MODULE 01 - 069: Planning the Prime Objective of an Application

---

Video Lesson Speech

[ENG]

Jordan Hudgens: In this guide, Jesse's going to focus on user stories and also hit upon the difference between user stories and a feature list. It's a subtle but very important difference. As a new developer, you may get your feature list think that's really all you need. When I was younger, I thought that's what I wanted. I always would tell a client, "Just send me bullet points with what you want the application to do." As much as it seems like that would work, using user stories can actually be a much more effective way of doing it.

Jesse Cook: *Absolutely. As you said, when you think of an app, you think of what it does, right? You think, "It can do this, it can do this, it can do this," but an app is not a one-dimensional thing. It's usually being interacted with by multiple different people. You need to identify all the different people that may ever touch this thing, all the way from the bottom, with a guest user, to the top, with a super admin.*

So once you've identified this prime objective–and you may have the bullet points of this feature list that the client, or your manager, or your boss has brought you–and that needs to be converted into user stories. Let's take messaging. The client says, "This app needs to be able to message." That's a feature. Jim, he's a standard user, non-paid. Can he message? Yes. Jim can message. ***There*** *is a user story.*

Then you have Susan, who is the super admin. She needs to message, but she also needs to CRUD messages. She needs to be able to do all those things, so she might have deeper access. Maybe she needs to be able to read everyone's messages. Hopefully not, but maybe. But she, at least, needs to be able to CRUD them. Identifying that one feature–which is just "messaging"–can turn into eight different user stories for eight different users. This is an obvious one, but just to show this line of reasoning, you say, OK, we want to have messaging. We have Jim, who's a standard user, and we have Susan, who's a standard user, and they can message back and forth. Can Jim flag inappropriate content inside of messaging? Yes, he can. So he flags it. Well, then what happens to it? You have to follow that data. Now that it's flagged, who manages the messaging? So Jim can send messages. Jim can receive messages. Jim can flag messages. Can Jim review his flagged messages? Maybe, maybe not.

JH: Right.

JC: *So Susan, who is the super admin, she needs to be able to do something with it. She needs to be able to review the flagged messages and CRUD these flagged messages.*

JH: Absolutely. There's some very nice overlap here on the UX side with more formal software engineering tasks. When it comes to user stories, if you understand UML, you can use case diagrams to formalize these user stories. You can set up your actors–the super admin, the regular users, the guest users–and then you can just have lines that go to each one of those use cases. With this, you can have an entire chart that outlines all of your user stories.

There's a lot of overlap between development, UX, and software engineering. You're going to go back and forth between these worlds quite a bit. I don't want you to think "Oh, right now, I'm just in UX mode, and all I do is I get on Sketch and I do nothing else." There's definitely a time and place when you do that, but then there's also a time where you switch over into LucidChart and you start building a UML use case diagram to design and model your user stories. Being able to understand that there is overlap there is definitely very helpful.

The other thing I really liked about your example is how we started with one feature: messaging. If you ask a client about messaging, they're probably not going to give you a lot of detail on what features need to be there. But if you say "What exactly do you want this guest user to do?" then it makes it a lot easier for them to tell you about that. I think it comes down to how people think. It's a lot easier for them to slide into the mental mindset of a guest user and say "Oh, this is what happens when I visit a page! **This** is what I should see as a guest user."

JC: *And it requires a lot of empathy, right? Don't just think about how* ***you're*** *going to interact with the app. You've got to be able to step outside yourself. It's about being able to see the app like all of the different users that may interact with it.*

JH: Yes. It's just another step along the path of building a proper UX. We started with a prime objective and right after that, we went into those user stories to be able to figure out exactly how this system is going to be interacted with.

JC: *There's one more thing that I have to bring up: the idea of having that crossover technical ability. Again with the messaging example: not all messaging is created equally. If you're using web sockets, you're going to have a different* *messaging experience. As a creative director, project manager, or developer, this may not be perfectly communicated to you and you'll need to ask those questions: Are we going to use web sockets? Is this going to be live messaging or is this going to behave more like an inbox?*

Those types of questions get asked to the client because, a lot of times, it's a budgetary thing. They're asking questions about cost, and how much time it will take to build it with live messaging, versus an MVP where you can prove the concept of messaging, and it will be a lot cheaper but it won't be as live as text messaging on your phone. Not everyone knows about web sockets, and they don't know what they don't know, so understanding the technical aspects so that you can bring up those types of questions is very helpful.

JH: Absolutely! That's a very good point when this process needs to occur and how important it is that it doesn't happen too late. Some of the things you just brought up will actually determine the type of technology stack that gets used to build the entire system.

JC: *Right!*

JH: So if you do this too late, then you might pick out the wrong stack because you either thought it was going to be much more or less resource-intensive than it ended up being.

I remember a great example of this, for me, where I had a pretty large project. Because it was so large, I didn't get the user stories first. I thought, "I want to build this in Angular, and I want to have this microservice backend with multiple Rails APIs," and I had it in my mind that that's what I was going to do. We started going into the user stories and it didn't need anything that Angular brought to the table.

JC: *Haha!*

JH: It didn't need web sockets. It didn't need live-updated data. It didn't need any of that! And, because we caught it before I started building it, I was able to build it in pure Rails, and I delivered the full set of functionality they asked for in half the time it would've taken me to build what I originally had in mind. And I was able to do that *because* we got into the user stories first.

JC: *Exactly.*

[SPA]