EPISODE 24

[INTRODUCTION]

[0:00:00.0] JM: Software architecture addresses the challenge of communicating and navigating large and complex systems to stakeholders, both technical and non-technical. Over the years, software architecture has gone in and out of fashion. Today we discuss why software architecture is important, what it means to have software architecture and how to properly structure teams and incorporate architecture.

Today's shows guest hosted by David Curry. David sits down with Simon Brown to discuss the importance of having a common language for software systems. Simon is an independent consultant specializing in software architecture. He's the author of S*oftware Architecture for Developers* and the founder of Structurizr.

If you're interested in hosting a show yourself like David, who is guest hosting the show. It's his second show he's guest hosted. Check out softwareengineeringdaily.com/host or you can email jeff@softwareengineeringdaily.com, that's me, to find out more about hosting a show. We'd like to get more external voices and turn Software Engineering Daily into more of a media channel with different voices, different types of content, and I'd love to hear your ideas. Send me an email, jeff@softwareengineeringdaily.com.

[SPONSOR MESSAGE]

[0:01:29.0] JM: For years when I started building a new app, I would use MongoDB. Now, I use MongoDB Atlas. MongoDB Atlas is the easiest way to use MongoDB in the cloud. It's never been easier to hit the ground running. MongoDB Atlas is the only database as a service from the engineers who built MongoDB. The dashboard is simple and intuitive, but it provides all the functionality that you need. The customer service is staffed by people who can respond to your technical questions about Mongo.

With continuous backup, VPC peering, monitoring, and security features, MongoDB Atlas gives you everything you need from MongoDB in an easy to use service, and you could forget about

needing to patch your Mongo instances and keep it up-to-date, because Atlas automatically updates its version.

Check out mongo.db.com/sedaily to get started with MongoDB Atlas and get $10 credit for free. Even if you're already running MongoDB in the cloud, Atlas makes migrating your deployment from another cloud service provider trivial with its live import feature.

Get started with a free three node replica set, no credit cards required. As an exclusive offer for Software Engineering Daily listeners, use code "sedaily" for $10 credit when you're ready to scale up. Go to mongodb.com/sedaily to check it out, and thanks to MongoDB for being a repeat sponsor of Software Engineering Daily. It means a whole lot to us.

[INTERVIEW]

[0:03:26.2] DC: Simon is an independent consultant specializing in software architecture. He's the author of *Software Architecture for Developers* and founder of Structurizr. Simone, welcome to Software Engineering Daily.

[0:03:38.8] SB: Hi, David.

[0:03:38.7] DC: Just so we can kind of set the frame about software architecture, can you tell me from your perspective what is software architecture.

[0:03:48.6] SB: I guess software architecture means so many different things that so many different people. From my perspective, it's about a couple of things really, it's about the structure for what you're building. If you have a software system, you can break it down into modules or components or services or subsystems, it's about structure and build the blocks and relationships. It's also about the significant decisions on, and this is something Grady Booch talks about a lot.

Imagine, you have a set of requirements, you were asked to build a software system. You have to go make a bunch of significant decisions. Really, it's that collection of decisions that's also included in architecture.

[0:04:26.3] DC: Yeah, that's a good broad overview of some of the things we will discuss today. Specifically, what problems does software architecture address when we look at those separate components and breaking things down? What are we trying to address with the model of software architecture?

[0:04:50.9] SB: I guess the main one is just chaos and avoiding chaos. I'm sure that you've probably seen this yourself. You are going to or a team is doing and if that team doesn't have any kind of obvious technical leadership and, therefore, obvious software architecture, the system they can be building often seem like big balls of mud or they're just horribly structured. They're not hard. They're not easy to maintain, hard to deploy. They got security issues, performance issues, that sort of stuff.

I guess, in a nutshell, architecture for me when we apply it to a team perspective is about introducing technical leadership and avoiding chaos basically so we could create software systems and software projects that work.

[0:05:35.0] DC: Yeah, within that, we're looking at better performance, better security, I guess structuring projects so they can be easier maintained. Are there other common software architecture problems?

I guess if we think about poorly, poorly, architected software, what may be some of the other problems that exist in that?

[0:06:04.1] SB: In terms of code basis, I guess a lot of the problems you see come down to poor structure. It's funny, we often have this discussion at the moment whenever we talk about microservices versus monoliths. I've been quote to saying that if people can build modular, well-structured monoliths properly, then they've got no hope for doing microservices.

There's definitely some truth in that because in order to create a highly modular, well-structured monolith, you do need to do some work around decomposition and you do need to do some work around design thinking to figure out a way or boundaries are. What sort of stuff you can to encapsulate, what sort of things you want to make publicly accessible in your codebase.

I often see people kind of skipping that step, and they just charge headlong into a software system. Let's imagine we're building a [inaudible 0:06:56.9] web app. They open up the books, they find, "Oh, layered architecture. That sounds like something we should do," and they just charge headlong into a layered architecture. Keep all their clusters public. That's fine to start with of course, but then after a few months or a few years you end up getting this horrible code system because there's just no integrity there, no structure.

I think that the structural issues are probably one of the most common ones I see and I guess they're the easiest to spot as well in many ways.

[0:07:29.9] DC: That makes me think about you start on a project, everybody's pretty excited. You may jot down some feature requirements or specifications and then everybody sort of jumps right in without really thinking about the overall direction and considerations to integrations of other systems and things. Yeah, that makes a lot of sense.

[0:07:55.0] SB: What's interesting here is if you go back 20 years, we used to have dedicated people called software architects who would do that job, and then Agile came around and a lot of teams dropped that kind of ivory towered dictatorship approach which is a good thing. That's not how we want our teams to be run.

They often didn't replace the technical leadership role within the thing, and I think that's, to some degree anyway, got us to where we are today. I do see lots of teams who are just in chaos. I see lots of teams who don't have explicit people looking after the architecture, whether that's one or many people. Yeah, it's a slippery slope unfortunately.

[0:08:35.8] DC: Are you saying that modern teams — It's better for modern teams not to have a specific architecture role and whether developers play that role in architecture.

[0:08:47.6] SB: I would say almost the total opposite. From my perspective, every software team needs technical leadership. That's the first easy thing we can talk about. More fundamentally, if you look at how that technical leadership role is kind of implemented and done

on project teams, I think, explicitly having the role is important. Making sure there are one or more people that are explicitly doing that role, and it can be one or more people.

I think most teams, to be honest, work better when it is just a single person kind of taking that leadership role, but if you have a team of more experienced people, they've worked together before, they know each other, then you can start to do power architecting and kind of showing the role out amongst the rest of the development team. That's not something I see working well a lot, to be honest.

[0:09:39.3] DC: For a develop who's focused on, say, one aspect or one component of the system, what can he or she do to think and work more architecturally?

[0:09:51.7] SB: Certainly, interesting question. The easiest way is just to work with the architect. I'm a big fan of making the role collaborative and dragging people in to architecture discussions where necessary. Of course, if you have a large enough system, you're not going to be the expert on all parts of that system, so it is often useful to gather people along with you inside meetings and discussions and stuff.

Also, I'm making the architecture — Flip the question around. If I'm an architect, how can I help my developers become more architecturally aware? Making a lot of the architecture information available is a really simple thing to do. Also, I'm explaining the rational for those significant decisions. Don't just use X version of this framework. Let's explain why. Maybe there are some specific principles that we want to adopt or some specific constraints that we have, for example, around support and maintenance of costs. Yeah, it's really about opening up that information stream.

[0:10:51.0] DC: I've read a blog post of yours from 2016, title; *Agile Software Architecture Documentation.* You also talked about why there's no conflict between Agile and architecture in your book. How does software architecture fit into the Agile environment?

[0:11:09.6] SB: You could look up for ways at looking at architecture within Agile environments. If you look at most of the definitions of Agile, they all tend to revolve around embracing change and moving fast and releasing often and that sort of stuff.

From a process perspective, we want to really create an architecture that supports and allows teams to move fast. For me, that's where the whole structure thing comes back into play again. If you're looking for agility in your team, if you're looking for agility in your software system, you tend to find that systems that are well-structured tend to have more agility. In other words, they're easier to change. That's why we're seeing lots of discussion around microservices architectures because, of course, if you have all of these small independent things then you can change these small independent things separately and individually.

A microservices architecture by definition gives you lots of agility. There are tradeoffs, of course. That's kind of one aspect, is if you want to move fast, make sure you create an architecture that allows you to move fast. That's the design thinking decomposition. That's something that you have to think about. You don't get that for free unfortunately.

In terms of the process perspective, 20 years ago, we're doing lots of all sorts of projects and it was all about upfront design and fixing all of the design issues upfront. Then we kind of flipped to the other extreme where people were saying, "Yay! We can just evolve the architecture, do a merchant design and therefore we don't need to do any upfront thinking. That's also foolish.

There's a nice happy midpoint in the middle somewhere. The obvious question here is, "How much upfront design should you do?" For me, the answer is very simply just enough. Which is a fantastic mantra but it's a completely useless piece of advice.

When I talk about doing just enough upfront design, and this is specifically applicable to Agile teams, of course, if I'm starting out with a blank sheet of paper and I'm designing a software system, I want to do enough upfront thinking either individually or as a pair, as a team, whatever, to understand the significant structured elements. In real terms, for me, this means going down to doing component level design or maybe service level design if you're doing microservices.

Being able to communicate those ideas and their architectural intent with the rest of your team, and that's where a good set of simple diagrams come into play. It's also about de-risking the

build. Essentially, this is about identifying and mitigating your highest priority risks and then doing some concrete experiments to prove out that stuff if necessary.

That's in a nutshell how architecture and Agile fit together from my perspective anyway.

[SPONSOR MESSAGE]

[0:13:58.0] JM: For more than 30 years, DNS has been one of the fundamental protocols of the internet. Yet, despite its accepted importance, it has never quite gotten the due that it deserves. Today's dynamic applications, hybrid clouds and volatile internet, demand that you rethink the strategic value and importance of your DNS choices.

Oracle Dyn provides DNS that is as dynamic and intelligent as your applications. Dyn DNS gets your users to the right cloud service, the right CDN, or the right datacenter using intelligent response to steer traffic based on business policies as well as real time internet conditions, like the security and the performance of the network path.

Dyn maps all internet pathways every 24 seconds via more than 500 million traceroutes. This is the equivalent of seven light years of distance, or 1.7 billion times around the circumference of the earth. With over 10 years of experience supporting the likes of Netflix, Twitter, Zappos, Etsy, and Salesforce, Dyn can scale to meet the demand of the largest web applications.

Get started with a free 30-day trial for your application by going to dyn.com/sedaily. After the free trial, Dyn's developer plans start at just $7 a month for world-class DNS. Rethink DNS, go to dyn.com/sedaily to learn more and get your free trial of Dyn DNS.

[INTERVIEW CONTINUED]

[0:15:58.0] DC: When you're starting out — Let's say you're starting out on the new project in an Agile team and you're creating your architecture. How far ahead are you thinking or the decisions that go into the current architecture for this particular sprint? How far ahead are you thinking and how much of that sort of thought of scalability goes into the current design or the

current architecture, because I know we're talking about just enough but I guess it's a balance between how much you're putting forth with the thought that this is the direction we're going in.

[0:16:37.6] SB: Yeah, and it's also tricky of course because you never quite know how something is going to evolve. Imagine you're in a startup and you're building some software as a service product. Because you're a startup and you probably have no money, you probably want to get something out there quickly, so this is the whole minimum viable product thing. That minimum viable product is probably not going to scale for a million concurrent users. That's one extreme.

The other extreme is you say, "Right, I know my startups can be super successful, so I want to put scalability in for a million concurrent users right at the start." There's a lot of engineer effort in doing that, isn't there?

[0:17:14.0] DC: Yeah. That's right.

[0:17:15.2] SB: Potentially, a lot of wasteful cost and effort if your startup fails. That is one of the big problems that we kind of talk about. It's not having that crystal ball to see far enough into the future. This is where you need to start using design tricks and tactics and architectural principles to build in flexibility into architecture so that you know that parts of your architecture can change if you want to.
Again, this is something that microservices architectures are relatively good at of course.

In terms of how far we're looking into the future then, I actually had this certain discussion with a team I worked with a few weeks ago, and we were talking about if you're starting up from scratch on an Agile project, you only really have the requirements of that sprint to work with, and I kind of question that. I said, "Well, somebody somewhere must have the "full set" of requirements, so I'm going to use full set in air quotes here. The full set of requirements even at a higher level or the business strategy or something about what this thing is going to need to do in the future, because they've obviously taken the requirements — The make-up of the sprint. It's a subset of the full backlog, essentially.

My answer was very simply the product owners and the business strategy people and the business sponsors. Ultimately, the people with the vision for this system need to have some involvement early or vice versa of course. The architecture people that the people designing the software need to have some early involvement with those sponsors so that they have a general idea of where the product is going to go.

I think a lot of teams don't do that. They seem to be focused very much in a kind of sprint sandbox, and that's fine, but I think you do need some forward-looking vision as well.

[0:19:09.2] DC: Again, when you were talking earlier having a dedicated kind of technical lead, this is probably another good job function for them to correspond with the other stakeholders as well as be an expert on the sort of software engineering tips and tricks you would implement to think for the future within the current sprint.

[0:19:38.4] SB: It's interesting. I've seen some resistance from some teams because they don't like the term software architect. Scott Ambler, I think it is, has another really nice name for this, and he calls it an architecture owner. Sometimes just reframing the architect role in terms of an architect owner is a really really simple trick, because then you've got the architecture owner working alongside, say, the product owner, and these people whether it's two people, one person, group of people, are together looking at the vision of where that thing is going.

[0:20:15.2] DC: I want to switch gears just a little bit, talk about your C4 model. You're the creator of the C4 architecture model. What is the C4 model and what problem does it solve?

[0:20:27.7] SB: One of the things I do a lot with teams around the world is I do software architecture workshops in Carters, and I've been running this for about 10 years now. The premise is very very simple, you get a group of people together, you break them up into smaller groups of, say, two to four people and you give them a two-page handout requirements. You say, "Right, what I'd like you to do in your groups is to go away and design a software solution for these requirements and draw some pictures to somehow visualize your architecture."

I gets about 1-1/2 hours to do this. There are really, really simple set of requirements. The actual design is super simple. The hardest thing about the whole exercise is actually visualizing the

solution. I've got 15 or 20 gigabytes of photos on my laptop of the various types of diagrams that people have tried to draw to describe their solutions, they're just crazy. Literally, every type of diagram you could possibly imagine I've probably got photos or somewhere. We have UML, of course, in the industry. I polled people at conferences and workshops around the world and I kind of asked something, "Who's using UML here?"

UML is definitely on the decline. I'm seeing a lot more teams now who don't use UML at all within their teams. In fact, I've met a lot of people recently who just plain don't know UML. Of course, in these diagrams they get during my workshops, there's very very little UML. In fact, even with teams who claim they do use UML in their data jobs, these same teams typically don't draw UML architecture pictures.

My C4 model is basically — This is a very long-winded way of answering the question by the way. The C4 model is basically a very simple set of hierarchical diagrams to describe a static structure. In other words, software architecture. Imagine you're building a system. A simple way to explain that system to people is to have a set of hierarchical diagrams much like a set of Google Maps.

Picture one is called a system context diagram. Basically, what you do is you draw a box in the center that represents your system and you draw a bunch of other boxes around it, and these boxes represent the people, the users, the actors, personas who interact with your system and the other systems that your system interacts with. Really, it's just a very very simple high-level map.

If you were to have a set of Google Maps, basically what you do is you want to kind of zoom in. You want to do the pinch to zoom in movement on to that system to kind of see what's inside it. Once you zoom into the system boundary, you get down to level two. This is what I call containers. This is a containers diagram.

What I mean by container, and it's a little bit unfortunate that Docker became popular, to be honest, because when I say container, I'm not referring to a Docker container. What I mean by a container is basically something like an application that runs your code or a data store, like a database, or a VAR system or something like this.

The container diagram basically shows applications and data stores. Maybe it shows mobile app talking to a web app, talking to a database. Nice and high-level, some simple text about responsibilities on there, some tech choices and protocols. If we want to know more about, say, the internals of the web application, we do the pinch zoom in movement once again and then we get to see inside it. Then you come up with a component diagram for a particular container.

This really shows you the internal structure of the thing that you're building in terms of components. You're going to zoom into a component and then you see the code inside it. It's a very simple sort of hierarchical diagrams to describe static structure essentially.

[0:24:08.0] DC: I imagine that each of these diagrams can communicate the same project to different stakeholders of different technical abilities.

[0:24:17.1] SB: Yeah, indeed. The context diagram is very useful for all audiences because it doesn't really have any technology on there, so it's good for business sponsors, product owners, developers to test these operations and so on. The container diagram, it's got a bit more technical stuff on there, so it's good developers and architects. It's also useful for operational-learned support staff. Here's the stuff that you need to look after. Don't worry about what's inside it. Of course, as you go deeper components and classes and code, that's really targeted much more on developers.

[0:24:46.7] DC: Why is having this type of documentation important?

[0:24:51.0] SB: The way that I'd like to think about this sort of documentation is that it's a way to help developers explore and navigate large and complex software systems. Whenever I go and see organizations around the world, most of them unfortunately don't have much documentation about their software systems. This kind of comes back to the agility thing again.

Imagine you have a software system and you get this big change request in or this new feature that needs to be added and you need to do some architecture refactoring. People often say that architecture refactoring is quite tricky because you never quite know what you're changing. My

view is that architecture refactoring shouldn't be tricky at all because it's just a simply transformation.

The thing that complicates architecture refactoring is that most teams, I think, don't have a good view of what their starting point is, and that's really where this documentation comes back into play. It's a nice high-level way to think about a software system and the structures in the software system, and it allows you to do things like architecture improvements or for change impacts and that sort of thing.

[0:25:54.5] DC: Is part of your goal with the C4 model to make kind of an industry standard for documentation?

[0:26:03.2] SB: I've been asked this question a few times, and I'm not entirely sure about industry standard, but I certainly do want to improve the way that people talk about describe and visualize their software systems. I think it's a bit too early to kind of go the industry standard root yet. A lot of people do find it very very useful, because it's just very very clean, very simple.

[0:26:27.0] DC: Documentation isn't the most desirable test for developers.

[0:26:31.9] SB: No.

[0:26:33.0] DC: From your perspective, why is there resistance to creating documentation?

[0:26:37.0] SB: It's just boring, basically. It's boring and it gets people — It's the honest answer, isn't it? In terms of software documentations, again, if you go back 10 or 20 years, you used to see teams building lots of software architecture documentation, SADs, they were often called, because SAD, that's how it makes you feel.

These things typically had good content in them, but the implementation and presentation wasn't ideal. Sometimes, these documents, hundreds of hundreds of pages. One of the things I kind of encourage teams do is to write documentation, but keep it nice and lean a light weight, a small number of pages versus hundreds.

The analogy I use here is like a travel guidebook. Imagine, you're going on vacation somewhere, you run through the airport, you pick up a travel guidebook for your destination. It's got a bunch of maps in the back somewhere. The maps help you navigate in an unfamiliar environment, that's all the diagrams of course. It's got information about sights and point of interest and history and culture and all the practical stuff, and that's a really nice simple way to think about software documentation; apply those same sort of sections to your documentation essentially, then you can make it more exciting from a technical perspective by not using a Word and SharePoint, but by using things Markdown and AsciiDoc.

[0:27:58.5] DC: A common complaint of documentation tends to be the upkeep of it. How could teams efficiently keep the architecture documentation up-to-date?

[0:28:08.7] SB: Yeah, you've got a couple of options, I guess. From a process perspective, it's no more complex than if you have a definition of done for your tasks, or your features, or whatever it is. whatever it is. You add a line to the bottom of that definition of [inaudible] that says, "Have you updated the architecture documentation?" Then it's just a really short, maybe zero-minute task, 10 minutes, 30 minutes, whatever, to update the documentation for the small parts of the system that you've just touched.

From a tooling perspective, one of the things I've created is a set of tooling around the C4 model called Structurizr. It's a kind of part open-source, part commercial thing. One of the things Structurizr lets you do is it lets you create architect in models using code. One of the things you can do, of course, is you can plug this whole mechanisms to your build environment and then when these sort of programmers, essentially, that you're writing, when these programs run, they have the ability to kind of scape information from your live codebase and use that as a way to update diagrams and documentation as well. Again, there's a bunch of techniques and tools and approaches there.

[0:29:18.9] DC: You are the founder of the product Structurizr. Is that what Structurizr does?

[0:29:24.1] SB: Yes. Structurizr is a collection of tooling that helps you visualize document and explore yourselves for architecture. It's kind of part open-source, part commercial. The open-

source thing currently consists of two open-source libraries, one for Java, one for C#, and these are really small classes that implement the C4 model.

Are you familiar with things like PlantUML and web sequence diagrams where you write texts to create diagrams? Yeah. It's the same kind of thing. What you're doing here is you're using text to create your architecture diagrams, but the text you're writing is actually Java code. If you were to look inside the structure or a Java open-source library, there are basically a bunch of classes in there representing people, software systems, containers, components, and all you do is you kind of create instances of these things and you wire them together.

You're really recreating a kind of simple model of your software system. There are also some component finder things in there that you can point at a codebase and you can tell it to go and find components of various types. That's kind of the open-source thing. Then I built a software as a service called Structurizr. Basically, that's a set of tooling that visualizes these models as web-based diagrams essentially. There's also a free plan that people can use as well.

[SPONSOR MESSAGE]

[0:30:48.0] JM: Blockchains are a fundamental breakthrough in computer science and consensus Academy is the place to learn about block chains. The Consensus Academy developer program is a free, highly selective, and carefully designed 10-week online curriculum where you will immerse yourself in block chain development and earn a consensus block chain certification. By completing the program, you will be eligible for immediate hire by Consensus; a leader in the block chain space focused on the Ethereum platform.

If you want to learn about block chains and become a developer for block chain technology, check out the Consensus Academy by going to softwareengineering daily.com/blockchain. The graduation ceremony is a spectacular all-expenses-paid trip to Dubai where you will meet block chain developers working on real-world solutions.

Dubai has announced ambitious plans to become the first city to run on block chains by 2020. If you like the idea of immersing yourself in block chain technology, graduating, and going to Dubai, check out softwareengineering daily.com/blockchain. Build your new career on the block

chain with the Consensus Academy developer program. Applications are open from now until July 1st, so you want to apply soon. For more details and to apply now, go to softwareengineering dailywe.com/blockchain.

To learn more about Consensus and Ethereum, please visit consensus.net, and sign up for the consensus weekly newsletter for everything you need to know about the block chain space. Thanks to Consensus being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:32:44.9] DC: Can the result of running Structurizr against your code help you improve the architecture in any way?

[0:32:52.1] SB: Yeah, definitely so. One of the features I've built in to the Structurizr software as a service is a set of exploration. It's a set of JavaScript D3 visualizations essentially that give you different perspectives on to a model. I've noticed this even with my own software. Once you have a model of your software system and that model is created automatically from the code, you can start to look at the model and analyze the model to find relationships and things that that make sense.

A really simple example is if you take any, say, Java or C# codebase, if you were to do a query upon that codebase to say, "I would like to find all of the public types that are only used once." That's a really really simple query that can help you go and identify classes that should be made more restrictive, so package protected, or private or internal in C#. Even things like this, having a model of your software system, being able to query in different ways can kind of help you improve your architecture which can hopefully eventually lead to a much much better well-structured architecture.

[0:33:59.0] DC: Are there any other languages outside of Java, and I believe you said C#, or .NET that Structurizr supports.

[0:34:08.1] SB: At the moment, the client libraries operates Java and C#. I think someone's building a Kotlin wrapper on top of the Java version. I think someone else is building a groovy

kind of DSL wrapper on top of the Java version. A friend of mine is starting a Python version, but that's at the moment.

[0:34:26.3] DC: People can surely participate in the open-source part of the product creating wrappers for different languages.

[0:34:33.8] SB: Yeah, indeed. Yeah. I've actually got some users who have a software system written in COC++, and what they've actually done is they've built some component finder codes in Java for the Java version of the library that basically passes COC++. Again, there are some other opportunities here.

[0:34:55.0] DC: Can you talk a little bit more about Structurizr's documentation, visualization component?

[0:35:00.4] SB: The visualization component, basically, in the client library, what you do is you create your model and then you create a bunch of views that map on to the C4 diagram, so the systems context diagram, the container diagram, the component diagram, and what the Structurizr's software as a service does, it basically creates those pictures for you. There's a set of boxes and lines, diagrams, very simple, very constrained notation. You can change the shapes and the colors and the size of the fonts and stuff, but it's a very very constrained notation.

That's essentially it for the visualization part. It's worth mentioning that in the open-source libraries, you can also export the views that you create into a format site PlantUML. You're not tied into using the software as a service.

In terms of the documentation, basically, what you do is you add Markdown or AsciiDoc content to the software architecture model that you create in the client library and then the Structurizr software as a service will render that for you.

A nice thing about having the documentation and the diagrams all in one place is it's very very easy to embed the life diagrams in the resulting HTML pages. Again, all of these stuff lives together and it all stays up-to-date at the same rate.

[0:36:15.7] DC: Looking back at our conversation about Agile teams and architecture, what's the ideal way of integrating Structurizr into the software development life cycle of, say, an Agile methodology?

[0:36:36.2] SB: What I tend doing and, really, my personal preference for all of these is I would start out doing architecture and design on a whiteboard or a large piece of paper. The reason I do that I because it's just a much bigger space. It's much easier to do collaborative architecture, pair architecture on a whiteboard. It's also easy to erase whiteboards and change your mind and all that sort of stuff.

I definitely use a whiteboard for the early stages of doing architecture design, and then when I had some code, I would start using tools like Structurizr to start formalizing g those visualizations and that documentation.You can also use Structurizr — I don't generally do that.

[0:37:17.6] SB: Yeah, it's a lot easier if you can kind of jot stuff down and erase and kind of wrap you mind around what you want to do, yeah. Looking at the future, how will software architecture change, say, over the next 5 to 10 years from your perspective?

[0:37:34.0] DC: Oh, it's an interesting question. I think it might come back into fashion again.

[0:37:39.6] SB: Right. All thing, it becomes a cycle.

[0:37:43.5] DC: Yeah, indeed. If I look back even maybe 10 years, I saw a lot of resistance against architecture because it's seen as this horrible ivory tower thing, it's very dictatorship-driven, forcing guidelines and principles and constraining what people could do. Then Agile came along and kind of killed all that stuff.

I think people have realized that teams do need technical leadership, and I think that's why there's a resurgence in people wanting to think more seriously about architecture. Whether my C4 standard becomes the de facto way to do architecture, who knows? Maybe there's a bunch of other architectures that might come in the future. I think one of the things we do need to do as an industry is really create common vocabulary, and that's one of the things the C4 model tries

to do. You've probably seen this yourself. You go and seat with a team and you see them having a discussion. They'll use words like component in very, very different ways.

I think one of the things we need to do as an industry is figure out what our vocabulary is. That's something I'd like to see happen in the 5 to 10 years. I think we may also see people starting to use some of the tooling that they perhaps used to use a while ago. Things like modeling tools have completely gone out of fashion, but there's definitely some value in having a model of your software architecture because you can do things like query and asking questions and stuff.

Also, things like static analysis tools never really took off as much as they should have done. I'd certainly like to see those sort of things taking much more importance. From a process perspective, I'd like to see the software architecture becoming much more formalized. It's funny, whenever you go to your organizations and you say, "What do your developers do?" They can tell you, "You're awesome. What does your scrum master do," they can tell you. You ask them, "What does your architects do," and they're kind of stumped.

Again, there's no single formal definition of architecture as well, and maybe that's something we need to do in the 5 to 10 years.

[0:39:43.1] SB: One of the things that I've always wanted to see is just sort of some collection or architecture documentation for solving different problems. A lot of times, it's a lot of trial and error between different teams and things. A lot of things are sort of open-source or open information but it doesn't seem that system planning is one of those, and I believe it'd be extremely helpful for different teams to look at the way people have solved problems from an architectural standpoint and maybe use some of those elements into their own projects.

[0:40:26.4] DC: Also, having a — Standardize is probably the wrong word, but a more or less consistent way of doing this as well. You have to see people documenting their systems, but they're documenting them in very very different ways, which adds as an extra layer of complexity in understanding what people do and, of course, the problems that they're solving.

[0:40:46.7] SB: Right now, there are so many cloud service options out there and cloud adoption rates continue to grow. Do you see this having an effect on the future of architecture?

[0:40:57.0] DC: Yes, definitely. You can kind of see this already through things on microservices and Docker, and also the new cloud native buzzword that people are throwing around, building your apps so that they are “ready for the cloud”.

Yeah, that’s definitely having an impact because if you’re creating cloud native applications, you can’t rely on persisting stuff to local disc anymore because your server instance, your warehouse instance might be moved at runtime. You could have to do things like log aggregation across many service. Yeah, definitely, cloud is having an impact in the things we do as architects these days. I’d also like to see platform as a service become more popular as well.

[0:41:40.4] DC: Simon, it’s been a pleasure speaking with you today, I really appreciate you coming on the show, and thanks for being a guest.

[0:41:46.0] SB: No problem at all. Thank you.

[END OF INTERVIEW]

[0:41:48.0] JM: Gatsby is a platform for improving your conversions. Let’s say that you’re selling cat food on your website. You’re an e-commerce merchant for cat food and I know that there’s a lot of these because I have cats. I spend a lot of time shopping for cat food, and whenever I hit a website that I haven’t been to before, or that’s trying to sell me cat food, the only way that I'm going to stay on that site for longer than two seconds is if something catches my eye, because I have a short attention span, probably most of you do too, and if something's not interesting I'm just going to click to Facebook or Twitter or whatever.

Gatsby provides you with a better UI, a better way to run contests, and a better way to showcase discounts. It's a conversion platform. What that means is that there are promotional widgets and contest widgets that you can easily integrate with your platform. You can have a contest that says, “Win a free box of cat food if you click this link and get 30% off your first Repurchase of you click this link.”

This is useful if you're selling cat food, if you're selling a database, really, anything, and it's developer friendly. They've got an API. You can go to thinkgatsby.com and find out more about that API. Gatby is great because it helps to increase your sales, it helps you capture data, and it helps you integrate with social media platforms in a way that provides a nice UX. You've seen those e-commerce plug-ins that don't look very nice, that don't really improve anything. They don't help anyone. Get started with Gatsby, go to thinkgatsby.com.

[END]