Architecture: Why Does it Matter?

Whitepaper

Contents

What kind of animal is app architecture?

Let’s say I’m an ordinary user with no development background. I pull out my phone and tap an app icon. What do I see? Well, there’s a program… with a nice design and cute buttons. I engage the app and voila! I immediately get a result: an answer from a calculator, an item added to my cart in an online store, or a sent message to a friend. I just tapped a few buttons and the result was mine. So simple. How does it work? Who knows — I’m just an ordinary user. Maybe it’s magic!

But what if I’m a programmer? I see an app icon. I imagine code, lots of code… an ocean of data that triggers the program when I tap the icon. This is the way it works. Magical, maybe, but purely in a technological sense.

When talking about an “ocean of data”, the more you structure the data into modules, the easier it is to determine the project details, locate bugs, and prevent future problems. An **app architecture** is a high-level structuring of code (sometimes involving miles and miles of code) into modules and estimating the relationships between them. In this fashion, I as a programmer convert seemingly cryptic data into a structured, coherent system. Abracadabra!!!

Some code lines look very similar, while being responsible for completely different features. To avoid confusion, programmers normally break the code into modules: they put the commands related to the interface in one module, and commands related to the business logic and calculations in another module. This is a convenient system — for example, when you want to change the computation procedure that calculates the cost of a mortgage in a finance app, you access the “business logic” module and leave the interface module untouched. If you need to change the coefficient, you enter the business logic module, then the submodule responsible for the calculation, and then a submodule of the algorithm. You open that file and change the coefficient.

On the flipside, trying to work with unstructured code is like looking for a needle in a haystack. For example, let’s assume the same coefficient is used in several code lines for a finance app, and that the original programmer messed up and entered the coefficient as a **number** but not a **parameter** in seven modules. Then that programmer leaves and a new one comes in. The new programmer sees that the coefficient in one code line is incorrect (and fixes it), but because of poor app architecture, he can’t find the other six mistakes. The result is that the client receives a flawed app, one that caused its managers to provide dozens of dead loans. Oh, and then the company went bankrupt. This is the price you pay when you don't take the time to create a well-designed app architecture that eliminates such unpleasant consequences.

High- and low-quality architecture

A new app architecture is like a building's blueprint in which the walls, doorways, utilities, and countless other details are shown. It's common sense that the blueprint is discussed and created **before** construction starts — and it's also common sense that such an important plan, which helps the entire construction team, must be created by an **experienced professional.** The only exception is when an app is so basic that the developer can conceive the entire blueprint in his head, or use existing layouts that fit the project (for example, a CMS like WordPress will often meet the requirements of a text-oriented website).

However, a complicated app requires much more time and thought. A programmer must consider which modules to create, the ideal way to group them (so that if a bug appears he only needs to change one module and not the entire project code), and what the inter-module relationships should be. Now, you might say, "Hey Sibers, there are plenty of ready-made layouts floating around. What's the point of making things so complicated?" Our response is, you're right — there are. But most of these are inapplicable for custom/complex projects, meaning the developer would have to **modify** the ready-made layout for every single project he designs.

An easy analogy can be made between **app architecture** and **home architecture**. Good home architecture means a door will open without coming off its hinges, and that the ceiling won't collapse in a strong breeze. Likewise, good app architecture prevents bugs from compromising performance, and has safeguards to prevent the app from crashing. Also, an app, just like the building, doesn't exist in a vacuum. Performance is affected by external factors: traffic volume, third-party services (i.e. payment services), and things like clearing the local cache. A good programmer will consider all of these variables. But when the app architecture is weak, surprises may occur, making it risky to open such an app, never mind actually purchasing it.

Unfortunately, the gaps lurking in low-quality architecture are not always readily apparent. When the app is small, it doesn't matter as much if the architecture is well-conceived or not, because the developer has the entire code in his head and can easily maintain it. But if the app grows bigger, or a new programmer is assigned

Good architecture!

Not so much...

to it (yes, even we hearty Siberians get sick once in a while, or drop everything for a two-week vacation to some sun-drenched island), “having the entire code in one’s head” becomes impractical: some processes are not readily visible, and if the project consists of jumbled parts of code that aren’t easily changeable, then one small alteration could cause the entire app to collapse. The lesson is, the **better** your app architecture, the **easier** it is to quickly find and remedy any problems.

GOOD (BUT INAPPROPRIATE) ARCHITECTURE

Sometimes architecture can be good, yet unsuitable for a specific project. This can happen even if the app was developed by pros who followed a carefully thought-out plan. Instances in which such a problem can occur include a client’s business focus changing, or a sudden shift in the development process’s priorities. In these cases the app architecture may struggle to meet the client’s new requirements. To draw an analogy between app architecture and home architecture again, it would be like trying to add a second story to a ranch house in a single day, or putting an Olympic-sized swimming pool on the roof without any structural support in place…

Here’s a real-world example:

A client with a small budget asks a company to develop an app. It’s a simplified clone of Twitter that only works when a user’s device is connected to the internet. The developer creates a simple, inexpensive architecture that excludes the possibility of adding features in the future. A few weeks later, the client comes back to the developer. Now he wants users to be able to “like” a tweet even when they’re offline, and he wants the *app to automatically send the "like" to the server when the user's internet connection is on again.*

Seems like a slam dunk, right? Just save information (in this case, a simple "like") on the user's device, and when the internet connection is on again simply synchronize the data. It should take only a few man-hours at most!

That's how the client might view the situation. However, every time a developer hears such statements, his eyes twitch and he begins speaking in tongues. He knows right away that "adding this one small feature" will crash the app. Why? Because the current architecture was **not designed** to accept future change. In order for the app to function correctly with this new feature, the developer must code several new pieces: one to save a "feed duplicate" locally for reading and liking posts offline, one to synchronize the statuses of local and real feeds, one to track the internet connection, and yet another piece to coordinate all of these processes. Ultimately, the app architecture **must be rebuilt** in order to add these new code lines, an undertaking that involves far more than one day of development time. No doubt the developer is left thinking, "If only I was approved to spend a few more hours on the architecture so that it accepted future changes, this entire project would've been much cheaper."

How to spot bad or inappropriate architecture

Here are two tips that a "non-programmer" can use to recognize a problem with his app's architecture:

#1. ADDING FEATURES BECOMES INCREASINGLY TIME-CONSUMING

In other words, when a new feature (even one with the same complexity as previously-added feature) takes more man-hours than the previous one.

#2. ONE BUG-FIX SPAWNS SEVERAL NEW BUGS

For example, the client asks the developer to fix a small problem in his existing project code. The developer fixes it, but three new bugs arise. He fixes these and three more arise, and so on. This is akin to a dog chasing its tail: the developer goes 'round in circles without ever completely fixing the problem.

What happens if the app architecture is flawed?

If the current app architecture is flawed, there are two ways out — depending on the extent of the damage: you can demolish the "rickety house" and build a new one from scratch, or undertake comprehensive repairs.

A FRESH START

If the house is no longer occupiable: its walls are collapsing, water is leaking through the ceiling, and its roof is collapsing, then renovating the home will cost more than building a new one. It's the same way with apps.

The reasons for a fresh start vary: low-quality development from the start; the app grows too big — there are too many features already, meaning an extra one might cost too much or compromise the app's performance; or, the client's local and/or global business priorities have changed. Whatever the reason, the bottom line is that the current architecture has become **untenable**. It's as if you built a charming three-bedroom house for you and your spouse, and then a couple of years later had three kids and suddenly required a garage, a playset in the backyard, and two new bedrooms. In sum, things have changed considerably and it's time for a new home — but this time, you'll design one with the flexibility for changes.

APP REFACTORING

If the home's foundation is ok but you'd like to remedy its squeaky floor, replace its windows for better insulation, and repaint its exterior, then repairs will suffice. App developers call such repairs **"app refactoring"**. This means that the app will be refactored (repaired) not all at once, but piece by piece. First the team makes several small changes and integrates them into the entire app structure. Then the same process is repeated for other problems.

What are the benefits of investing in high-quality architecture from the start?

Fact: high-quality app architecture costs more than mid- or low-quality architecture. As with anything else, you get what you pay for. However, similar to playing the stock market, it isn't always clear to the client when his return will surpass his investment. So when is the "gamble" on high quality app architecture worth taking? Let's summarize.

PROJECT SIZE MATTERS

If an app consists of only two screens, has no specific business logic, and future additions are not anticipated, then expensive, well-crafted architecture may not be worth the investment. In this case, the client would spend a huge amount of time and money for implementation, but never utilize 100% of the app's capabilities. It's like buying a Porsche but only driving it three miles to and from work.

Now, if the project **is more complicated** (i.e. it includes a database, integration with outside services, and numerous complex machinations), then paying for accurate, complex architecture from the start is a smart move. In this scenario, the budget would include money for future additions as well as changes of business philosophy. Thus, a higher investment at the project's beginning offers terrific value and will **save** the **client money** in the long run.

How does the client save money? Simple: the more accurate the app architecture, the fewer man-hours are needed for future changes/additions (in fact, any changes made will not compromise the code — on the contrary, they'll make it even better). Also, a big project with solid architecture will not lead to unpleasant surprises, i.e. a small addition/change suddenly becoming a bug-filled, multi-day disaster cleanup.

WHAT IF THE CLIENT'S BUDGET IS TOO SMALL FOR "GOOD ARCHITECTURE"?

If the project budget is tight, it would be irrational to spend a whole week on small architecture details. Instead, the top-tier development team, together with the client, will determine how to manage priorities and find the optimal solution. It's also important to understand that the majority of the development teams with low rates often design apps that are technically functioning, but lack good architecture and cannot be reconfigured in the future. As we mentioned before, you get what you pay for, and so your expectations regarding quality (of the app itself and the development team) should adjust accordingly.

THE MVP CONCEPT

Further to the above, some clients are ok with the development team designing the simplest, most cost-effective application. This is known in the development industry as MVP **(minimum viable product),** with the goal being to create an app that looks nice and demonstrates **key features.** In this scenario, the client saves money by development a simple, ground-floor application. However, be aware that this app has no future. If one year later the client receives a large seed investment and decides to discard the current app and produce a brand new one, this new version must be created **from scratch**, which will cost much more than the original, inexpensive version. You pay less for the present, but the future may be very expensive.

Sibers recommends this approach for projects with considerable potential but for which it's unclear whether they'll score the "big win" the client is hoping for. This approach is also useful for projects that will be shown to potential investors/aboard of directors. For example, a client asks us to develop a clone of Facebook. It's obvious that this clone could potentially cost a gigantic amount of money. A smart developer will start simple, and if the project makes a big splash and investors jump on board by the dozen, we'll rewrite the app from scratch in the proper manner. It bears repeating that if you still want a premium of an inexpensive trial app, it won't be usable for future, more complex versions.

Conclusion

High-quality architecture is just as important for developing an app as it is for building a skyscraper. App architecture affects everything: development process, user experience, and business metrics. Our advice when discussing app architecture is to be receptive to your development team's suggestions. Keeping an open mind provides reassurance that your app will never collapse, and that its users will always enjoy a warm, satisfying experience.