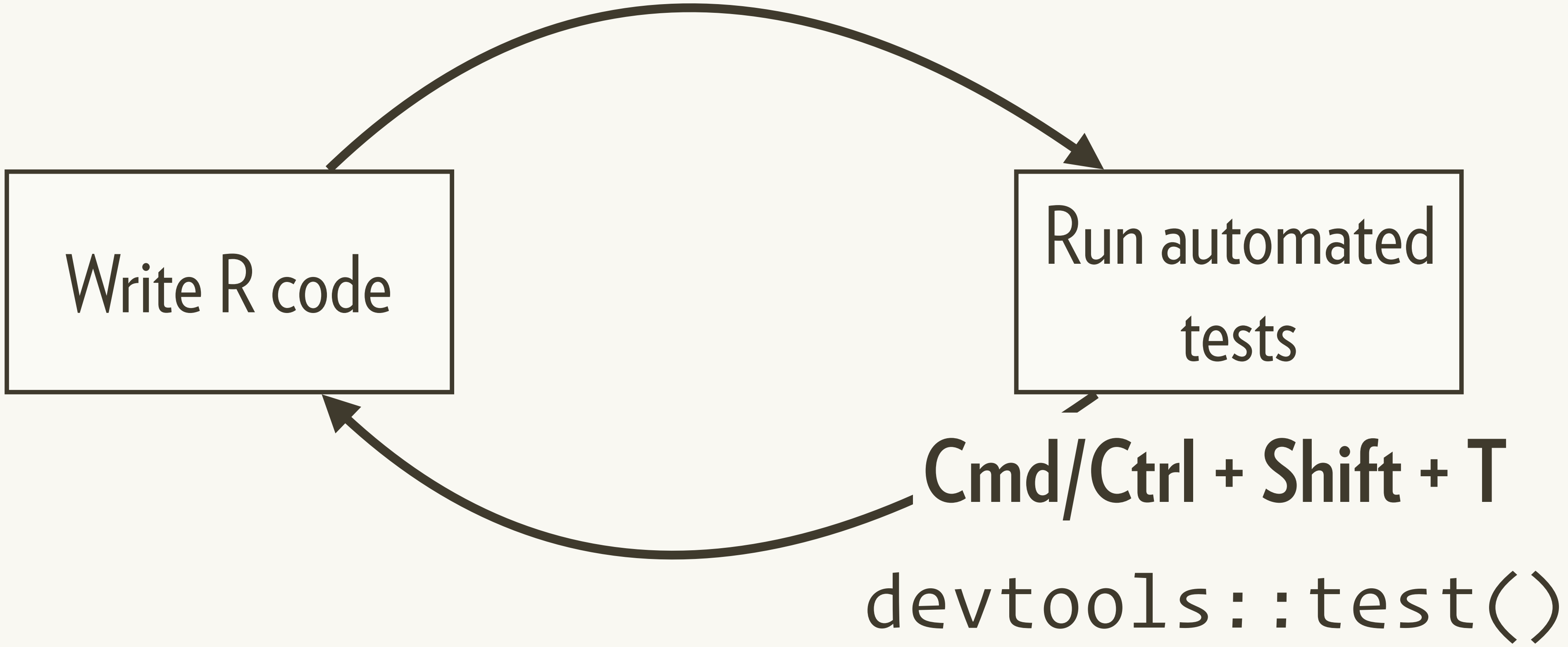
`devtools::load_all()`

Run automated tests

Cmd/Ctrl + Shift + T

`devtools::test()`

But why load the code?



Write R code

Run automated tests

Cmd/Ctrl + Shift + T

```
devtools::test()
```

Key idea of unit testing is to automate!

Helper function to reduce duplication

```
at_pos <- function(i) {
  add_cols(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"))
```

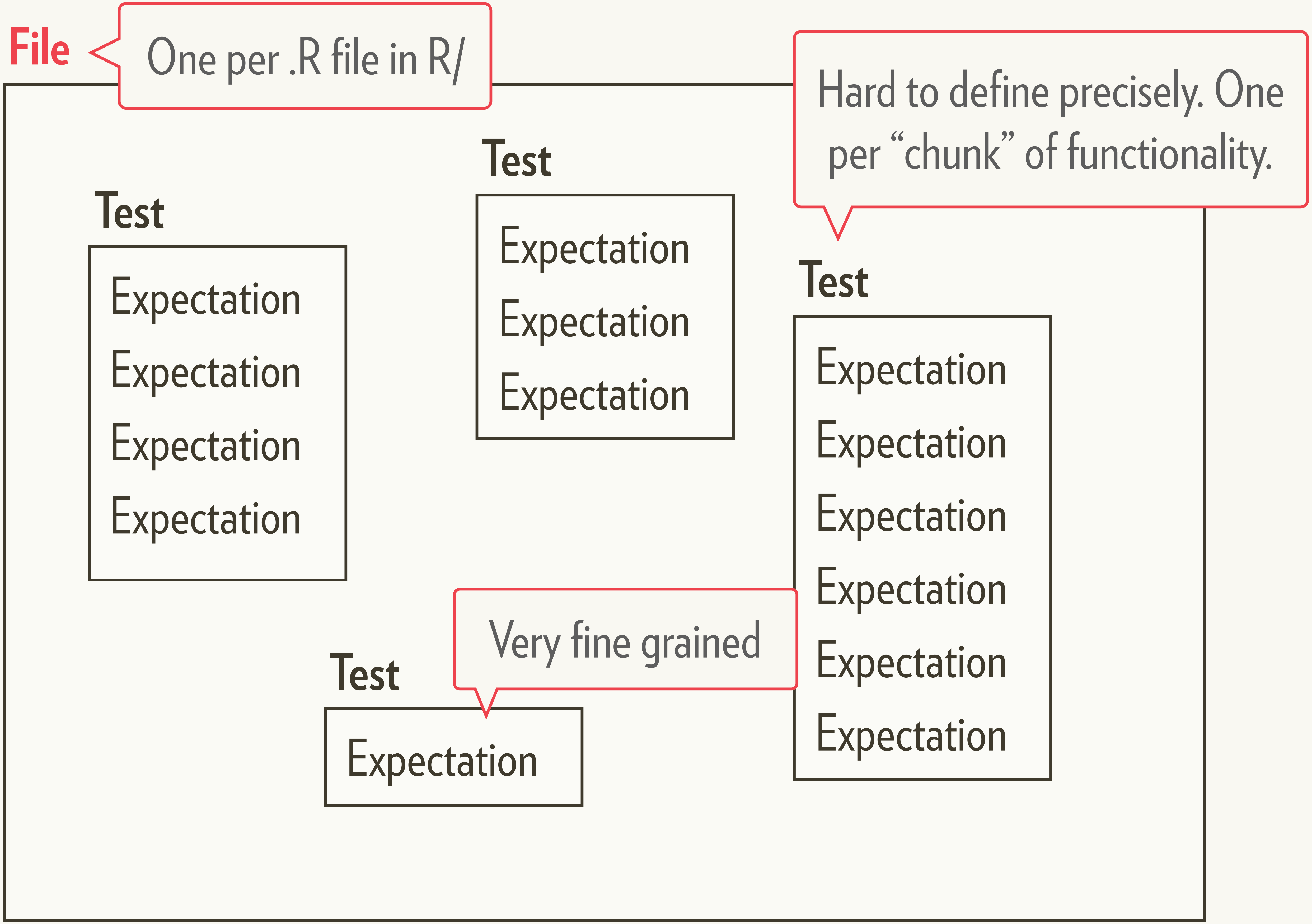
Describes an expected property of the output

And this automation n ventions

Tests for R/add_cols.R

```
# In tests/testthat/test-add_cols.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) {
    add_cols(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/

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

Very fine grained

Test

Expectation
Expectation
Expectation
Expectation

Test

Expectation
Expectation
Expectation

Test

Expectation
Expectation
Expectation
Expectation
Expectation
Expectation

Test

Expectation

Practice the workflow

```
usethis::create_package("~/desktop/hadcol")
usethis::use_r("add_col")
# Copy add_cols() from slides
# Check all is ok with load_all()
usethis::use_test()
# Copy expectations from slides
# Run tests with keyboard shortcut
# Confirm that if you break add_cols() the
# tests fail.
```

You should now be in freshly created

[hadcol]

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

Other advantages

Writing tests improves your API

https://unsplash.com/photos/TL5Vy1lM-uA

Improve readability or performance without changing behaviour.

https://unsplash.com/photos/gYBMinOIAUE

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

```r
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

```
# use_test("add_col")
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)
  )
})
# Some hints on next slide
```

Work through up to fail fast

resume at 12:10pm

Hints

```
# 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
```

More hints

```
# You'll need to use add_cols

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

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

My solution

```
add_col <- function(x, name, value, where) {
  df <- setNames(data.frame(value), name)
  add_cols(x, df, where = where)
}
```

Make this test pass

```
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

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

Make this test pass

```
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)
    add_cols(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?

```r
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?

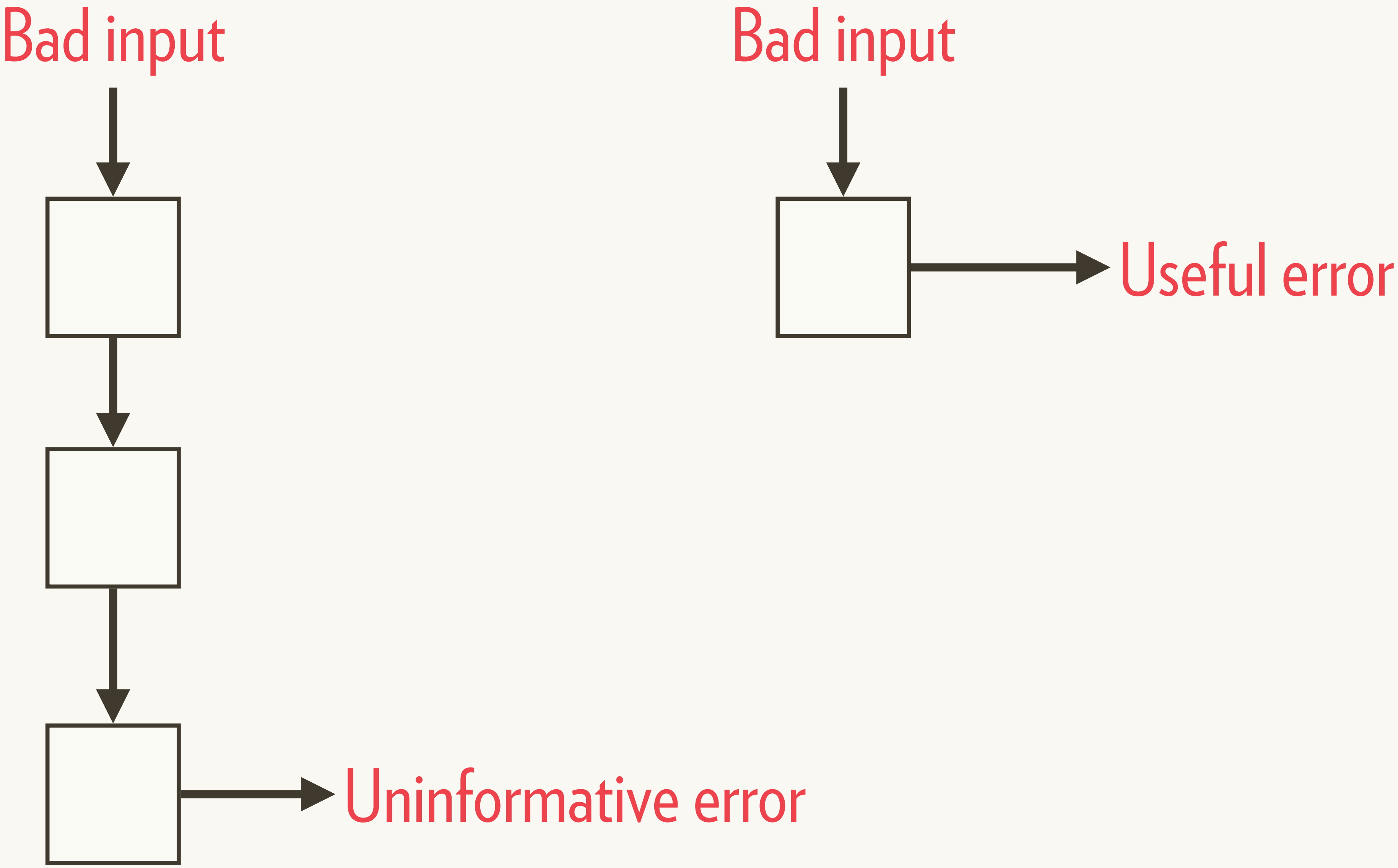
```
# Prefix?
add_col()
move_col()
remove_col()

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

Fail fast

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)

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

For robust code, fail early



We *could* add to add_cols directly

```
add_cols <- function(x, y, where = 1) {
  if (!is.numeric(where) || length(where) != 1) {
    stop("`where` is not a number", call. = FALSE)
  } else if (where == 0 || is.na(where)) {
    stop("`where` must not be 0 or NA", call. = FALSE)
  } else if (where == 1 || where <= -ncol(x)) {
    cbind(x, y)
  } else if (where >= ncol(x) || where == -1) {
    cbind(y, x)
  } else {
    if (where < 0) where <- nrow(x) + where
    cbind(x[1:where], y, x[where:nrow(x)])
  }
}
```

But this confuses the intent of add_cols

```
# Better to have one function responsible
# for checking for valid inputs

check_where <- function(where) {
  ...
}

# This also makes it easier to test because
# it's independent of add_cols
```

Your turn

Back at 1:30pm

```
# Write down the error message that you think
# each of these lines should generate

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

Error message structure

1. Problem statement
   (use must or can’t)
2. Error location
   (where possible)
3. Hint
   (if common)

http://style.tidyverse.org/error-messages.html

Punctuation

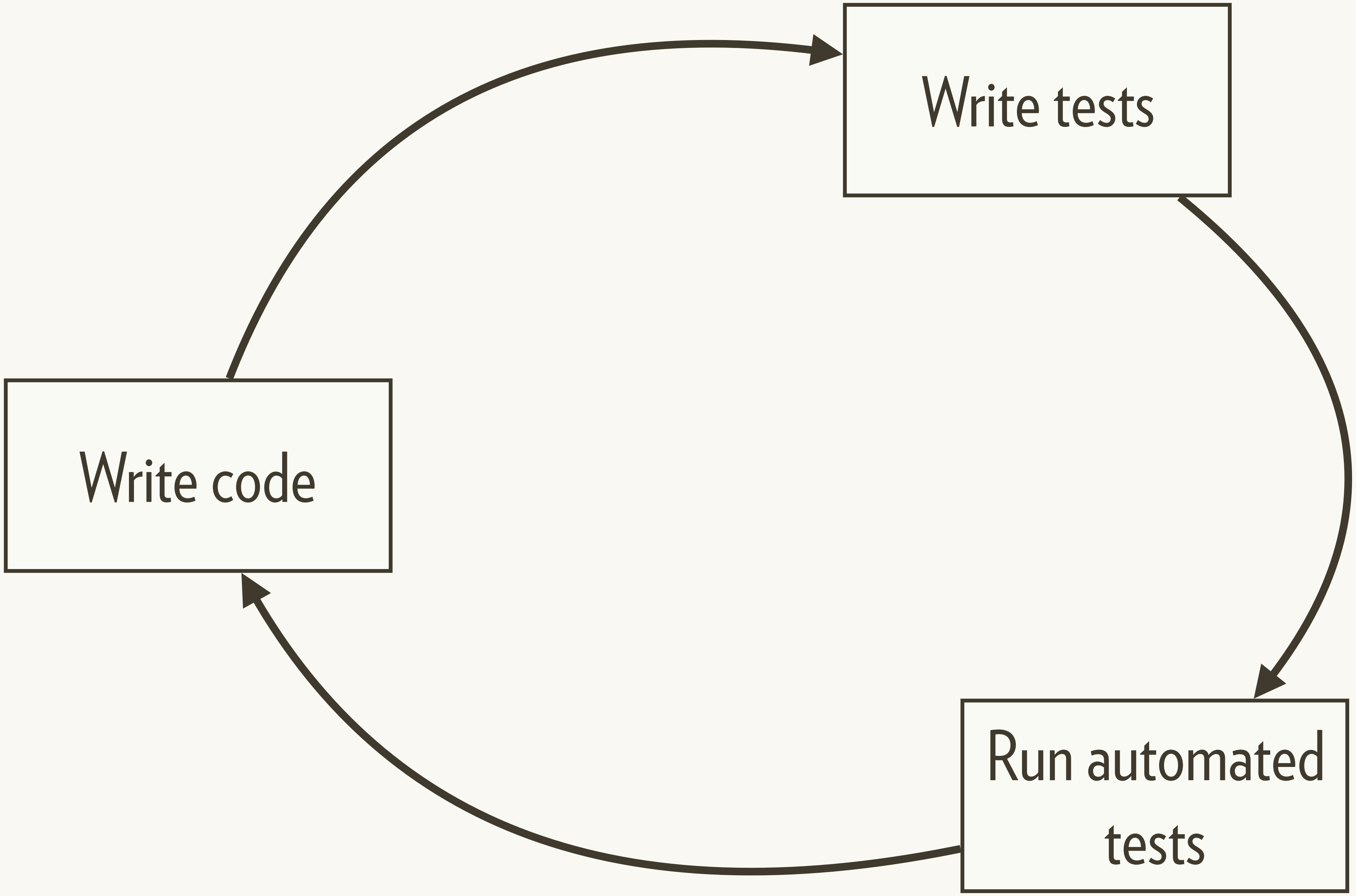
- Always use `call. = FALSE`
- Surround variable names in \`...\`, and strings in '...'
- Sentence case

My results

```
check_where(0)
#> Error: `where` must not be zero or missing.
check_where(NA)
#> Error: `where` must not be zero or missing.
check_where(1:10)
#> Error: `where` must be a length one numeric vector.
check_where("a")
#> Error: `where` must be a length one numeric vector.
```

Test driven development

So far we’ve written code, then tests

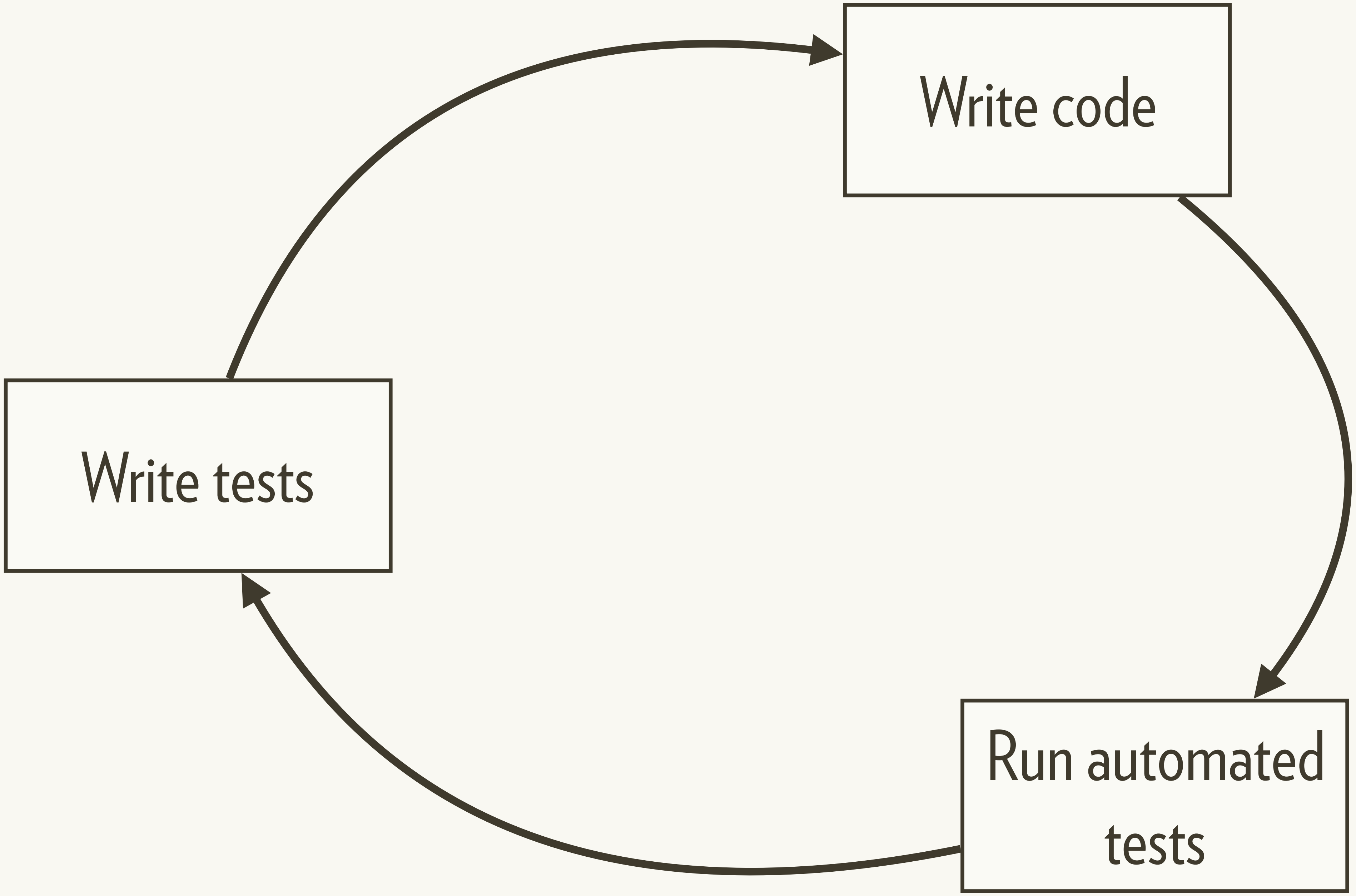


Write tests

Run automated tests

Write code

What happens if we write the tests first?



Write code

Run automated tests

Write tests

Use expect_error() to test for errors

```
expect_error(
  check_where("a")
)

expect_error(
  check_where("a"),
  "not a number"
)
```

A regular expression

Your turn

Write tests to ensure that check_where() only allows valid inputs. (Where should the tests live? How many tests do you need? How many expectations?)

My tests

```
# I think should live in tests/testthat/test-add_cols.R

test_that("where must be valid value", {
  expect_error(check_where("a"), "length one numeric vector")
  expect_error(check_where(1:10), "length one numeric vector")

  expect_error(check_where(0), "not be zero or missing")
  expect_error(check_where(NA), "not be zero or missing")
})
```

Your turn

Write check_where(). It should throw an error if the input is incorrect. I suggest you put in the same file as add_cols().

```
check_where(0)
check_where(NA)
check_where(1:10)
check_where("a")
```

My answer

```r
check_where <- function(x) {
  if (length(x) != 1 || !is.numeric(x)) {
    stop("`where` must be a length one numeric vector.", call. = FALSE)
  }
  x <- as.integer(x)

  if (x == 0 || is.na(x)) {
    stop("`where` must not be zero or missing", call. = FALSE)
  } else {
    x
  }
}
```

My tests

```
# check_where() lives in same file as add_cols()
# so tests should live in test-add_cols()

test_that("where must be valid value", {
  expect_error(check_where("a"), "length one numeric vector")
  expect_error(check_where(1:10), "length one numeric vector")

  expect_error(check_where(0), "not be zero or missing")
  expect_error(check_where(NA_real_), "not be zero or missing")
})
```

Test coverage

Test coverage shows you what you've tested

```
# Development version of devtools
devtools::test_coverage()

library(covr)
report(package_coverage())
```