00;00;00;00 - 00;00;29;09

Hi, my name is Casper Lassenius and I want to welcome you to this podcast about software architecture on Aalto University. This course in Software Engineering. My guest today is Professor Philippe Kruchten, who is one of the leading experts in software architecture. He has an extensive over 30 year long industrial experience of software development and software architectures.

00;00;29;12 - 00;00;55;07

When he was in industry, he worked mostly with large software intensive systems design in the domains of telecommunications, defense, aerospace, and transportation, while Philippe was, for example, the main architect for the Canadian air traffic Control system. He directed the development of the Rational Unified process RUP that you have read about from 96 till 2003, when Rational Software was bought by IBM.

00;00;55;09 - 00;01;22;22

Philippe is also the author of several books and his textbook on the Rational Unified Process, called Rational Unified Process, and the introduction is required reading in many universities. Philippe Kruchten is also the mind behind the four plus one views to software architecture that you have read about, both in your textbook and perhaps elsewhere. It's also one of the most cited papers ever in the field of software architecture.

00;01;22;24 - 00;01;50;25

So it's a real pleasure to have such an expert on software architecture with me today. And thank you very much, Philippe, for being here. And let's start with the really basics. So what is software architecture? Casper, thank you for having me this morning. Architecture. Well, it's a bit of a difficult concept to explain because it's pretty invisible to the external user of a software intensive system Architecture,

00;01;50;25 - 00;02;12;23

it's the overall structure, how we organize a software system. It's the result of the major decisions, the one that will be relatively difficult to change later on during the life of the system. Things about which technology do we use? Which programing language do we use? How do we integrate with other systems? Some of those decisions address some of the big ilities.

00;02;12;23 - 00;02;38;29

You know, performance, response time, availability of the system, security. All these things are not easy to sprinkle. When the system is finished. You cannot have a complete system and say, oh, let's add a little bit of security in it. So it's deep down in the guts of the system it represents, like the skeleton of the system. Now, in the agile world, a lot of people have sort of dismissed architecture as.

00;02;38;29 - 00;03;00;09

Oh, yeah, something from the past. We don't need any big up front design. Well, yeah, in many circumstances, they're right. When they arrive on the project, there is already a de facto architecture. All the major choices have been done before they were hired for the project. Or there is some standard architecture in the domain where they were. Yeah.

00;03;00;12 - 00;03;28;15

You know, most little e-commerce system, they have a little database on the back end. They have a front end, they have a, load balancer. They have a web server. And two choices are relatively obvious. when you arrive in the project, maybe they've been done two years ago already, when the project is a little bit more novel, or when you want to stretch some of the capabilities of the system, make it work for a thousand users instead of 100, making it ten times faster.

00;03;28;18 - 00;03;53;00

well, then you may have to revise even what look to you as, as a relatively reasonable architecture. And you thought that you didn't need to do any architectural work. So, unfortunately, a lot of project, when they scale up, they realize that, sometimes a little bit late, sometimes when they're really in deep doo doo, that maybe they should have thought about architecture.

00;03;53;02 - 00;04;37;20

But Philippe, in the Agile Manifesto and the agile principles, there is clearly this idea of architecture as something that emerges, not something that you plan. Yes, architecture will gradually emerge. Actually, that's one of the beauty of using iterative development, such as agile, most agile software development, because it allows architecture to be built gradually, not just plan, build validated gradually, with actually running code, you can make spread your architectural decision over the first few iteration of your project and have what at least Coburn calls a working skeleton, on top of which you're going to build up more of, your system, but you're going to validate your architecture.

00;04;37;20 - 00;04;59;28

You can build test, test harnesses to shake the performance, shake, you know, the load distribution, shake the database, make sure that you have the right bones in place before you start putting too much flesh. flesh on top of it. So emergence will happen, but somebody will have to drive it. It doesn't happen by accident. You cannot just launch a project.

00;05;00;05 - 00;05;26;27

And that iteration 17 say, oh, anybody has seen a new architecture emerge. Well, it hasn't yet emerged. Let's continue to pile up some code. Actually, you're digging yourself slowly to a grave

because if you nobody pays any attention to architecture, you're probably accumulating something very nasty called technical debt. The little extra skill or knack of one of the developer is to be able to have a little bit of flair and time.

00;05;27;00 - 00;05;54;18

We're going to do this this week or during this spring, thus is this architecturally significant? Is this a choice that we may regret later on? Do we want to spend just one more hour speaking about it, thinking about it, looking what other people have done? Because once you've we've done that, yes, we can do it in a few hours, but in two months from now it may be relatively difficult to add this feature, you know, get this kind of performance.

00;05;54;18 - 00;06;16;20

So sometimes you will make shortsighted, choices, you know, that you have to get something out of the door from, you know, a better version by the day after tomorrow, you will make that shortcut. But you need to remember that shortcut. And maybe after the day after tomorrow, you need to make the right choice. Repay this temporary little technical debt, which we could call a wise investment.

00;06;16;22 - 00;06;47;21

So how do you see the difference between software architecture in the waterfall model and in agile in a big waterfall project from last century? Yes, you had a massive amount of requirement, but finding what was architecturally significant, you may realize that it's missing. Nobody tells you what response time your system should have in this or that circumstance. You had vague requirements like the system must be user friendly or the system must work all the time.

00;06;47;23 - 00;07;08;06

you know, that's today. yeah. It requires a little bit of flair and a little bit of experience for a more senior developer to sort of ask the right question to the business analyst, or to the customer, or to whoever or to the product owner. What if where are we going with this? Yes, we could do that within two weeks.

00;07;08;06 - 00;07;32;07

Where do we want to be in six weeks or in two years? So you need to anticipate a little bit in the future in order to guide some immediate decision. And that, well, it doesn't come naturally just because you've just attended a course in programing. It requires a little bit of experience of the domain, of the technology, of the whole process of developing new product, but it's not that difficult to do.

00;07;32;09 - 00;07;56;18

So drive with your eyes open. So here you talk about doing the architecture and who should do it. So we typically talk about the role of software architects. Do we always need a full time software architect. As you said it's a role. It's not necessarily a job title. It's not something that somebody must have on his business card and get, you know, a 20 person pay raise.

00;07;56;20 - 00;08;27;01

Well, maybe this is a role that is played collectively by the team if the appropriate skills are distributed throughout the team. However, pretty much like you have a product owner in scrum, I suggest that for bold new project, you appoint somebody who's an architecture owner. He may not be the chief architect, but he is the person who reminds the team at regular intervals that there are certain architectural decisions that need to be done and the Kanban is called that have a little bit of a different color.

00;08;27;01 - 00;09;09;01

The architectural things to do, if you use a Kanban, appoint somebody to be the architecture owner, make sure that that person understand what an architecture is. And, what, architectural significance decision might be. What are the drivers, the factors that drive architectural decision, somebody that's, you know, relatively familiar with the technologies that are available, somebody that has good contact with the product owner or who will represent the source of requirement, or even be familiar with, with the domain and, somebody that is a little bit persistent and curious and ask questions and doesn't, doesn't just sit on his hand or her hand.

00;09;09;04 - 00;09;38;18

Okay. So in traditional development, you do architecture as a quite upfront stage, right after requirements and specification. How do you see the role or and place of architecture in iterative and incremental development or agile development? Naturally, it's not the kind of decision you can make just two days before delivery. So architectural decision they tend to be in the, I would say early third or half of the project.

00;09;38;23 - 00;10;14;11

That doesn't mean that they all need to be done upfront very early on the first week, actually. Pace yourself. Make some that will allow your team to start doing some work. Like maybe we pick the development environment, maybe we pick the programing language, maybe we pick some general object oriented framework, maybe we pick Apache as our web server or whatever, but don't make all the decisions, you know, start building some code, look at what is architecturally significant and organize your architectural decision, the one that you still have to make.

00;10;14;11 - 00;10;44;00

Organize them over time by bringing forward the one that if you do them, you would enable the team to make some progress and pushing much backwards. The ones that they are not that important. Maybe they're more reversible. We can we can still change them and make up make up our mind. And what drives the sequencing is mostly enabling the rest of the team to make some progress by building up functionality on top of it.

00;10;44;05 - 00;11;07;04

Sometimes you'll have to stub one. Let's say you're pretty uncertain about what kind of database are we going to use. and everybody tells you, well, I wrote my code. Where do I put the data now? Well, don't rush it. Maybe put a stub there, put some temporary interface that looks like it's a database, but actually it just goes to some rough, binary file or text file.

00;11;07;06 - 00;11;33;17

This will allow you to postpone maybe for 1 or 2 sprint or iteration, the choice of a database, maybe two sprint later. You say, let's do something lightweight. Let's do some open source stuff like MySQL. And then after that, for some reason, you realize that you need to sort, or store all kind of relatively complex objects. Then you replace it, you know, three iteration later by MongoDB or some NoSQL database.

00;11;33;22 - 00;12;03;16

So you can actually pace your, your decision. You not have to make them and freeze them. You have as an architect to remain relatively adaptive. Okay. But how do I, as a software architect, understand what decision is architecturally significant? Take a decision and think, if I make a choice, what is the impact of that choice? Is it local to that module or is it local to that subsystem?

00;12;03;16 - 00;12;26;17

Does it affect one developer to developer, or is it pretty widespread? Does it affect all modules? Does it affect all teams? So what is the range of the impact of the decision. So that if you make a decision now and you imagine that I have to change it in three months, how many people will be mad at me and how much code we will have to redo?

00;12;26;19 - 00;12;52;01

So that's one criterion. The other thing that you may have, to use as a criterion is, are some of the big qualities affected? Does this affect security in some way? Does this affect scalability of the system? Yes, our first prototype. It's okay if it supports only 100 user. If we were to go to a million user, would that be the totally wrong solution?

00;12;52;03 - 00;13;12;19

Then there are some more difficult choices to make. Maybe you want to sort of say, let's do that, but let's not implement the simple solution in a way that ties us completely with it. Let's make sure that we have the escape of putting the deluxe version later on. Let me give you an example. I live in a bilingual country.

00;13;12;19 - 00;13;32;03

You also in a bilingual country, we have a minority of French speaker. I'm French speaker. You probably hear heard by now. you build a system in western Canada and you, you want to go to a better version rapidly. So product owners say, yeah, yeah, forget forget about the French. Let's just do a version in English. So everybody pretty happy.

00;13;32;03 - 00;13;56;04

Do a version in English. Shipped big success on the Twitterverse. And and now some people in Quebec say, oh, this is scandalous. You know why it isn't the thing, speaking French. So product goes to the team and say, well, you know, we really need to speak about, these people in Quebec and New Brunswick, so I need a version that does French also by the day after tomorrow.

00;13;56;07 - 00;14;23;10

Otherwise, you know, we get so much bad press in these that, so developers say, well, you know, this is complicated. You didn't want something bilingual. So we haven't externalize all the strings and. Yeah, just do it. So they just go to the code and they put if-and-else all over the code, with a Boolean flag that says in French, then and three days later they say thus we have a bilingual version.

00;14;23;13 - 00;14;48;25

Things go fine, everybody's happy. They go to version two. Things go fine, everybody's happy, bus goes to travel to Japan, and Japanese customers say, very nice product. We'd like to have that for Japan. Is product multilingual? Oh yes. Yes. Look, I can switch from English to French. Okay. Send me a version that does Japanese. Bus goes back to Calgary and says, okay guys, we have a big order for Japan.

00;14;49;00 - 00;15;09;20

Can I have a version? Speaking Japanese by the day after tomorrow, everybody turns pale. No, we can’t do that. Come on. You did the French version in two days. You know, I'm hiring you two nice student intern from Japan to help you with the text. I'm pretty sure you can crank us Japanese version for the day after tomorrow.

00;15;09;23 - 00;15;35;08

No, it can’t be done. We have to undo first all the stuff that we've done for the French. And we have to buy this package and this framework in order to externalize all this. Are you out of your mind? This is going to take, you know, 4 or 5 days. No, it's going to take three weeks. So that's the kind of, you know, if you don't anticipate a little bit the consequences, you can make some short sighted, decision.

00;15;35;08 - 00;15;58;20

Maybe it's a good thing because you need to deliver something, but they have consequences. So this is an example of an architecture decision where maybe not the ultimate optimal decision was taken on day one. Now, maybe on day one you would have picked, you know, the proper tool to do localization, internationalization and yes, this would have added a little bit of delay this content continuously.

00;15;58;20 - 00;16;26;06

This kind of compromise between immediate result with a relatively short development time, but long term consequence of these days, we often run into something that people like to call agile architecture. Now, Philippe, what is that all about? An agile architecture would be an architecture that is flexible enough so that it can accommodate relatively easily a certain number of changes in the future.

00;16;26;06 - 00;16;52;24

Some changes may have been anticipated, some not. So that would be an agile architecture, something that you know, can basically support easily a lot of anticipated or non anticipated enhancement and changes. Agile architecting. This is the process by which you know you use some of the agile principles. Rapid iteration and so on to develop an architecture.

00;16;52;27 - 00;17;37;17

Now you can have agile architecture to develop an architecture that's not very agile and vice versa. So the two are not necessarily linked by a relationship of consequence. Do that and you get the other agility supports very well architecture. Contrary to what some people thought, architecture is not tied to the big ugly waterfall. Agility supports architecture in the sense that you can actually try out things over iteration and undo them and pivot and change trajectory as you learn more about the requirement, and as you learn more about the system that you're building, you can test validate by running code some of your architectural choices, as opposed to a more waterfall approach where you can

00;17;37;17 - 00;17;58;00

plan the architecture but you have no clue if it's going to work. It's only in six months from now when you start integrating bits and pieces of your code that you say, oh, the performance is not there. Oh,

this is very late to discover that. Sorry, we'll deliver it like this. So agility allows to validate, architectural decision easily.

00;17;58;07 - 00;18;22;05

It also allows you to pace the rhythm at which you, you embed architectural decision in the system. And you can tie that by, the dependencies with the feature that you would build on top of it. So you validate your architecture by running actually some features rather than building some ideal architecture, hoping that it will satisfy everything that, the application developers, need.

00;18;22;10 - 00;18;49;21

So it's a, it's a pretty good support. On another hand, architecture support agility. It supports agility in the sense that it allows the partition of work between people. It allows to get some relatively early feedback on some of the ilities, you know, performance and all that kind of thing. So it, it both support each other. A focus on architecture, supports agility, and agility, supports a better development of an architecture.

00;18;49;24 - 00;19;13;24

So, Philippe, these days when agile and lean is the hottest thing on earth, we often hear the idea that we are supposed to deliver features as quickly as possible. That's what our customers want to deliver working software as quickly as possible, and architecture, isn't that something that is extra work that the customers typically aren't willing to pay for it.

00;19;13;24 - 00;19;45;26

So yes, maybe you want something cheap now, but it's going to cost you more later, you know? So architecture is really like, you know, putting an investment for the future. Now, you may not want to invest a lot for the future. If the future is very uncertain, but I think it comes with, some education of, your stakeholders, product owner or customer about what is the value of architecture rather than hiding the architecture as a dark dug deep secret inside the development team expose the value of architecture.

00;19;45;29 - 00;20;05;01

You know, explain to people, use analogy, explain to people. Yes, we could build a building, but if we or meet some major beams, that building may not resist an earthquake. Or you may not have, you know more than three floors, you know, so I can show you a demo building with the first two floors. But if you want something quick and cheap, it will be just two floor.

00;20;05;06 - 00;20;25;01

Oh, yeah, but we could always add the floor 4 or 5, six, seven later. nope. Sorry. So what if I want two floors now? But I want to be able to add four floor, 4 or 5, six, seven, and then we have to put some beams inside. It will not change the shape of floor one and two, but it will take a little bit more time and money.

00;20;25;04 - 00;20;46;07

But then if you want you can get four, 3, 4, 5, six, seven. So I don't know, it's a maybe a crude, analogy, but that's basically what happens if you put no focus on architecture. Yes, you can get something quick and dirty relatively early, but getting to the next step if you're successful is going to take some more work.

00;20;46;07 - 00;21;10;12

Scrapping out completely what you had done for the first release. And I've seen that case for one big financial system, six months of work, they got somewhere for 100,000 line of Java code was doing the job, but in unable to do any further progress because they had to completely ignore completely the concept of architecture and they were unable to go any further with the development.

00;21;10;14 - 00;21;35;26

They had expected that an architecture will gradually emerge out of weekly refactoring. While it didn't. So Philippe say, I'm getting all fired up about software architecture and like to become a really good software architect. What should I study? Basic stuff? Know how to code. Know how to code using 2 or 3 different programing language, 2 or 3 different frameworks.

00;21;36;01 - 00;22;04;21

Not that an architect must code necessarily all the time, but if you don't understand the basic, you know, skills of our trade, they may be old kind of thing that you're missing. Then be curious. Look at how systems are organized. Ask questions to your colleagues. people working in other organization, read blogs, try to understand the vast palette of architecture that exists out there.

00;22;04;24 - 00;22;35;28

So you have to be curious and, ask questions, challenge things, don't take things for granted. So it takes people with, relatively inquisitive and sometimes I would say, even skeptical mind to not take, you know, things, at face value and dig a little bit further down, play scenarios in your head. as an architect, you're constantly looking at risk, technical risk, programmatic risk by saying, well, yes, we can do that.

00;22;36;01 - 00;23;18;10

But what about tomorrow? What about next week? What about next month? So play scenarios in your head about what could happen and going down mentally into various branches, rather than just saying, well, I'm doing my job for today and I don't care about what's going to happen next week. Another thing as as if you are in a not too small team, the architect has to play the role of communicator, communicate between various stakeholders inside and outside of the organization, and this becomes very important if your organization is split into multiple side, it's even more important if two sides are not on the same time zone and populated with people who are not in the

00;23;18;10 - 00;23;43;29

same country. So the architect will play a role of building this internal company or team culture, be a sort of, node of communication for technical issues. Don't take somebody who is the ultimate shy guy who cannot speak to, any people. You need somebody who is not afraid to stand up and communicate and listen to, issues from various parts and reconcile them.

00;23;43;29 - 00;24;08;02

And I suggest to, people also to keep their hands in the code, once in a while, participate. So it's not I'm the architect of the coders. I'm an architect six hours a day, and I'm coding the other 2 or 3 hours. So that you better understand what's going on. You feel the issues that, your fellow developers feel the same way.

00;24;08;04 - 00;24;30;16

Don't sit your architect in another building. You know, the architect is embedded with the rest of the software developer. Unless it's a humongous project with 200 people and you have a little segment of your building with 2 or 3 architects there, but in general, it's not a good idea to separate them from the rest of the team. The best architect also have, good knowledge of the domain.

00;24;30;18 - 00;24;53;22

To give you an, personal example, I was the chief architect for the Canadian air traffic control system, but I've worked before on air defence system and I'm. I'm my private pilot. I flew in, in Europe, and I flew, airplanes in North America. Well, I understood that, you know, a little bit more. What air traffic control was doing than the average Ada or C++ developer.

00;24;54;00 - 00;25;16;10

So that that helps. It's not an absolute must, but that helps a lot to understand the problem domain. Unless you're paired with, product managers or product owner who can compensate your lack of knowledge of the domain. But then you need to have a relatively good dialog. Thank you. Philippe. It's very interesting to hear about the different aspects of the role of an architect.

00;25;16;12 - 00;28;45;08

And I finally want to thank you very much, Philippe, for taking the time to talk to me about software architecture. I'm sure our students will really enjoy this podcast. Thank you very much. Very nice. Thank you Casper.