Fad-Free Architecture: Sticking to the Basics for Success

Justin Munger
CocoaHeads
July 12, 2016

Introduction

- Professional software developer for 18 years
- Independent contractor/consultant for 11 years
- Mobile development for 7 years
- Started in iOS in 2009, Android in 2010
- Shipped/fixed over 20 iOS/Android apps

Disclaimer

- This is one developer's opinion based on his industry experience
- It's going to be biased
- There is no perfect architecture out there
- This is just one possibility that is working for me
- Sharing what I've learned

A Detour through Contractorland

My Experience

- As a contractor, I've worked with a wide range of companies on a wide range of apps
- Have worked with companies ranging in size from small startups to multinational corporations
- Each company is like its own country with its own unique culture and customs
- I get to see a LOT of different ways companies operate, develop, and ship (or not ship) mobile apps

Why Do Companies Use Contractors?

- Unable/unwilling to invest in software development as a core component of their business
- Unable/unwilling to find/acquire/invest in talent
- Broken app/teams in need of rescue
- Companies that operate with a "Units of Work" environment

Units of Work Environment

The "Units of Work" Environment

- Software development is considered a means to an end rather than a core business function
- Management treats software developers like unskilled labor, line items in a budget, where developers are interchangeable/ replaceable units of work at an internal or external hourly rate
- Developers are put into play on teams, switched back and forth between teams, or removed from teams as budgets shift and time constraints change
- Needs specific to the thought-intensive nature of software development are disregarded in favor of spatial efficiency

GARMENTS OR GITHUB
IT'S STILL A SWEATSHOP

What Problems Does a "Units of Work" Environment Cause?

- Development processes and management are lacking or non-existent
- Individual units of work drive development processes
- Parkinson's Law of Triviality (bike-shedding) starts to rule development decisions with more of a focus on trivial but exciting topics that avoid dealing with the mundane but critical aspects of delivering successful software
- This can lead to avoiding vanilla first-party SDKs in favor of trendy techniques and third-party libraries that are perceived as more interesting
- The trendy techniques and third-party libraries used may not be part of an architecture that is compatible with delivering sustainable working software
- Software developed in this manner may need to be scrapped, heavily reworked, or will have a limited lifespan after its initial release

So What's Wrong with Trendy Techniques and Third-Party Libraries?

- Nothing at all, they are merely techniques and tools to solve problems
- Potentially everything if they don't solve the problem well (or correctly), work against first-party frameworks/architectures, or are poorly-or-not-at-all documented/supported
- Third-party libraries become YOUR code that YOU must now own and maintain when it (inevitably) breaks
- If the third-party library is no longer maintained by the original source and breaks, guess who becomes the new maintainer?

What's So Special about First-Party Components?

- For iOS development, they are made by Apple, who makes the software and hardware that runs the platform
- They are made by people much smarter than most of us
- They are optimized to work best with other Apple frameworks
- They (usually) come very-well tested to ensure they work properly (more so for more established frameworks than those that are brand-new)
- They come with complete documentation
- They live and are supported for as long as the platform is alive (unless specifically deprecated)
- They have a VERY large community of support behind them

So What Does This All Have To Do with Fad-Free Architecture?

- I'm often brought in to help clients who have a "Units of Work" environment to get their applications working and under control with a stable architecture
- The trendy techniques and third-party libraries used solved an immediate problem (in a fun and interesting way) in one part of the app, but failed to integrate well with other parts of the app
- Many times, by the time I'm brought in, the only way to get the app functioning is to gut the entire application, removing trendy implementations, stripping out third-party components, and re-architecting the app piece-by-piece using first-party components and tried-and-tested techniques

Fad-Free Architecture

Inspiration for Fad-Free Architecture

- Marcus Zarra's MVC-N presentation at CocoaConf Chicago 2016
- Gives a good basis to start with understanding how to create a solid app architecture based on MVC and first-party components from Apple
- Handles happy-path scenarios
- Needed some more definition to be usable for a foundation for an actual application
- This talk will try to expand upon MVC-N to give a more usable blueprint for application architecture
- Comes with a working implementation that you can clone from Github and try yourself (https://github.com/justinkmunger/FadFreeArchitecture)

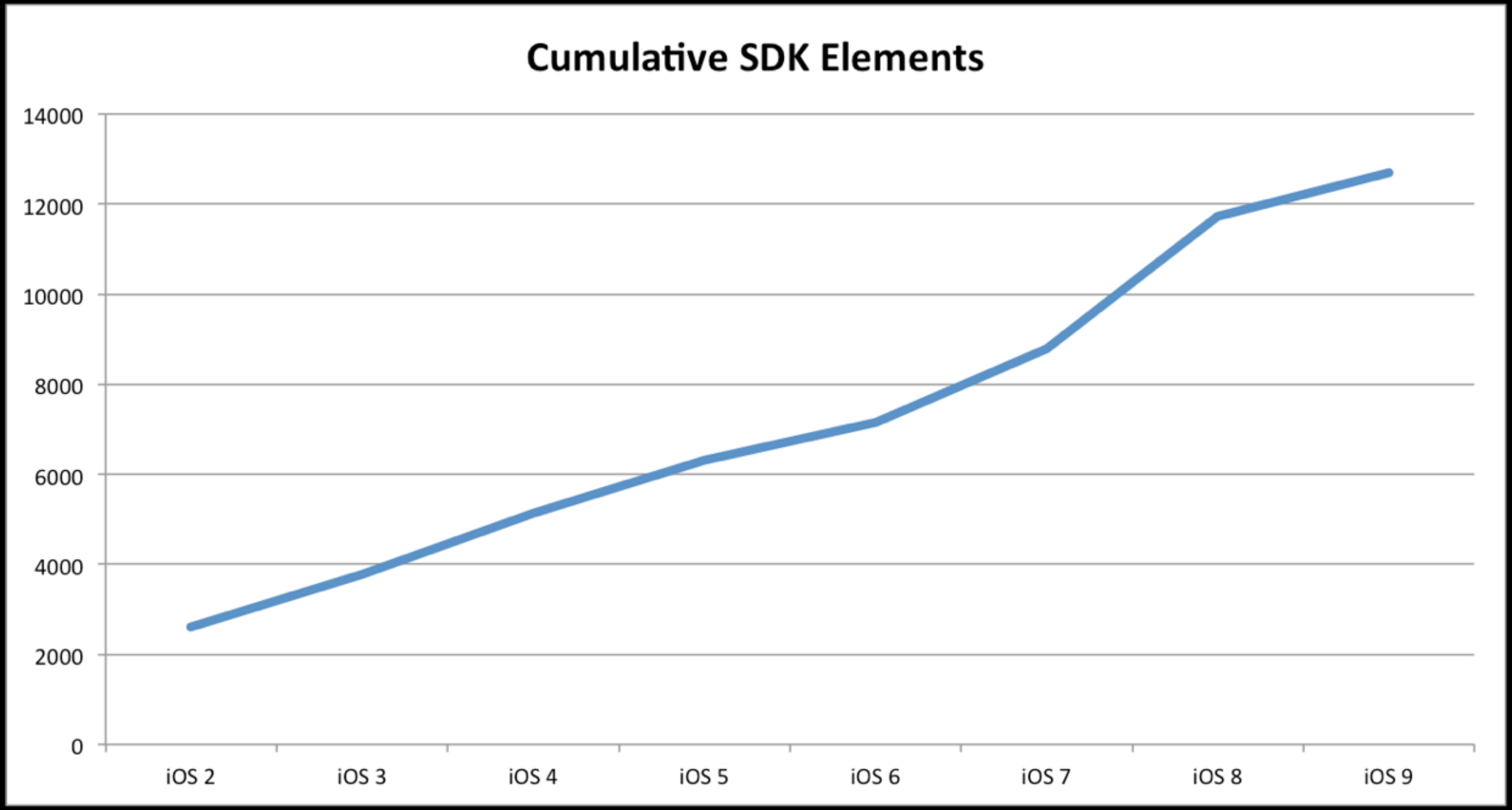
Fad-Free Architecture

- Keep UI and non-UI parts of the app separate as much as possible
- Contain non-UI parts of the app in a single-instance object, created by AppDelegate and passed between view controllers through standard segues and dependency injection (in prepareForSegue)
- "MVC is central to a good design for a Cocoa application" - Apple
    - https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html
- Rely on first-party (Apple) solutions as much as possible
- Only use third-party libraries if you're willing to learn how their code works and be able to support them yourself when they break

Benefits of a Fad-Free Architecture

- Separation of UI and non-UI concerns
- Helps enforce UI/non-UI code boundaries, allowing for potential separation of non-UI code into its own reusable framework
- Easier to unit-test
- Developers can focus their energy on solving business problems with mature first-party frameworks
- Easier to staff projects when there's a larger pool of developers that understand your code base

Increasing Complexity of iOS SDK



https://david-smith.org/blog/2015/09/03/the-growing-ios-sdk/

Every year, it's becoming increasingly difficult for a single developer to successfully master the entire iOS platform

Consequences of a Fad-Free Architecture

- "Front-end" and "Back-end" iOS development
- Allows team members to specialize and focus on specific parts of the application
- Platform is maturing to where applications can be more effectively developed if specialization occurs

When to Use a Fad-Free Architecture

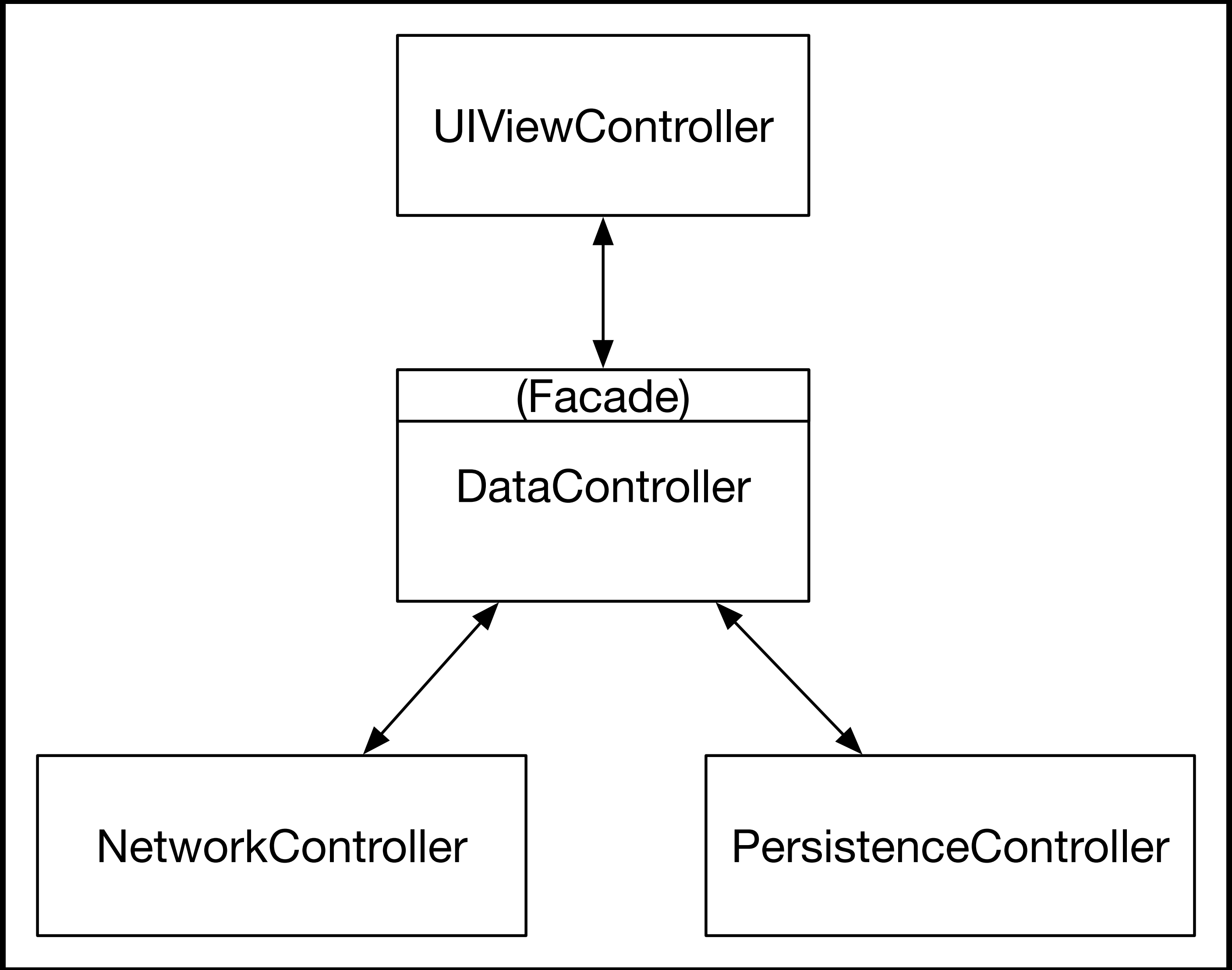
- When you're working on a project with inexperienced team members
- When you need to make the application easy to understand for other team members
- When you want to leverage tools designed specifically for the platform
- When the application is code is going to change hands to other developers or another client

When Not to Use a Fad-Free Architecture

- When all team members are guaranteed to be mid- or senior-level who are staying with a company long-term and will continue to be working on the project throughout its entire lifespan
- When the company retains ownership of the code and it does not exchange hands for someone else to take over or maintain
- When experimenting with new technologies in applications that are for internal use and/or are experimental

High-Level Overview

What Does a Fad-Free Architecture Look Like



Fad-Free Architecture
Key Components

UIViewController

- Only responsible for displaying UI and receiving input from the user
- Receives an instance of DataController when it is instantiated
- First visible view controller kicks off persistence initialization in DataController
- Once initialization is complete, it can start using DataController methods
- Accesses the DataController for anything non-UI related

DataController

- Access point into the non-UI portion of the application
- Instantiated in AppDelegate as a single instance (not a singleton)
- Passed through to all UIViewControllers that need access to it using dependency injection
- Holds references to and coordinates activity between all other controllers for the application
- Vends configured components to the UI layer
- Provides a "facade" to allow access into it

DataController Facade

- Implemented as extensions on DataController with functions and computed variables
- Functions take raw parameters and completion closure used to indicate success/failure in UI
- Functions create instances of NSOperations from other controllers and coordinate their execution through dependencies
- Functions implement NSOperation completion closures which use the closures passed in to the functions to call back to the UI code with success/failure

NetworkController

- Handles all access to network resources
- Implements NSURLSession with a delegate callback interface
- Wraps NSURLSessionTasks in NSOperations that are handled by NSOperationQueue
- Returns JSON data to DataController facade function (through NSOperation completion callback) for processing

PersistenceController

- Handles all persistence in the application
- Relies heavily on Core Data and associated classes to implement persistence
- Contains Core Data initialization and managed object context save logic

Walkthrough

Any Questions?

Thanks!

- Slides and code are at: https://github.com/justinkmunger/FadFreeArchitecture
- Email: justinkmunger@gmail.com
- Twitter: @justinmunger