Testing

Hadley Wickham
Assistant Professor / Dobelman Family Junior Chair
Department of Statistics / Rice University

June 2012

1. Motivation
2. Overview
3. Expectations
4. Tests
5. Context
6. Running tests

Motivation

You already know how to debug

- `traceback()` tells you where the problem is
- `browser()` starts an interactive debugger where it's called
- `options(error = recover)` starts interactive debugger automatically on error
- `options(warn = 2)` turns warnings into errors so you can find them more easily

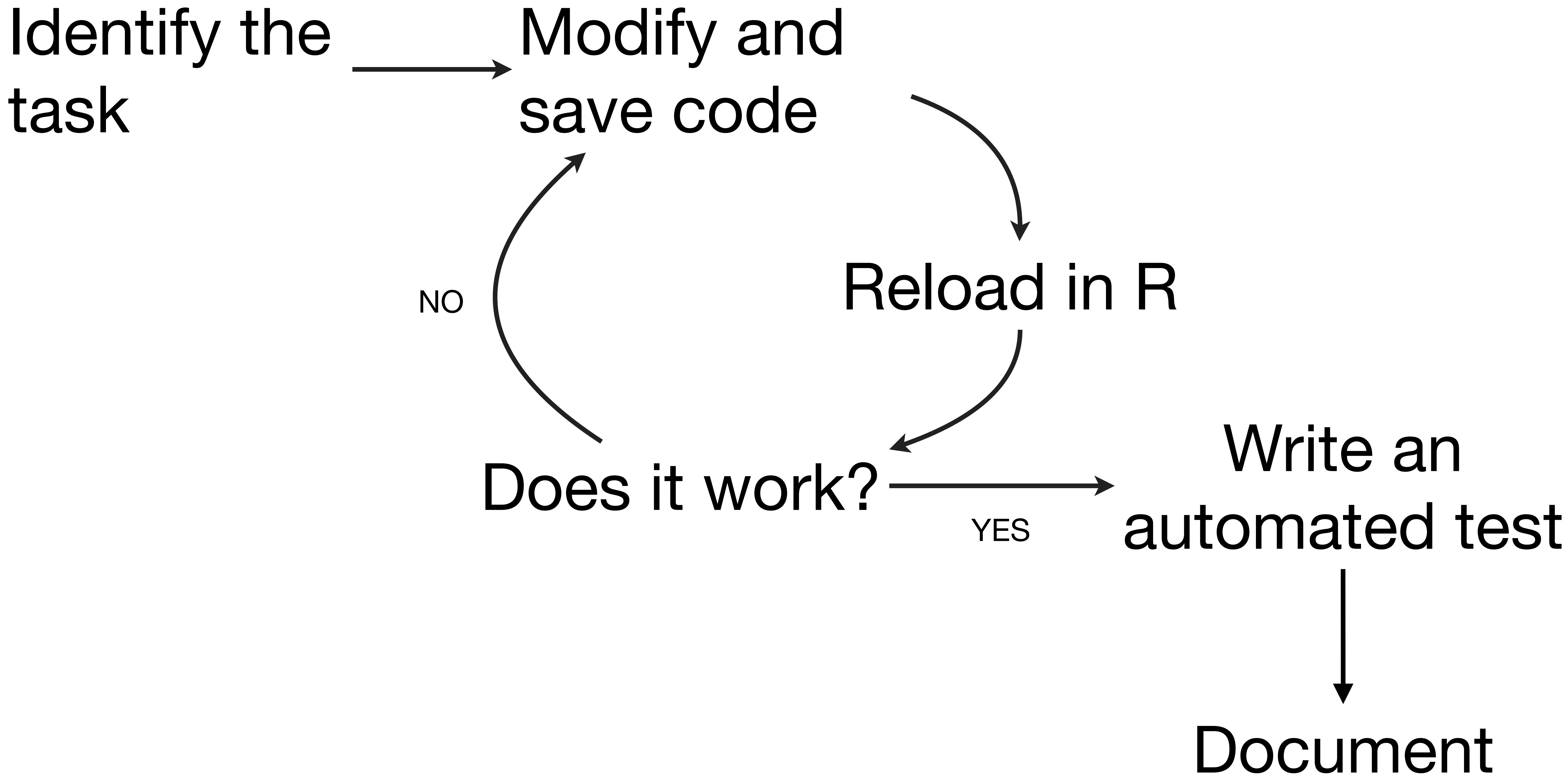
Automated tests

How do you keep bugs from coming back?

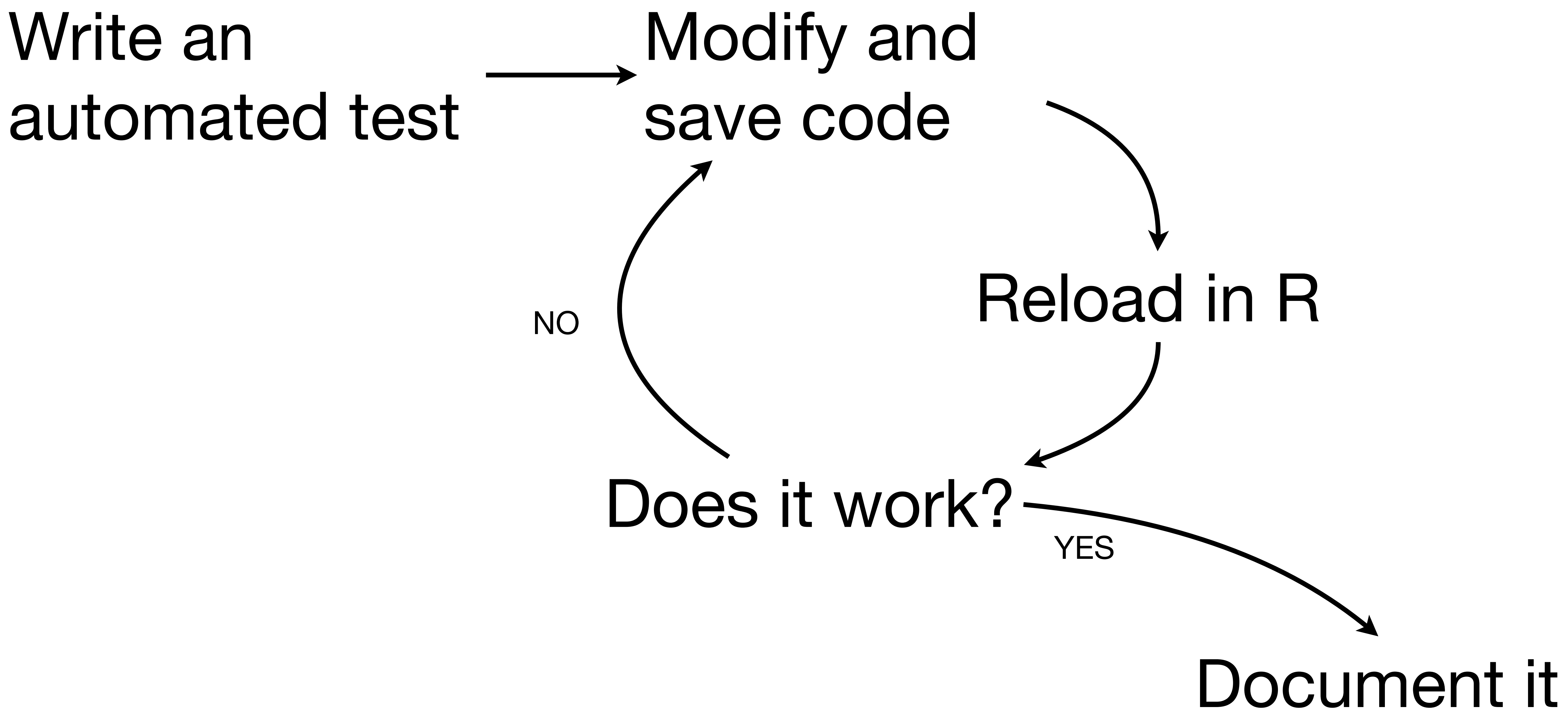
You can’t manually check every function every time you make a change – it takes too long

Solution: automate your testing so you can quickly run tests after every change

Exploratory programming



Confirmatory programming



aka test driven development (**TDD**)

Other benefits

- Code that can be tested easily, often has a better, more modular, design
- When you stop working, leave a test failing. You’ll know what to work on when you come back
- Make big changes without fear of accidentally breaking anything

Testing packages

- `RUnit`
- `svUnit`
- `testthat`

http://bitly.com/testthat

Why test that?

- Easy transition from informal to formal tests. Can be used in wide variety of situations
- Wide range of expectations/assertions
- Fun, colourful output that keeps you motivated

`https://github.com/hadley/devtools/wiki/Testing`

Example packages

- `testthat`, `stringr`, `plyr`, `lubridate`
- `ffbase, ISOweek`
- Reverse suggests from: `http://cran.r-project.org/web/packages/testthat/`

Overview

Key components

- **Expectations**: what do you expect a function to do?
- **Tests**: a group of expectations that tests a small piece
- **Contexts**: a group of tests that tests behaviour of a large piece of functionality (function, class, etc)

```
context("Expectation")

dice <- rv(1:6)
coin <- rv(c(-1, 1))

test_that("expectation is additive", {
  expect_that(E(dice + coin), equals(E(dice) + E(coin)))

  expect_that(E(dice + dice), equals(2 * E(dice)))
  expect_that(E(dice + dice + dice), equals(3 * E(dice)))
})
```

Your turn

Look at `some-tests`. Where are the tests? How are they structured?

Run the tests using `test("some-tests")`. What do you see? What does each dot represent?

Where is the failing test?

```
# Green . = passing test
# Red number = failing test (or error)
# Numbers index list of all failed expectation
#  giving message and test name.


VAR <- function(x) E((x - E(x) ^ 2))
# should be
VAR <- function(x) E((x - E(x)) ^ 2)

# fix and then re-run tests
```

Expectations

Expectation	Test	Abbreviation
equals	all.equals	expect_equal
is_identical_to	identical	expect_identical
is_equivalent_to	all.equals, check.attributes = FALSE	expect_equivalent
is_a	inherits	expect_is
is_true / is_false	identical	expect_true / expect_false

Expectation	Test	Abbreviation
`matches`	`grepl + any`	`expect_matches`
`prints_text`	`matches applied to output`	`expect_output`
`shows_message`	`matches applied to messages`	`expect_message`
`gives_warning`	`matches applied to warnings`	`expect_warning`
`throws_error`	`matches on errors`	`expect_error`

Your turn

Add a new file (and context) for testing probabilities. Test that rv's have probability 1 of being greater than -Inf and smaller than Inf.

What happens if you supply a missing value? What should happen? Write a test.

```
context("Probability")

test_that("0 probability of being infinite", {
  X <- rv(1:10)
  expect_equal(P(X > -Inf), 1)
  expect_equal(P(X < -Inf), 0)
  expect_equal(P(X > Inf), 0)
  expect_equal(P(X < Inf), 1)
})

test_that("missing comparison means 100% of NA", {
  X <- rv(1:5)
  expect_equal(P(X > NA), NA_real_)
}
```

Running tests

```
# Casually, during development
# (automatically reloads all code)
test(”some-tests”)

# More formally
install(”some-tests”)
test_package(”some-tests”)

# Even more formally (and the next topic)
check(”some-tests”)
```

Package tests

Store all tests in `inst/tests` so they are installed with the package. Then users can run to check their installation/OS is ok.

Include the following code in `tests/test-all.R` (note capital `R`). This ensures `R CMD check` will not pass unless all tests pass

```
library(testthat)
# This loads the version being tested
library(rv)

test_package("rv")
```

This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 United States License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/us/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.