00;00;00;00 - 00;00;31;26

I'm sitting here with Professor Tomi Männistö. We are going to have a talk about a very important aspect of software engineering, and that is software architecture. Thank you very much, Tomi, for being here. Thank you for inviting me. And if we start from the objective of this podcast and, we start from the view that each of the architectures is about the dealing, the complexity of, systems of all kinds of software systems that seem to be becoming more and more complex in many ways.

00;00;31;26 - 00;01;04;02

And, and that's the basic motivation for software architecture we are going to use today. And then after that, we'll take a little bit of a look at, what is software architecture. What is the role of a software architecture in software development? And then briefly about architecture design when we are going to talk a little bit about a concept of architecturally significant decisions, which is the one key concept in software architecture design and then take a look at the role of the software architect.

00;01;04;05 - 00;01;30;12

Why should we care about software architecture? One ways is that what I already said about dealing with complexity. But then if you take a look at what companies are, for example, interested in, one source you might Google into is cnnmoney.com which actually lists those top rated best shops in America. And I think software architecture has been there at quite a top in many ways.

00;01;30;12 - 00;01;53;20

And that's one indication of that. There is something in the job description that actually makes the companies interested, and there's something that they those are the people they actually want to have. I understand that, and that's a very good motivation for learning software architecture. What about if we look at architecture from a software engineering point of view? Why do we need something like software architecture?

00;01;53;23 - 00;02;13;28

We can start from this. Complexity is that if you're building a simple thing, I sometimes use an analogy of building a dog house. You build something that you can start building. Just build it. If it goes a little bit wrong, you fix it and that's design process or building process where you don't need that much of the overall thinking.

00;02;13;28 - 00;02;40;27

But then when you go to build something a little bit more complex. Well, Sydney Opera House or whatever. You just can't start building it from one corner and go one from there. And that's way where you do need something else. Kind of a situations in which you are building complex things that won't go for, build and fix or you can't just first do one thing thing, but you have to do something else.

00;02;40;28 - 00;03;01;18

The role of a software architecture is this thinking before you start doing what you are about to do these big things. If you're building something simple, something you've done millions of times, web shop that you've already have delivered. You don't need software, so all you have is of the architecture there implicitly, somewhere. You don't have to care about it.

00;03;01;20 - 00;03;24;00

But then when you're doing something complex, new things you don't understand, then you typically have to care about software architecture, then software engineering. So where does this complexity that we need architecture for? Where does it come from? It's not only one kind of a complexity, of course one thing is size. When you are doing big things, then you typically need to organize them somehow.

00;03;24;00 - 00;03;48;11

And they might be pick in different terms, the size of the code in terms of lines of code or number of nodes you have any and whatever means or amount of data you need to handle or manipulate or transfer or whatever. Those are size related things that might make your thing complex. And you have to think about, okay, how are you going to deal with that in a reasonable time then?

00;03;48;17 - 00;04;19;28

But there are also other aspects to complexity, like typical ones in software architecture design are important quality attributes or sometimes called nonfunctional requirements like performance. If you have strict requirements for performance or something that you need for a competitive advantage, or then you have hard constraints, like if you are trying to grind an airplane, well, there's a certain limits within which you have to have to actually accomplish things.

00;04;19;28 - 00;04;46;09

Or there might be other similarly issues like security or there might be limitations with respect to such qualities, like physical resources, so typical of mobile devices. Or today they need to be very aware of how much they consume energy, and you have to build them typically to minimize the energy consumption in order to increase the battery life. So this one other thing is broad category of different quality attributes.

00;04;46;12 - 00;05;09;21

But there are also others that might come from the application domain, such as if you are doing something for medical or military, they have very, very strict quality attributes or procedures or processes that you need to prove that your system does things properly. So this kind of a different

tough domain requirements might be one way of increasing or making things more complex than just doing them.

00;05;09;21 - 00;05;36;18

And one particular reason is a typical these days it is this variability, which means that in many cases when someone makes a software product, it's not just one single product, but a family of products. So you have multiple products and then you need built your system in order to enable this variability. You can't build every single product, for every market, for every different customer group or every different price group.

00;05;36;18 - 00;06;03;05

You can't do them separately from scratch. You have to do them from one set of assets. And in that software architecture is typically a central asset. We have talked about software architecture, but we haven't defined it yet. Before we actually define it, we can start from seeing the software architecture from different perspectives, because that's actually depending on the situation of a particular company or whatever where you are doing your software.

00;06;03;08 - 00;06;28;05

It might be that different things are important. And so one way of seeing is that what do you try to do with software architecture is that you try to manage the risks related to the complexity. So for example, if you have tough performance requirements, so before you start building, you try to do enough design, enough overall thinking that are we going to meet these performance requirements or how how risky is that

00;06;28;06 - 00;06;48;13

and then you do architecture design in order to come up with some initial ideas or overall structure of the system. That gives you some little bit more certainty about the risk. Related to that one way of seeing software architecture is that one can say that every system has an architecture defined as a highest level of abstraction of systems.

00;06;48;13 - 00;07;11;20

So that's the architecture. It may be that nobody even knows that, but in the system there is architecture. And then there's separate issue, whether there's any documentation about that. But that's one way of defining architecture to the highest level of abstraction of a system. Of course, there are also formal definitions to or standard definitions that are architecture and one of the most used one.

00;07;11;20 - 00;07;46;05

The quite widely used one is from a IEEE -standard that actually says that architecture is the fundamental organization of the system, basically boils down to that it defines the system components and how they relate. Sometimes architecture is called that being a component connector view of the system. So that's one way of defining the architecture from more or less this kind of a structural point of view of a and then actually the definition from standard goes on and takes also into account the environment into which that goes.

00;07;46;05 - 00;08;09;24

It's not done in the system, but how it relates to the environment and the principles you use to guide the development. So architecture is not only the structure but how you communicate that to the developers, how to build the system so they actually will have such a structure. Who is interested in the architecture? When you think of a system, there are always people to whom you build the system and one can define that.

00;08;09;24 - 00;08;32;04

Those are the stakeholders, the system and that's where architecture design starts, that you try to understand that to whom you are building the system for. And in architecture design, you typically talk about concerns of those stakeholders and what they have. They might have different views someone wants about money, someone wants speed, someone wants performance, security and so forth.

00;08;32;05 - 00;08;59;24

And the typical stakeholders include, of course, the systems users that how they use and how they see the system, how much they actually tolerate, for example, latency from the system responses and so forth. And then another thing is, of course, the customer who's paying for the system. Those are the typical typical stakeholders. But there are many others you need to consider, or at least think about when you are thinking about building the system and then making the architecture for it.

00;08;59;24 - 00;09;20;21

Starting from managers and project managers who might have deadlines there. And there would be, of course, developers who develop the system. For example, simple thing is that when you think of building a system, you might select some technologies. If the people who are supposed to develop the system don't know the technologies, you have to take that into account.

00;09;20;21 - 00;09;47;08

And they're capabilities. So there are many things. And of course there are legal issues that are issues from sales and marketing. You have to make it so such that things that include the systems are easy to build and so forth. So that's the idea to architecture is to see that what are the concerns of the

stakeholders, whether there are qualities or some constraints over the system, what kind of concerns that you have and try to understand can you meet them?

00;09;47;08 - 00;10;07;02

So before you start building the system, you try to figure out that what are those concerns and how are you going to meet them. So what is the actual requirement for latency? And can we meet with this kind of a architecture if we use some client server architecture is that, you know, can we process everything that's needed in, in due time and so forth?

00;10;07;04 - 00;10;29;11

In that sense, one can say that the roles of software architecture include first building. So in the big picture, the overall picture of the system so that it helps you to communicate, okay. What are the main elements of the system and how they interrelate. And that's just kind of a standard definition of what's your overall system or highest level of abstraction.

00;10;29;13 - 00;11;01;22

But in addition to that, it's very important tool for documenting the early design decisions. These are sometimes called architecturally significant design decisions. So, making those decisions and communicating them enabling the communication between different stakeholders about those decisions. So for example, if there's a someone interested in that the security system is very important. But if it has some implications, for example the performance then you have to in many cases you might have to make some trade offs.

00;11;01;22 - 00;11;23;02

You can't have the fully secure system and have to perform as, best possible performance out of there. And now you need to make a decision about that. And actually who makes the decisions are those stakeholders. But they now have to understand what they are making the decision about. They have to understand that, okay, we can't have the highest level of security.

00;11;23;02 - 00;11;44;27

Can we be satisfied with the level of security, or can we sacrifice the performance? I mean, this kind of a trade off discussions, they are basically making technical side decisions. But the input for those decisions comes from the stakeholders. And you have to be able to engage those stakeholders into those decision process in order to make the system for them and not for you.

00;11;45;01 - 00;12;08;13

Doing the architectural design. How do you do that? And and who does it? The one view was that you make the architecturally significant design decisions. The idea is that those are the ones you

made in the beginning, and you have to identify that, okay, that you make them first because they are characterized typically so that they are the ones that are difficult to fix or change if you get them wrong.

00;12;08;18 - 00;12;39;00

If you build your system from the front component, from kind of a components from from structure, or you use some technologies that cannot scale up or whatever, it's very hard to change that underlying technology later on. So that's the typical kind of architecturally significant design decisions. They are typically also kind of decisions that are cross-cutting in the sense that they affect multiple components if you want to change them, if you want to add authentication to all components, it's going to affect all components.

00;12;39;01 - 00;13;12;07

And it's better to decide that early on than trying to do it later, because then it's harder. You have to go through everything. And then then to also mentioned this system quality is there enables that. If you need high performance, you better start thinking about it early enough before you start building, because it may be that it's not fine tuning in the one place you have to understand where the performance comes and where you lose it, and it's good to understand it before you build the system, rather than realizing from the ready made ready system that, okay, well, we won't get anything out.

00;13;12;11 - 00;13;42;22

So those are the architectural design decisions. This architecturally significant decision, some very, very critical to me. How do you know what an architecturally significant decision is. You know them typically if they fail. And the idea to know them is to make them before you fail. So I mean, this is it's kind of a chicken egg problem. And the main point in architecture design is that when you when architect starts his or her job, these are not given.

00;13;42;24 - 00;14;09;02

Nobody is going to tell you most probably as well. If you think of the stakeholders, they think about one thing they don't know about your solutions and what you're going to build and whether it's going to work out. So nobody's going to tell you which ones are the architecturally significant decisions you need to make. And that's also most probably one of the difficult parts of architecting, figuring out first what is the problem.

00;14;09;02 - 00;14;32;01

Because you don't just solve the problem, you have to first figure out what is the problem, and you get the problem by talking to stakeholders. And that's that's where you start identifying the

architecturally significant design decisions that you need to make. Although the best, best advice for that is that is the responsibility of the architect to decide what's architecturally significant.

00;14;32;03 - 00;14;55;20

And that's the important thing to understand that nobody's going to tell you that you have to do that. Most probably it's your responsibility. Of course they might tell you, but still it's your responsibility to figure out whether it's really the important thing or not. And that's one part of the expertise of the architect. To understand that and take the responsibility of figuring out what, what's really needed there.

00;14;55;20 - 00;15;13;25

And if we go this way, is this still going to be a performance problem? Are we going to have a security issues if we build the system that way? And those lots of things that the architects just need to take the responsibility off. So it sounds to me like a good architect knows what these are, but there's no clear rule.

00;15;13;25 - 00;15;37;06

They don't show that you. I mean, architecturally significant decision is something you need based upon experience, domain and system understanding to figure out yourself. Exactly. That's correct. Yeah. So the idea is, is exactly the one key element is the experience. And when you think over these kind of a core CS experience, it's very hard to talk. Which experience comes from the root.

00;15;37;10 - 00;15;58;15

What you can learn from this kind of course is, is to understand the role and some methods that when you start building your purpose, then you can actually utilize it into those to making it. So it's this why software architects are paid so well. That's probably I think that's one part of it. In other bodies that there is these risk management issues because they are down the line there.

00;15;58;15 - 00;16;25;17

If the system doesn't scale up, they're going to show that the software architect first, they somehow bridge the gap between the problem domain and the solution domain. So they have one foot close to requirements engineering and one foot very close to development. And understand both exactly when you start making software, this architecture and security design decisions fix something or think about, okay, rebuilding this kind of a system, you typically induce other kind of requirements.

00;16;25;17 - 00;16;45;27

So if you think, okay, we are going to use mobile applications here and then mobile devices for some things, then of course if that wasn't on the table the requirements earlier, now you have a

requirement. What are the what how much we need processing power memory there, how much we need to screen. So these kind of issues come up.

00;16;45;27 - 00;17;07;10

So you are actually generating requirements while you start making a decision. So it's it's kind of a two way process I think this is something we could explore a bit more because we have talked about requirements engineering which is engaging the stakeholders. And now you say, well, the architect does that too. So is the architect part of requirements engineering or is this some different engagement with the stakeholders.

00;17;07;10 - 00;17;31;04

So what is the relationship between architecture and requirement? In many cases the answer is that they are separate, which most probably is not that good. But one other way we've seen is that when you do the requirements, you list quite a many requirements which most are not architecturally significant, so they don't matter that much. You can just implement them.

00;17;31;06 - 00;17;56;04

Architect doesn't need to be involved in collecting those and defining refining those and understanding those decisions. Architect comes into play when you start already thinking about the system, and then architect starts looking at those requirements and trying to pick up architecturally important one. Typically because the usual requirements process, in most cases it concentrates on the functional requirements.

00;17;56;09 - 00;18;22;26

So what the system is supposed to do rather than that much on, for example, performance or security. And they might if they are really important, they might come up already there. But many of them, the maintainability of the system, of these kind of things come up at all in that way. So when architects come, the architects try to figure out which of the functional requirements are architecturally significant and which potentially other requirements that are not yet stated, there are there.

00;18;22;26 - 00;18;39;11

And that means that architect needs to come into the process. At what stage they come there might depend in mitigation, might be a little bit too late, so they have to redo some of the work with their stakeholders. And they I think that there's no way out of that. You just they have to engage the stakeholders. I think this clarifies it quite a bit.

00;18;39;13 - 00;19;00;11

Let's look at the other interface that is downstream to the developer. So what is the interface between the architect and actual developer? The architect makes the high level decisions of the system about the structure, how you build the system. And clearly that is one way of instructing the developers, okay, this is how we want to build the system.

00;19;00;11 - 00;19;19;00

So for example, if architecture is just we are going to divide. It used to mean these kind of a layers. When developers start with they need to understand okay, these are the layers and these are the rules for actually how will you if you build something in this layer, what things you should be able to access. Because those things it might be the same code base.

00;19;19;03 - 00;19;43;08

They are not restricted in any other way but by the convention. And that's one way of seeing architecture for the developers. Is that architect actually put some constraints on developers and how they can do their job, and that's very important for architects to understand that architecture is not supposed to do the design decisions for the developers. The low level design decisions or coding decisions by the architect should make the big decisions.

00;19;43;08 - 00;19;59;26

Okay, these are the things we are going to do in the server. These are things we are going to do in a client or this is how we structure the system at the higher level and then communicate that to the developers. This sounds like you need to be a superhuman to be a good architect. It sounds like you need to have in-depth technical knowledge.

00;19;59;26 - 00;20;19;11

You must be good with people and understand both the problem and solution domains. So how do you learn to do all that? It's not easy, but I think that's very good. Architects are kind of that. Or at least they have two out of three. Of that said, they are technically very, very competent. If they can't talk to the people or they can talk to people, they understand some.

00;20;19;11 - 00;20;37;14

And of course, in many cases with the big systems, you have a team of architects working there, so you might have a little bit of different roles and that's it. But but that's the way it's a very demanding job. And typically those best programmers tend to get this kind of a position as an architect at later on in their careers.

00;20;37;16 - 00;20;57;19

So that's not not uncommon. So how grow into the role of an architect? And one part of that is not the experience. So it's not that difficult when you have seen enough of the system. So easier is your decision making. Then you already have a knowledge okay, this works and it doesn't work and so forth. So you are not just starting from scratch, making those big decisions.

00;20;57;19 - 00;21;19;23

Architecture and its place is somehow easy to understand in the waterfall model. But now these days, we see more and more organizations turning toward agile development. Agile development has, as far as I know, fairly little to say about software architecture. So could you comment on the role of software architecture architecture in agile development? There are a couple of ways of commenting for that.

00;21;19;23 - 00;21;45;15

One. First one is that whether you really need an architecture, well, whether the architecture is simple, that comes from the where we started from the complexity. So if you are doing a very highly complex system, which you have a high risk of a failing, then agility might be a very, very hard to do. But if you have a system you have done, you know how to do that, know you have, or you are using technologies that basically restrict your architecture to more of your control.

00;21;45;15 - 00;22;05;22

Other than that, that seems to be good for kind of applications you are building. Then you don't have to worry about the architecture and you can go with your agile development. Then your job is more about development, then architecture design. But then of course there are cases where you have tough architecture and you would still like to do some, development in an agile manner.

00;22;05;24 - 00;22;25;13

And there have been different ways companies try to deal with that. Sometimes they have a separate board of architects making sure that nothing, goes wrong with the architecture. They have all the architecture, some kind of guidelines or, and try to see that when you develop the code that you will stay within the architecture. Architecture doesn't go bad at any stage.

00;22;25;15 - 00;22;49;12

Or then you might have different sprints that you do some architecture design in the middle and try to keep the architecture good and try to concentrate a little bit more on the architecture. And then there are some approaches in which you take these, let's say performance. You try to put it down as a requirement that you try to develop also bit by bit, okay, that in the first sprint be try to get this level of performance.

00;22;49;12 - 00;23;11;17

And then later on we are going to go through a trend. So which means that you also have to think about the architecture all the time. I don't believe that the architecture just emerges without somehow magically emerges when you just code. Well, it just you just come up with a good architect. That's something I don't believe. You have to somehow put effort in that.

00;23;11;17 - 00;23;31;24

It might be that there's a couple of persons who know so well the area that they are actually doing the good job, but they still, at least in the back of the mind, they have to be very conscious about this. Okay. They are going to come up with a good architecture. So would I paraphrase you correctly, if I say that I interpret you as saying that good architectures don't emerge by refactoring.

00;23;31;24 - 00;23;53;16

They emerge by planning and thinking. I think you are doing a good job in rephrasing this, because you may or you might refactor into a good architecture. If it's simple enough and you I the knew you'd never got too much away from the good architecture you made a good case this or you have a good intuition from the beginning.

00;23;53;16 - 00;24;17;10

But in general, I don't believe it's a lot like starting to build a bridge without first thinking at how long the bridge will be, or how you build it. I mean, this kind of a thinking that you have to have some, you have to get that idea from somewhere. You can't rely that that you do whatever you do, then you refactor it into order, of course, or you are going to do quite a lot of refactoring if you all might be very, very expensive.

00;24;17;10 - 00;24;48;21

Exactly. Because that was the issue with the architecturally significant decisions that they are hard to fix. So if you build a total mess and it doesn't scale up, then fixing that is starting more or less from scratch, thinking about the system and doing it again. So you can refactor but take couple of years. So that means would for example, mean that for architecturally significant design decisions, a later change is certainly not cheap even in an agile process.

00;24;48;23 - 00;25;14;14

Yeah, exactly. So and that's a actually very good point because one of the thing is that it's clear that even if you want to embrace change, there are things that are harder to change than the others. You can't build a system where you can change everything cheaply. And I think that's one way of putting architecturally significant decisions in is that if they are hard to change, they are architecturally significant.

00;25;14;14 - 00;25;36;21

If you've got them right, good. If you didn't, then they are going to be hard to even regard as what how agile your development is. I think that's an excellent point. So let's stop with this very important thing to keep in mind. But the final question is, I guess many students now got very interested in software architecture and understand both the role importance and the good pay you get as a software architect.

00;25;36;21 - 00;26;06;06

So how can they learn more? This is what's an interaction do then we have a little bit more architecture on the software development course. And then we have a special course at Aalto for software architecture. So there's a specific course where you actually go on a little bit deeper into the methods how to actually make these architecturally significant design decisions. What are the methods for that and means for validating or evaluating your or your decision before you start building?

00;26;06;09 - 00;26;15;20

Okay. Thank you very much, Professor Männistö, for your time. This was a very interesting discussion. Thank you.