How to Properly Structure a Mobile App Development Project to Avoid Unnecessary Technical Debt

By Brian Westendorf, Principal Consultant

Digital Experience and Mobile

Introduction

Apps are now doing things only dreamed of a few years ago and we're continuing to move from a Mobile First to Mobile Only mindset. But while many aspects of the market are maturing, sloppy practices can erode the foundation of mobile projects, create sustainment nightmares and even cause projects to fail completely. In software development, the accumulation of errors in architecture, design, and poorly written code are referred to as technical debt.

Technical debt is a serious issue that is impossible to avoid, but some organizations add to it unnecessarily. Errors made innocently enough at the beginning of a project can snowball into an avalanche. For example, AIM was once brought in by a new client to analyze why their enterprise-level iOS mobile app, which had been in development for over a year, had stalled and failed Apple submissions. We discovered that the code, which the company had developed in-house, did not follow any discernable design patterns, was all original, and was therefore unreliable and difficult to debug. The resulting technical debt was so high that the client would actually be served best in the long run by scrapping the project and starting over, ultimately costing the company millions of dollars.

This situation is more common than you might think. Too often companies plunge into mobile development without properly setting up the project and end up creating something that is very costly to deploy and maintain. Avoiding these mistakes and the technical debt associated with them becomes more manageable if you follow some basic best practices while laying the foundation for your mobile projects, primarily:

- Adhering to Standard Design Patterns
- Embracing Open Source Code
- Understanding the Project APIs

Adhering to Standard Design Patterns

Design patterns are simply common ways for engineers to create reusable solutions. In many cases, engineers may be writing designs patterns without knowing their technical definitions. Early and often it's important for engineers to follow patterns to create a solid project structure and well-organized set of de-coupled solutions. This way, when you're in the middle of development and the project pivots to a place where you may need to re-factor, it's easier to know the role of each solution component.

The following are some examples of popular design patterns (and not just for mobile):

- **MVC design pattern**: Classes or objects in your application will take on one of three roles: Model (representing your application data), View (user interface) or Controller (manages the communication between the Model and View, taking the data from the Model and feeding it to the View for display). Conforming to this pattern helps to ensure that your code is easier to understand. A common mistake is putting everything into the Controller (known as the MassiveViewController in iOS). Properly breaking it up makes the code much easier to manage.
- **Observer design pattern**: The Observer pattern handles and manages all communications between its dependents. Think of a parent communicating to children either individually or all at once. Within iOS this can be done very broadly with NSNotifications or more granularly with Key Value Observing (KVO).
- **Façade design pattern**: Commonly used in object-oriented programming, this design pattern provides a simplified interface to a larger, more complex body of code. The façade acts as a gatekeeper of sorts, communicating only with specific components of an application at specific times, enabling efficiency in the app. It is often used in very complex or difficult to understand systems.
- **Singleton design pattern:** At its core, Singletons are a very simple design pattern, but one that I use often. They create one instance of a class or object and give shared access throughout the app. For example, you can imagine an app having one User and you can set this up using the singleton pattern. Once this is established in your code, you can access the one instance of User throughout your app.

When considering the possibility of *not* using one of these design patterns, it's easy to see how the technical debt in your development projects can accumulate quickly! When things in your project pivot and re-factoring discussions begin, having these design patterns in place will help.

Embracing Open Source Code

Another major way for organizations to avoid massive technical debt is to leverage reusable solutions from open source libraries. Open source communities have done a terrific job creating, contributing and maintaining reusable frameworks that organizations of all sizes use time and time again. When developing solutions, it used to be a build vs buy discussion. Now there's typically a public GitHub or other repository where someone has already figured out the problem you are trying to solve and provided a solution.

The important factor is knowing which to use and which not to use. When reviewing open-source code, look at how often it's been updated — if the code is fresh and has been used and updated by many contributors, it's likely solid. However, if it was written two years ago and has never been updated, it might be too old and you'd be better off avoiding it. You can pull open-source code into your apps and alter it in whatever ways you want or need, then redeposit it to the open-source community. This makes it a better product for the community.

While it's typical for enterprises to make use of reusable solutions, some organizations desire to write it themselves. They often do this for the purpose of owning the copyright to the solution or product. Creating all of your own code from scratch carries the risks inherent in adding more code, complexity, and bugs to the app, increasing the potential technical debt for the project. However, depending on the project, this might be the only way to go.

The only other consideration of note when leveraging open source reusable code is attribution. Within the About section, most apps normally include a license view. Contributors only ask that you give them credit for using their open source solutions in this section. Oftentimes the attribution is referenced by the MIT license, which is a free software license originating at the Massachusetts Institute of Technology (MIT). This license permits use of reusable code in a proprietary software application provided all copies of the software reference the MIT license.

Understanding the Project APIs

In simplest terms, APIs allows programs to communicate with each other. The most common API in use today is REST (Representational State Transfer, a term often used interchangeably with RESTful), which governs how applications communicate over the web via web services. RESTful APIs enable this communication with a very light amount of data exchange compared with previously popular protocols, a tremendous benefit for the development and use of mobile applications.

The important thing to note about APIs is making sure that they are accessible, capable of supporting the features and overall project before the beginning. In most cases, current APIs don't fully support the needs of project and updates are done in parallel (or not at all). It's important to be collaborative and flexible while always working towards a solution.

The Lasting Benefits of Avoiding Technical Debt

By establishing solid practices and patterns at the start of your mobile development projects, you build a solid foundation for long-term success. When AIM works with clients on mobile projects, we start with strategy and approach. Part of this effort involves setting up the architecture and establishing the design patterns. We'll almost always pull from third-party frameworks (open source code) when viable to save time and resources and we always talk to clients about the state of their APIs. Approaching projects strategically helps reduce the chances of having to devote valuable resources to fix the potholes, if they're even able to be fixed in the first place. Ultimately, your development resources will be used far more efficiently, devoted to enhancing already solid applications, and your technical debt, if any, will be small.

About the Author:

Brian Westendorf has over 15 years of experience in software engineering with an emphasis on mobile applications, enterprise e-commerce solutions, and content distribution systems. His experience include app development for big box retailers and commercial enterprises, where he has integrated cutting-edge technology into applications including check-ins with iBeacons, loyalty with Passbook, and enhanced customer engagement with geo-fencing for in-store experiences. He lives in Seattle, WA.

About AIM Consulting

AIM Consulting is a rapidly growing, nationally recognized leader in technology solutions and services. We have the people, processes, and tools to provide companies with strategic guidance on business-critical initiatives and deliver end-to-end solutions. We meet the highest standard of excellence in technology, for better value than other consulting companies, because we are 100% focused on forging long-term relationships with deeply experienced consultants and building high-performance, service-oriented teams that produce results.

Ready for a Solution?

- Digital Experience and Mobile
- Application Development
- Delivery Leadership
- Business Systems and Strategy
- IT Infrastructure, Cloud, and ITSM
- Data and Analytics

Tell Us About Your Challenge

Offices

- Denver
- Minneapolis
- Seattle