What Software Engineers can share with Data Scientists: … with Automatic Tests

Andrea Melloncelli
andrea.melloncelli@quantide.com

sponsored by
Outline

1. Conway’s Game of Life
2. Why do tests
   2.1. Validation
   2.2. Working documentation
   2.3. Readable code
3. Testing Strategies
   3.1. Test Driven Development (TDD)
   3.2. Test After Development (TAD)
4. Testing tools
   4.1. Testthat package
   4.2. Shinytest package
Summary
1. Conway’s Game of Life

Legend:

- Dead
- Alive
1. Conway’s Game of Life

Legend:
- **Dead**
- **Alive**

1. Any live cell with fewer than two live neighbors dies, as if by isolation.

```
Alive  Alive  Alive
       Dies
       Alive
```

2. Any live cell with more than three live neighbors dies, as if by overcrowding.

```
Alive  Alive
Alive  Alive
       Dies
```

3. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

```
Alive  Alive  Alive
       Born
```

4. Any live cell with two or three live neighbors lives on to the next generation.

```
Alive  Alive  Alive
Alive  Lives
       Alive
```

http://jonathan-jackson.net/life-in-a-shade-of-ruby
https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
Outline

1. Conway’s Game of Life
2. Why do tests
   2.1. Validation
   2.2. Working documentation
   2.3. Readable code
3. Testing Strategies
   3.1. Test Driven Development (TDD)
   3.2. Test After Development (TAD)
4. Testing tools
   4.1. Testthat package
   4.2. Shinytest package
Summary
2. Why do tests?

1. Validation
2. Working documentation
3. Readable and reusable code
2. Why do tests?

2.1. Validation

```java
context("evolution of a single cell")
...
```

Context: Evolve
2. Why do tests?

2.1. Validation

```r
context("evolution of a single cell")

... test_that(desc = "rule1", ...)

... test_that(desc = "rule2", ...)

... test_that(desc = "rule3", ...)

... test_that(desc = "rule4", ...)
```

Context: Evolve

1. Test 1: rule 1
   1.1. ...

2. Test 2: rule 2
   2.1. ...

3. Test 3: rule 3
   3.1. ...

4. Tests ...
2. Why do tests?

2.1. Validation

context("evolution of a single cell")
...

test_that(desc =
    paste("Any dead cell",
          "with exactly three live neighbours",
          "becomes a live cell,
          "as if by reproduction.")
),
...

Context: Evolve

1. Test 1: rule 1
   1.1. ...
2. Test 2: rule 2
   2.1. ...
3. Test 3: rule 3
   3.1. ...
4. Tests ...
2. Why do tests?

2.1. Validation

```r
context("evolution of a single cell")
...

test_that(desc =
    paste("Any dead cell",
      "with exactly three live neighbours",
      "becomes a live cell,",
      "as if by reproduction.") ),
    code = {
      state <- dead
      evolved_state <- evolve(state, neigb = 3)
      expect_equal(evolved_state, alive)
    })
...
```

Context: **Evolve**

1. Test 1: rule 1
   1.1. ...

2. Test 2: rule 2
   2.1. ...

3. Test 3: rule 3
   3.1. Setup
   3.2. Function run
   3.3. Validation

4. Tests ...
2. Why do tests?

2.2. Working documentation

A test file provides:

1. Information about the feature (context)
2. Some working examples of how that feature is implemented (test_that)
2. Why do tests?

2.3. Readable code

**Refactoring**: improving the code without adding further functionalities.

```r
if (wday(now) > 2 &&
    wday(now) < 6 &&
    hour(now) > 8 &&
    hour(now) < 17 )
{
    cat("I'm working."
}
else {
    cat("I'm out of the office.")
}
```
2. Why do tests?

2.3. Readable code

**Refactoring**: improving the code without adding further functionalities.

```r
if (wday(now) > 2 &&
    wday(now) < 6 &&
    hour(now) > 8 &&
    hour(now) < 17)
{
    cat("I'm working.")
} else {
    cat("I'm out of the office.")
}

is_working_time <- function(time) {
    wday(time) > 2 &&
    wday(time) < 6 &&
    hour(time) > 8 &&
    hour(time) < 17
}

if (is_working_time(now))
{
    cat("I'm working.")
} else {
    cat("I'm out of office.")
}
```
Outline

1. Conway’s Game of Life
2. Why do tests
   2.1. Validation
   2.2. Working documentation
   2.3. Readable code
3. Testing Strategies
   3.1. Test Driven Development (TDD)
   3.2. Test After Development (TAD)
4. Testing tools
   4.1. Testthat package
   4.2. Shinytest package
Summary
3. Testing Strategies

1. Test Driven Development (TDD)

Tests then Implementation

2. Test After Development (TAD)

Implementation then Tests
3. Testing Strategies

3.1. Test Driven Development (TDD)

Add a Test

Tests

Pass

Fail
3. Testing Strategies

3.1. Test Driven Development (TDD)

Add a Test

Tests

Pass

Fail

Make slight changes

Tests

Fail

Pass
3. Testing Strategies

3.1. Test Driven Development (TDD)

Refactor

Add a Test

Make slight changes

Tests

Pass

Fail

Refactor

Make slight changes

Tests
3. Testing Strategies

3.2. Test After Development (TAD)

Test After Development (TAD)

Implementation then Tests

1. Old way of operate, I only need to add tests to my implementation
2. It is always available
3. Useful when the result can’t be predicted (models, ... )
Outline

1. Conway’s Game of Life
2. Why do tests
   2.1. Validation
   2.2. Working documentation
   2.3. Readable code
3. Testing Strategies
   3.1. Test Driven Development (TDD)
   3.2. Test After Development (TAD)
4. Testing tools
   4.1. Testthat package
   4.2. Shinytest package
Summary
4. Testing tools

1. Testthat package (TDD + TAD)

2. Shinytest package (TAD)
4. Testing tools

4.1. Testthat Package

- Complete set Testing tools
- Developed by RStudio
- Compatible with different Testing Strategies (TAD and TDD)
4. Testing tools

4.1. Testthat Package

context("evolution of a single cell")
...

test_that(desc =
    paste("Any dead cell",
    "with exactly three live neighbours",
    "becomes a live cell,",
    "as if by reproduction."),
    code = {
        state <- dead

        evolved_state <- evolve(state, neigb = 3)

        expect_equal(evolved_state, alive)
    })
...

Context: Evolve

1. Test 1: rule 1
   1.1. ...
2. Test 2: rule 2
   2.1. ...
3. Test 3: rule 3
   3.1. Setup
   3.2. Function run
   3.3. Validation
4. Tests ...
4. Testing tools

4.2. Shinytest

The strategy is TAD (Test After Development), therefore:

1. Having a working Shiny application
2. Record a test using the application as the final user
3. Run all tests sequentially
4. And if something is wrong....
4. Testing tools

4.2. Shinytest

4. If something is wrong... Get notified graphically!

You see the differences between the recorded run and the current coloured out.
Summary

Why are tests so important in our work?
Questions?