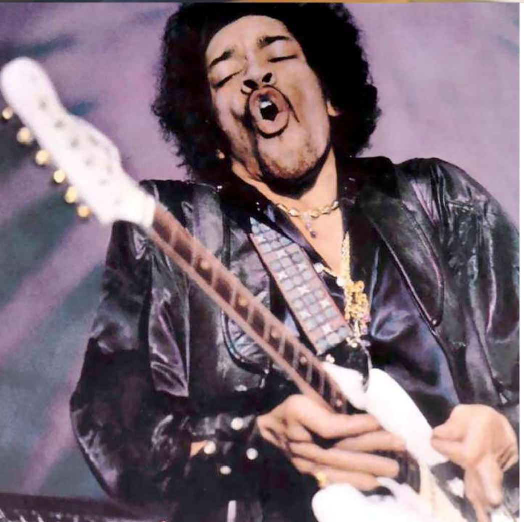
Software Engineering with Shiny

Or, how to work smarter so that you can do more ~~fun~~ important stuff

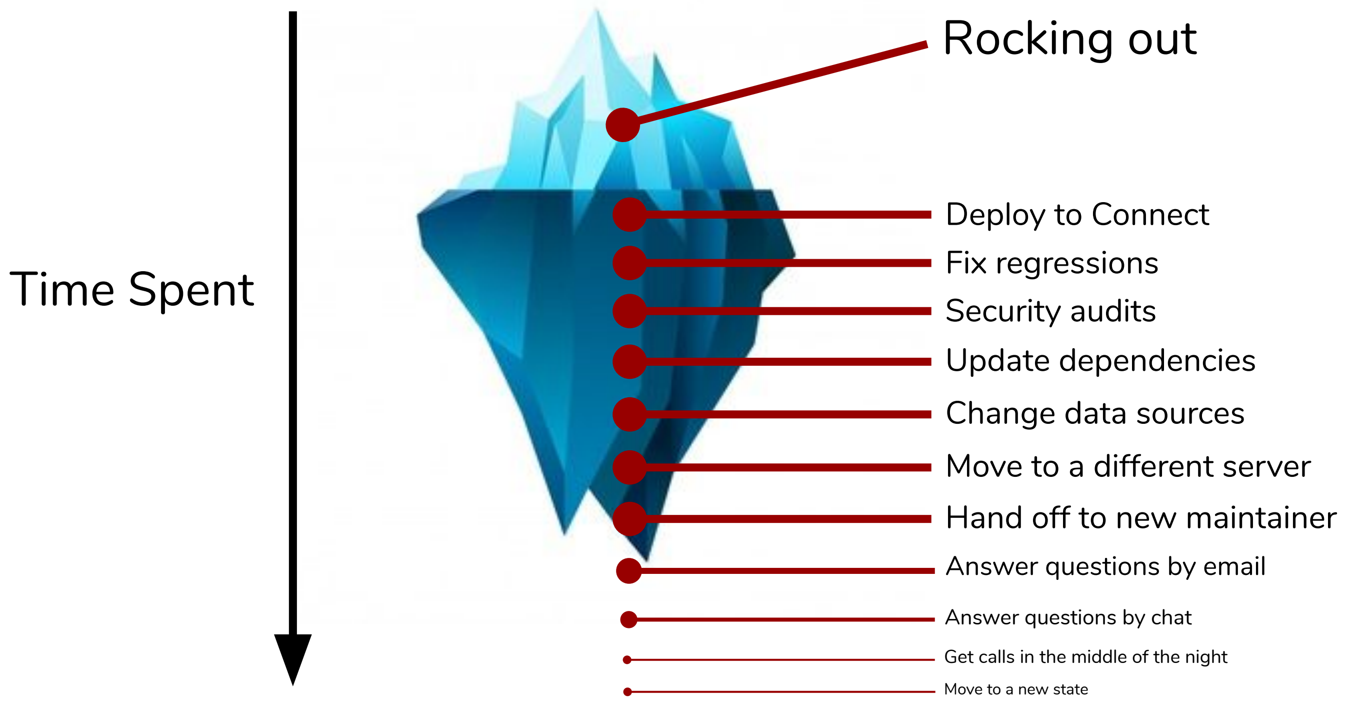
Alan Dipert
Software Engineer, RStudio
@alandipert



Shiny rocks.

For building apps.

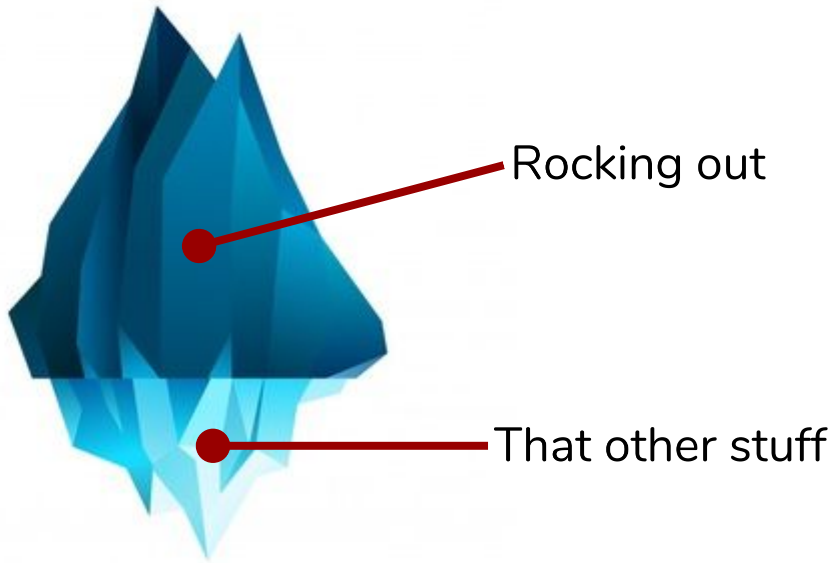
“Total Cost of Ownership”



Software is hard.

To maintain. To fix. To deploy.

#FlipTheIceberg



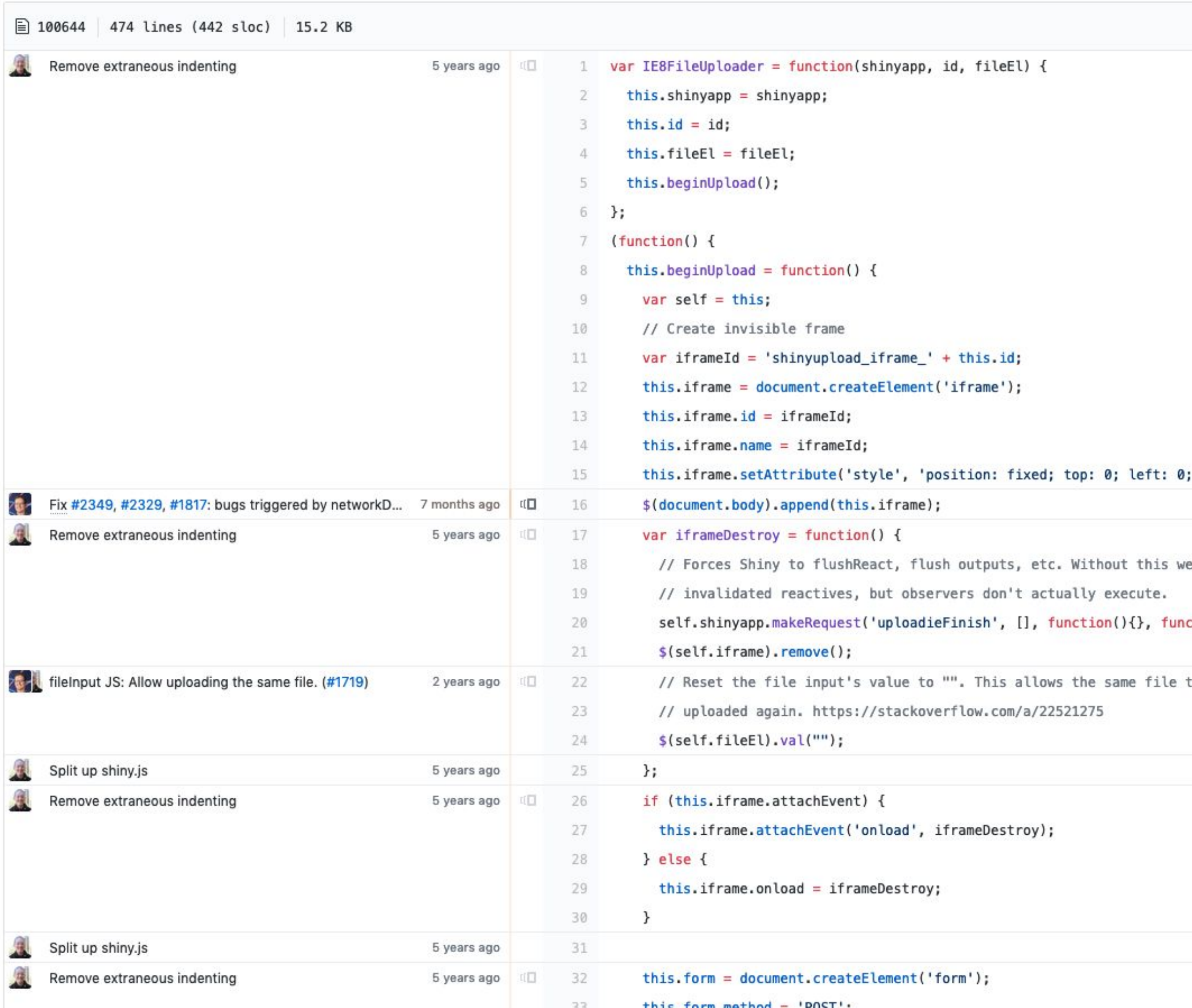
Objective

Empower you to build more, and higher quality, Shiny applications by leveraging the following techniques:

1. Source control
2. Dependency management
3. Scientific debugging
4. Modular construction
5. Unit testing
6. Functional testing
7. Performance optimization
8. Load testing

Source Control

- Software tools for retaining information about changes to source code. **git** is one.
- Nothing deleted; only changed.
- Ultimate “Undo”.
- Premier way to collaborate on software, such as with **GitHub**.
- Useful even as an individual.
- **If you do nothing else: learn to use source control.**
- https://happygitwithr.com/



100644 | 474 lines (442 sloc) | 15.2 KB

Remove extraneous indenting — 5 years ago

```
var IE8FileUploader = function(shinyapp, id, fileEl) {
  this.shinyapp = shinyapp;
  this.id = id;
  this.fileEl = fileEl;
  this.beginUpload();
};
(function() {
  this.beginUpload = function() {
    var self = this;
    // Create invisible frame
    var iframeId = 'shinyupload_iframe_' + this.id;
    this.iframe = document.createElement('iframe');
    this.iframe.id = iframeId;
    this.iframe.name = iframeId;
    this.iframe.setAttribute('style', 'position: fixed; top: 0; left: 0;
```

Fix #2349, #2329, #1817: bugs triggered by networkD... — 7 months ago

```
    $(document.body).append(this.iframe);
```

Remove extraneous indenting — 5 years ago

```
    var iframeDestroy = function() {
      // Forces Shiny to flushReact, flush outputs, etc. Without this we
      // invalidated reactives, but observers don't actually execute.
      self.shinyapp.makeRequest('uploadieFinish', [], function(){}, func
      $(self.iframe).remove();
```

fileInput JS: Allow uploading the same file. (#1719) — 2 years ago

```
      // Reset the file input's value to "". This allows the same file t
      // uploaded again. https://stackoverflow.com/a/22521275
      $(self.fileEl).val("");
```

Split up shiny.js — 5 years ago

```
    };
```

Remove extraneous indenting — 5 years ago

```
    if (this.iframe.attachEvent) {
      this.iframe.attachEvent('onload', iframeDestroy);
    } else {
      this.iframe.onload = iframeDestroy;
    }
```

Split up shiny.js — 5 years ago

```

```

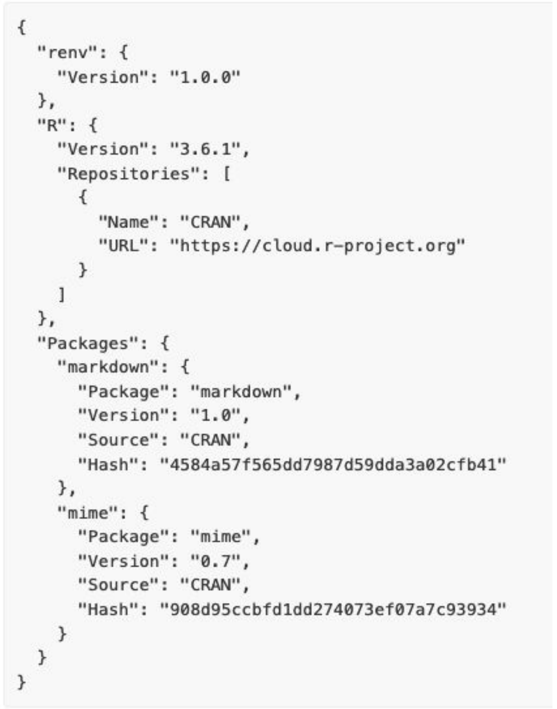
Remove extraneous indenting — 5 years ago

```
    this.form = document.createElement('form');
```

Dependency Management

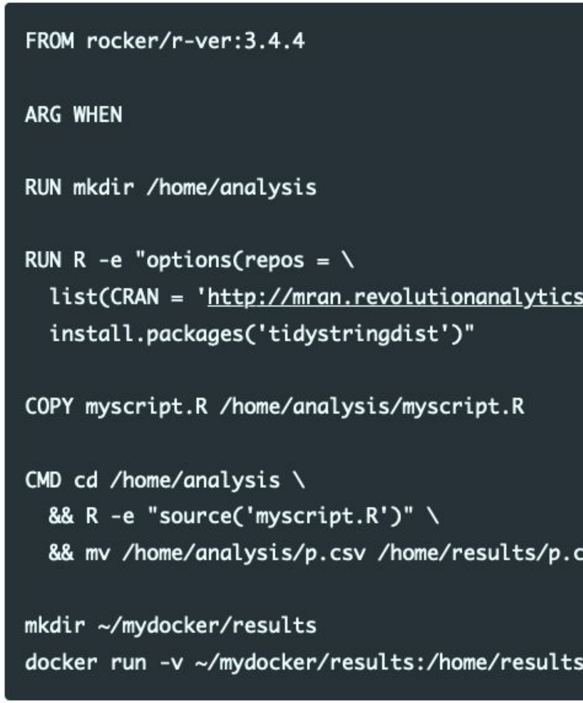
- Many kinds of dependency
  - R Package
  - System
  - Data
  - Usernames/passwords
- Apps with implicit dependencies are hard to relocate
- **Make dependencies explicit**
  - **Put them in files***
  - Bonus: put files in git
- renv (Packrat successor)
- RStudio Package Manager
- RStudio Connect
- https://environments.rstudio.com/
- https://environments.rstudio.com/docker

renv lockfile



```
{
  "renv": {
    "Version": "1.0.0"
  },
  "R": {
    "Version": "3.6.1",
    "Repositories": [
      {
        "Name": "CRAN",
        "URL": "https://cloud.r-project.org"
      }
    ]
  },
  "Packages": {
    "markdown": {
      "Package": "markdown",
      "Version": "1.0",
      "Source": "CRAN",
      "Hash": "4584a57f565dd7987d59dda3a02cfb41"
    },
    "mime": {
      "Package": "mime",
      "Version": "0.7",
      "Source": "CRAN",
      "Hash": "908d95ccbfd1dd274073ef07a7c93934"
    }
  }
}
```

Dockerfile



```
FROM rocker/r-ver:3.4.4

ARG WHEN

RUN mkdir /home/analysis

RUN R -e "options(repos = \
  list(CRAN = 'http://mran.revolutionanalytics
  install.packages('tidystringdist')"

COPY myscript.R /home/analysis/myscript.R

CMD cd /home/analysis \
  && R -e "source('myscript.R')" \
  && mv /home/analysis/p.csv /home/results/p.c

mkdir ~/mydocker/results
docker run -v ~/mydocker/results:/home/results
```

**Except, generally, passwords.*

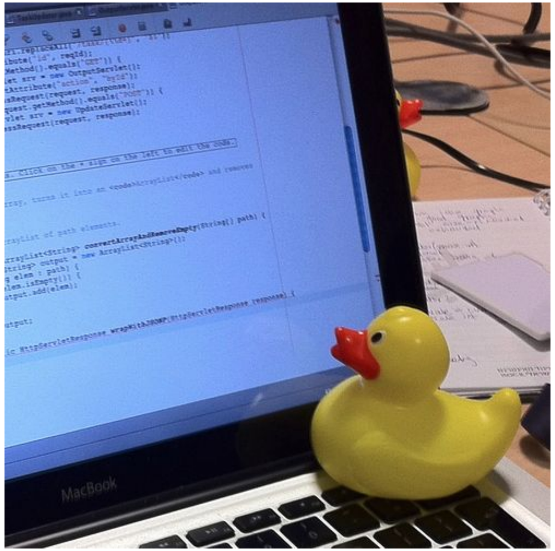
Debugging

..as The Scientific Method

1. Determine how to reliably reproduce the problem
2. Develop a theory
3. Test your theory by making changes and trying to reproduce
4. Repeat from Step 2 until you can't reproduce the problem

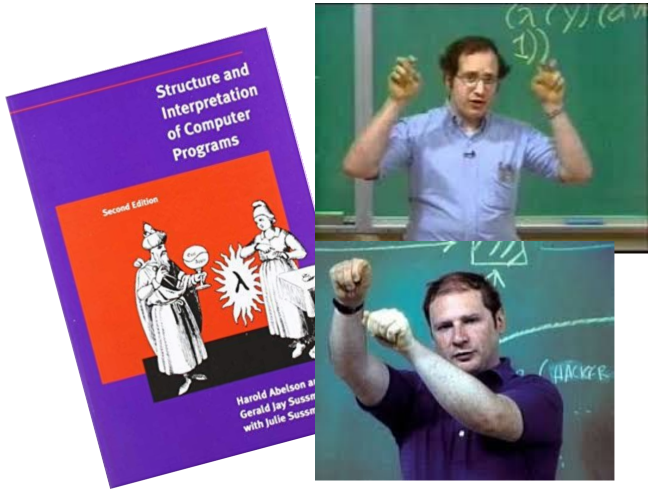
The Tools

- Master the R and RStudio debugging tools; see Advanced R
- Learn how to interpret stack traces in Shiny apps
- `base::browser()` is life
- Keep a work journal, "lab notes", so you don't go crazy
- Enlist rubber ducks
- Diagram or whiteboard copiously
- Take a break from the problem
- Get a good night's sleep



Modular Construction

“Modular construction is a powerful strategy for controlling complexity in engineering design.”

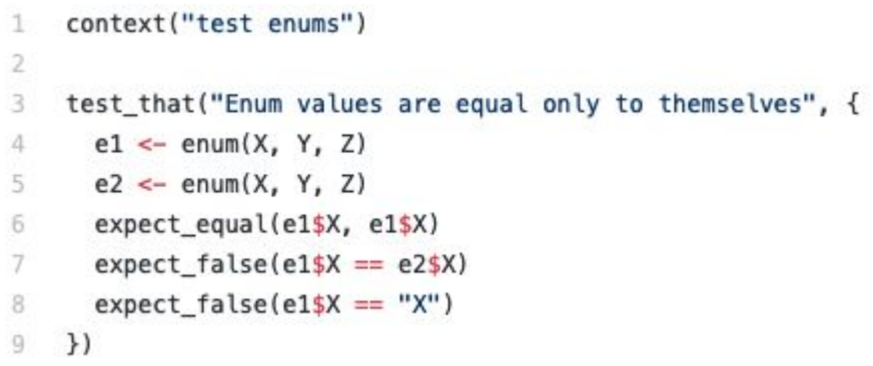


The Techniques

- **Extract code from reactives and complex functions into smaller, pure functions**
- Isolate and consolidate interactions with file system, network, databases, APIs, and other “side-effects”
- Reduce maintenance cost by making shared private packages instead of copying common code to each new app.
- Learn to use Shiny Modules to encapsulate UI code

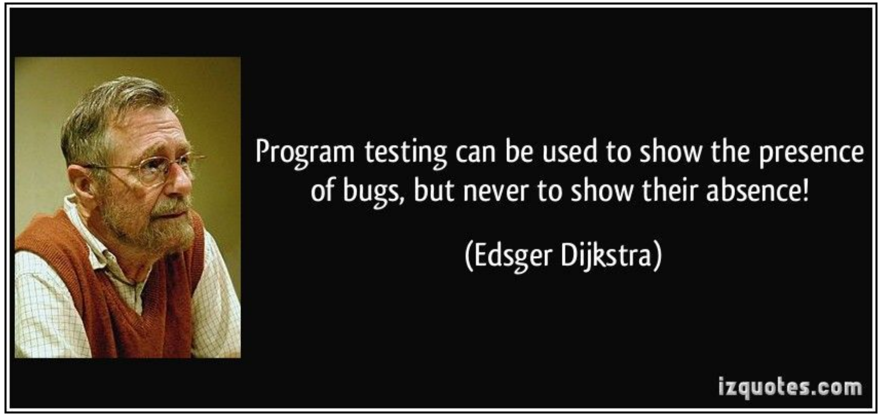
Unit Testing

- R code to check that other R code works properly
- Benefit: change code with higher confidence
- In R, the “unit” is usually a function
- Comparatively “low-level”
- Usually managed by a testing framework like testthat
- Can be run automatically by TravisCI or Github Actions
- Incredibly helpful
- Famously unhelpful
- Martin Fowler on Unit Tests



```r
context("test enums")

test_that("Enum values are equal only to themselves", {
  e1 <- enum(X, Y, Z)
  e2 <- enum(X, Y, Z)
  expect_equal(e1$X, e1$X)
  expect_false(e1$X == e2$X)
  expect_false(e1$X == "X")
})
```



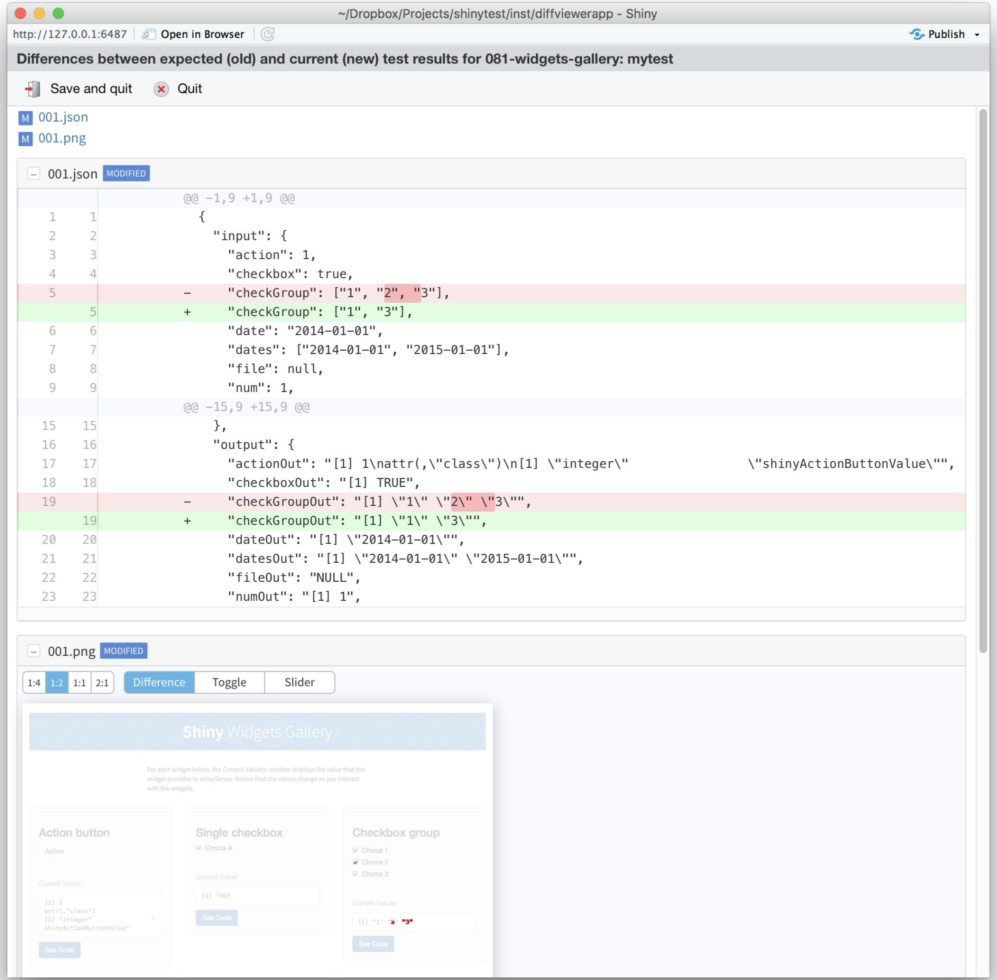
Program testing can be used to show the presence of bugs, but never to show their absence!

(Edsger Dijkstra)

izquotes.com

Functional Testing

- Test that requirements are met
- Difficult for user interfaces
- State of the art: automate user input, compare screenshots
- Benefit: Less manual testing
- Use shinytest



Differences between expected (old) and current (new) test results for 081-widgets-gallery: mytest

```
                @@ -1,9 +1,9 @@
                  {
                    "input": {
                      "action": 1,
                      "checkbox": true,
-                     "checkGroup": ["1", "2", "3"],
+                     "checkGroup": ["1", "3"],
                      "date": "2014-01-01",
                      "dates": ["2014-01-01", "2015-01-01"],
                      "file": null,
                      "num": 1,
                @@ -15,9 +15,9 @@
                    },
                    "output": {
                      "actionOut": "[1] 1\nattr(,\"class\")\n[1] \"integer\"                \"shinyActionButtonValue\"",
                      "checkboxOut": "[1] TRUE",
-                     "checkGroupOut": "[1] \"1\" \"2\" \"3\"",
+                     "checkGroupOut": "[1] \"1\" \"3\"",
                      "dateOut": "[1] \"2014-01-01\"",
                      "datesOut": "[1] \"2014-01-01\" \"2015-01-01\"",
                      "fileOut": "NULL",
                      "numOut": "[1] 1",
```

For each widget below, the Current Value(s) window displays the value that the widget provides to shinyServer. Notice that the values change as you interact with the widgets.

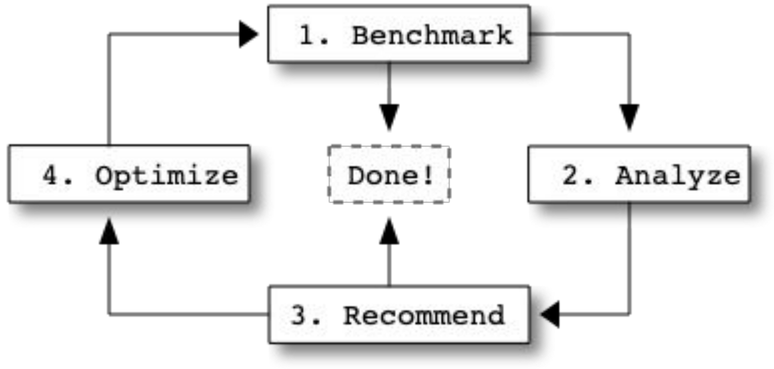
```
[1] 1
attr(,"class")
[1] "integer"                "
shinyActionButtonValue"
```

```
[1] TRUE
```

```
[1] "1" "2" "3"
```

Performance Optimization

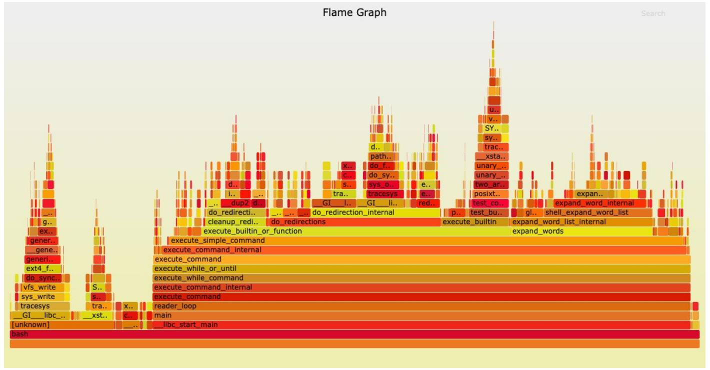
The Method: Optimization Loop



From "*Make Shiny fast by doing as little work as possible*"

The Tools

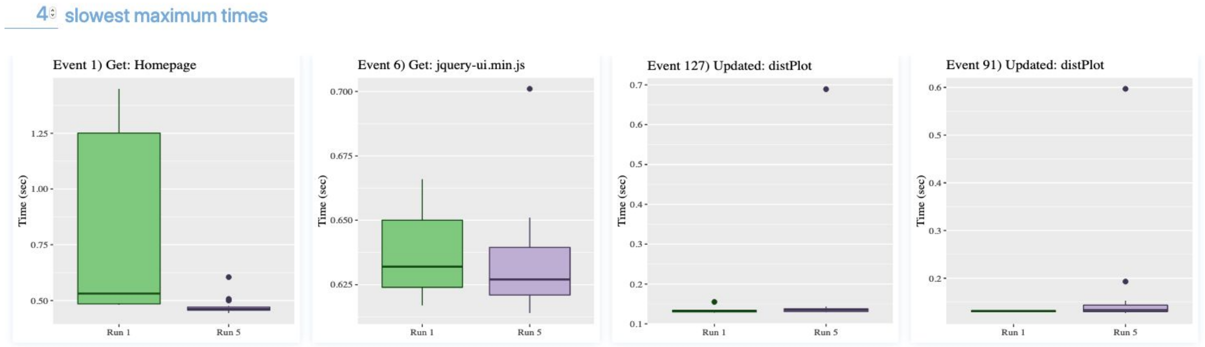
- Advanced R: Performance
- The microbenchmark package
- The profvis package and its RStudio IDE integration
- Profvis - Profiling tools for Faster R code | RStudio Webinar

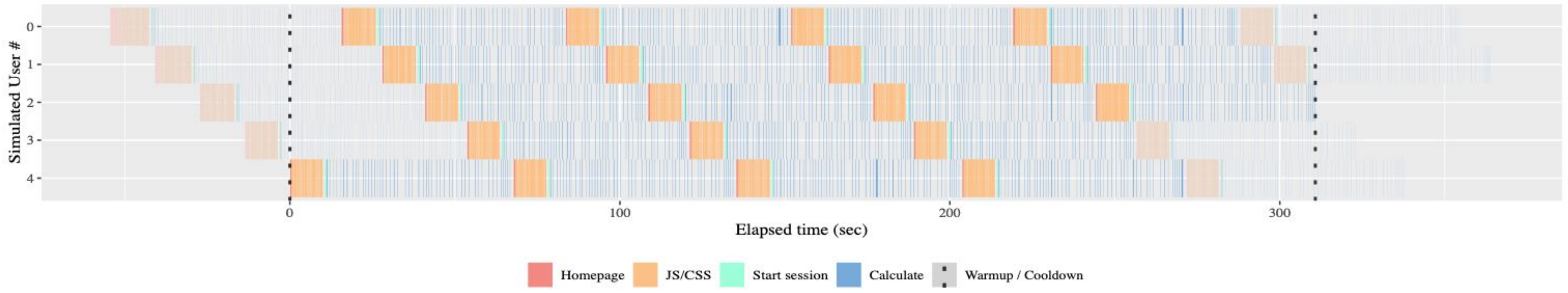


A CPU Flame Graph

Load Testing

- A kind of performance testing
- "How many concurrent users does the app support?"
- shinyloadtest R package (recording, analysis)
- `shinycannon` command-line tool (generating load)





Summary

1. Foundational: source control
    - Use git
2. Dependency management
    - Make dependencies explicit, use renv
3. Debugging
    - Scientific method, read Advanced R
4. Modular construction
    - Extract pure functions and Shiny Modules, read SICP
5. Unit testing
    - testthat
6. Functional testing
    - shinytest
7. Performance optimization
    - Scientific method again, profvis
8. Load testing
    - Surprise! Scientific method, shinyloadtest