0:14

As you design software to satisfy requirements, you have to make many important decisions. Certain design decisions involve trade-offs in different quality attributes such as performance, convenience and security. Think about balancing such qualities when designing a front door to a house. You want to make your home more secure in the most direct way. So, you decide to add locks. You add a single lock and don't feel very secure. It's easy to open and close the door very quickly. So, you continue to add extra security. You keep adding locks until you realize it's very time consuming and inconvenient to unlock your door. Instead of mindlessly adding locks, you must come up with a design that balances security with both convenience and performance in mind. When designing software, it is important to consider how qualities can compete in a proposed solution under different situations and determine a suitable compromise. This is a constant balancing act for a software architect. Let's talk to an expert about how competing qualities constantly influence their implementations. Yeah, there's always a lot of competing concerns with software design and I think, obviously, from a product perspective, you have things like usability and performance. Obviously, those are are critical. Scalability is always important. Where the trade-offs really seem to be important is when it comes to security, code quality, time to market. These kinds of things where everybody wants and you need to be secure, right? And you need to have high quality. But the pressure that you have from product, to get something out the door, is sometimes at odds with those requirements. And as an architect, its your job to fight for those things and to advocate for them in the context of the business. So, it's not slavishly sticking to a point and to the point that you're damaging your business, but it's about advocating responsibly for the quality of the code base that you're producing. So, I think if I would summarize, quality is the job of the architect. You're ensuring the quality of the code base and you're trading that off against lots of things all the time. Architecture is about producing a quality product and you have to define what quality is. That's what architecture is, right? It is that you're defining what are those quality attributes that you're chasing after. And you're going to deal with performance, you're going to deal with scalability, you're going to deal with maintainability, security, all that stuff. Also, you want your job to advocate for that quality for all of the stakeholders, right? Engineers need quality, customers need quality, the business needs quality. So, I think overall, I mean you're always bouncing a lot of things. Architecture is all about balancing competing concerns. But if I boil it down to the most important, it's you're balancing quality versus time to market. Right? So, as customers, as businesses, we want to get things into market. We want to start making money off of them or serving our customers or whatever it might be. And as engineers, we have a tendency to say we want it to be perfect. We want it to be 100% code coverage. We want it to be tested every way possible. It's got to support as much load as you could ever possibly throw at it. But the reality is there's always a trade-off there. And so, as an architect, you have to be able to establish what is good enough, right? What are the non-negotiables? What must you do? And then, what can you negotiate on? And once you establish that as an architect, you've made a really good job of establishing those guardrails and then the team can execute from there. The business and the engineering team can execute from there. Context is important to determine what choice of solution is right for the balance of qualities. For example, a home located in a low crime area will require different security needs from one located in a high crime area. For software, talking to its stakeholders will help you to understand the context. Sometimes choices made in your software designs can have unintended consequences.For example, an idea that seems to work fine for a relatively small amount of data may become impractical if there is a need to deal with a lot more data. It's good to get other perspectives on your technical designs for a more rounded implementation. These perspectives can be in the form of asking other developers for their opinion or having a design review session. In the home example, you may decide to install bars on the window of a house built in high crime neighborhood. However, the bars might have the unintended consequence of preventing an escape through the windows in case of fire. These consequences could have been avoided if a fire marshal's perspective had been considered before implementation. Besides design reviews, it is worthwhile to slow down while implementing a system and test it carefully. You can prototype alternative ideas and run tests to see what works best. If a design decision has unintended consequences, tests can help to catch them. In the same way for the home example, running an emergency escape drill would detect the bars as impeding a fast escape. Let's see how various qualities arise which influence your software design. For software, there are functional requirements that describe what the system or application is expected to do. For example, a media app has a functional requirement of being able to download a full length movie. Naturally, a software design needs to outline a solution to meet such requirements correctly. So, a key quality to satisfy is simply correctness. Besides functional requirements, there are also non-functional requirements that specify how well the system or application does what it does. Such requirements may describe how well the software runs in particular situations. For example, the media app can have non-functional requirements to download a full length movie at a specific speed and to play such a movie within a certain memory limit. Beyond correctness, other qualities to satisfy include performance, resource usage and efficiency, in terms that can be measured from the running software. Both functional and non-functional requirements are important to satisfy. You'll need to discuss what is acceptable with the stakeholders. The desired qualities will put constraints on your system's design. Consider an analogy involving categories of cars. Both sports cars and minivans meet the functional requirement of providing transportation. But each category, indeed each model of car, offers a different balance of achieved non-functional requirements with different factors like

acceleration, handling, cargo capacity, weight and fuel economy. Another kind of non-functional requirement concerns how well the code of the software can evolve. For example, parts of the implementation may have to support use in other similar software products. Also, the implementation may have to allow for future changes. So, other qualities to satisfy for the software can include re-usability, flexibility and maintainability. As the design gets detailed and the implementation is constructed, the required quality should be verified through techniques like reviews and tests. As well, certain qualities can be validated with feedback from end users. You need to keep many qualities in mind when designing software. It's not enough to write any code you want if it works to meet the desired functional requirements. There are multiple perspectives to consider. You must satisfy qualities that matter to the users of the software, as well as those for its developers. In software design, your starting point is ensuring your software structure suits the balance of qualities desired. How the structure is organized may affect the performance as seen by the users, as well as the re-usability and maintainability as seen by the developers. In particular, there is a common trade-off between performance and maintainability. High performance code may be less clear and less modular making it less maintainable. Another trade-off is security and performance. The extra overhead for high security may reduce performance. Extra code for backward compatibility can worsen both performance and maintainability. Achieving the extremes in such competing qualities can, in a way, pit users against developers or users against other users. Generally, you have to strike a balance during design. You should ask how much performance, maintainability, security or backward compatibility is needed. Can you cut back on performance to gain more security? Can you drop some backward compatibility to have better performance? Finally project realities will impose compromises on your design. You must balance the software qualities with the resources you have to develop your product. Thinking about quality attributes gives a broader view on how you can achieve the desired requirements in your design. You now understand these qualities as competing ideals that must be balanced. When asked to satisfy a quality, you will consider multiple perspectives and look for potential trade-offs.