

Unidades de prueba

(Con un poco de diseño de APIs)

Julio 2019

Traducido por Leonardo Collado-Torres

@fellgernon

lcolladotor@gmail.com

lcolladotor.github.io

Desarrollado por Charlotte Wickham para rstudio::conf(2019)

@cvwickham

cwickham@gmail.com

cwick.co.nz

1

Adapted from *Tidy Tools* by Hadley Wickham



Motivación

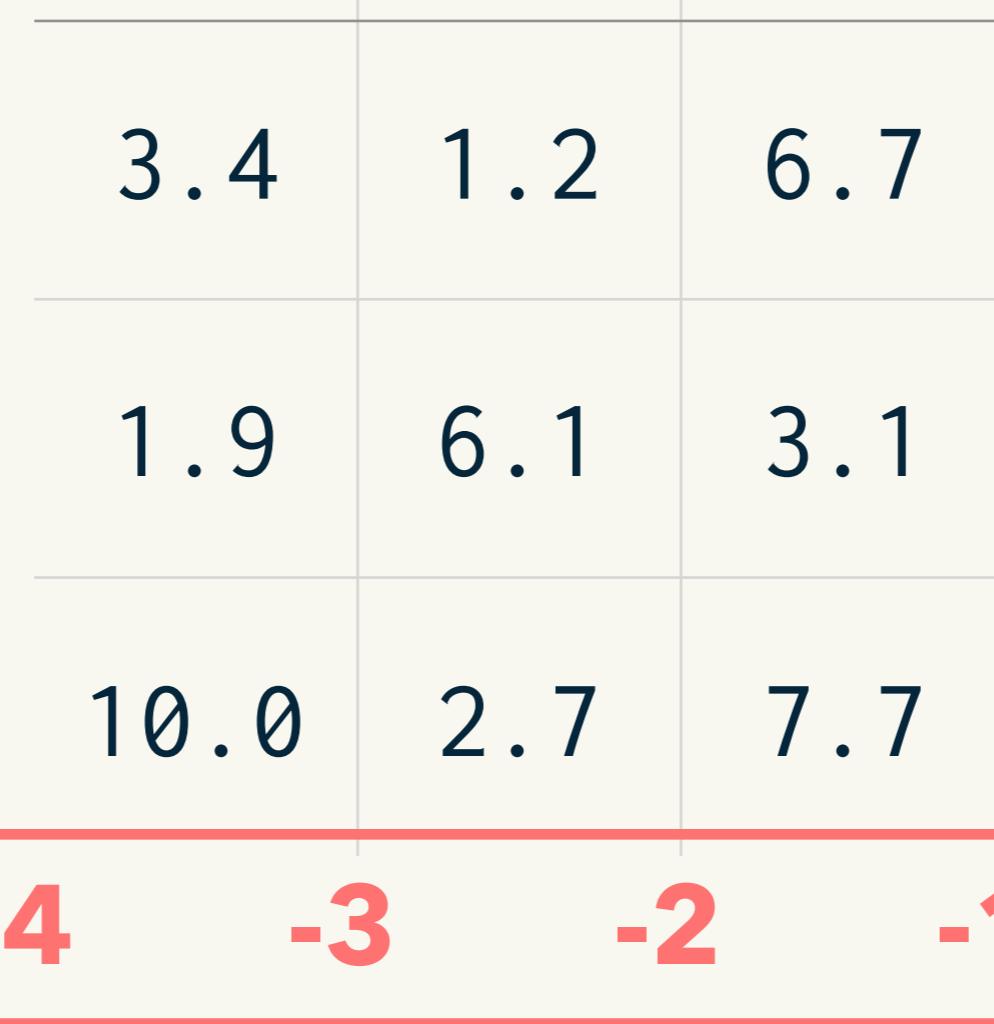
Agregemos una columna a un data frame

```
# Objetivo:  
# Escribir una función que nos permita  
# agregar una nueva columna a un  
# data frame en la posición que especifiquemos  
  
add_col(df, "name", value, where = 1)  
add_col(df, "name", value, where = 2)  
  
# Empecemos simple y probémosla mientras  
avanzamos
```

where =

1 2 3 4

↓ ↓ ↓ ↓
x **y** **z**



Estaría bien tener esta parte, pero no la implementaremos hoy

Empecemos con insert_into()

Funciona como cbind() pero podemos insertar en donde sea

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

Agrega las columnas de df2 a df1 en la posición where

Tu turno

¿Qué va en ...?

```
# Pista: cbind() será útil
# Agrega las columnas de df2 a df1 en la posición where
insert_into <- function(x, y, where = 1) {
  if (where == 1) { # primera columna
    ...
  } else if (where > ncol(x)) { # última columna
    ...
  } else {
    ...
  }
}
```

Mi primer intento

```
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)])  
  }  
}
```

Función correcta

```
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])  
  }  
}
```

¿Cómo escribí este código?

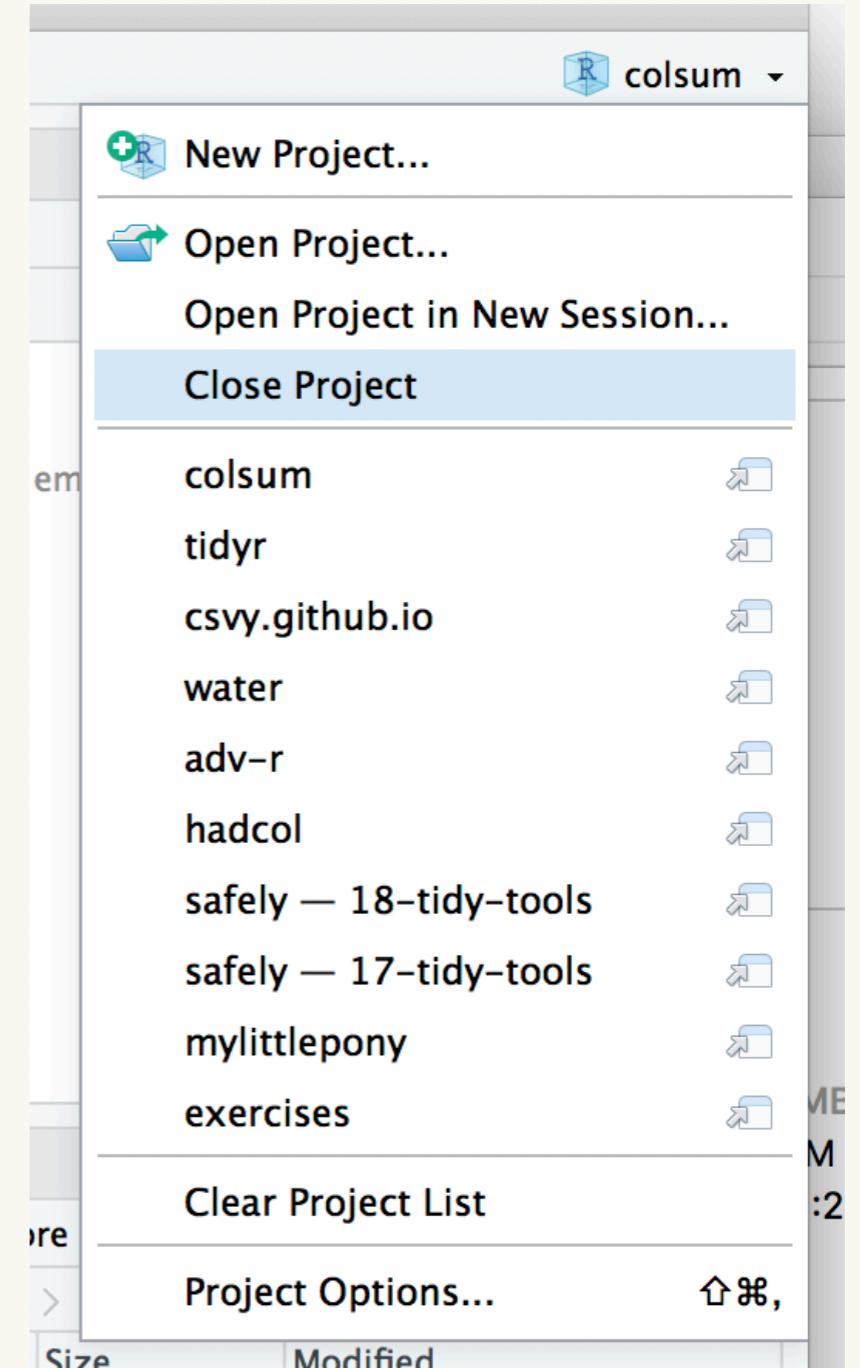
```
# Unos valores de prueba
df1 <- data.frame(a = 3, b = 4, c = 5)
df2 <- data.frame(X = 1, Y = 2)

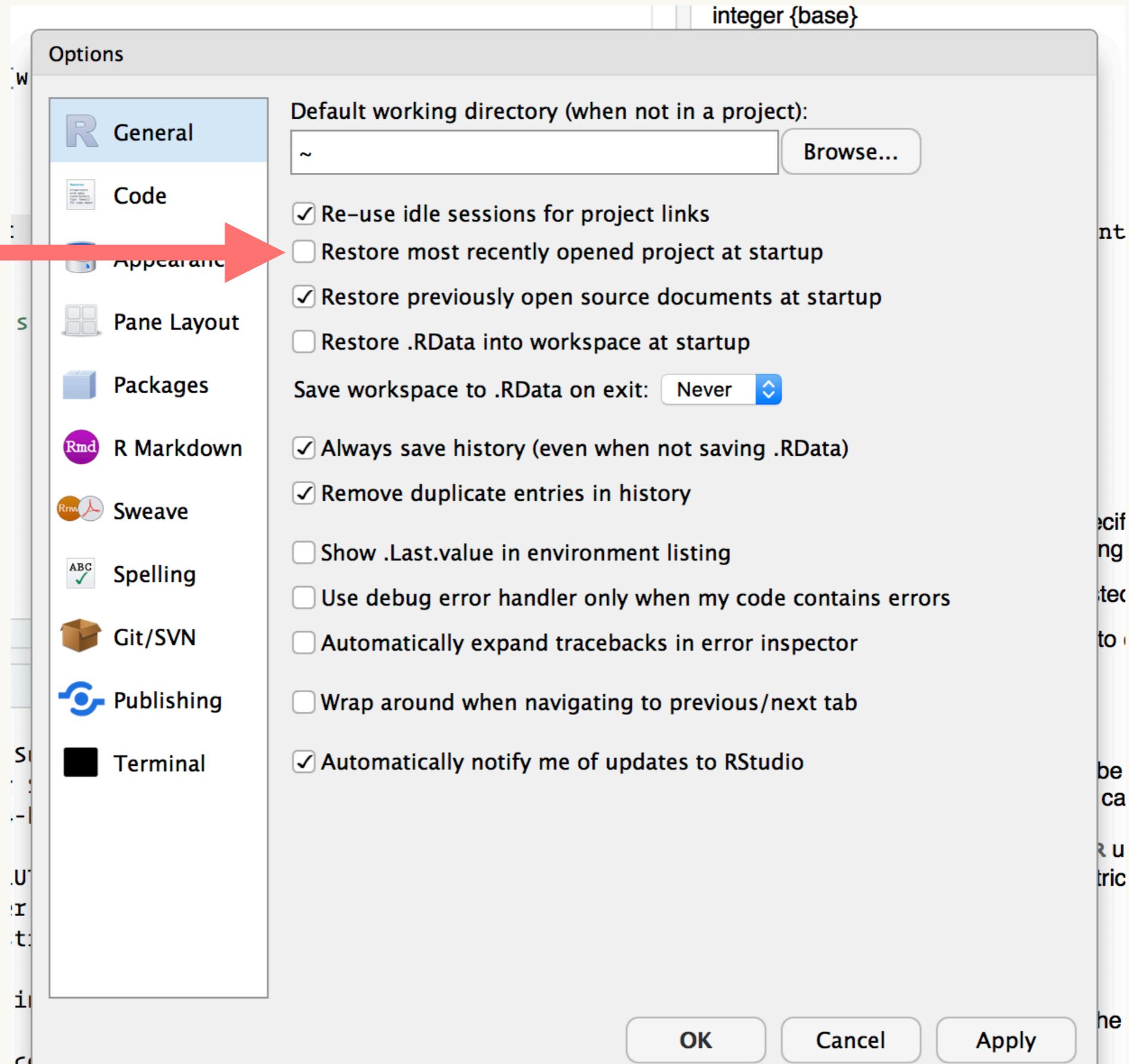
# Después cada vez que la modificaba,
# volví a correr los siguientes casos
insert_into(df1, df2, where = 1)
insert_into(df1, df2, where = 2)
insert_into(df1, df2, where = 3)
insert_into(df1, df2, where = 4)
```

¿Cómo escribí este código?



Al igual que RStudios asociados a algún proyecto, también podemos obtener uno asociado a ningún proyecto





ion and

as.integer attempts to coerce its argu

Dos desafíos

Cmd + Enter es susceptible a errores

**Ver todos los resultados de las
diferentes iteraciones es tedioso**

We need a new workflow!

Cmd + Enter es susceptible a errores

Pon el código en R/ y usa devtools::**load_all()**

Ver todos los resultados de las diferentes iteraciones es tedioso

Escribe unidades de prueba y usa devtools::**test()**

Flujo de trabajo para pruebas

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

Primera, crea un paquete

```
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])
  }
}
```

copia y pega esto
en
insert_into.R

Luego, configura la infraestructura para pruebas

```
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'

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

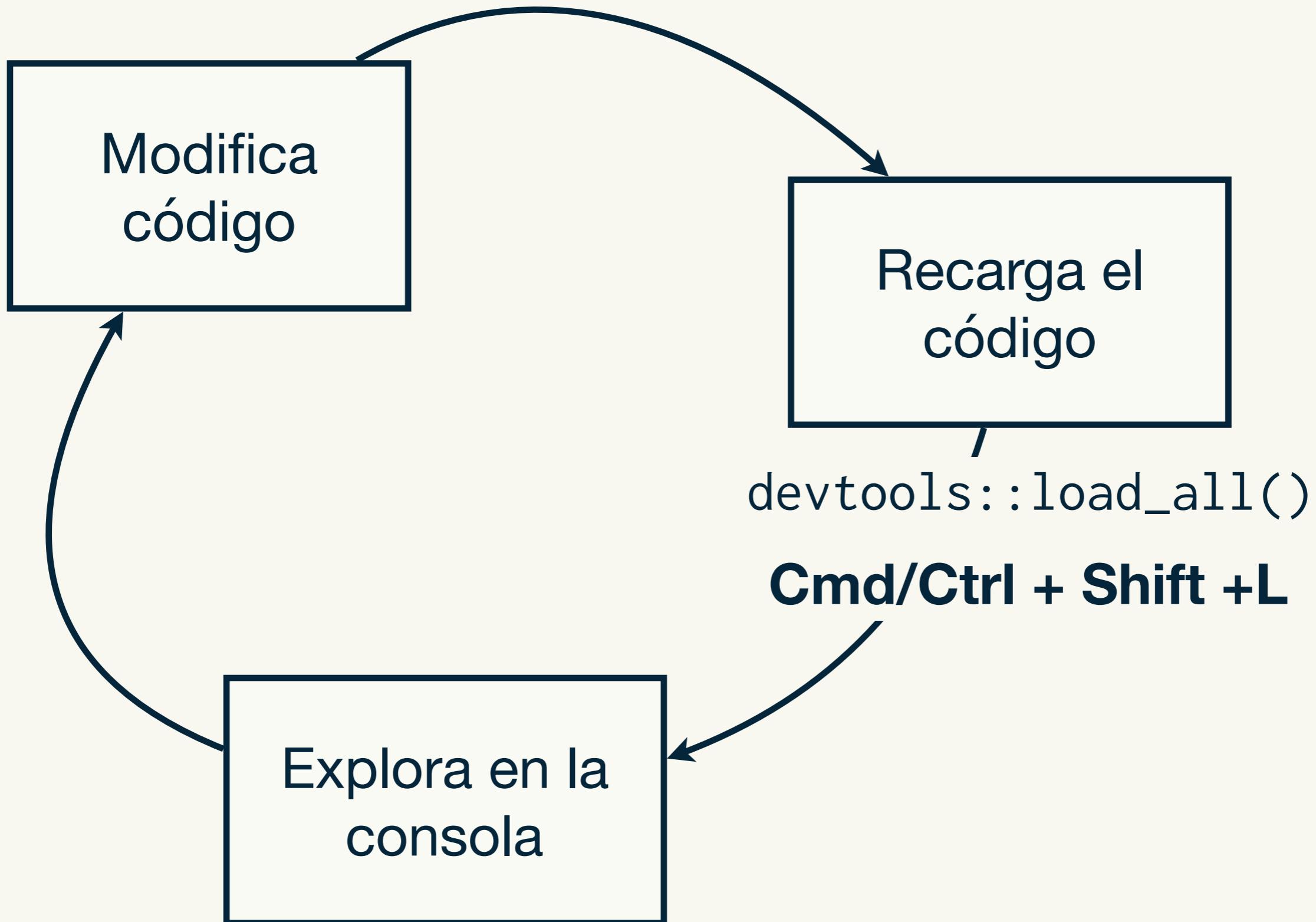
```
# ⌘ ⌥ ⌘ Command + Shift + T
```

Corre las

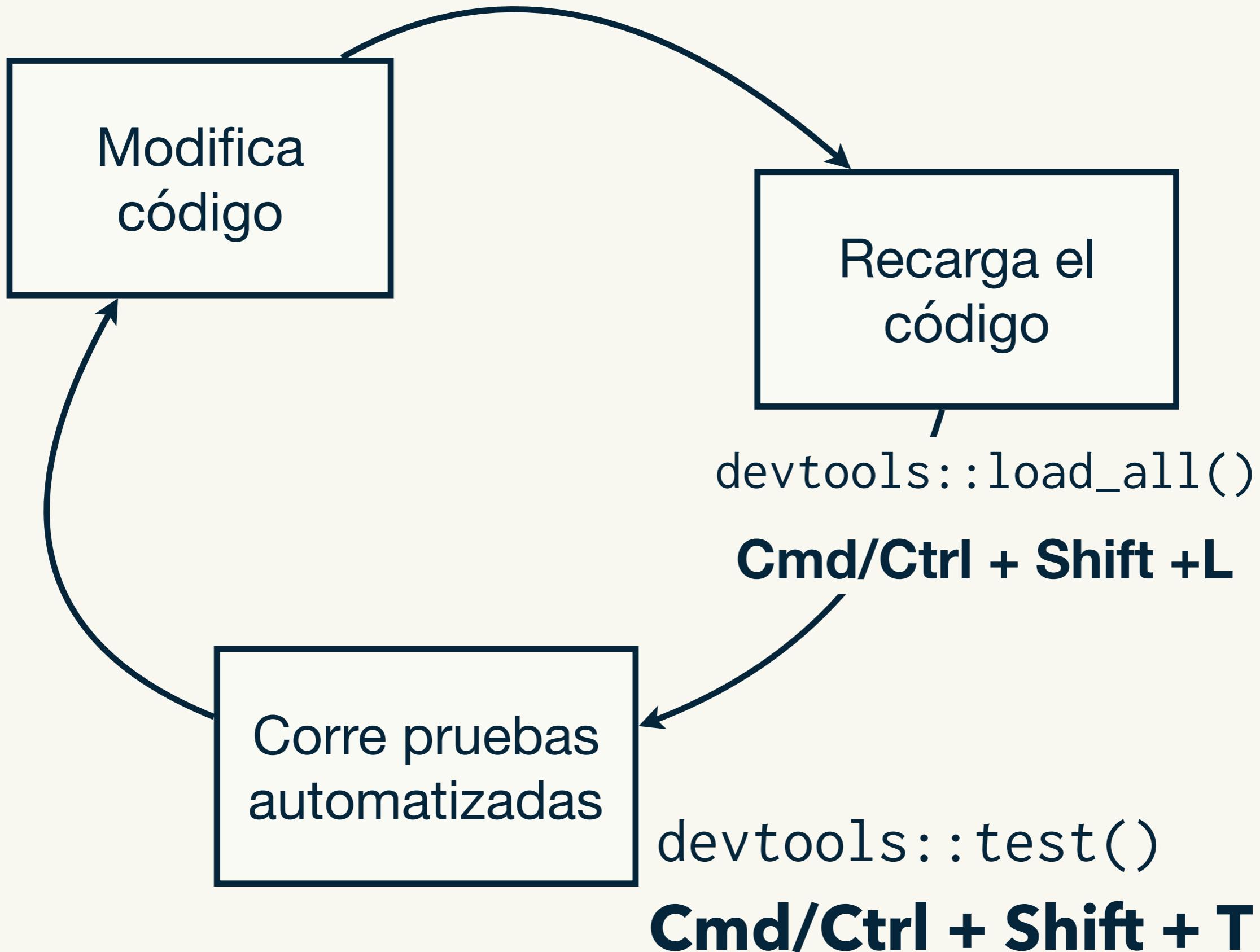
Configura la estructura de test

Crea un archivo de pruebas basado en el script

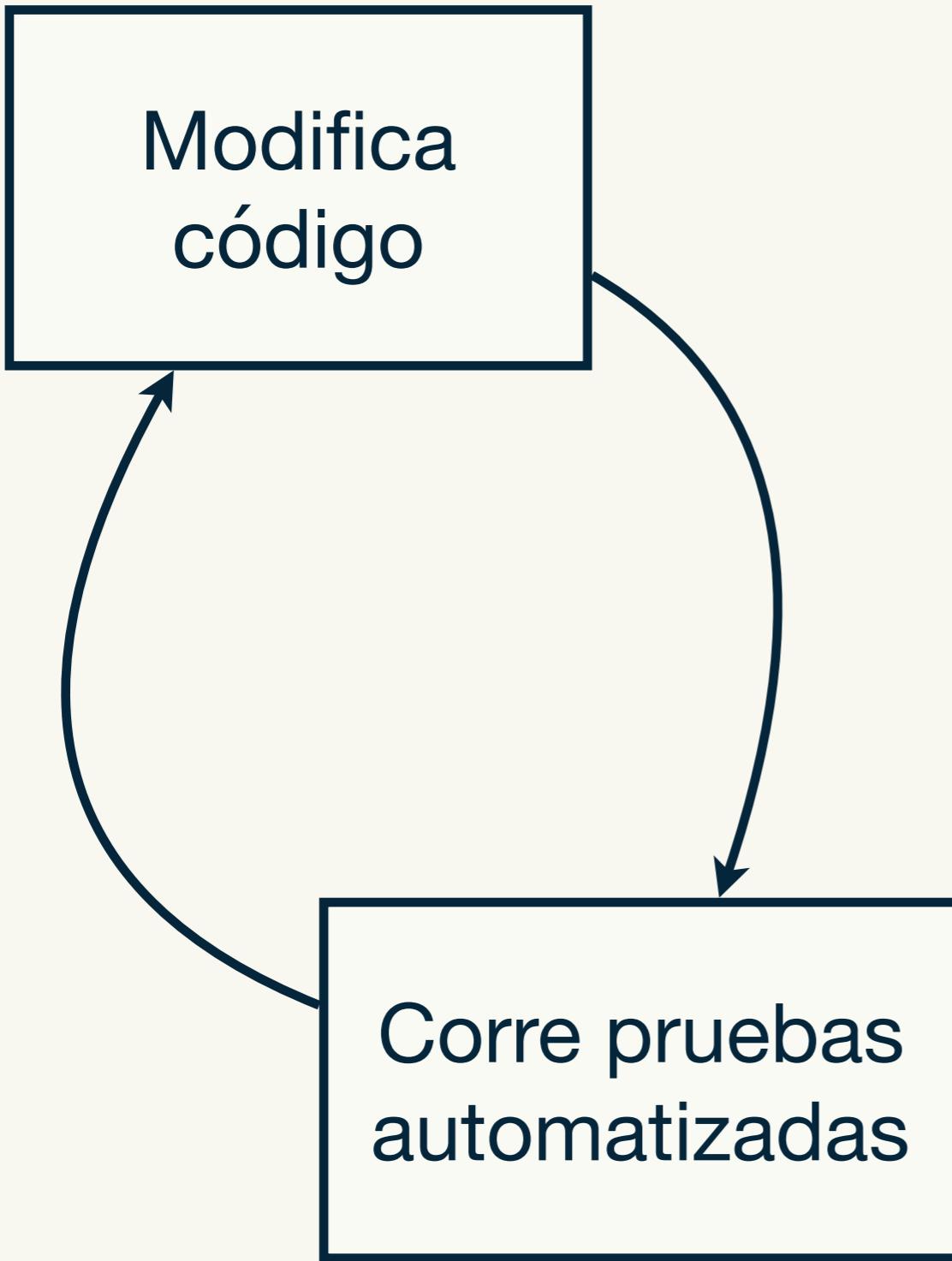
Hasta ahora hemos echo esto:



Testthat nos provee un nuevo flujo de trabajo



¿Pero por qué recargamos el código?



`devtools::test()`

Cmd/Ctrl + Shift + T

¡Idea clave de las unidades de pruebas es automatizar!

Función ayudante para
reducir duplicaciones

```
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"))
```

Describe una propiedad
esperada

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"))
})
```

Pruebas están organizadas en tres capas

Archivo

Uno por archivo.R en

Prueba

Expectación
Expectación
Expectación
Expectación

Difícil de
definir
precisamente.

Uno por grupo de

Prueba

Expectación
Expectación

Prueba muy
detallada

Prueba

Expectación

Practica este flujo de trabajo

```
usethis::create_package("~/Desktop/hadcol")  
  
usethis::use_r("insert_into")  
# Copia insert_into() de la siguiente diapositiva  
# Checa que todo esté bien con load_all()  
  
usethis::use_test()  
# Copia valores esperados de la próxima  
diapositiva  
# Corre pruebas con atajos del teclado  
# Confirma que si rompes insert_into()  
# entonces las pruebas fallan.
```

insert_into()

```
# En 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])
  }
}
```

Valores esperados (expectaciones)

```
# Dentro de tests/testthat/test-insert_into.R
context("prueba-insert_into")

test_that("podemos agregar una columna en cualquier posición", {
  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)
  }

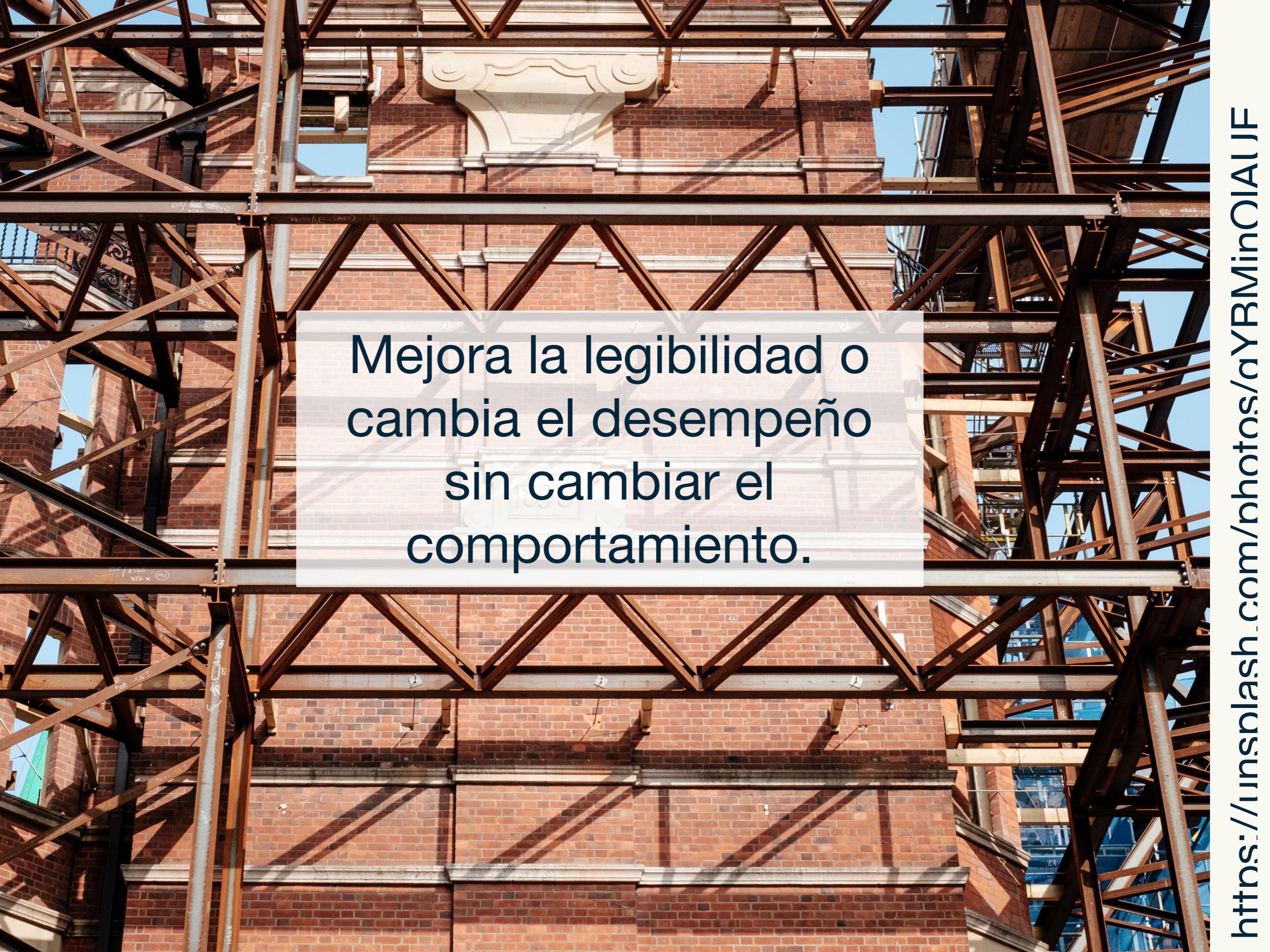
  expect_named(at_pos(1), c("X", "Y", "a", "b", "c"))
  expect_named(at_pos(2), c("a", "X", "Y", "b", "c"))
  expect_named(at_pos(3), c("a", "b", "X", "Y", "c"))
  expect_named(at_pos(4), c("a", "b", "c", "X", "Y"))
})
```

Deberías estar en el paquete que
acabamos de crear

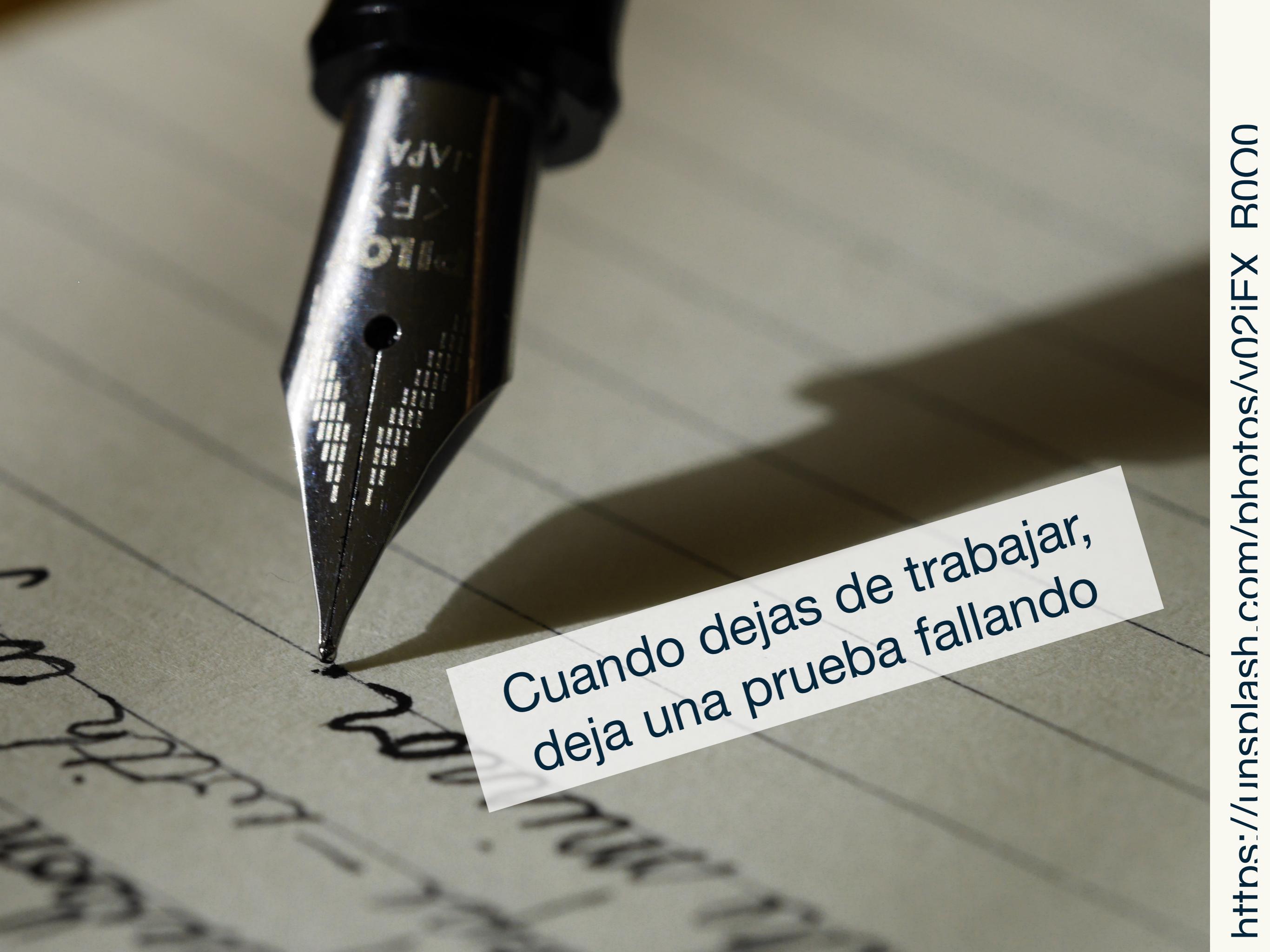
[hadcol]

(El curso también viene con hadcol-
test si te atoraste)

Otras ventajas



Mejora la legibilidad o
cambia el desempeño
sin cambiar el
comportamiento.



Cuando dejas de trabajar,
deja una prueba fallando

add_col

Siguiente reto es implementar 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)
```

Dos expectaciones cubren el 80% de los casos

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

```
# Aprenderás más durante el curso.  
# La lista completa está en:  
# http://testthat.r-lib.org/reference
```

Haz que estas pruebas pasen

```
usethis::use_test("add_col")
# Copia esta prueba:
test_that("where controla la posición", {
  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)
  )
})
# Corre las pruebas con los atajos del teclado
# Algunas pistas en la siguiente diapositiva
```

Pista: empezando el proceso

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

```
# En R/add_col.R
```

```
# Comienza por establecer la forma básica de  
# la función y establece el caso de prueba.
```

```
add_col <- function(x, name, value, where) {
```

```
}
```

```
# Asegúrate de que puedas correr Cmd + Shift + T  
# y ver dos pruebas fallidas antes de continuar
```

```
# Más pistas en la próxima diapositiva
```

Pistas: add_col()

```
# Necesitarás usar insert_into()  
  
# insert_into() toma dos data frames y  
# tu tienes un data frame y un vector  
  
# setNames() te permite caminar los nombres  
# de un data frame
```

Mi solución

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

Haz que esta prueba funcione

```
# agrégame en test-add_col.R
test_that("puede reemplazar columnas", {
  df <- data.frame(x = 1)

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

Mi solución

```
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)  
  }  
}
```

Tu turno

```
# modifica add_col() para que cumpla con esta
# expectativa, agrégala a test-add_cols.R
test_that("where es por default hasta la
derecha", {
  df <- data.frame(x = 1)

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

Haz que esta prueba funcione

```
# agrégame a test-add_col.R
test_that("where es por default hasta la derecha", {
  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

Mi solución

```
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)  
  }  
}
```

¿Podemos usar add_col() para **eliminar** columnas?

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

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

```
# ¿Deberíamos? Si no, ¿que debería add_col()  
# regresar cuando value es NULL? ¿Sería una buena idea  
tener  
# otra función remove_col()?
```

¿Qué pasa si las columnas no son de la misma longitud?

```
# ¿Qué debería pasar aquí?
```

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

```
# ¿Deberíamos reciclar y de forma silenciosa?
```

```
# ¿Reciclar con una advertencia?
```

```
# ¿Regresar un error?
```

¿Podemos usar add_col() para **mover** columnas?

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

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

# ¿Deberíamos?
# ¿Sería mejor una función move_col()?
```

¿Cómo deberíamos nombrar esta colección de funciones?

¿Con un prefijo?

add_col()

move_col()

remove_col()

¿Con un sufijo?

col_add()

col_remove()

col_move()

¿Y qué hacemos con valores incorrectos proveídos por el usuario?

También necesitamos revisar que no haya errores

```
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)
```

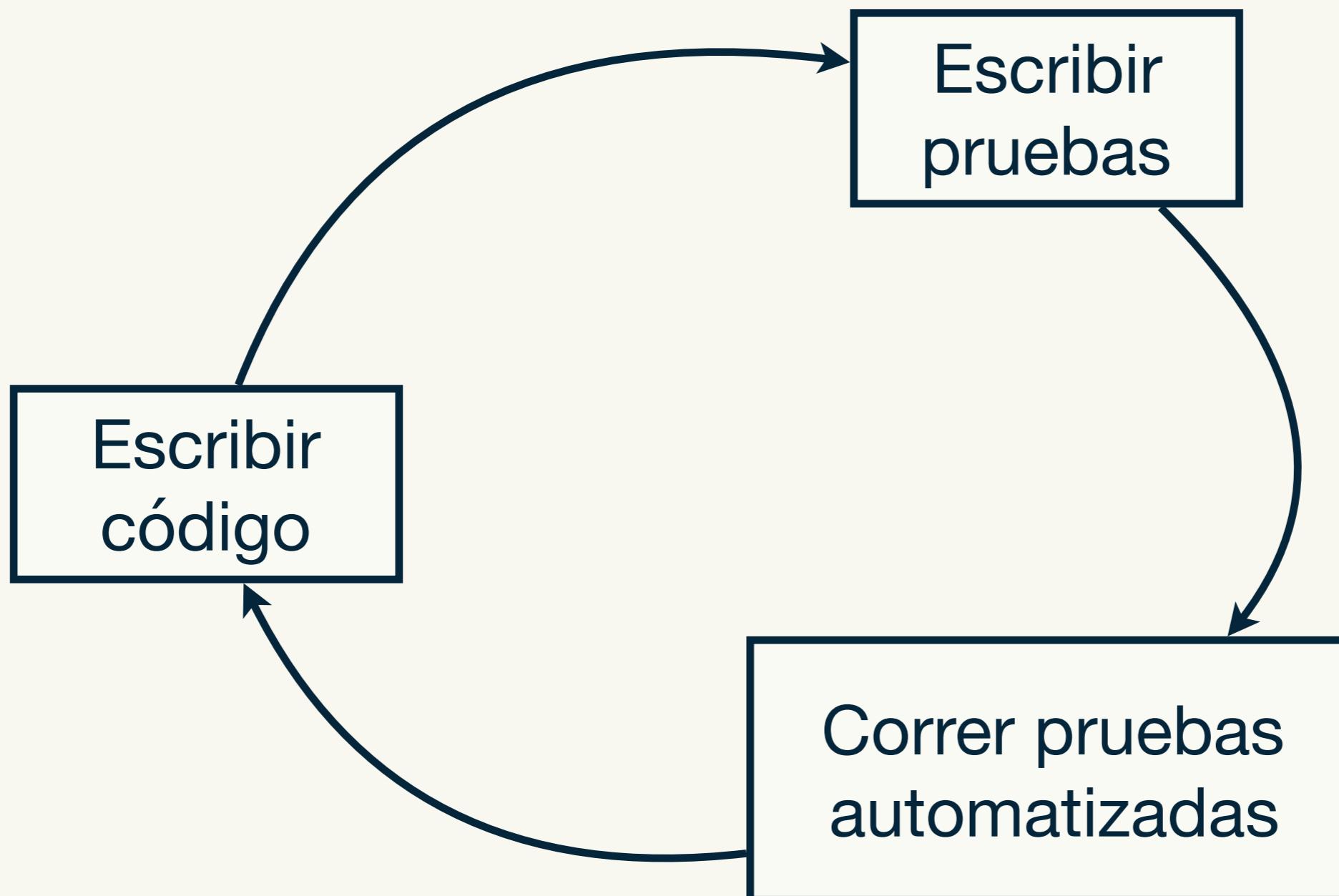
```
insert_into(df1, df2, where = 1:10)
```

```
insert_into(df1, df2, where = "a")
```

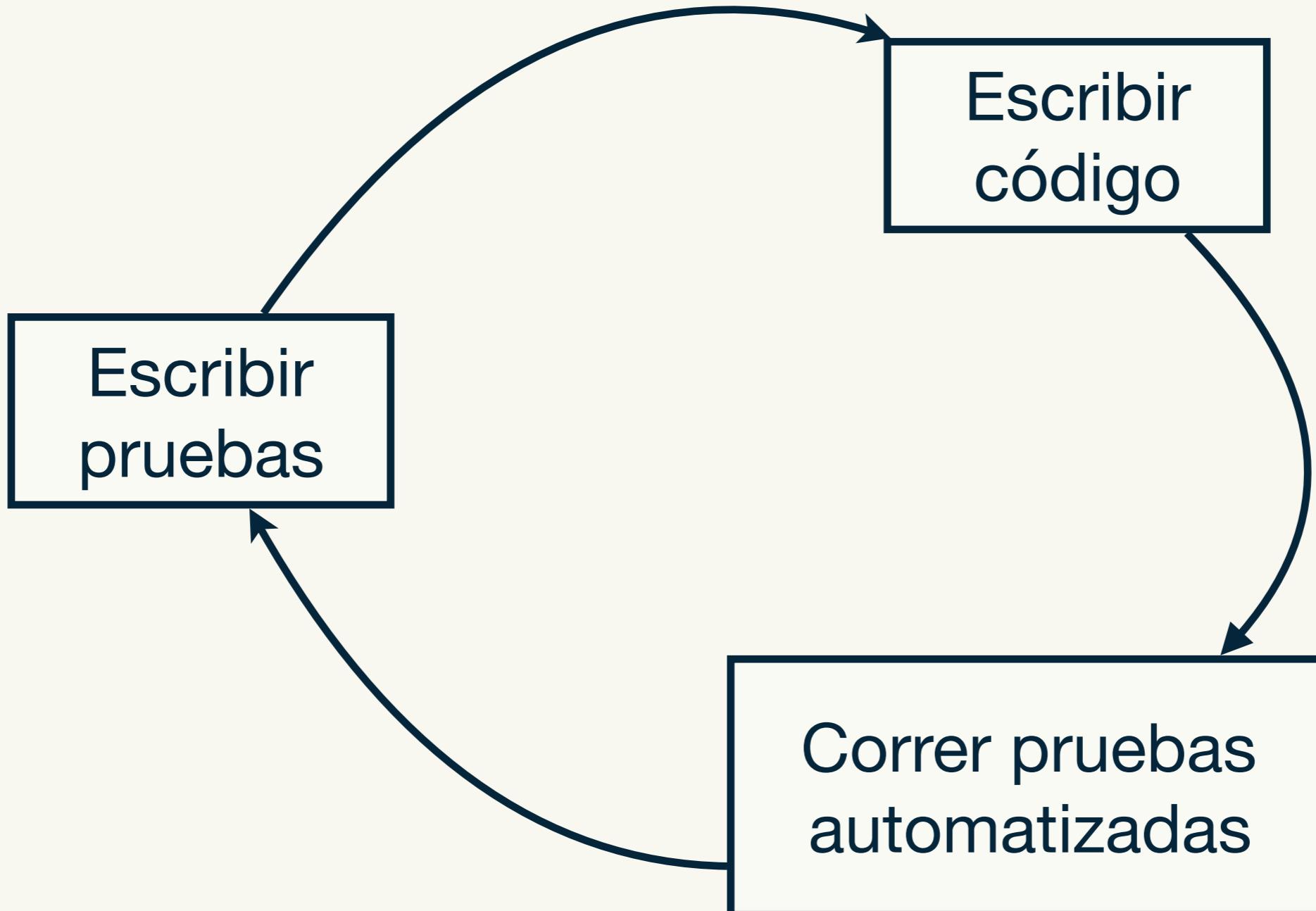
Regresaremos a esto mañana...

Desarrollo
motivado por
pruebas

Hasta ahora hemos escrito código, y luego pruebas



¿Qué pasa si escribimos las pruebas primero?



Desarrollo motivado por pruebas

Cobertura de pruebas

Test coverage nos muestra lo que hemos probado

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


Adapted from *Tidy Tools* by Hadley Wickham

This work is licensed as
Creative Commons
Attribution-ShareAlike 4.0
International

To view a copy of this license, visit
[https://creativecommons.org/
licenses/by-sa/4.0/](https://creativecommons.org/licenses/by-sa/4.0/)