Functionality vs. Classy: The “90/90 Percent” App Rule

Whitepaper

Sibers

"The first 90% of the code accounts for the first 90% of the development time.
The remaining 10% of the code accounts for the other 90% of the development time."

Tom Cargill, Bell Labs

What does this 90/90 percent rule mean when discussing app development? Can we somehow make sense of this strange arithmetic?

Main flow

DETECTIVE GAME EXAMPLE

Let's imagine you're contemplating the plot for the first, very short level of your new detective game. You start a conversation with a developer and explain the premise: the main character must reach a parking lot, where the criminal is hiding in his old, rusty minivan. He arrests the criminal, and shazam! — the player's first mission is complete.

You describe to the developer the story's main flow: the player starts the game on an empty street. He walks to the end of it, turns left, and runs straight ahead, following a set of footprints. A boy running in the opposite direction runs into him. There's an "Oomph!" sound and then the player's pistol and handcuffs fall to the ground. He picks them up and sees the parking lot. He goes to the only car parked there, sees the criminal, pulls out his gun and slaps a pair of handcuffs on him.

This imaginary developer, without any additional questions, codes first level's main flow just as you described. The first level is published, and if a user behaves exactly you expect him to, everything will go smoothly.

But let's imagine that the user plays without headphones. Or the sound on his device is switched off. One way or another, he never heard the gun and handcuffs fall to the ground and proceeded without them. Can he arrest the criminal another way, i.e. by fighting? And if he can't should we display a notification summarizing why he failed the mission and what he needs to do next time? Or if he eventually noticed his missing gear before approaching the criminal, could he turn around and retrieve them?

If you, a novice storyteller, haven't foreseen every possibility, and your novice developer hasn't either (in order to save his time and your money), he might not have

implemented the possibility of returning to a spot the player passed earlier. As such, when the user tries to go back, the game will crash. Or if he keeps playing without headphones, he won't understand what the problem is and will give up on your wonderful game before even finishing the very first level.

WHAT DOES THE MAIN FLOW INCLUDE?

Further to the game analogy, the main flow of events in any app is the functionality it provides and the tasks a user can solve while using it. The main flow can be estimated: for example 200 hours for initial development and another 200 hours to perfect it and make it "retail ready". This is a common situation, and ideal too.

However, the client could still ask his outsourcing provider: "Can you create the app in 200 hours, since my budget is only for this amount of hours? What can we do with 200 hours?"

HireRussians, as a software outsourcing provider, frequently hears this question, and because of its popularity, we've decided an explanation will be useful for potential clients wondering what is included in the first and second 90%.

The first 90% is dedicated to creating a functioning app, i.e. implementing all of the features that make an app an app. Alas, the first 90% doesn't make the app "retail ready", because there are still many details small and large that must be addressed.

Thus, we need to consider these details, which are outside "the main flow", and how to process them.

Incidentally, this is a big reason why it is important to create a project specification document. At the moment the client contacts an outsourcing provider, he probably knows how the app should work, even without a specification doc. What the specification document does is describe how the app should "fail gracefully", which means evaluating how the app should perform when uncommon issues occur.

If we don't spend the necessary time accounting for such situations, here's what could happen: the app is functioning, everything is great, but then something goes wrong (a user makes an unusual action). He sees an "Error" message and doesn't understand what to do next. In some cases, he even won't see a message. The app will just crash (close) or hang up. During the first app development phase (first 90%, or 200 hours), we don't address these unusual cases and error handling. Rather, the first version is

meant to be 1) appealing (to the eye and the mind), 2) suitable for demonstrating to investors, because the client will have it in his own hands, swiping and clicking, and ensuring everything goes smoothly, since he already knows where and how to click.

While the app's author (the client) and the developer know exactly how the app should work, someone seeing the app for the first time will act in a way we haven't foreseen, and an unexpected situation will arise.

So, what are the most common "rare situations" when speaking about app development?

The second 90%: special-cases handling

LACK OF STORAGE PLACE

This case is specific to mobile apps. The situation is simple: the device has run out of storage. Here's a real-life example: we worked on a smartphone app where a user records himself reading a fairy tale. But what if his phone runs out of storage while he's still recording his tale?

The main flow — the "normal scenario" — goes like so: a user clicks "REC", records the tale, his phone has plenty of memory. When finished, he clicks "Stop", then "Save", and his recording is saved. If we hadn't foreseen the lack of storage place issue, the app would 1) stop recording, 2) crash, leading to some not-so-happy endings.

What's the best way to handle the "lack of storage" error? First, we must ensure that the user can record what he wants to, without fear of losing anything. So even before he starts recording, we display a notification that alerts him to an insufficient amount of storage on his device, it could be too low for his task, and that he must delete something before continuing. If the user ignores the notification, we show another pop-up alert that says, "Please free up storage on your device without closing the app". Simultaneously, we must save whatever he's managed to record and give him the possibility of proceeding from the exact point where the "insufficient memory" alert appeared.

SERVER CONNECTION

Let's imagine we are working on a web or mobile app that lets users search for restaurants, read menus, and leave reviews: photos, comments, and ratings.

The following situation is common: a user uploads a picture from his device's gallery, types in a review, makes a rating, and clicks on "Publish" to post the review.

What problems might appear at this step?

- Airplane mode is switched on (speaking of mobile apps specifically)
- No Internet connection (i.e. no money in the account)
- Internet is connected but the server isn't answering
- The server answers, but the review didn't publish

Ideally, each of these four cases is handled separately and a custom notification is shown for each one.

But what does "handled" mean?

1. Foresee these cases at the app's architectural design stage (ideally)
2. Write code that addresses each specific case
3. Writing the notification/alert text that the user will see (if needed)
4. Discuss the text with the client and get approval (if needed)
5. Test the code created for step two
6. Create additional scenarios, i.e. the possibility of sending a support request to the developer.

The ultimate goal of "handling" is to avoid the user losing any data he's already uploaded/inputted. Regarding server connection, there are two main resolutions:

The easier one: after completing a review and sending it to a server, an error notification is shown to the user. The data already added is not lost, and the user can resend it again.

The more complex fix is one implemented by a company like Twitter. If for some reason a user-created message couldn't be sent to the server and published, it can be saved as a draft, and as soon as the error is fixed the message is sent.

For web-based apps, an offline mode is critical too, and it's important to think about it at the app's architectural design stage: give a user the opportunity to use the app when there is no server connection, and easily/transparently synchronize data when the connection is resumed. In this scenario we must foresee a local database, double-ended synchronization, etc.

The problems resulting in poor/no server connection are varied: server settings,

architecture, tech problems on the cloud or router crash along the way, are a few examples. Regardless of the issue, every case should be handled transparently.

In fact there could be many different cases, and each one should be addressed carefully. The simplest rule is not to answer them all in the same way, i.e. by showing an “Error” message. For example, let’s imagine that a user enters his login but forgets to enter his password before clicking “Ok”. The server, instead of just showing an “Error” message, should instead say: “We tried fulfilling your request, but a mistake was made: the password field is empty”. In other words, we should track and explain at what step the problem occurred, and what exactly has happened.

Conclusion

In each concrete application, dozens of specific (and often times unforeseen) situations can appear. That’s why we need the second 90% of development time to handle these issues and make the app consumer-ready. This exacting, sometimes painstaking work is exactly what distinguishes ordinary apps from the ones featured in the App Store: the latter do not crash, hang up, or make confuse their users. And that’s why the second 90% is integral to a successful, bug-free app.