
Metrics & Methodologies for Test Suite Design

Sean Olszewski

Cocoaheads Boston - September 2018



Hello!

Engineer for Pivotal Labs

Daily TDD Practitioner

Musician & Sound Designer

@_chefski_



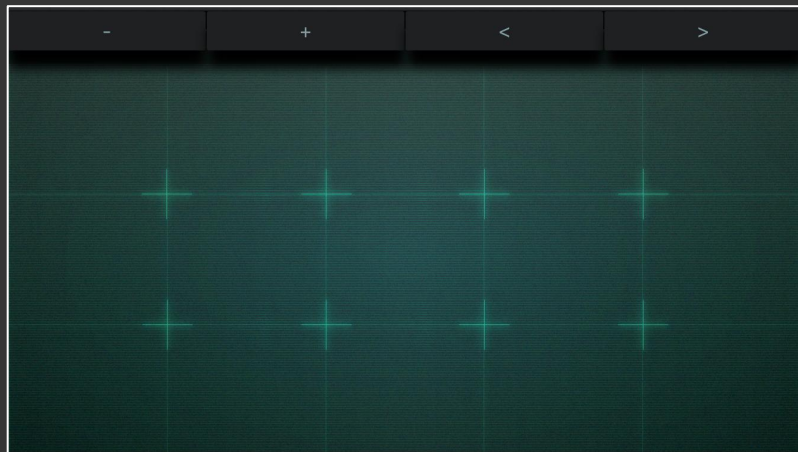
Session Overview

- **My App - Arper**
Ground the talk in something concrete
- **Test Suite Engineering**
Talk about ways to engineer effective test suites
- **Review**
Go over the topics we went through

Arper

— Arper

- Music app
- Ambient/drone music
- Entirely test-driven
 - Quick/Nimble
- Under active development



Arper - Components

ButtonBankView

Lays out buttons so they can be played. Notifies a delegate that a button is pressed.

MIDINoteMapping

Determines which button corresponds to a note.

AudioEngineManager

Handles note routing into the audio engine. Contains logic for interpreting various controls.

AudioEngine

Handles converting notes into sound.

Test Suite Engineering



Concepts

- **Responsibilities**
The purpose and benefits of a test suite
- **Metrics**
Measurable and meaningful details for assessing the efficacy of your test suites
- **Patterns**
Clearly defined and repeatable ways to code test suites
- **Methodologies**
Ways that you can use patterns & metrics to engineer an effective test suite

Terminology

→ **Test**

A way to prove something works, usually automatically.

→ **Test Subject**

The component you are interested in proving works.

→ **Behavior**

What a test subject is supposed to do; what we are interested in testing

→ **Test Double**

A component which stands-in for a dependency of the test subject (**mocks**, **spies**, **fakes**, etc)

Responsibilities

The purpose and benefits of a test suite

—

Proving what you're
building is **coded**
correctly.

—

Showing how your
code **works by**
example.

—

**Improving your
ability to make
changes to your
code base.**

—

Create software at a
lower cost and a
faster rate.

Metrics

Measurable and meaningful details for
assessing the efficacy of your test
suites

Metrics

Signal-to-Noise Ratio

how clearly a test failure indicates a specific fault or issue in your code base

Maintenance Cost

how much effort must go into keeping a test or test suite effective

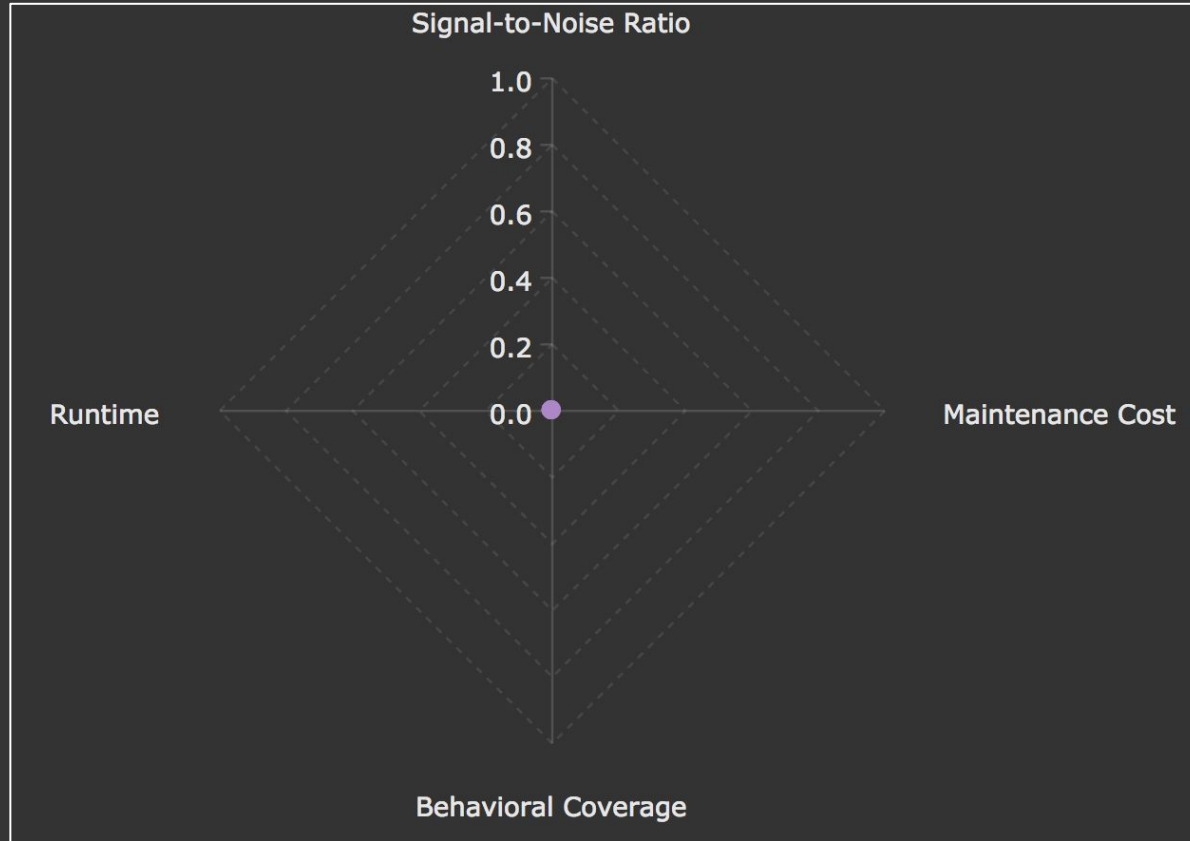
Behavioral Coverage

how many of the system's behaviors are exercised by the test or test suite

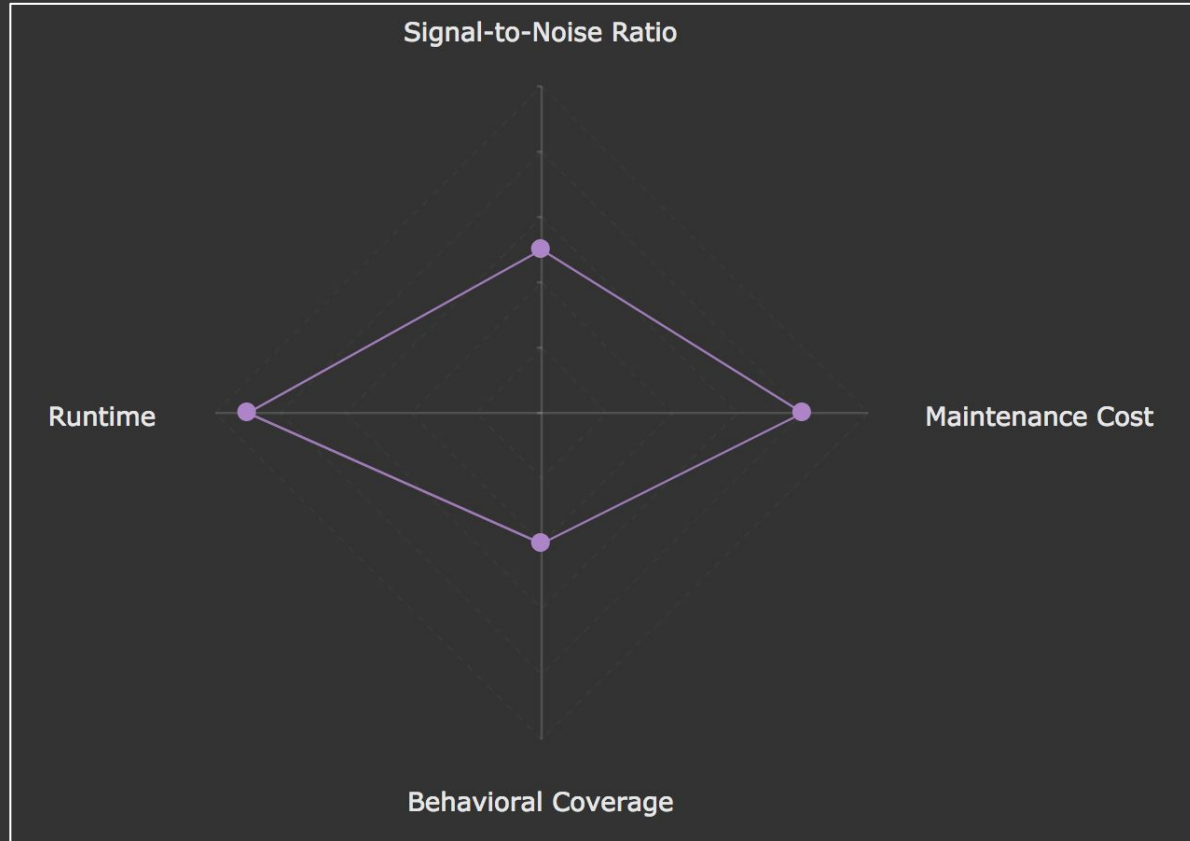
Runtime

how long a test or test suite must run for before it reveals an issue

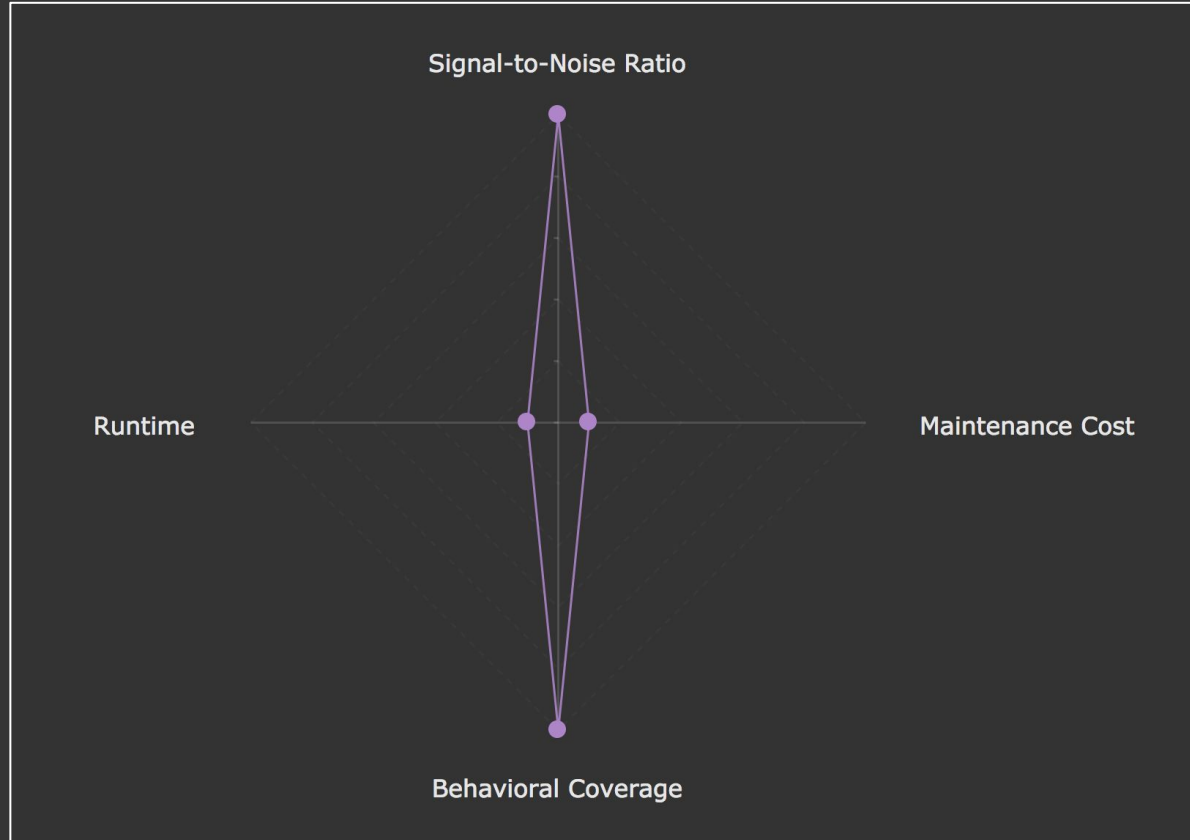
— Metrics



— Metrics



— Metrics



Patterns

Clearly defined and repeatable ways to
code test suites

Patterns

Overview

Collaboration Test

A test that proves a subject uses a dependency correctly

“Does this method call another method on a passed-in dependency?”

Functional Test

A test that proves the subject returns a specific output for a specific input

“Does this method return y when I give it x ?”

Contract Test

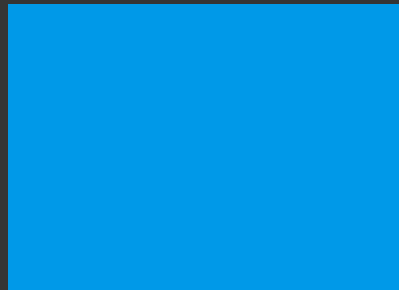
A test of an abstract interface that proves certain behaviors about all the implementers of an interface

“Does every implementation of this method return 15 unique elements?”

Arper - Components

ButtonBankView

Lays out buttons so they can be played. Notifies a delegate that a button is pressed.



Problem

A user must be able to press a button so that the synthesizer can know to make a sound

View layer must not have any other logic

Solution

Create a *UIView* which encapsulates handling the buttons

Delegate out responding to button presses

Use *IndexPaths* to refer to a particular button

```
@objc protocol ButtonBankViewDelegate: class {  
    func received(noteEvent: NoteEvent, from indexPath: IndexPath)  
}
```

```
class ButtonBankView: UIView {  
    var buttonBank: [[PressureButton]]  
  
    weak var delegate: ButtonBankViewDelegate?  
  
    init(frame: CGRect, delegate: ButtonBankViewDelegate)  
}
```

Collaboration Test

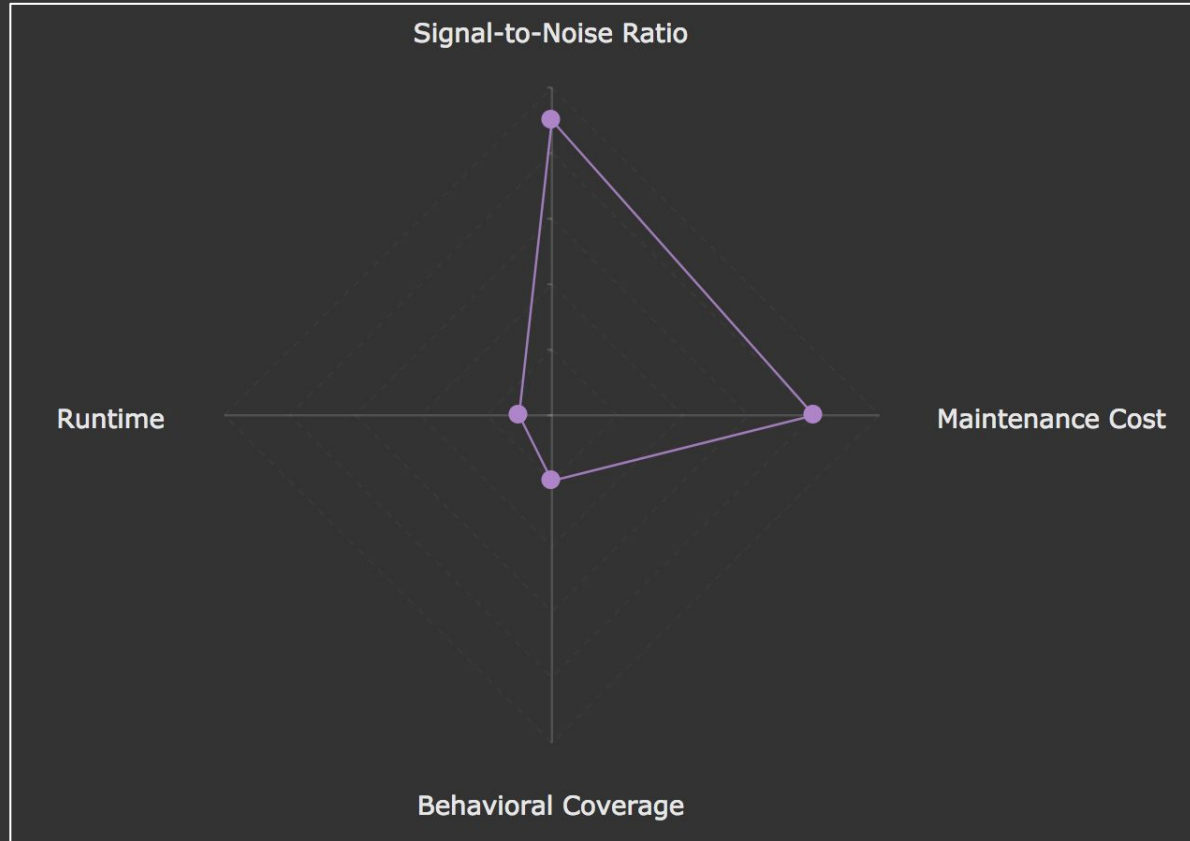
A test which proves a subject uses a dependency correctly

Uses test doubles for validation

Not a test of behavior

Effective for testing against non-deterministic or complex dependencies

— Metrics



```
class ButtonBankViewSpec: QuickSpec {
    override fun spec() {

        var delegateSpy: ButtonBankDelegateSpy! // conforms to ButtonBankViewDelegate
        var subject: ButtonBankView!

        describe("ButtonBankView") {

            beforeEach {
                delegateSpy = ButtonBankDelegateSpy()
                subject = ButtonBankView(frame: .zero, delegate: delegateSpy)
            }

            it("calls its delegate any time a button receives a touch event") {}
        }
    }
}
```

```
delegateSpy = ButtonBankDelegateSpy() // conforms to ButtonBankViewDelegate
subject = ButtonBankView(frame: .zero, delegate: delegateSpy)
```

```
protocol Spy {  
    var methodCalls: [String] { get }  
}
```

```
@objc protocol ButtonBankViewDelegate: class {  
    func received(noteEvent: NoteEvent, from indexPath: IndexPath)  
}
```

```
class ButtonBankDelegateSpy: Spy, ButtonBankViewDelegate {  
  
    private(set) var methodCalls = [String]()  
    private(set) var noteEvents = [NoteEvent]()  
    private(set) var indexPaths = [IndexPath]()  
  
    func received(noteEvent: NoteEvent, from indexPath: IndexPath) {  
        methodCalls.append(#function)  
        noteEvents.append(noteEvent)  
        indexPaths.append(indexPath)  
    }  
}
```

```
it("calls its delegate any time a button receives a touch event") {  
    let expectedNoteEvents: [NoteEvent] = [.noteOn, .noteOff]  
    let expectedIndexPaths: [IndexPath] = [IndexPath(row: 0, section: 0),  
                                           IndexPath(row: 0, section: 0)]  
  
    // subject is an instance of ButtonBankView  
    let firstButton = subject.buttonBank.first?.first  
    firstButton.sendActions(for: .touchDown)  
    firstButton.sendActions(for: .touchUpInside)  
  
    // delegateSpy updates when received(noteEvent:for:) gets called  
    expect(delegateSpy.noteEvents).to(equal(expectedNoteEvents))  
    expect(delegateSpy.indexPaths).to(equal(expectedIndexPaths))  
}
```

Arper - Components

MIDINoteMapping

Determines which button corresponds to a note.

Problem

A user must be able to have a button press correspond to a note

Must support many permutations, as there are many ways to associate buttons with notes

Must use numbers to represent notes

Should use MIDI as inspiration

Solution

Create a *MIDINoteMapping* abstraction which encapsulates mapping buttons to note numbers

Receive an *IndexPath*, return a note number that's valid per MIDI spec

Use *UInt8* to refer to a note number

Support mapping from any note number

```
protocol AnyMIDINoteMapping {  
    var baseNote: UInt8 { get }  
    init(baseNote: UInt8)  
    func noteForButton(at indexPath: IndexPath) -> UInt8  
}
```

```
struct ChromaticNoteMapping: AnyMIDINoteMapping {  
    let baseNote: UInt8  
    func noteForButton(at indexPath: IndexPath) -> UInt8  
}
```

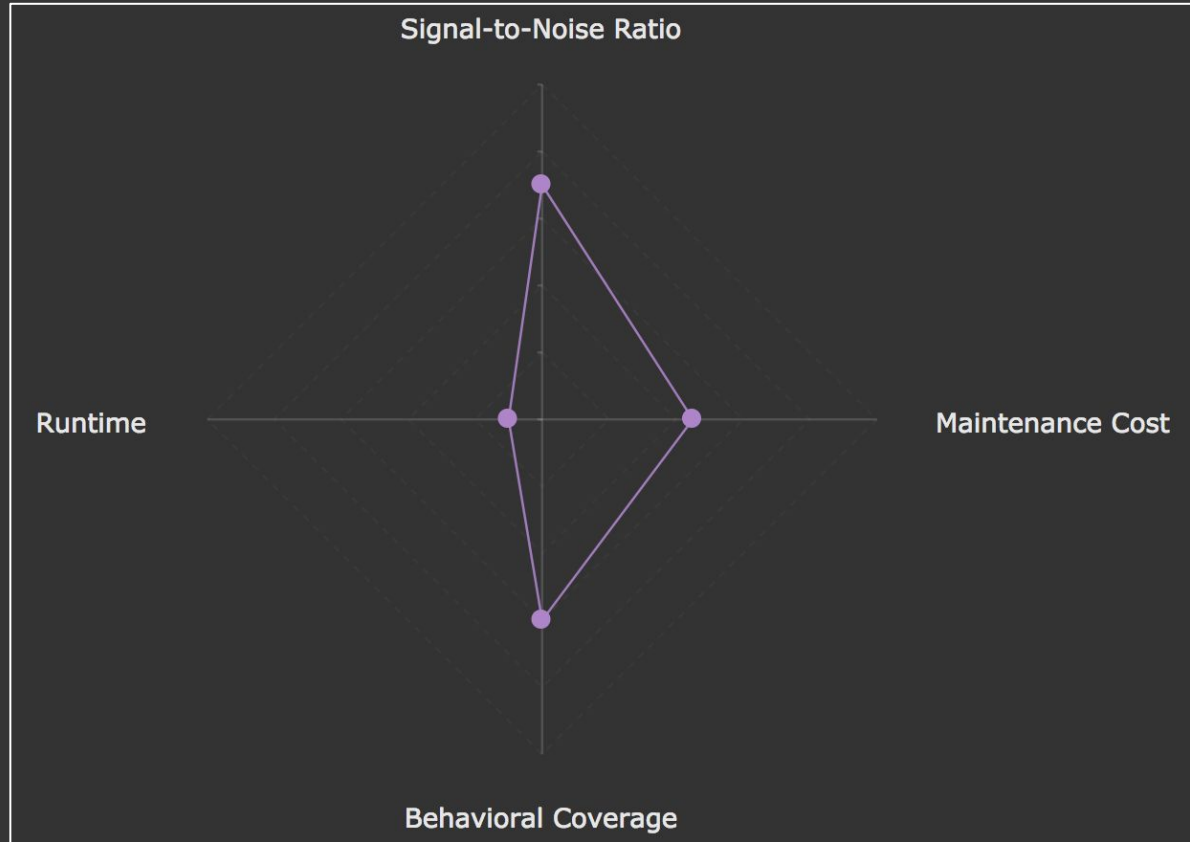
Functional Test

A test which proves the subject's interface works, with no validation of how the subject uses its dependencies.

Proves the subject returns a specific output for specific input

Sometimes called a blackbox test

— Metrics



```
describe("ChromaticNoteMapping") {  
  it("maps IndexPath into note numbers using the chromatic scale"){  
  
    let chromaticScale: [UInt8] = [60, 61, 62, 63, 64,  
                                   65, 66, 67, 68, 69,  
                                   70, 71, 72, 73, 74]  
  
    let chromaticNoteMapping = ChromaticNoteMapping(baseNote: 60)  
  
    // notes(using:) calls noteForButton(at:) 15 times  
    expect(notes(using: chromaticNoteMapping)).to(equal(chromaticScale))  
  }  
}
```

```
func notes(using mapping: AnyMIDINoteMapping) -> [UInt8] {  
    return (0..2).flatMap { notesForRow(number: $0, using: mapping) }  
}
```

```
func notesForRow(number: Int, using mapping: AnyMIDINoteMapping) -> [UInt8] {  
    return (0..4).map {  
        let indexPath = IndexPath(row: number, section: $0)  
        return mapping.noteForButton(at: indexPath) // noteForButton(at:) is the test subject  
    }  
}
```



```
protocol AnyMIDINoteMapping {
    var baseNote: UInt8 { get }
    init(baseNote: UInt8)
    func noteForButton(at indexPath: IndexPath) -> UInt8
}

struct ChromaticNoteMapping: AnyMIDINoteMapping {
    let baseNote: UInt8
    func noteForButton(at indexPath: IndexPath) -> UInt8
}

describe("ChromaticNoteMapping") {
    it("maps IndexPaths into note numbers using the chromatic scale"){

        let chromaticScale: [UInt8] = [60, 61, 62, 63, 64,
                                       65, 66, 67, 68, 69,
                                       70, 71, 72, 73, 74]

        let chromaticNoteMapping = ChromaticNoteMapping(baseNote: 60)

        // notes(using:) calls noteForButton(at:) 15 times
        expect(notes(using: chromaticNoteMapping)).to(equal(chromaticScale))
    }
}
```

Arper - Components



AudioEngineManager

Handles note routing into the audio engine. Contains logic for interpreting various controls.

AudioEngine

Handles converting notes into sound.

Example of Balancing Metrics

SNR, Maintenance Cost, & Behavioral Coverage

```
class AKAudioEngineManagerSpec: QuickSpec {
  override fun spec() {
    // variable declarations are omitted...

    describe("AKAudioEngineManager") {
      beforeEach {
        audioEngineSpy = AudioEngineSpy()
        noteMappings = [ChromaticNoteMapping(baseNote: 0)]
        subject = AKAudioEngineManager(audioEngine: audioEngineSpy,
                                       noteMappings: noteMappings)
      }

      it("routes button presses to the audio engine") {
        subject.received(noteEvent: .noteOn, from: IndexPath(row: 0, section: 0))
        subject.received(noteEvent: .noteOff, from: IndexPath(row: 0, section: 0))

        expect(audioEngineSpy.methodCalls.first).to(equal("render(notesNumbered:)"))
        expect(audioEngineSpy.methodCalls.last).to(equal("stopRendering(of:)"))

        expect(audioEngineSpy.renderedNoteNumbers).to(equal([0]))
        expect(audioEngineSpy.stoppedNoteNumbers).to(equal([0]))
      } // 12 more tests are omitted...
    }
  }
}
```

Balancing Metrics



Balancing Metrics

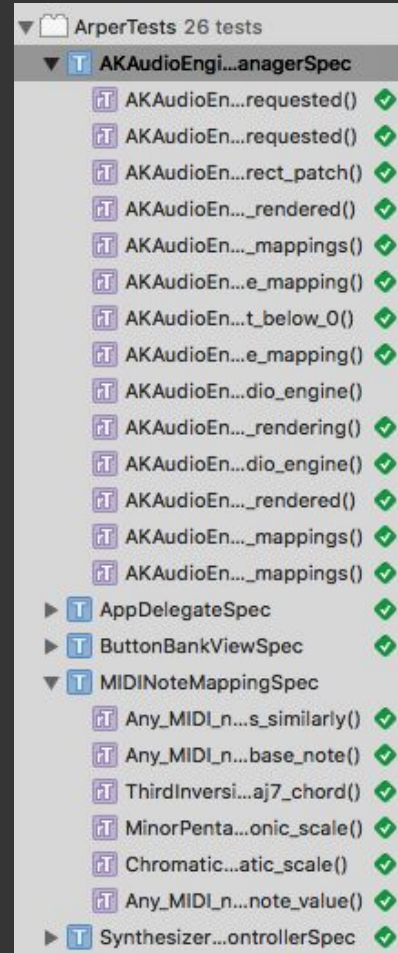
```
describe("AKAudioEngineManager") {  
    beforeEach {  
        audioEngineSpy = AudioEngineSpy()  
        noteMappings = [ChromaticNoteMapping(baseNote: 0)]  
  
        subject = AKAudioEngineManager(audioEngine: audioEngineSpy,  
                                       noteMappings: noteMappings)  
    }  
    // 13 tests omitted...  
}
```

Mutation Testing

The process of intentionally introducing issues into a code base to assess its test suite's efficacy

Balancing Metrics

```
describe("AKAAudioEngineManager") {  
  beforeEach {  
    audioEngineSpy = AudioEngineSpy()  
    noteMappings = [ChromaticNoteMapping(baseNote: 0)]  
  
    subject = AKAAudioEngineManager(audioEngine: audioEngineSpy,  
                                     noteMappings: noteMappings)  
  }  
  // ...  
}
```



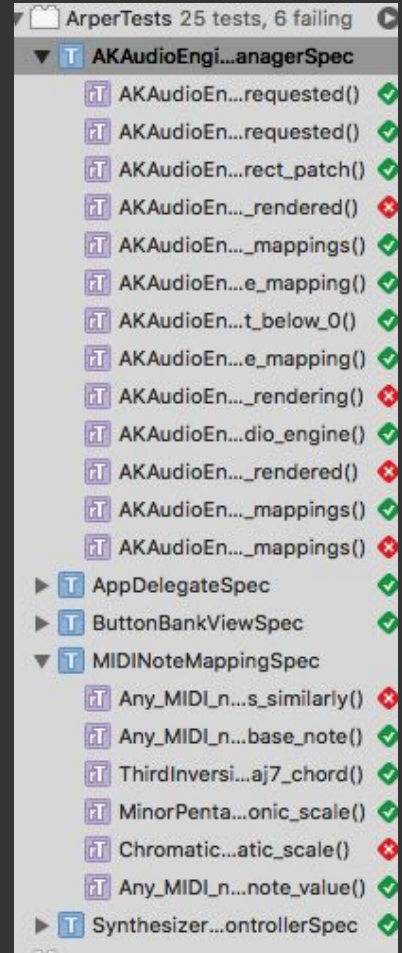
Balancing Metrics

// BEFORE

```
struct ChromaticNoteMapping: MIDINoteMapping {  
    let intervals: [UInt8] = [1]  
}
```

// AFTER

```
struct ChromaticNoteMapping: MIDINoteMapping {  
    let intervals: [UInt8] = [0]  
}
```



Balancing Metrics

Possible Solution

Isolate *AudioEngineManager* from changes in *ChromaticNoteMapping*
with a test double

Contract Test

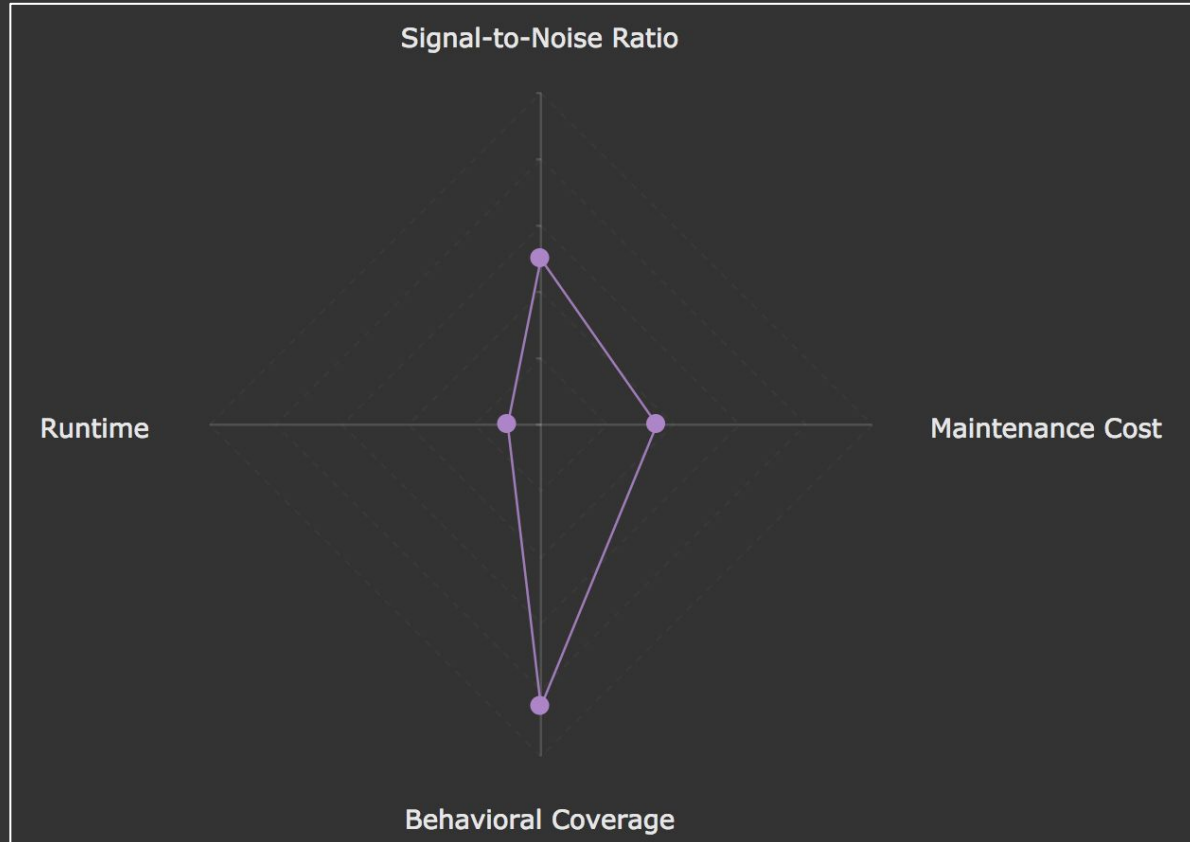
A test of an abstract interface that proves certain behaviors about all the implementers of an interface.

In Swift, used for types that conform to protocols or subclass.

Used when a type system doesn't completely cover the behavior of the subject's interface.

Test multiple types at once

— Metrics



— Balancing Metrics

```
protocol AnyMIDIINoteMapping {  
    var baseNote: UInt8 { get }  
    init(baseNote: UInt8)  
    func noteForButton(at indexPath: IndexPath) -> UInt8  
}
```

Balancing Metrics

```
describe("Any MIDI note mapping") {  
  it("maps 15 notes with no repeating notes") {}  
  
  it("maps notes starting from a provided base note") {}  
  
  it("defaults notes above the allowable range to the maximum MIDI note value") {}  
}
```

Balancing Metrics

```
describe("Any MIDI note mapping") {  
    it("maps 15 notes with no repeating notes") {  
        let mappings = allMIDINoteMappings(usingBaseNote: 60)  
  
        for anyMapping in mappings {  
            // ...  
        }  
    }  
}
```

— Balancing Metrics

```
func allMIDIINoteMappings(usingBaseNote note: UInt8) -> [AnyMIDIINoteMapping] {  
    return [ChromaticNoteMapping(baseNote: note),  
           // ...  
           MIDIINoteMappingFake(baseNote: note)]  
}
```


Balancing Metrics

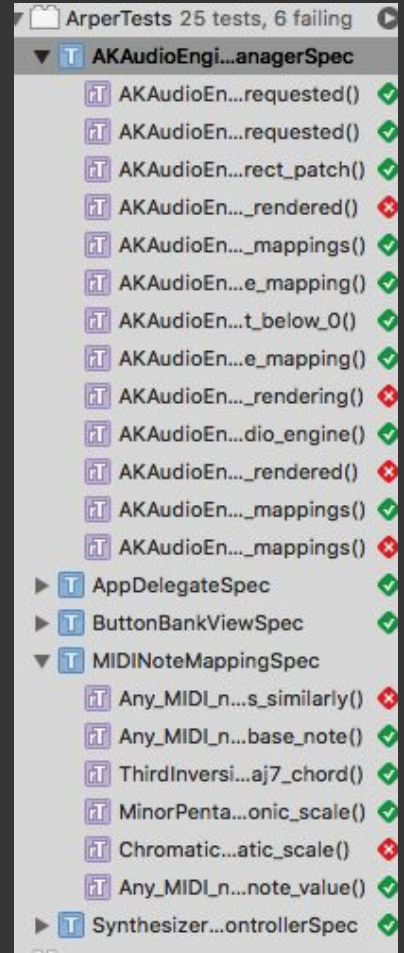
```
it("maps 15 notes with no repeating notes") {  
    let mappings = allMIDINoteMappings(usingBaseNote: 60)  
  
    for anyMapping in mappings {  
  
        // notes(using:) calls anyMapping.noteForButton(at:) 15 times  
        expect(notes(using: anyMapping)).to(haveCount(15))  
        expect(notes(using: anyMapping)).to(haveUniqueElements(15))  
  
        let theFirstNote = anyMapping.noteForButton(at: IndexPath(row: 0, section: 0))  
        let theFirstNoteMappedAgain = anyMapping.noteForButton(at: IndexPath(row: 0, section: 0))  
        expect(theFirstNote).to(equal(theFirstNoteMappedAgain))  
    }  
}
```

```
class MIDINoteMappingFake: AnyMIDINoteMapping {  
  
    let baseNote: UInt8  
  
    required init(baseNote: UInt8) {  
        self.baseNote = baseNote  
    }  
  
    func noteForButton(at indexPath: IndexPath) -> UInt8 {  
        return noteMapping[indexPath.row][indexPath.section]  
    }  
}
```

```
private extension MIDINoteMappingFake {  
    var noteMapping: [[UInt8]] {  
  
        let mapping: [[UInt8]] = (0...2).map { rowNumber in  
            return notesForRow(number: rowNumber)  
        }  
  
        return mapping  
    }  
  
    func notesForRow(number: UInt8) -> [UInt8] {  
        return (0...4).map { columnNumber in  
  
            if number == 0 && columnNumber == 0 {  
                return min(baseNote, 127)  
            }  
  
            let rowOffset = UInt8(number) * 5  
            let columnOffset = UInt8(columnNumber)  
            let note = baseNote + rowOffset + columnOffset  
  
            return min(note, 127)  
        }  
    }  
}
```

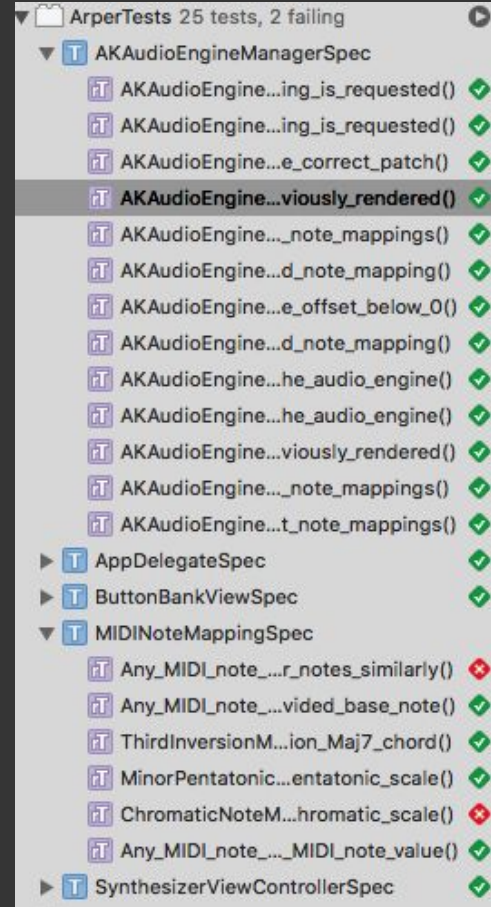
Balancing Metrics

```
describe("AKAudioEngineManager") {  
    beforeEach {  
        // ...  
        noteMappings = [ChromaticNoteMapping(baseNote: 0)]  
        // ...  
    }  
}
```



Balancing Metrics

```
describe("AKAudioEngineManager") {  
    beforeEach {  
        // ...  
        noteMappings = [MIDINoteMappingFake(baseNote: 0)]  
        // ...  
    }  
}
```



Balancing Metrics

What was improved?

Signal-to-Noise Ratio

Balancing Metrics

What was worsened?

—
Balancing Metrics

Maintenance Cost

—
Balancing Metrics

Behavioral Coverage

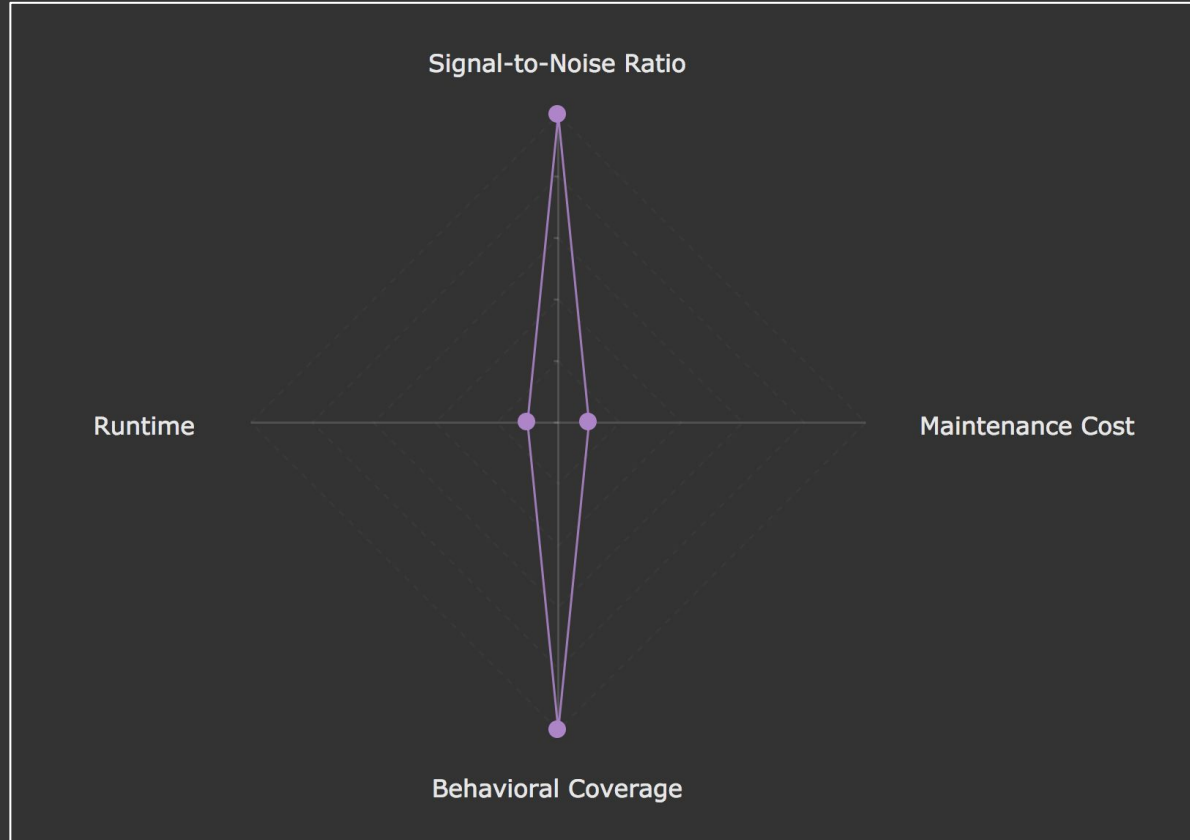
Balancing Metrics

**Was it the right decision to
make?**

Balancing Metrics

**Depends on what you
need your test suite to do
for you**

— Metrics



Review



Review

Metrics

Signal-to-Noise Ratio

how clearly a test failure indicates a specific fault or issue in your code base

Maintenance Cost

how much effort must go into keeping a test or test suite effective

Behavioral Coverage

how many of the system's behaviors are exercised by the test or test suite

Runtime

how long a test or test suite must run for before it reveals an issue

Review

Testing Patterns

Collaboration Test

A test that proves a subject uses a dependency correctly

“Does this method call another method on a passed-in dependency?”

Functional Test

A test that proves the subject returns a specific output for a specific input

“Does this method return y when I give it x ?”

Contract Test

A test of an abstract interface that proves certain behaviors about all the implementers of an interface

“Does every implementation of this method return exactly 15 elements?”

Review

Testing Methodologies

Test Doubling

Replacing a subject's dependency with an implementation only meant for testing.

Use sparingly.

Mutation Testing

Introducing issues into your code to assess how effective your test suite is.

Practice often.

Behavior Driven Development

Designing components and tests with a focus on what something is supposed to do, without regard for implementation.

Practice always.

Review

Additional Resources - Books

Test Driven Development: By Example

Kent Beck

Xunit Test Patterns

Gerard Meszaros

Working Effectively with Legacy Code

Michael Feathers

Review

Additional Resources - Websites/Blogs

Introducing BDD

<https://dannorth.net/introducing-bdd/>

The Test Double Rule of Thumb

<https://engineering.pivotal.io/post/the-test-double-rule-of-thumb/>

Joe Masilotti's Blog

<http://masilotti.com/>

Thank you!

Sean Olszewski

github.com/SeanROlszewski

@__chefski__

