Proper-ly Testing Elixir

Paul Daigle

2020-Aug-12

Outline

Outline

Why Do We Test

The Sixth Law of Software Design

The degree to which you know how your software behaves is the degree to which you have accurately tested it.
- Max Kanat-Alexander, “Code Simplicity”

- Testing captures intent
- Testing verifies behavior
- Test cases define the edges of behavior

Typical Types of Testing

Unit	Test behavior of individual functions
Integration	Test interaction of modules
Behavior	Test system inputs/outputs

Limitations of Testing

In all types of test, it is up to the test writer to determine the test cases.

Good Practices in Testing

- Avoid hard coded values
- Look for edge cases
- Make tests documentary

Property Based Testing

Works by defining what the code should do and letting the system generate cases.

Outline

Parts of a Property Test

Invariants

Defines what is always true about the output.
Examples:

- In a sorted list, every item is <= the next item.
- An html document is enclosed by `<html><\html>` tags.

Generators

Defines the edges of the input.
Examples:

- a list of integers
- a valid http request

Property Testing Framework

A property testing framework should provide *generators*, a way to create new generators, and run inputs based on those generators against defined *invariants*.

Outline

Example Problem

Partiphification

We have *n* jobs to distribute over *m* processes. Our goal is to write a function that partitions the jobs equally.

1. Jobs are input as an array
2. We want the array split into subarrays
3. The split should be as even as possible
4. The function should be as generic as possible

Invariants

- We should have *m* sublists
- Every job should be in a sublist
- Every job should appear once in all sublists
- The length of every sublist should be +-1 of any other

Example Code

It's time to look at some code!

Outline

General Thoughts

Property testing generates test cases for you, leaving you free to focus on the rules of your data transformations and code behavior.

Good Practices!

- Avoid hard coded values (Generators)
- Look for edge cases (Generators)
- Make tests documentary (Invariants)

Challenges

- Unfamiliar
- Complex generators
- Not a lot of guidance

Questions?

Twitter: @philosodad
Twitch: @philosodad
github: github.com/philosodad
medium: medium.com/perplexinomicon-of-philosodad
Really, I'm pretty sure if you just search "philosodad" whatever comes back is probably me.