



*Everything not tested will eventually fail*

2020-07-28 - NYC Data Hackers Meetup

Colin Fay - ThinkR

# \$ whoami

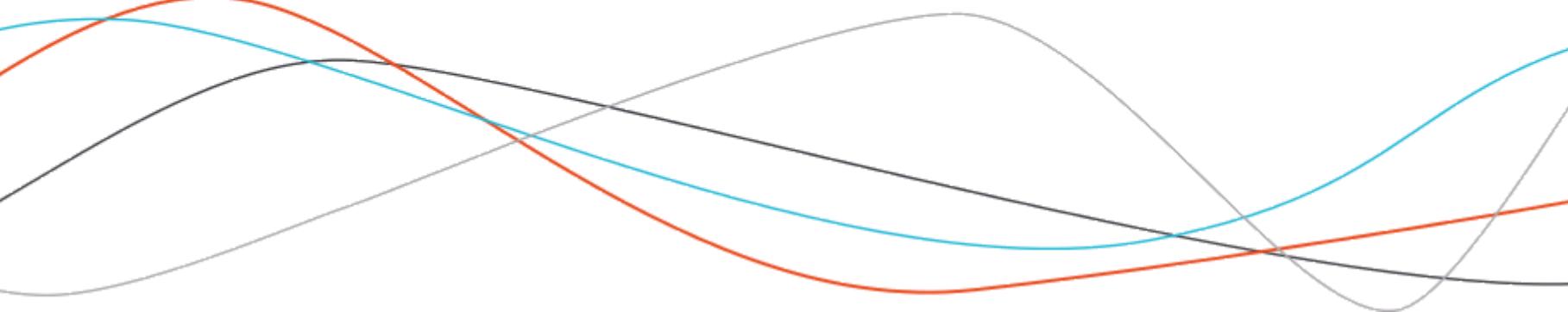
Colin FAY

Data Scientist & R-Hacker at ThinkR, a french company focused on Data Science & R.

Hyperactive open source developer, lead developer of the `{golem}` project.

- <https://thinkr.fr>
- <https://rtask.thinkr.fr>
- [https://twitter.com/\\_colinfay](https://twitter.com/_colinfay)
- <https://github.com/colinfay>
- <https://colinfay.me>

# ThinkR



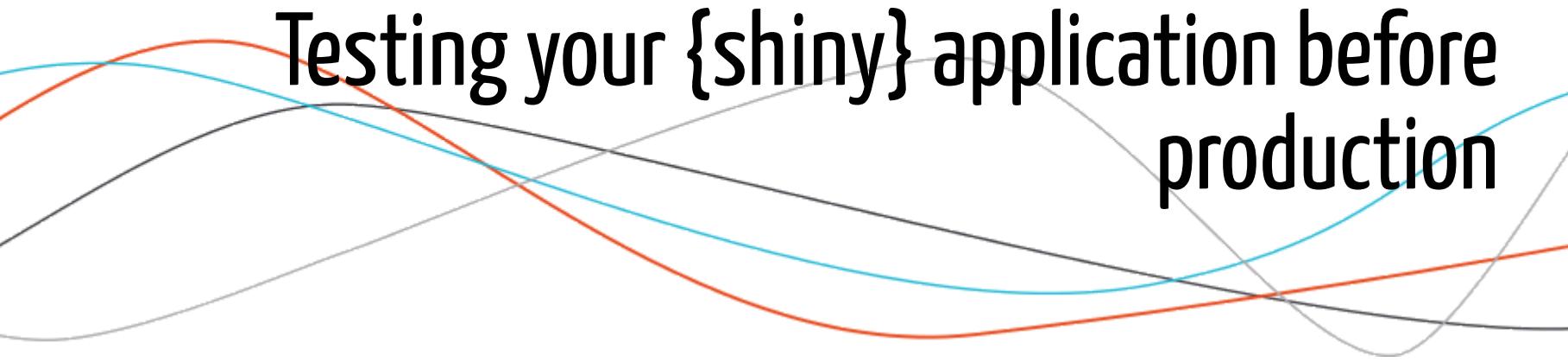
# ThinkR

Data Science engineering, focused on R.

- Training
- Software Engineering
- R in production
- Consulting



# Everything not tested will eventually fail



Testing your {shiny} application before production

A decorative graphic at the bottom of the slide features three overlapping bell-shaped curves. One curve is red, one is light blue, and one is dark grey. They overlap in the center, creating a sense of data distribution or quality assurance.

## <aside>

What does "production" even mean?

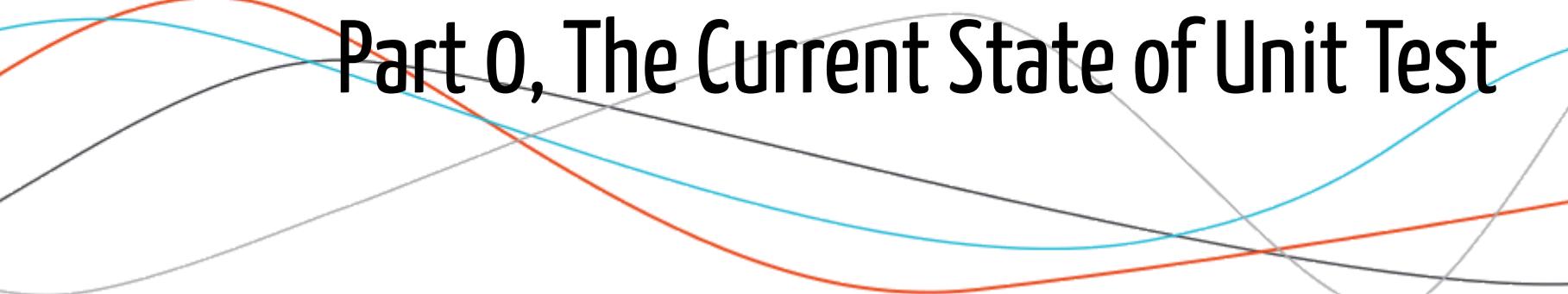
A piece of software that is:

- **USED**, even if by only one person
- **RELIED UPON**
  - by the user: to be available, and accurate
  - by the developer: to be available, accurate, modular, and resilient
- **HAS REAL LIFE IMPACT**
  - on the user, who needs the software
  - on the developer, who works on the software

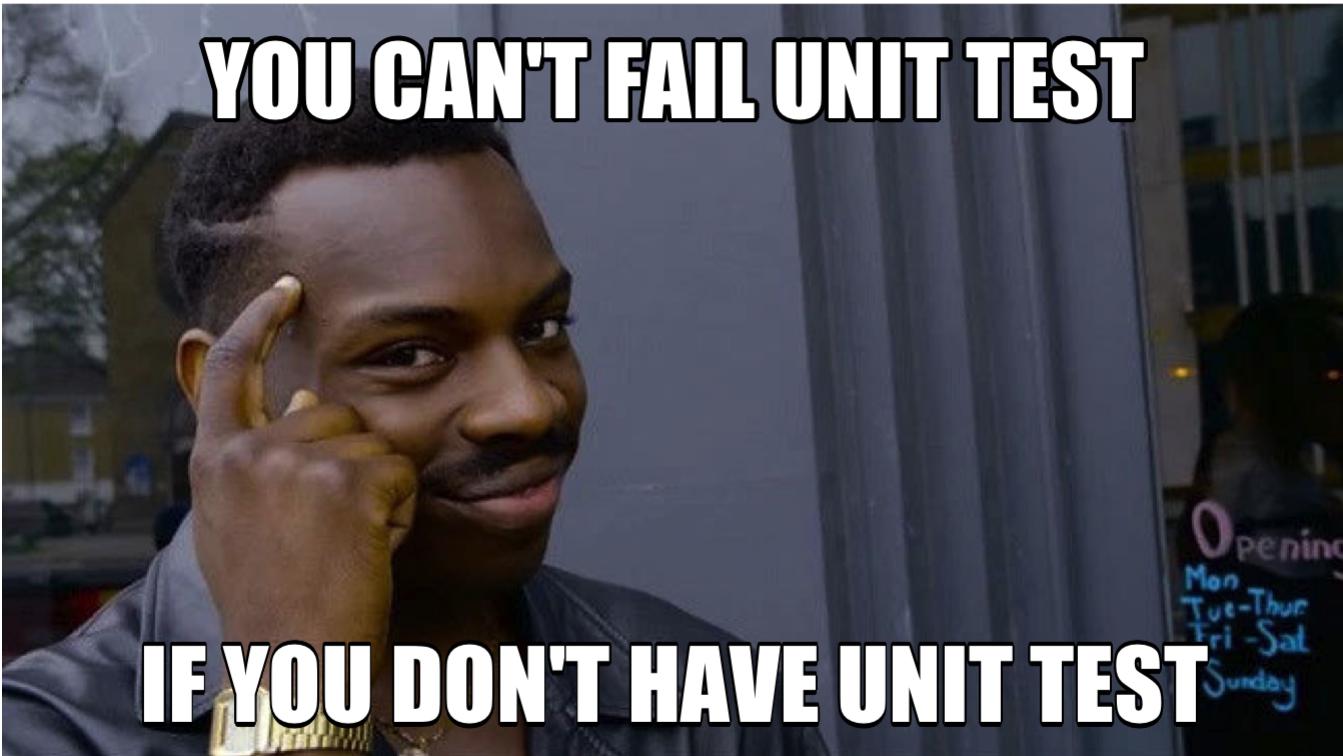
## </aside>

# Everything not tested will eventually fail

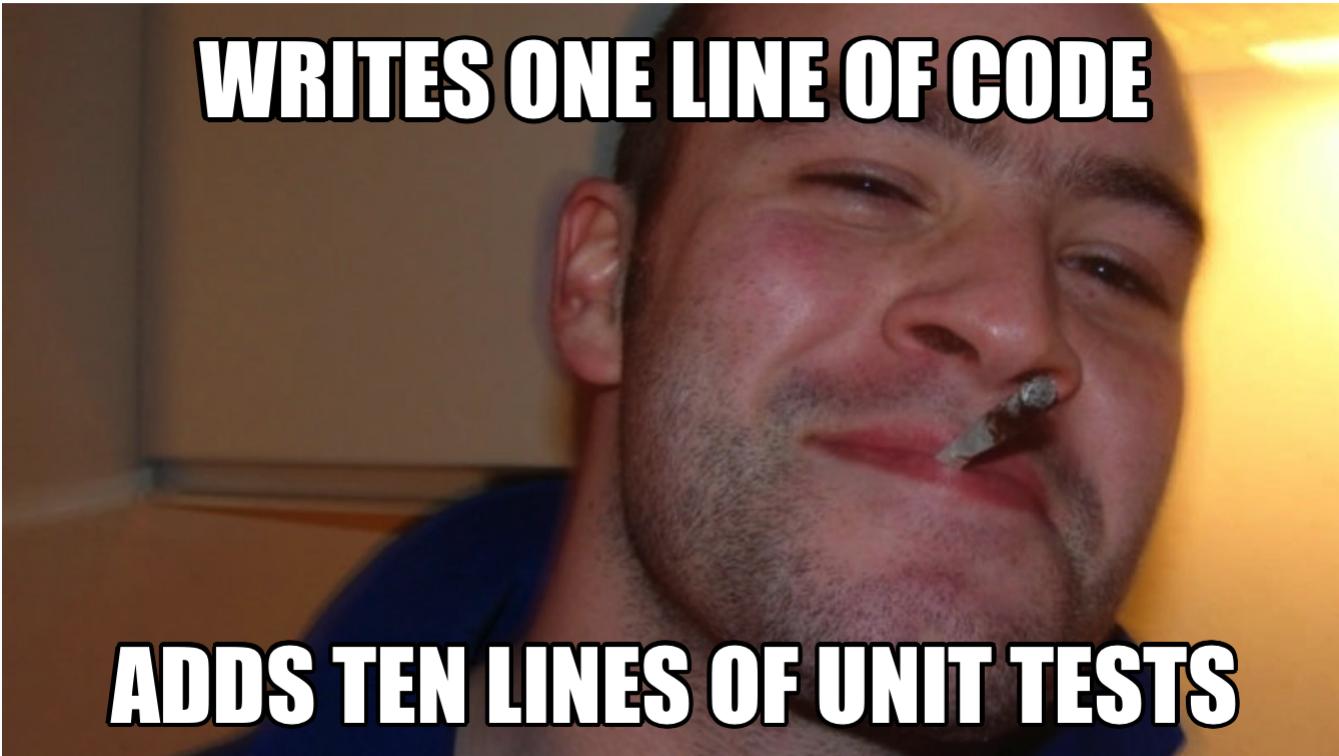
## Part 0, The Current State of Unit Test

A decorative graphic at the bottom of the slide consists of several abstract, smooth curves in shades of grey, black, red, and blue, overlapping each other against a white background.

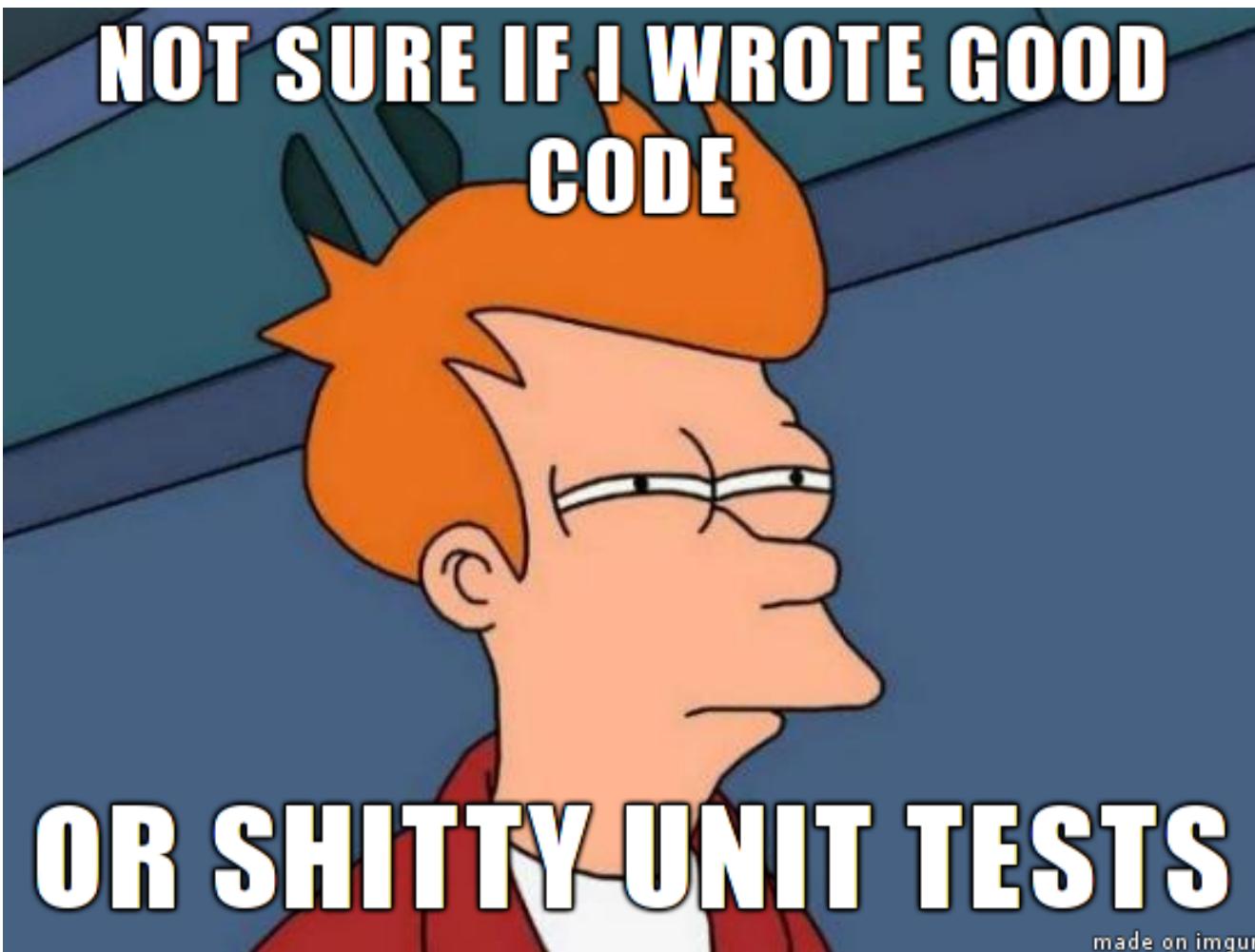
# The two states of unit test



# The two states of unit test

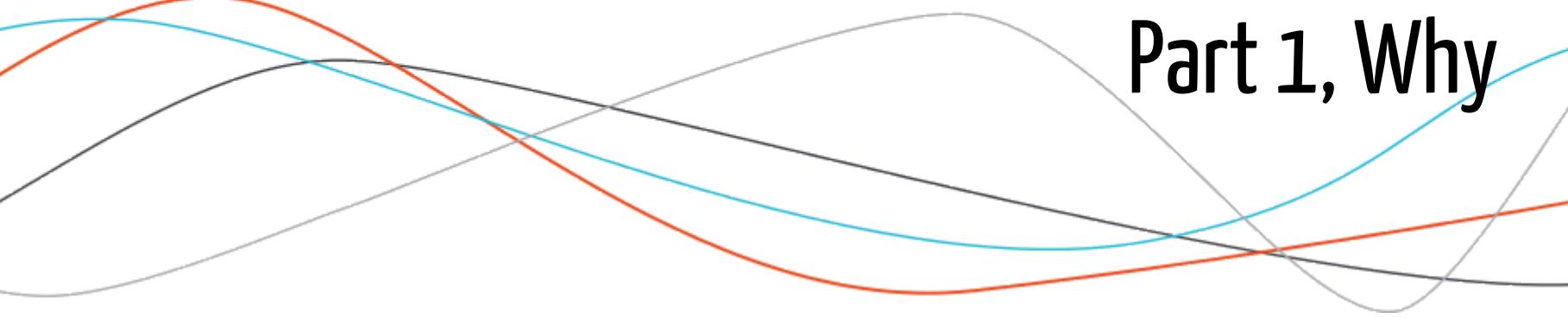


# Or somewhere in the middle?



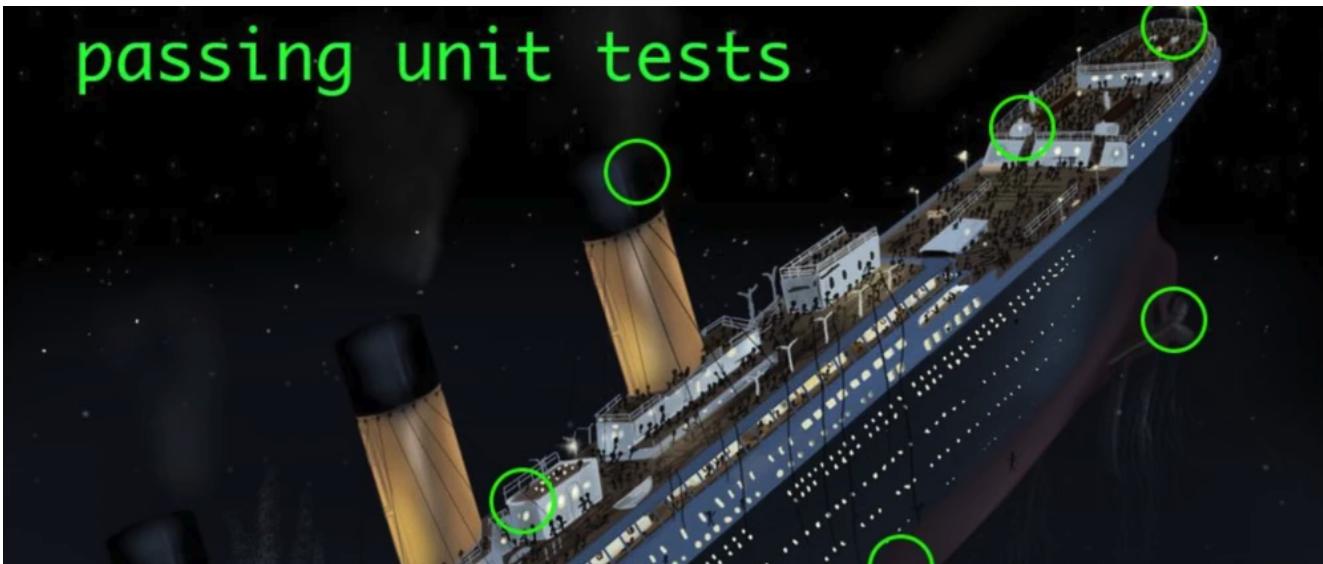
made on imgur

# Everything not tested will eventually fail



Part 1, Why

# Why



Everything That's Not Tested Will Eventually Break

# Why



Jenny Bryan  
@JennyBryan

If you use software that lacks automated tests, you are the tests.

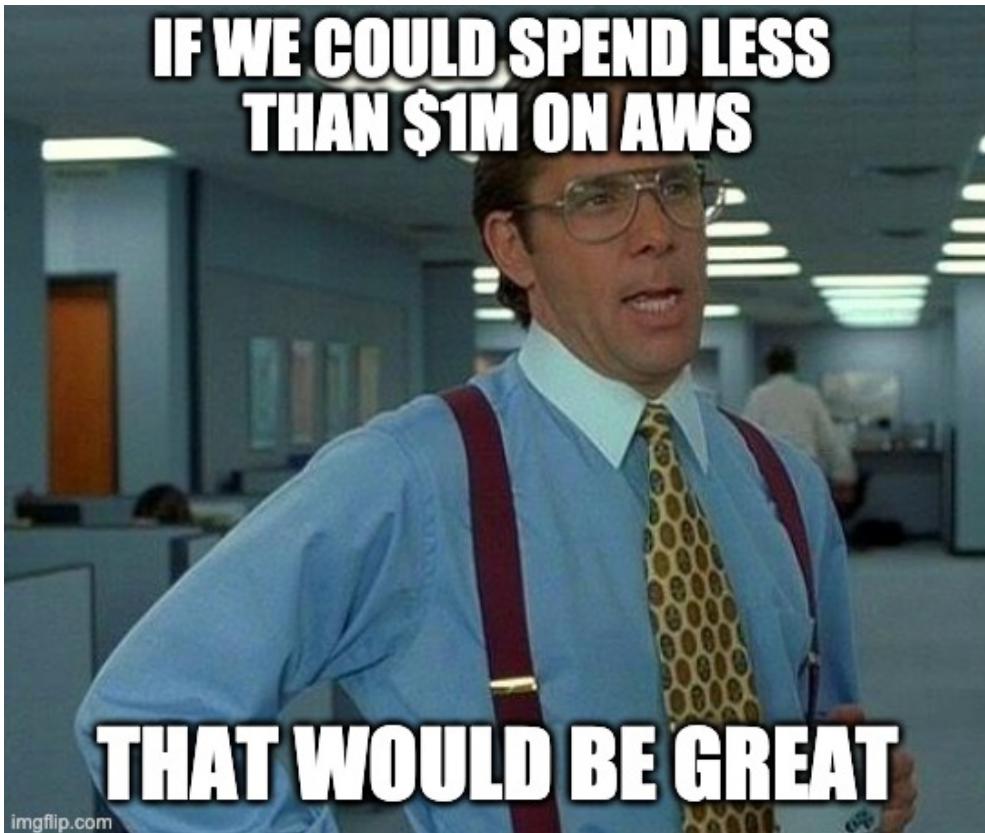
3:13 AM · 22 sept. 2018 · Tweetbot for Mac

294 Retweets 1 k J'aime



Don't let your users be your unit test

# Why



Control the application load

See also: Don't DoS your own server

# During dev

## Safely collaborate & change elements on a project

- Making changes should be pain-free
- Bugs should be detected quickly 🐞
- New collaborators should be able to integrate a team smoothly

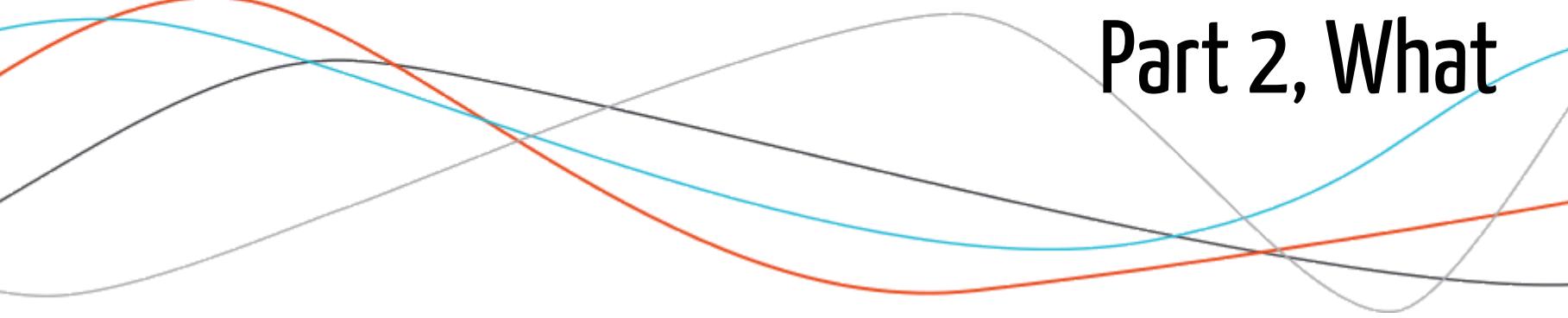
# In production



## Safely serve application

- Your users should not be your unit tests
- Serving application cost money 💰
- (corollary) You shouldn't spend 1 million bucks on AWS (Jeff Bezos is rich enough)
- You shouldn't DoS your own server 🚧

# Everything not tested will eventually fail



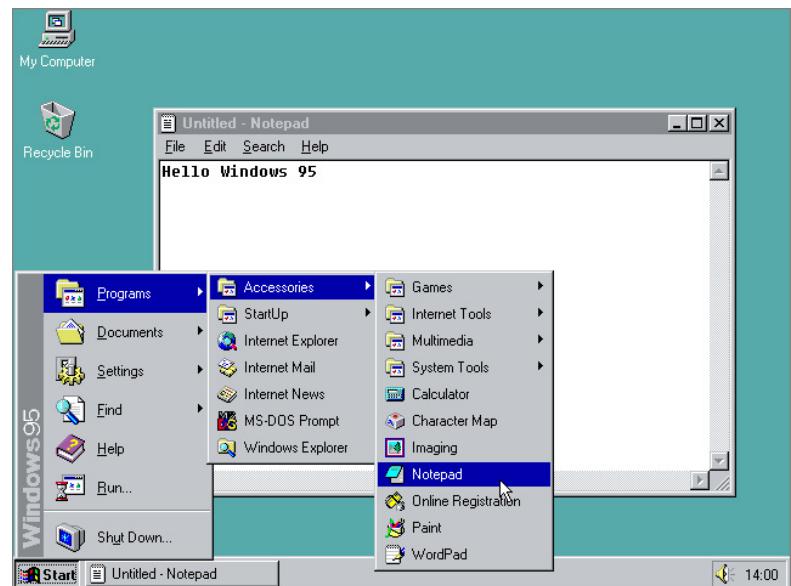
Part 2, What

A decorative graphic at the bottom of the slide features three overlapping bell-shaped curves. One curve is red, one is blue, and one is grey. They are positioned in a way that suggests they are all peaking at the same point, which corresponds to the word 'What' in the text above.

# What - User Interface

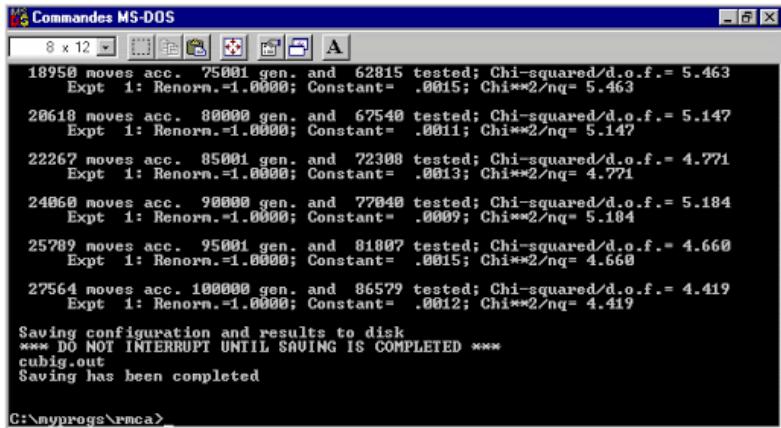
Your time is limited, so if you have to choose don't focus too much on testing the UI only

- What the users see
- What the users interact with
- General front-end/design



# What - Business logic

Try to test business logic as extensively as possible



A screenshot of a MS-DOS Command Prompt window titled "Commandes MS-DOS". The window shows the results of a genetic algorithm run. The text output includes various statistics such as moves, generations, and Chi-squared values, along with configuration details and a warning about saving to disk.

```
8 x 12 A
18950 moves acc. 75001 gen. and 62815 tested; Chi-squared/d.o.f.= 5.463
Expt 1: Renorm.-1.0000; Constant= .0015; Chi**2/nq= 5.463
20618 moves acc. 80000 gen. and 67540 tested; Chi-squared/d.o.f.= 5.147
Expt 1: Renorm.-1.0000; Constant= .0011; Chi**2/nq= 5.147
22267 moves acc. 85001 gen. and 72300 tested; Chi-squared/d.o.f.= 4.771
Expt 1: Renorm.-1.0000; Constant= .0013; Chi**2/nq= 4.771
24060 moves acc. 90000 gen. and 77040 tested; Chi-squared/d.o.f.= 5.184
Expt 1: Renorm.-1.0000; Constant= .0009; Chi**2/nq= 5.184
25789 moves acc. 95001 gen. and 81807 tested; Chi-squared/d.o.f.= 4.660
Expt 1: Renorm.-1.0000; Constant= .0015; Chi**2/nq= 4.660
27564 moves acc. 100000 gen. and 86579 tested; Chi-squared/d.o.f.= 4.419
Expt 1: Renorm.-1.0000; Constant= .0012; Chi**2/nq= 4.419

Saving configuration and results to disk
*** DO NOT INTERRUPT UNTIL SAVING IS COMPLETED ***
cubig.out
Saving has been completed

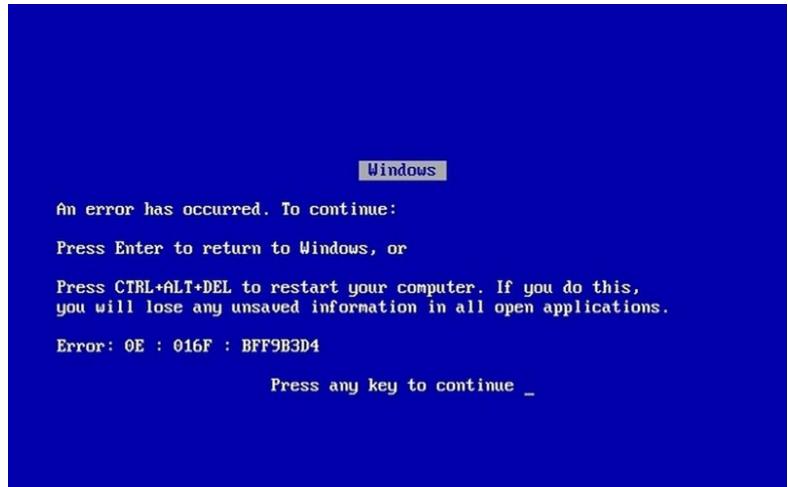
C:\nyprog\rmca>
```

- Core algorithms that make your app "unique"
- Business knowledge
- What your users rely on

# What - Application Load

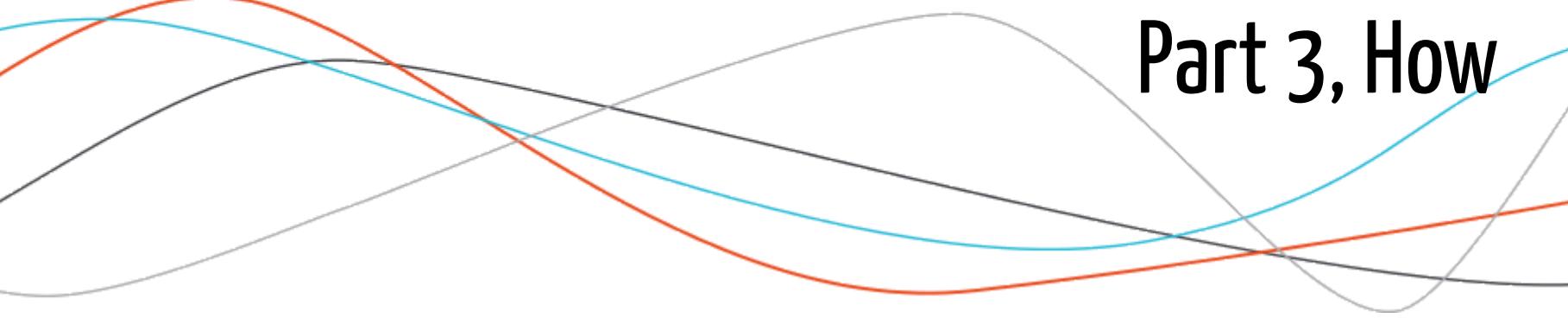
Poor app performances lead to bad UX, and potentially cost 

- How much CPU & RAM does your application need
- Bad estimate will lead to slow application performances
- If the app needs to scale, it's crucial to know it upfront



# Everything not tested will eventually fail

Part 3, How



# How - Business logic/backend

## Shiny App as a package

-> Leverage standard testing frameworks

```
test_that("The meaning of life is  
42", {  
  expect_equal(  
    meaning_of_life(),  
    42  
})
```



# How - User interface/frontend

## {shinytests}

- 🕵️ Test visual regression of your application

## puppeteer

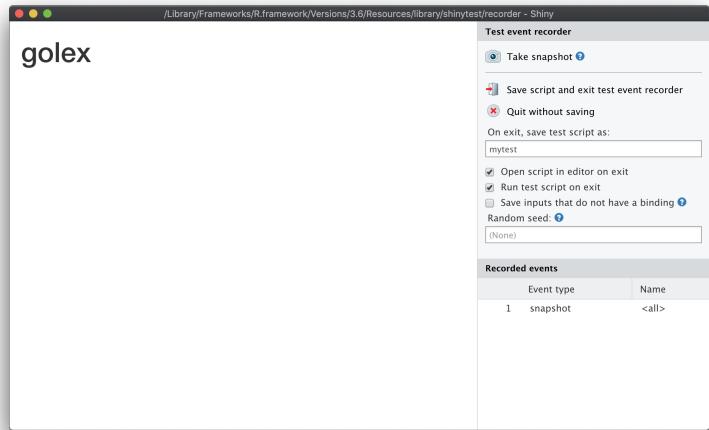
- 🚻 Command line tool to mock a web session, in NodeJS

## {crrry}

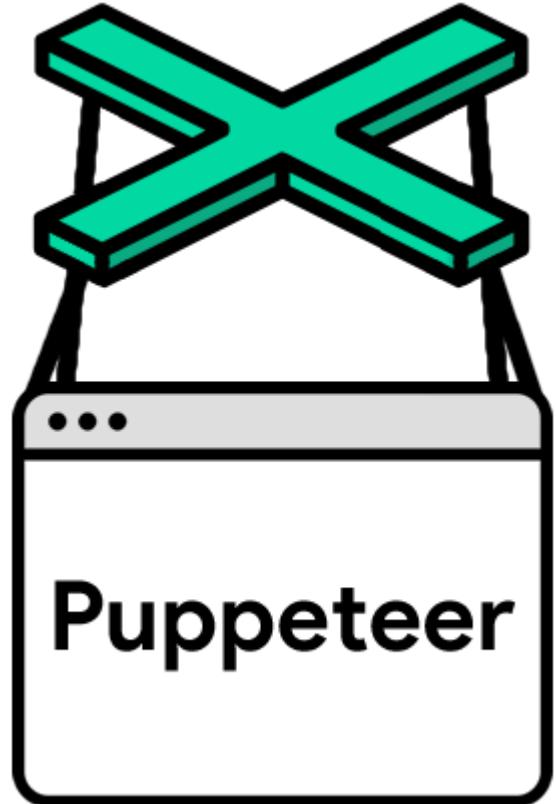
- 🚗 R tool to drive a {shiny} session

# How - {shinytests}

1. Records snapshots of an app
2. Replays the application
3. Detects visual regression



# How - puppeteer



1. Records interaction with the application
2. Replays the application with NodeJS
3. Detects application logic changes

# How - puppeteer

```
// Require the node module
const puppeteer = require('puppeteer');
(async () => {
  // launch puppeteer and connect to the page
  const browser = await puppeteer.launch()
  const page = await browser.newPage()
  await page.goto('http://localhost:2811/')

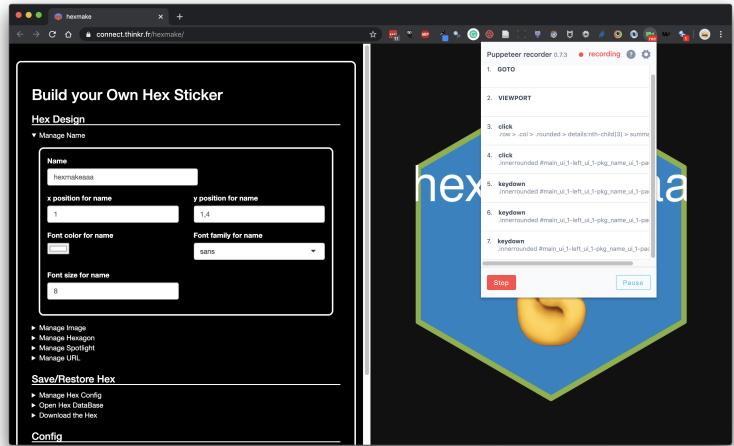
  // We're waiting for a DOM element to be ready
  await page.waitForSelector('.row > .col > \
    .rounded > details:nth-child(3) > summary')
  // Now it's ready, we can click on it
  await page.click('.row > .col > .rounded > \
    details:nth-child(3) > summary')

  // Now our test is over, we can close the connection
  await browser.close()
})()
```

# How - puppeteer

And it has a Chrome extension 

- Records the app "live"
- Returns the NodeJS code to replay the application



# How - {crrry}



- Wrapper around the {crrri} 📦, for remote orchestration of Chrome, with recipes for {shiny} apps
- Connects to a running app and interact with it
- Allows to script everything

# How - {crrry}

## Create a new testing session

```
test <- crrry::CrrryOnPage$new(  
  # Find the Chrome binary  
  chrome_bin = pagedown::find_chrome(),  
  # Get a random port for Chrome to use  
  chrome_port = httpuv::randomPort(),  
  # Connect to a page  
  url = "https://connect.thinkr.fr/hexmake/"  
)
```

```
test$wait_for_shiny_ready()
```

Shiny is computing

✓ Shiny is still running

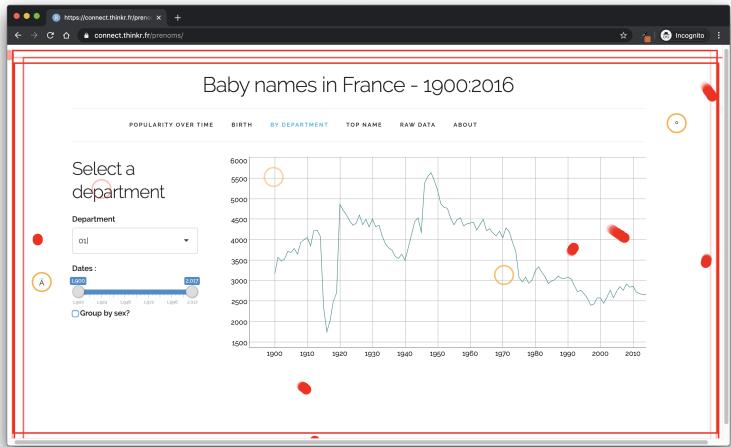
# How - {crrry}

## Set a series of input

```
for (i in letters[1:3]){
  # Set a value for a given input
  test$shiny_set_input(id = "main_ui_1-left_ui_1-pkg_name_ui_1-package", i)
}
```

- Setting id main\_ui\_1-left\_ui\_1-pkg\_name\_ui\_1-package with value a  
Shiny is computing
- ✓ Shiny is still running
- Setting id main\_ui\_1-left\_ui\_1-pkg\_name\_ui\_1-package with value b  
Shiny is computing
- ✓ Shiny is still running
- Setting id main\_ui\_1-left\_ui\_1-pkg\_name\_ui\_1-package with value c  
Shiny is computing
- ✓ Shiny is still running

# How - gremlins



- JavaScript library to simulate a hordes of gremlins using an app
- Will click and scroll at random on the app
- Allows to detect unexpected behaviors

# How - gremlins

```
test <- crrry::CrrryOnPage$new(  
  chrome_bin = pagedown::find_chrome(),  
  chrome_port = httpuv::randomPort(),  
  url = "https://connect.thinkr.fr/hexmake/",  
  headless = TRUE  
)  
test$wait_for_shiny_ready()  
# Launch a hordes of gremlins  
test$gremlins_horde()  
# Wait for them to finish their work  
Sys.sleep(20)  
# Check that everything is ready  
test$wait_for_shiny_ready()  
# Stop the test  
test$stop()
```

# How - Testing the app load

{shinyloadtest}

- 🏆: native R package + Cli to record and replay load tests

{dockerstats}

- 🐳: get Docker stats inside R

{crrry} + {dockerstats}

- 🎭: replay session and watch the Docker stats

# How - {shinyloadtest}

- Records a visit on the application
- Replays the app with multiple users
- Gives detailed stats about response time and load



# How - {dockerstats}



- Wrapper around `docker stats`
- Turns the stats from Docker into an R dataframe
- Can be called recursively

# How - {dockerstats}

```
# Launch the container from the terminal
```

```
docker run --name hexmake --rm -p 2811:80 colinfay/hexmake
```

```
# Call dockerstats, only selecting specific columns
```

```
dockerstats::dockerstats("hexmake")[, c("Name", "MemUsage", "CPUPerc",
"MemPerc", "record_time")]
```

```
Warning in system(com, intern = TRUE): running command 'docker stats --no-
stream hexmake --format "{{.Container}}",
{{.Name}}, {{.ID}}, {{.CPUPerc}}, {{.MemUsage}}, {{.NetIO}}, {{.BlockIO}},
{{.MemPerc}}, {{.PIDs}}' had status 1
```

Unable to find any container running.

```
[1] Name      MemUsage     CPUPerc     MemPerc     record_time
<0 rows> (or 0-length row.names)
```

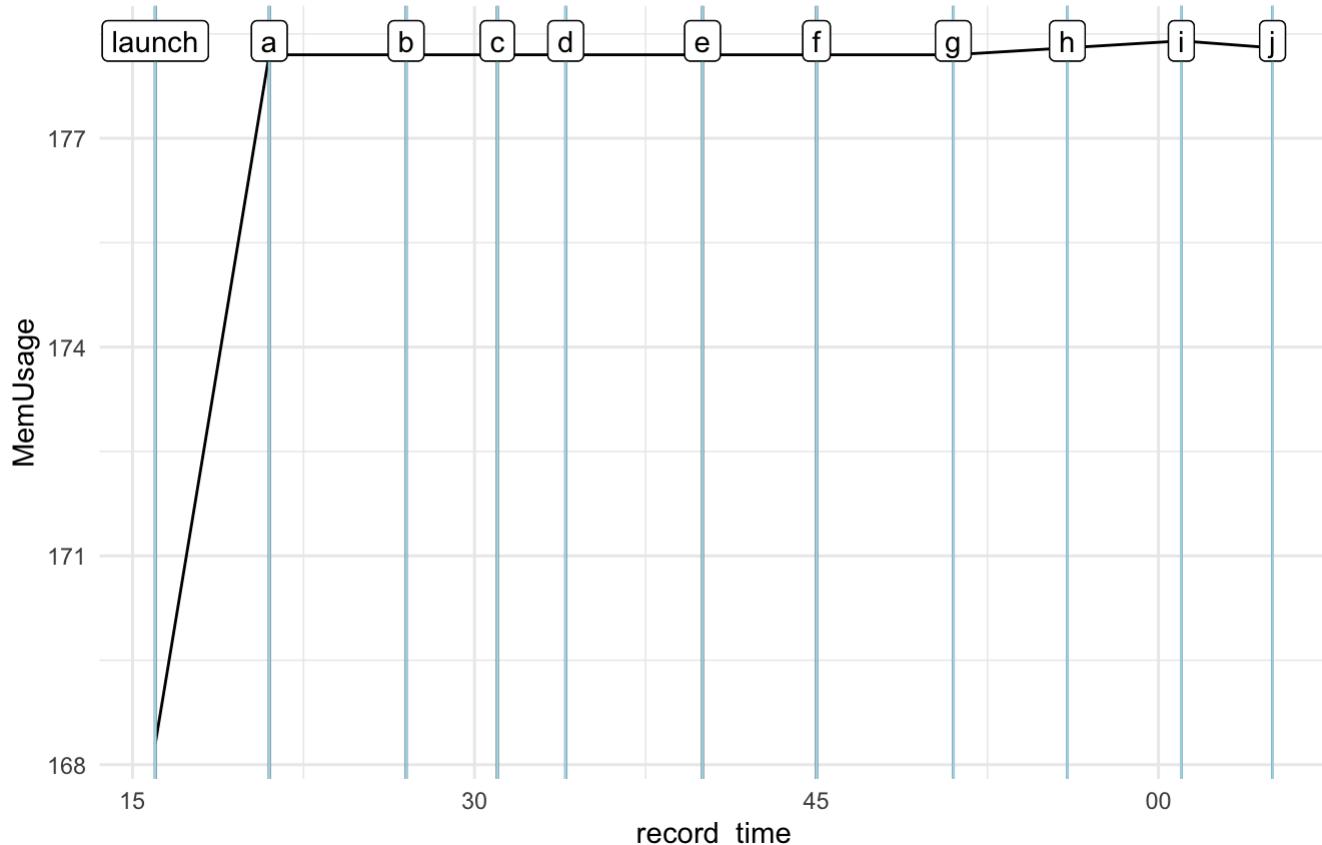
# {dockerstats} + {crrry}

```
# Getting the result from the first launch
results <- dockerstats::dockerstats("hexmake", extra = "launch")

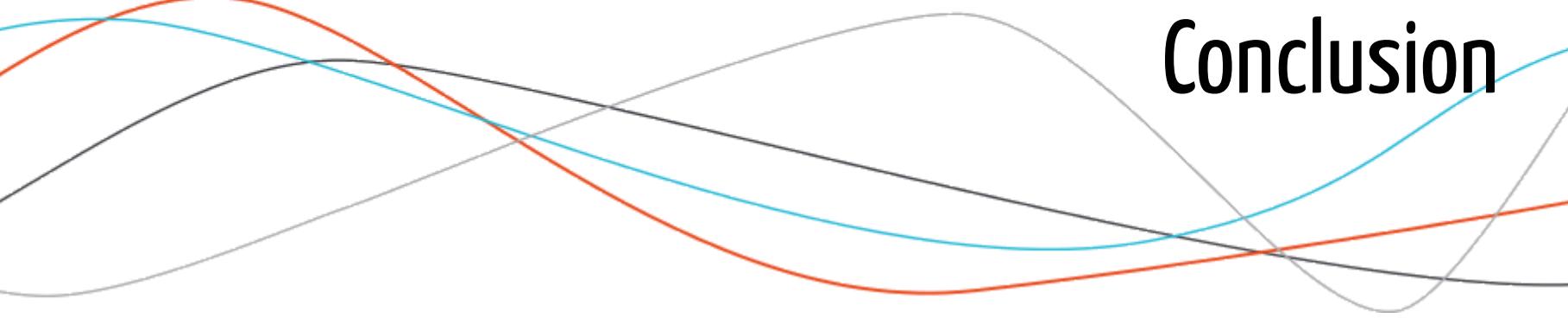
# Setting a series of letters as input
for (i in letters[1:10]){
  test$shiny_set_input(
    "main_ui_1-left_ui_1-pkg_name_ui_1-package",
    i
  )
  # Binding the current snapshot to the results
  results <- rbind(
    results,
    dockerstats::dockerstats("hexmake", extra = i)
  )
}
```

# {dockerstats}

MemUsage of 10 inputs for package name  
Black line is memory usage, blue vertical lines are input changes



# Everything not tested will eventually fail



Conclusion

# To conclude

- Test often, test soon
- If you have to chose, focus on the backend
- Interactive tests can (and should) be scripted
- Don't forget to test the load



- [testthat.r-lib.org](https://testthat.r-lib.org)
- [rstudio.github.io/shinytest](https://rstudio.github.io/shinytest)
- [pptr.dev](https://pptr.dev)
- [github.com/ColinFay/crrry](https://github.com/ColinFay/crrry)
- [github.com/marmelab/gremlins.js](https://github.com/marmelab/gremlins.js)
- [rstudio.github.io/shinyloadtest](https://rstudio.github.io/shinyloadtest)
- [github.com/ColinFay/dockerstats](https://github.com/ColinFay/dockerstats)



## Introduction

I Building Successful Shiny Apps

1 About Successful Shiny Apps

2 Planning Ahead

3 Structuring your Project

4 Introduction to {golem}

5 The workflow

II Step 1: Design

6 UX Matters

7 Don't rush into coding

8 A Gentle Introduction to CSS

III Step 2: Prototype

9 Setting up for success with {golem}

10 Building an "ipsum-app"

IV Step 3: Build

11 Building app with {golem}

V Step 4: Strengthen

12 Build yourself a safety net

13 Version Control

14 Deploy your application

VI Optimizing

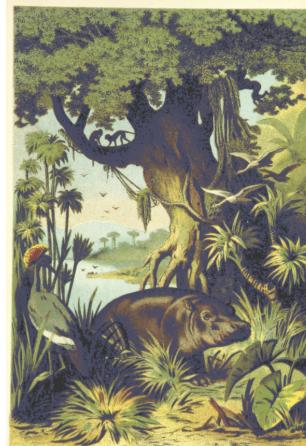
15 The Need for Optimization

# Engineering Production-Grade Shiny Apps

Colin Fay, Sébastien Rochette, Vincent Guyader, Cervan Girard

2020-04-25

## Introduction



step further.

This book is currently under development. It will be published in 2020 in the [R Series](#) by Chapman & Hall.

## Motivation

This book will not **get you started with Shiny**, nor **talk about how to deploy into production and scale your app**. What we'll see is the **process of building the app**. Why? Lots of blog posts and books talk about starting to use {shiny} (???) or putting apps in production. Very few (if any) talk about this grey area between getting started and pushing into production.

So this is what this book is going to talk about: building Shiny application. We'll focus on the process, the workflow, and the tools we use at ThinkR when building big Shiny Apps.

Hence, if you are starting to read this book, we assume you have a working knowledge of how to build a small application, and want to know how to go one

# Thx! Questions?

Colin Fay

## Online

- [colin@thinkr.fr](mailto:colin@thinkr.fr)
- [http://twitter.com/\\_colinfay](http://twitter.com/_colinfay)
- [http://twitter.com/thinkr\\_fr](http://twitter.com/thinkr_fr)
- <https://github.com/ColinFay>
- <https://thinkr.fr/>
- <https://rtask.thinkr.fr/>
- <https://colinfay.me/>

## Related projects

- [engineering-shiny.org](http://engineering-shiny.org)
- [{golem}](#)
- [{shinipsum}](#)
- [{fakir}](#)
- [{shinysnippets}](#)