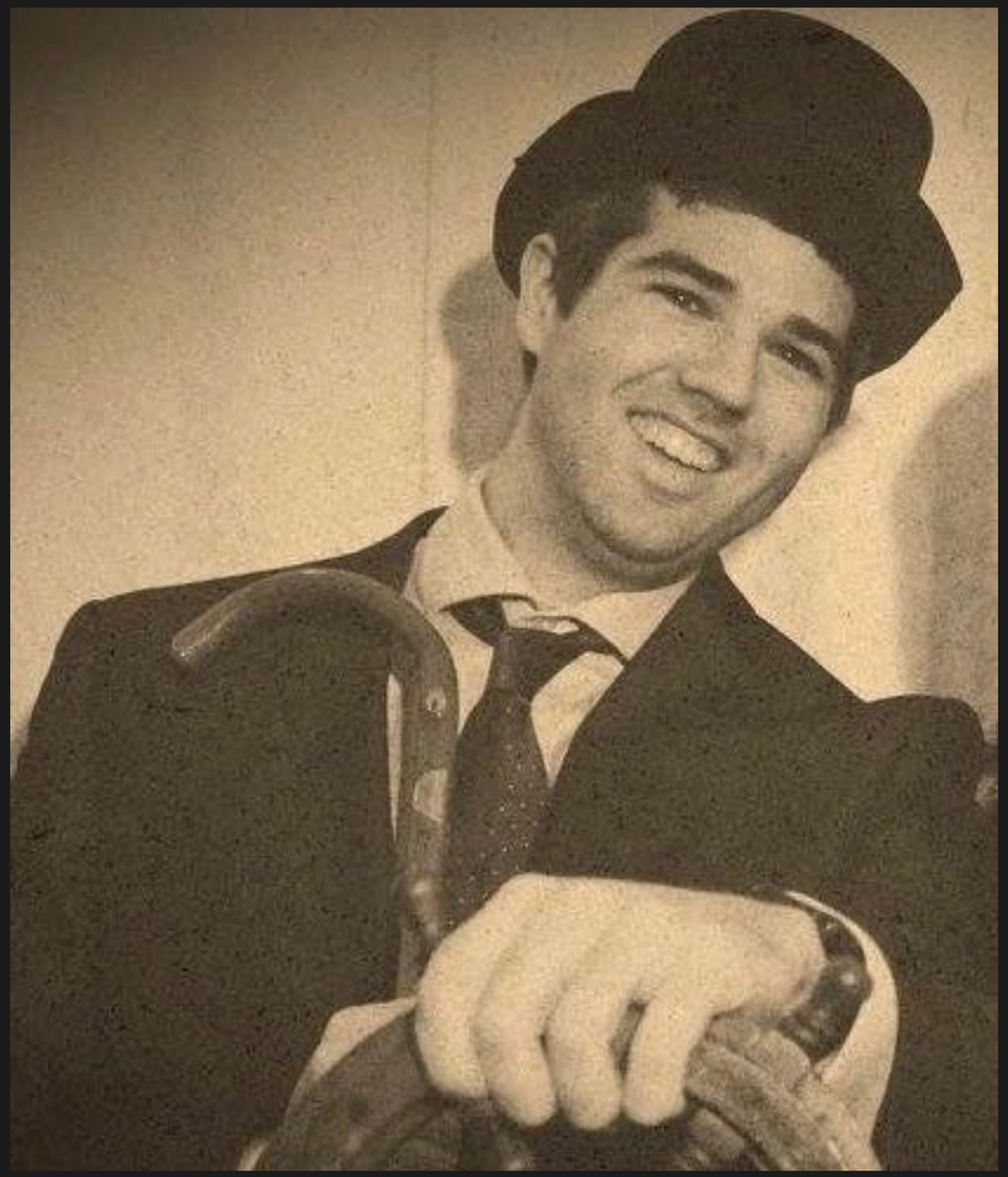
Swift in a Large Objective-C Codebase

Greg Spiers – @gspiers – March 2017

About Me

- Working as a Principal Software Engineer on iPlayer Radio at the BBC in Salford
- Long time Objective-C developer
- Ruby and Ruby on Rails before mobile
- Started shipping Swift in production this year 🎉



Agenda

- iPlayer Radio history
- Why Swift now?
- Rules we followed
- Problems and solutions
- Where we are now

iPlayer Radio

- Not small, ~97 Kloc and an additional ~80 Kloc of libraries
- Current code base three years old with code brought over from original outsourced codebase
- All Objective-C
- Millions of happy users

Why Swift Now?

That was me while watching the 2014 WWDC keynote

Why Swift Now?

- Swift 3 had just shipped
  - 2 -> 3 migration pain* won't be repeated
- BBC has a culture of learning
- We had a dependency coming in shortly that was written in Swift

* Firefox/Mozilla Swift migration, https://mozilla-mobile.github.io/ios/firefox/swift/core/2017/02/22/migrating-to-swift-3.0.html

Stop Fighting it and Embrace it

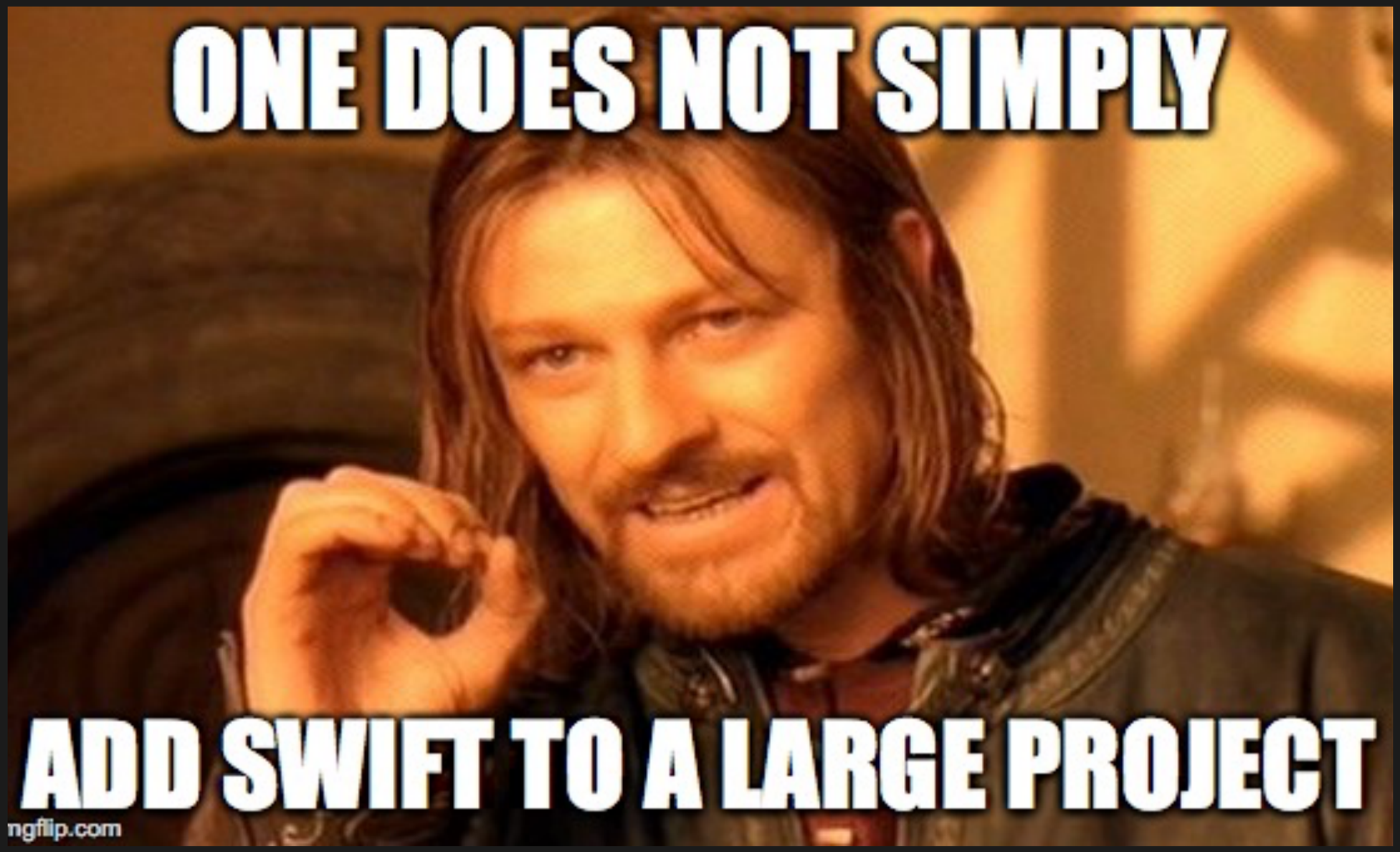
*In a controlled manner

Other Advantages

- Safer and more stable codebase in the long run
- Some iOS devs already out there that only know Swift (we'll be hiring these people soon)
- The devs wanted it as our careers depended on having the knowledge
- We needed to use Swift in anger not just in our side projects
- We had buy in from product managers (there will be some rework)
- We have an awesome Tech Lead who helped us develop a strategy and some ground rules

Disclaimer

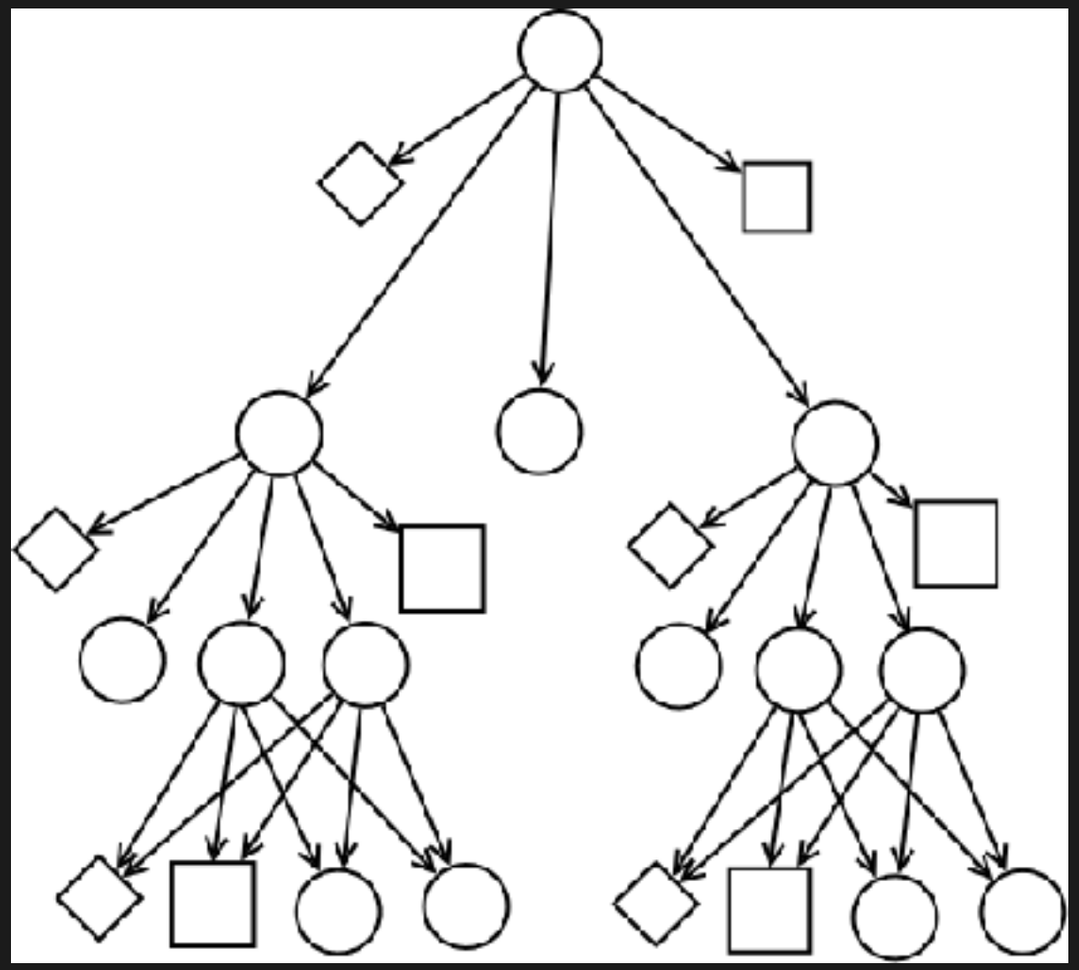
* This may not work for you, but it worked for us



Ground Rules

Ground Rules

1. Start at the leaves of the dependancy graph
   - Add Swift where it doesn't interact with very much of the system



Ground Rules

2. Write Swift code to interact with Objective-C
   - No structs/enums/tuples and other Swift only features
   - Be happy with the type safety and other awesome language features we get
   - Don’t get hung up on writing “Pure Swift”

“Pure Swift”

When asked about the tension regarding writing “Pure Swift” that doesn’t touch the Objective-C runtime **Chris Lattner** said,

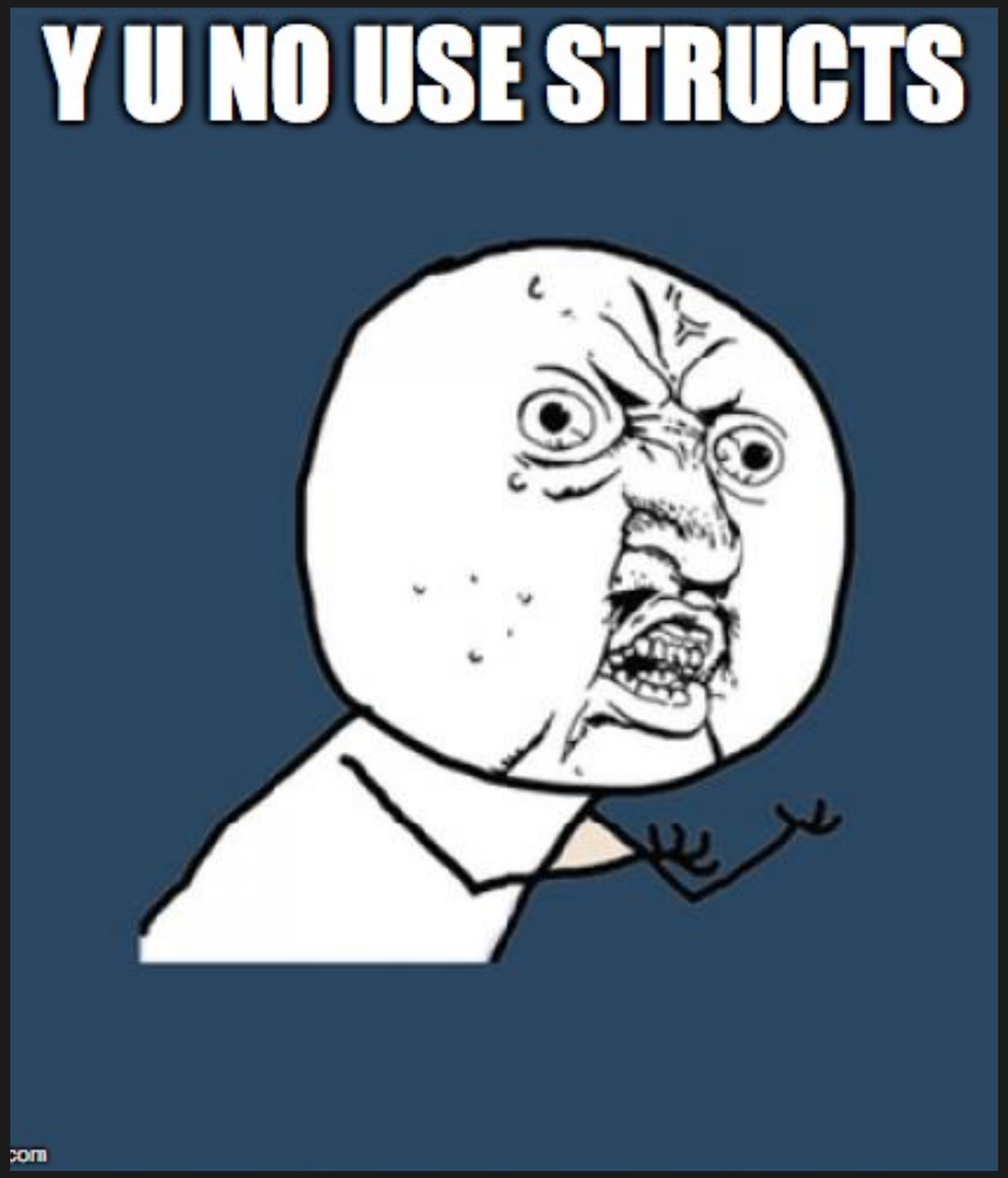
“I guess it makes some people feel good?”

and went on to explain the features missing from Swift that the Obj-C runtime provides will come but it’s not bad to use the Obj-C subset of the language.

Accidental Tech Podcast, episode 205, http://atp.fm/episodes/205

“Pure Swift”

- Use classes for now
  - No ‘wrappers’ for structs/ enums*
  - All of UIKit is classes anyways!
  - Speed isn’t a problem for our app
- Use Swift when it speeds up development, not slows it down



*Bridging Swift Types to Objective-C, http://blog.benjamin-encz.de/post/bridging-swift-types-to-objective-c/

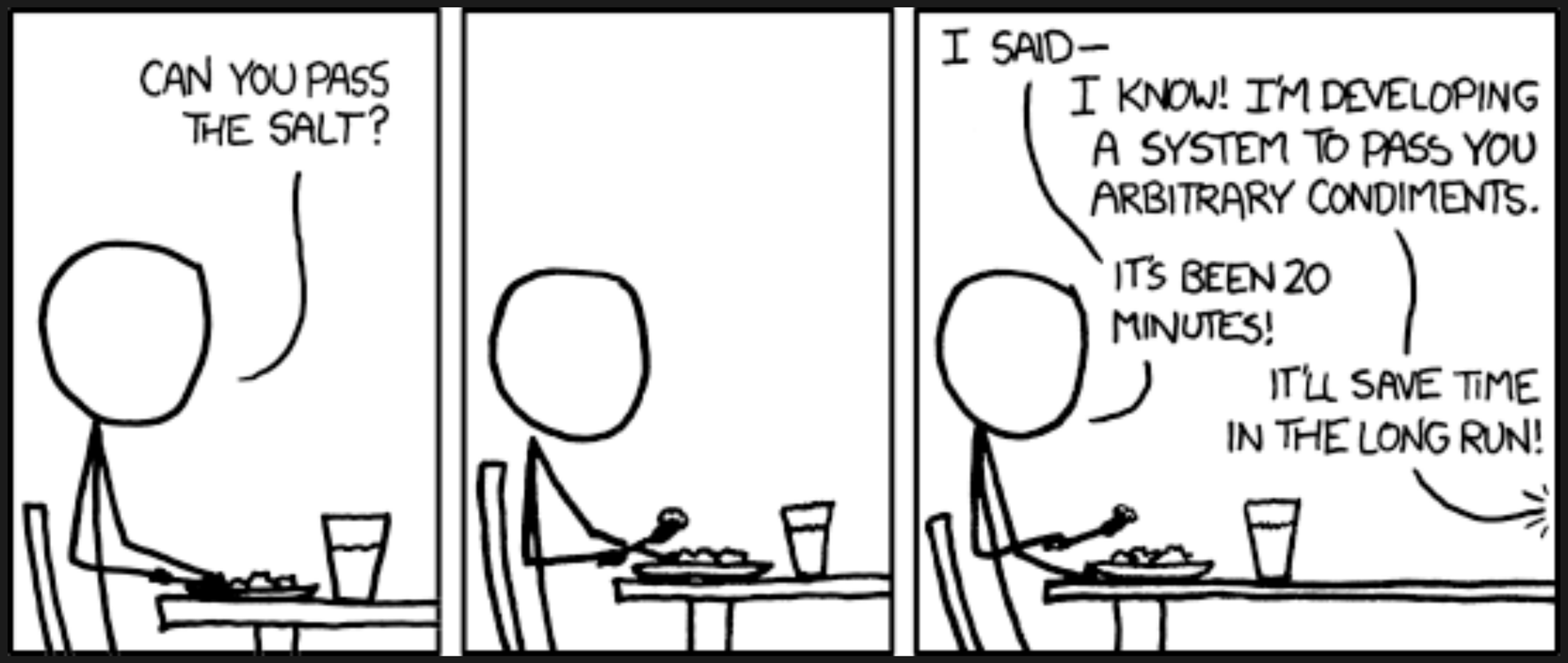
Ground Rules

3. Be pragmatic and continue shipping frequently

- If it’s easier in Objective-C do it in Objective-C
- Extend and change an Objective-C class if that makes more sense

Be Pragmatic

Be careful of the slippery slope of adding in Swift where it will cause more work but “save” time in the future. You might not get that time back.



XKCD – https://xkcd.com/974/

Ground Rules

4. No re-writing code!

So we added Swift
and all went well…

Problems

Problems

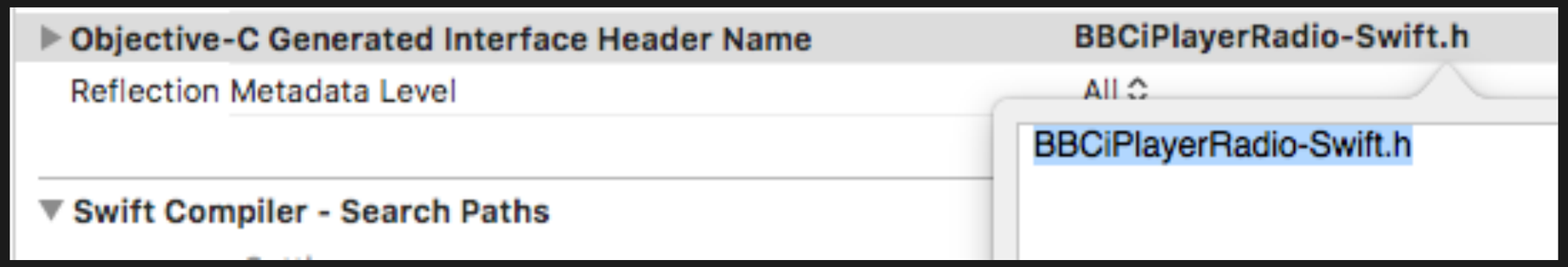
- Two targets is difficult
- Building an IPA that Apple would accept
- Explicitly unwrapped optionals everywhere
- Our Swift wasn’t very Swifty

Two Targets

- We have a UK and international target
- This results in each target having a different Swift “Objective-C Generated Interface Header Name”
- Defaults to “ModuleName-Swift.h” which is different for each target
- Xcode build errors while working in .m files

Two Targets, Attempted Fixes

- Override the Obj-C interface filename so it's the same as both targets

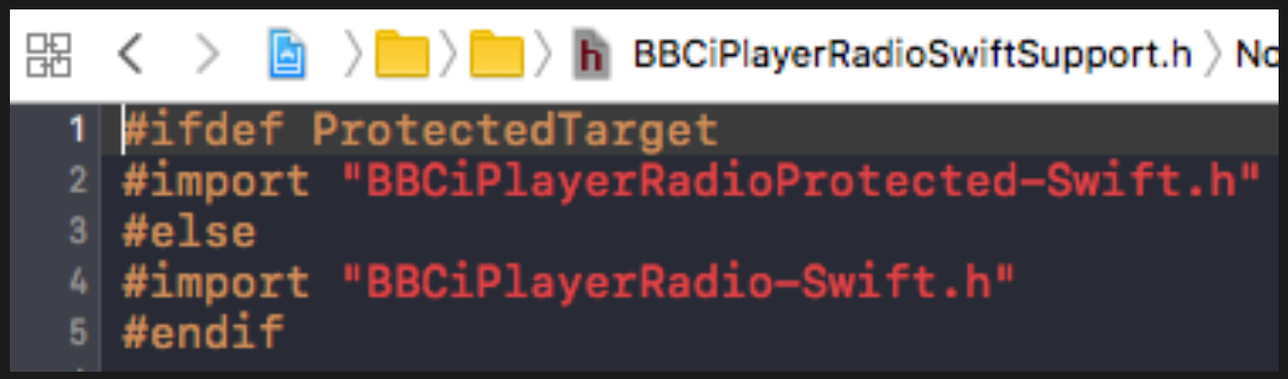


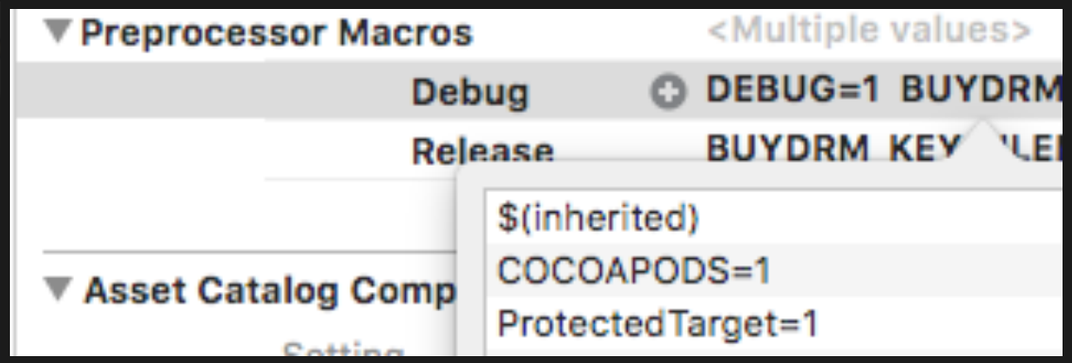
🚫 Xcode gets confused while doing incremental compiles, still shows build errors

🚫 Can resolve by building the other target once in a while but very annoying

Two Targets, Attempted Fixes

- Compiler defines to include correct file based on target



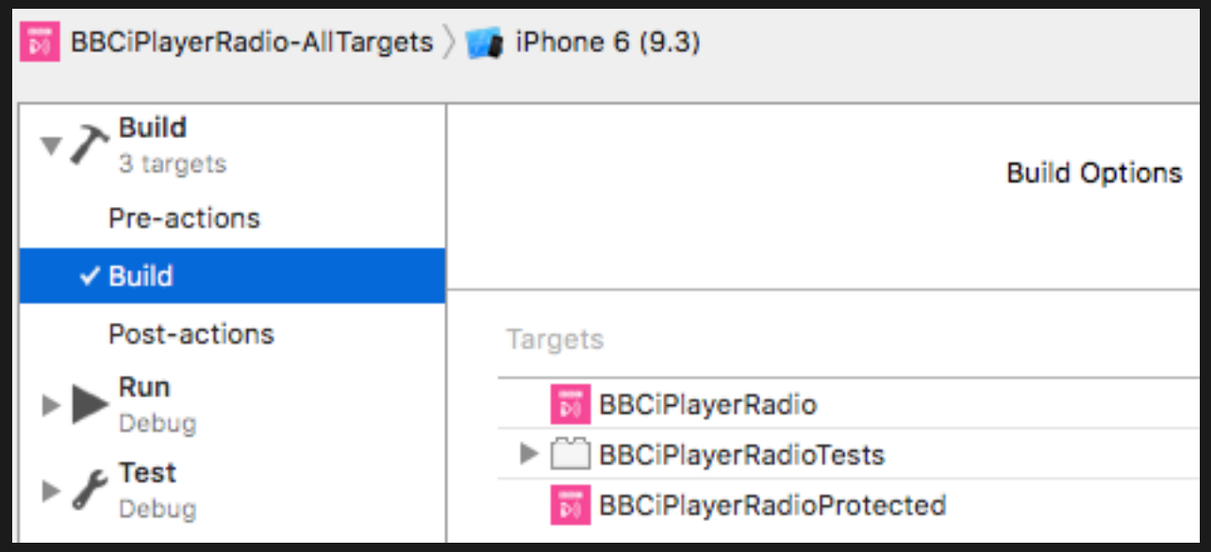


🚫 Xcode still occasionally gets confused while doing incremental compiles, still shows build errors

🚫 Can resolve by building the other target once in a while but very annoying

Two Targets, Fixed

- Compiler defines to include correct file based on target and scheme changes to build both targets



🚫 Takes longer to do a clean build

✅ No build errors while doing incremental builds and working in a .m file

One Target, Better Fix

- Longer term solution is to switch to one target and have UK/International builds differ by build configurations

✅ No build errors while doing incremental builds and working in a .m file

CI/Xcodebuild

- We were using xcodebuild and “xcrun PackageApplication” which had been deprecated for a few years at least
- Can’t build an IPA with Swift support using the old way
- You will get a rejection email from iTunesConnect saying “Invalid Swift Support”

CI/Xcodebuild, Fixed

- New method to archive a build is using xcodebuild with exportOptions
- See xcodebuild -help for possible options

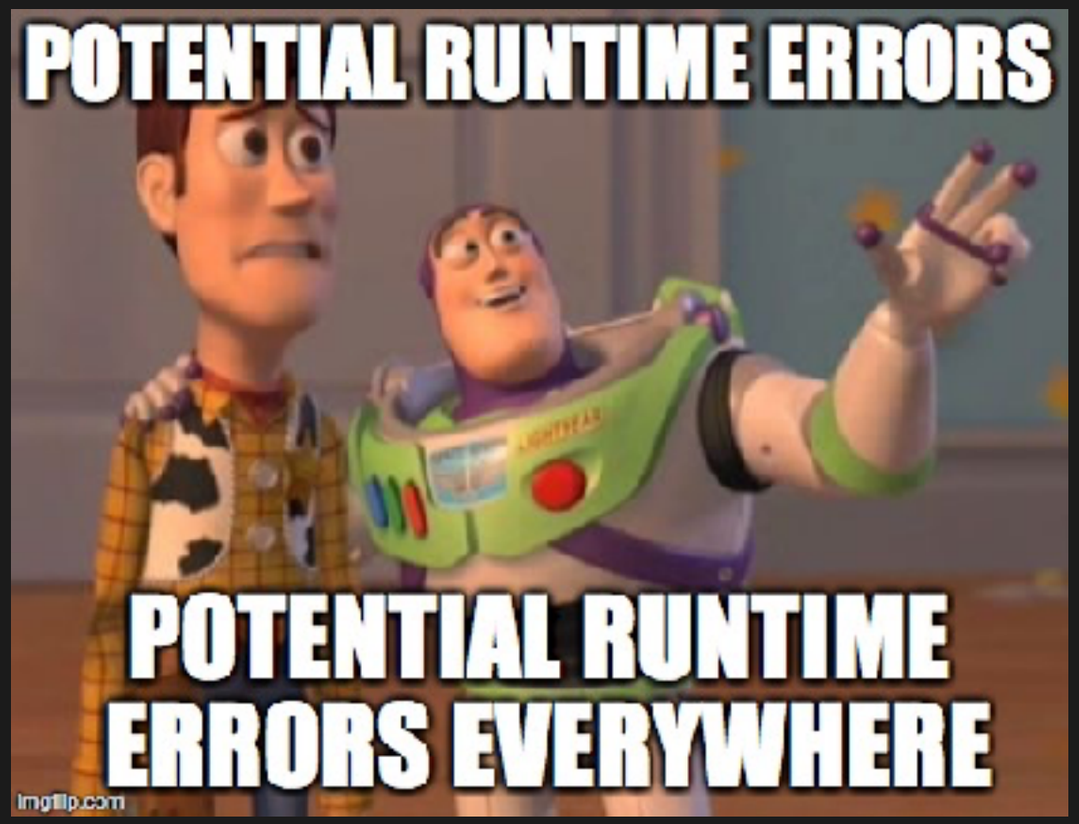
```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>teamID</key>
        <string>YOURTEAMID</string>
        <key>method</key>
        <string>app-store</string>
        <key>uploadSymbols</key>
        <true/>
        <key>uploadBitcode</key>
        <false/>
    </dict>
</plist>
```

CI/Xcodebuild, Fixed

- The IPA produced with Swift support will be much larger than your previous binary
- This will be thinned out when delivered to device
- We saw an increase of 9 MB when installed on the device from the App Store

Explicitly Unwrapped Optionals

- iPlayer Radio did not have any nullability auditing in our headers **(NS_ASSUME_NONNULL_BEGIN and END)**
- Everything in the bridging header could potentially crash our app with **“unexpectedly found nil while unwrapping an Optional value"**



Explicitly Unwrapped Optionals, Fixed

- Any .h file imported in the bridging header needs auditing for nullability
  - Any .h file in that imported header needs auditing for nullability
    - And so on…

Idiomatic Swift

- We were writing Swift, but in an Objective-C style
- Not taking advantages of any Swift language features
- Translating the way we would do something in Obj-C into Swift
- It’s hard to fight years of experience

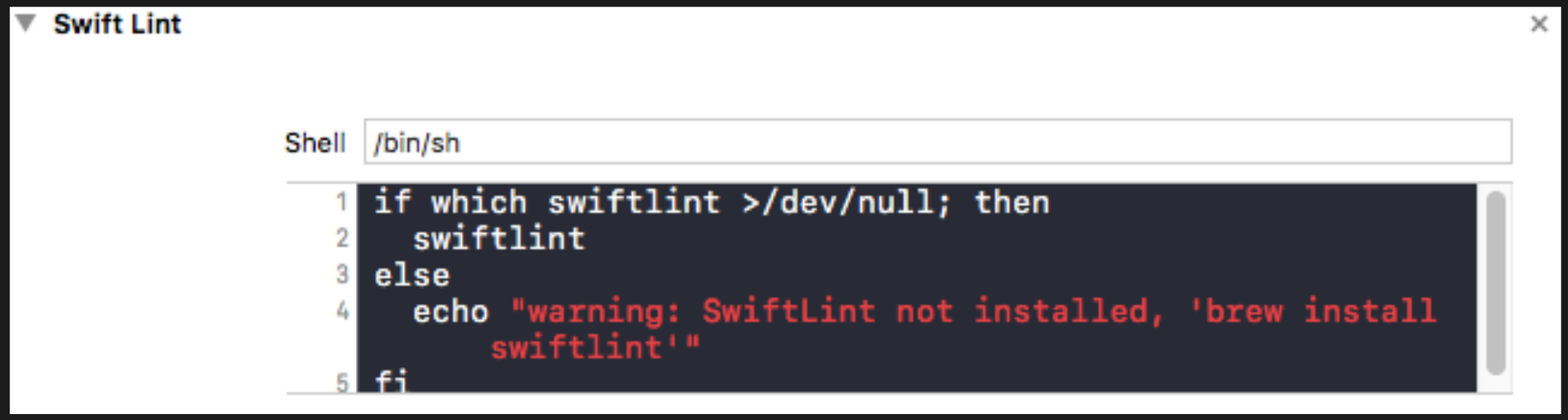
Swift Linter

- Nudges us in the right direction
- Swift is more opinionated than Objective-C, there is a style the community is following
- Start with most rules ignored, add them slowly and fix the linting errors
- Don’t follow it blindly, apply the rules that make sense to your team

Swift Linter, https://github.com/realm/SwiftLint

Swift Linter

- Install: brew install swiftlint
- Run as a build phase script



Swift Linter, https://github.com/realm/SwiftLint

Swift Linter

.swiftlint.yml

```yaml
excluded:
  - Pods
  - BBCiPlayerRadio/MediaButtonStyleKit.swift

disabled_rules:
  - line_length
  - type_body_length
  - file_length
  - trailing_whitespace
  - cyclomatic_complexity

trailing_whitespace:
  ignores_empty_lines: true
```

Swift Linter

- Great talk from JP Simard – https://realm.io/news/slug-jp-simard-swiftlint/

Books

- objc.io books have been a good use as a discussion
- We're currently working through the Advanced Swift book

Current Situation

Current Situation

- We experienced the pain, we worked past it, and we are stronger for it
- Context switching isn’t a problem (knowledge of the frameworks is what matters)
- We walked before we tried to run
- It was time to revisit the rules

Ground Rules Revisited

1. Start at the leaves of the dependancy graph
2. Write Swift code to interact with Objective-C
3. Be pragmatic and continue shipping frequently
4. No re-writing code!

Ground Rules Revisited

1. Write Swift code to interact with Objective-C
2. Be pragmatic and continue shipping frequently
3. No re-writing code!

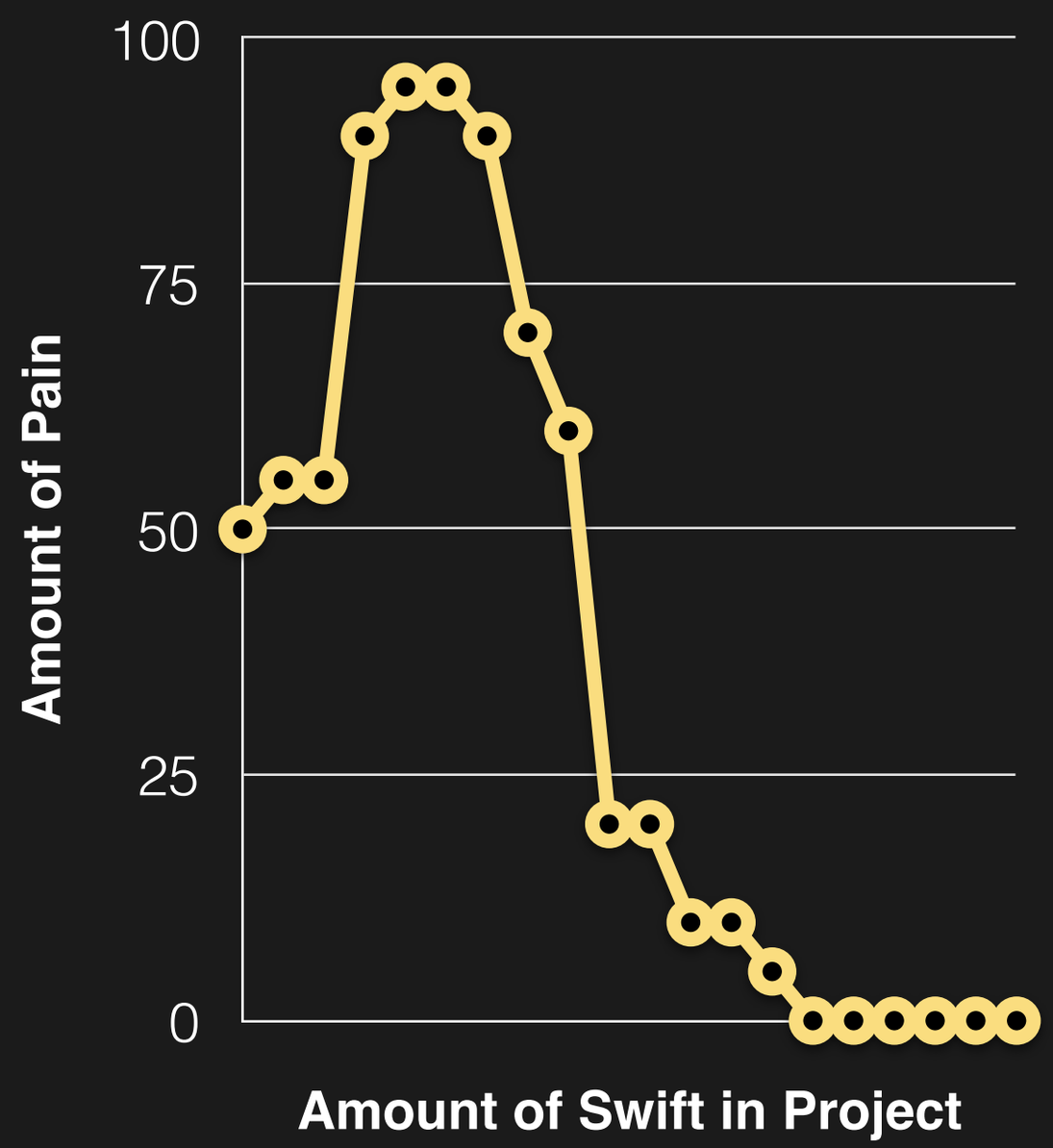
Ground Rules Revisited

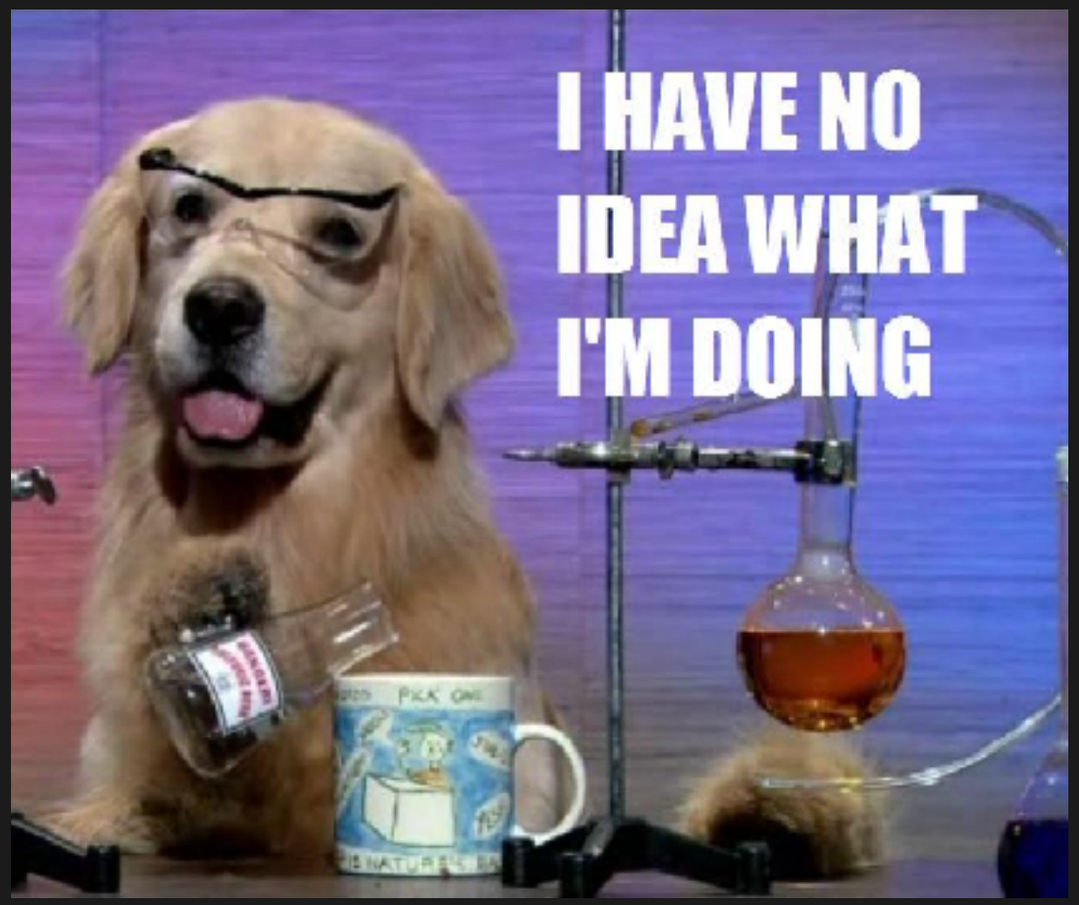
1. Write Swift code to interact with Objective-C, unless it's isolated
2. Be pragmatic and continue shipping frequently
3. No re-writing code! Unless you would have re-written it in Objective-C anyways.

Current Situation

- Devs are happier and as productive
- We are on our way to writing safer better quality code
- We want to use new Swift features and this will happen naturally as the amount of Swift increases in the project

Final Advice





* Based on no scientific data whatsoever

Final Advice

If you don’t know where to start, just start. But learn to walk before you run 👍

Questions?

References/Links

- **Firefox/Mozilla Swift 2 -> 3 migration pain**, https://mozilla-mobile.github.io/ios/firefox/swift/core/2017/02/22/migrating-to-swift-3.0.html
- **Bridging Swift Types to Objective-C**, http://blog.benjamin-encz.de/post/bridging-swift-types-to-objective-c/
- **Swift Linter**, https://github.com/realm/SwiftLint
- **objc.io** Books, https://www.objc.io/books/