Test-Driven Data Wrangling in R

And A Little on [S]OLID

Donald Sawyer

(https://www.linkedin.com/in/donaldsawyer)

Presentation & Source Code on GitHub:

https://github.com/donaldsawyer/test-driven-data-wrangling-r

About Me

phData

- Sr. Solution Architect
- We're Hiring!

- Adjunct
- Designed & Teach Big Data Engineering & Analytics

UNIVERSITY OF MINNESOTA

What is Unit Testing?

- Types of testing
  - Unit
  - Integration
  - System
  - Data Quality, Functional, Acceptance, UX, and many more…
- Unit Testing
  - Testing of an individual code module, function, or “unit”
  - Done by a developer before deployment
  - Verifies input/output of a “unit”
- Why?
  - Verify the individual components perform as expected
  - Gain confidence that “done” code is still working as the program changes

Concepts of Automated Testing

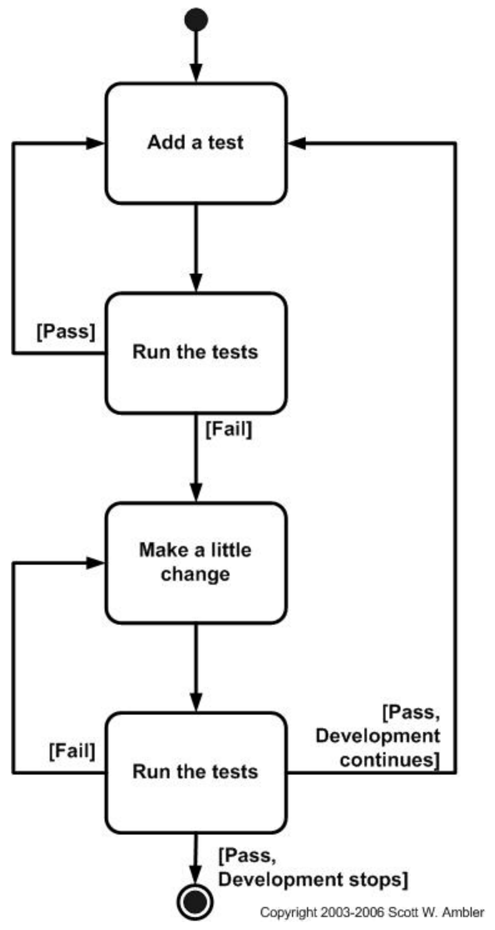
- Coded tests
  - Your tests are code
  - They need code review too
- SOLID is the key
- They should be run by your CI tools (Jenkins, Drone, etc.)
- Try to achieve high [statement, branch] coverage
- You’ll use a framework

Let’s Get Started!

An Introduction to SOLID

- SOLID Principles of Object-Oriented Design
  - S – Single Responsibility
  - O – Open/Closed Principle
  - L – Liskov Substitution
  - I – Interface Segregation
  - D – Dependency Inversion
- Single Responsibility
  - A module should have a single responsibility and that responsibility should be entirely encapsulated in the module
  - A module should have one, and only one, reason to change
  - ***THIS is what we'll unit test in this talk***

Test-Driven Development (TDD)



Add a test
[Pass]
Run the tests
[Fail]
Make a little change
[Fail]
Run the tests
[Pass, Development continues]
[Pass, Development stops]
Copyright 2003-2006 Scott W. Ambler

- Helps you understand your requirements
- Forces you to test (YAY!)
- If your test doesn't fail first, it's a bad test
  - Avoid false positives
  - Ensure better tests
- When you have tests, you can CONFIDENTLY change your code in the future
- Incorporate SOLID to make it easier
- It takes practice!

Demo App in R

Coded Unit Testing with RUnit

- Why?
  - Every time you change your code, you want to re-test it, ENTIRELY
  - Without coded testing, we skip testing pieces we’ve already tested
  - Gain confidence in testing changes to large data sets
  - What if we broke something we previously built?
    - We won’t know until a defect is reported
  - Integrate testing within the deployment process
- Package: Runit
  - Allows automated verification of code units
  - Allows a test suites to be defined and tested together
  - Allows for test results to be reported
  - Also: testthat

A Doorbuster Metric Program

Requirements

1. Shall read two product csv files
    1. Doorbuster1.csv
    2. Doorbuster2.csv
2. Shall add two metrics to the product data
    1. Products that are doorbusters but have no price assigned
    2. Products that are doorbusters and online, but are out of stock
3. Shall write out a csv with the following:
    1. Product id
    2. No price metric
    3. Online but out of stock metric

The R Example

- 0_monolith.R
    - A typical R Script
    - Contains a lot of code that all runs in sequence
- What the script does
    - Reads 2 data sets from csv (doorbuster data)
        - doorbuster1.csv & doorbuster2.csv
    - Combines the datasets
    - Adds metrics for
        - Doorbuster items missing a price
        - Doorbuster items that are online, but out of stock
    - Writes metric data to csv called doorbuster_metrics.csv

To Refactor (the TDD Way)

1. Stub out a function for the code being refactored
2. Write the appropriate unit tests
3. RUN the tests → They BETTER fail!
4. Fill the code into the method
5. RUN the tests & fix function until the tests pass
6. Move on
7. Every change to the program you make, RUN ALL THE TESTS
8. Every defect that gets reported → CREATE NEW TESTS

Refactor: Extract Method

- Purpose
  - Take a chunk of code and create a method/function so it can be tested.
  - Make a small function to follow the Single Responsibility principle
- In 0_monolith.R
  - Create a method for reading a single doorbuster csv file
  - Create a method that reads both doorbuster csv files
  - Create a method that adds metric for doorbusters with no price
  - Create a method that adds metric for doorbusters that are online but out of stock

Refactor 1: Reading Files

- Create a function that reads a csv file
  - read.doorbuster.csv()
- Create a function that reads the specific files and combines them
  - read.all.doorbuster.files()
- Why two functions?

- Replace data acquisition functionality in monolith

Refactor 2: Adding Metrics

- Create two new functions for the metrics required
  - add.column.db.noprice
  - add.column.db.online.outofstock

- Replace manipulations in monolith

Feature Add: Snake Case Columns

- Convert columns from column.name format to column_name format
- Functions
  - Convert a string to snake case
  - Convert array of strings to snake case

- Add to monolith

Quick Tips

1 Test > No Tests. Always.

Where there’s smoke,
there’s fire.

You’ll need to learn a lot of new things.

If you can “find time” to support defects and product issues, you can make time to start testing.

Helpful Resources

- TDD
  - https://en.wikipedia.org/wiki/Test-driven_development
  - http://www.agiledata.org/essays/tdd.html
- Refactoring
  - http://refactoring.com/
  - Code Smells: https://en.wikipedia.org/wiki/Code_smell