PUBLIC

Podcast: Process Transformers

Episode 31: Healthy Software at Scale: Blueprints for Adaptable Architecture | Feat. Chris Richardson

Fig. 1 – Cover art of Process Transformers podcast
Figure description – A square cover image with decorative geometric designs representing workflows and connections with blocks, circles, and arrows in varying shades of blue. The title “Process Transformers” is featured at the top of the image, and logos for “SAP” and “SAP Signavio” are featured underneath the image

Transcript

Lukas Egger: Hello, and welcome to *Process Transformers*, the podcast that talks about business transformation at the intersection of processes and AI. For those of you who have listened before, welcome back. And if you're new to the show, thanks for tuning in. My name is Lukas Egger. I'm the head of innovation at SAP Signavio. I'll be your host for today's episode titled "*Healthy Software at Scale: Blueprints for Adaptable Architecture*". And I'm thrilled to introduce today's guest, Chris Richardson. Chris is a renowned software architecture consultant and creator of microservices.io. He's also the best-selling author of the book *Microservices Patterns*. Chris, welcome to the show.

Chris Richardson: Well, hi. It's good to be talking to you. Thanks for having me on the show.

Lukas Egger: Of course. Chris let's start with a simple but also very hard question. Why is good software architecture important?

Chris Richardson: It is important because in order for the software to have certain properties. Or a more common way of putting it is in order for software to satisfy non-functional requirements, it needs to have the right architecture. And there's obvious non-functional requirements or properties like scalability and availability, those framework from architecture, architecture affects more than just the runtime characteristics of the application, but also the development time characteristics. So it's interesting, if you think about it, a software system has to satisfy two constituencies. One is the users, the classic thing. The other one is the development teams. And the reason for that is the design decisions that software developers make today actually determine how easy it will be to change software in the future. A lot of things really don't work like that in design, right? What you do today doesn't make it harder to do something tomorrow, but with software, you can very easily do that. And that actually all ties back to architecture.

Lukas Egger: I think you were touching on something that is at the core of what people are interested in, right? The ability to change. Now, architecture mostly is connected with, in a lot of people's heads, as technical debt. So how does one think about architecture in terms of being able to adapt? Like, are there clear guidelines or patterns? Like, what is it that sets apart architecture that can enable an organization to change quickly?

Chris Richardson: Yeah. And I mean, that's a good question. I think it's got a; it's actually quite a deep question in a way. So, the way I like to introduce this topic is as follows, right? So, you know, the world that we live in is volatile, it's uncertain, it's complex, it's ambiguous, which I guess you could condense down into, it's crazy, right? It's very unpredictable. And in order for businesses to thrive in that world, they actually need to be very agile, very nimble. And given that businesses are powered by software, that means that software delivery needs to be agile and nimble. It has to be able to readily adapt to change. And another way of looking at the same thing is that the heart of software development is a feedback loop. We build something, we implement some features, we deploy it into production. And as soon as we do that, we get validation of the technical decisions that we made. What worked well, what bad decisions we make, and we also get feedback from the users, whether we've actually met their needs. In today's crazy volatile world, we need a very fast feedback loop so we can constantly learn about how to build our software and also what the users need so we can constantly course correct. We need an architecture that actually facilitates short feedback loops and adaptability.

Lukas Egger: Now, the goals you're laying out, there's no one out there who wouldn't like that, right? They want code that is a pleasure to maintain, that works for the users, that scales, that does all these things. But again, mostly when we talk about architecture, it sounds like a bit like a liability on the one hand. And on the other hand, it feels like it's very driven by a couple of flashy names or concepts, right? There is monoliths, there is microservices, there is cloud native, there is streaming. And for a lot of people, it's hard to cut through all of that. So maybe you could help disentangle a little bit like what are the most important concepts to maybe get a bit closer to what it truly means to get to architecture.

Chris Richardson: Well, I think there's a few. One of them, which is ironically an absolutely ancient concept, is that of loose design time coupling. That is a concept that first was discussed by, or he might not use these words exactly, but David Parnas talked about this as long ago as 1972. And it's the whole idea of modularity and encapsulation. So, when you make a change to your software, like you're implementing a feature or fixing a bug, the scope of that change should be as small as possible, right? Like ideally, we could be down to a single line of code in a single class or a class or a package or if we're going up the design hierarchy it could be an application or it could be a microservice right but it should be as far down and as small as possible and that's really a property property of loose design time coupling and cohesion i change something here I don't have to change it in five million places, And the reason that matters is especially problematic when change spans team boundaries, right? So, I need to make a change to my software that requires you to make a change to your software. Right? Because of tight coupling, we have to go have a meeting. And everyone's schedule is busy because they're actually having way too many meetings aligning and coordinating and so on. And so, it really slows us down. Whereas if the discussion was within my team, we did that at our, say, daily stand-up, we're getting very fast decision-making and rapid iteration. I think most enterprises actually suffer from a tightly design time coupled architecture because they are constantly aligning and coordinating and getting anything done requires a very large room packed full of people on a regular basis. And I don't know if that happens at your organization, but that to me is that is enterprise software development.

Lukas Egger: Them I’m sure we are all guilty of that and there is well i would say most of the time concepts veer off course it's not because of malicious intent it's because of good intentions right just like the road to hell is paved with good intentions so where do you think that comes from because it's not given that people would not adhere to loosely coupled designs right What draws people in to build something that's hard to disentangle and maintain? What do you think are the motivations?

Chris Richardson: You certainly have to apply a set of design principles, right? Say software elements need to look like, I mean, I think of them as icebergs. Like very small visible surface area that's hiding a much larger implementation. So, there are sort of basic design principles like that. And then I think it requires discipline. Right? Sort of eternal vigilance to actually ensure that your software is constant. Is loosely design time coupled and that people are not taking shortcuts and introducing coupling because there's a deadline to meet. I think organizations actually often just constantly take shortcuts and ignore good design practices because shipping this feature is far more important. Yeah, sometimes, obviously, you kind of have to ship a feature to make revenue. Because without revenue, you can't, quote, survive. And so, there's an aspect of that. But if you are incurring all of this technical debt, then, well, tomorrow, when it comes to make the next change, it's going to take even longer. And it just builds up. And the productivity just spirals down and down and down. Now, you actually have to devote time to this. You actually have to let architects do architecture and make sure the architecture is loosely coupled rather than just have them 100% of the time work on features.

Lukas Egger: I think it's now over 10 years that this famous article came out like software is eating the world. And essentially, it's stipulated that pretty much every company, no matter what industry they're in, they're also in the software industry. They're also building software. And we see that more and more companies want to differentiate. And one part of that story is building the right tools and being able of creating good software. So, what is a good approach or a good pattern so that companies can respect architecture? Because again, what you say makes perfect sense. And I think everybody would want to get on that bandwagon, right? Of not having a terrible time further down the line and want to do it. But very often it's hard to know what good looks like especially when it's hidden behind sometimes arcane looking symbols or something that is not easily.

Chris Richardson: Understood so yeah but you know it's funny it makes me think about everyone should eat healthy work out and drink lots of water, I'm only at like 45% of my prescribed water volume today, right? And so, it has analogies to that. There's a lot of different things. And so we kind of jumped ahead and talked about architecture, but I think it's also useful to take a step back and talk about what I call the success triangle, which says that if an organization wants, shall we say, an agile, nimble IT department, wants to deliver changes rapidly, frequently, and reliably, which is a concept that is called fast flow, then they actually need to have the right development process or way of working which goes by the name of DevOps as defined by the DevOps handbook. So, it's a set of principles and practices for delivering software rapidly, pleasantly, reliably. That's the handbook on how you worked. And then you also need to have the right organizational structure which is essentially a collection of loosely coupled teams that are able to work independently and more specifically the book team topologies defines the organizational structure that you want to follow and I’ve i mean there's a whole level of detail you could skip into but it's sort of in a set work according to devops handbook right and then structure your organization according to team topologies and then you need an architecture that supports that those two things That's where this whole monolithic architecture versus microservice architecture comes into play.

Lukas Egger: That is something that is somehow bubbled up to general conversations, right? Like monoliths versus microservices. And people tend to get quite opinionated about their due diligence and technical depth in terms of, well, if it's a monolith, it has to be like a bad design choice. And if it's microservices, it's automatically the right approach. Can you maybe lay out a little bit the ideas behind it and what design choice is right under which circumstances? So, I kind of wanted to spell the myth that if it's a microservice architecture, it's probably wonderful and everything else is bad.

Chris Richardson: Oh, no, no. I mean, yeah, it's quite the opposite. I mean, you can have a good monolithic architecture, and you can have a bad microservice architecture. And in fact, you almost could say that a bad microservices architecture is far far worse than a bad monolithic architecture because you sort of end up, at worst case, you've got the worst attributes of both. So yeah, so monolithic architecture. I mean, my definition, and there's some variability, but the one I like to use is that the application is structured as a single deployable unit. So, it's kind of one thing, typically in its own code, just single code repository with its own deployment pipeline, which is something I should really emphasize, go into in a little bit more detailed concept of a deployment pipeline. Whereas a microservice architecture, you have functionally decomposed what would otherwise be a monolith into a set of deployable units called services, each one of which typically has its own code repository, although that's not exactly a defining characteristic. But each service does have its own deployment pipeline. And then those services have two important properties one is they're loosely design time coupled right which means that when you work when a team that owns a service works on that service they rarely have to change any other services so as i mentioned loose design time coupling is a pretty fundamental property then each service is independently deployable so it can be tested in isolation without any other services by its deployment pipeline. And then when those tests pass, it can be deployed into production. It's production ready just by testing that one piece of the system. Sort of the reason that matters is because I've already talked about loose design time coupling and the importance of that and how that facilitates fast change. And monoliths can be loosely designed time coupled mostly but there's even though there's a single code base and so that doesn't force lots of teams to coordinate their work under some situations, but then the other aspect of this is having a fast deployment pipeline right so that's it, and this should be a fully automated mechanism it's the fully automated path from a developer's laptop to production. So, the developer does a git push amidst their changes, that triggers the deployment pipeline, it builds it, well, compiles it, tests it, runs security checks, static analysis, so on and so forth, and then deploys it into production. And this is an essential part of this fast feedback loop because part of the set development cycle, right, is I make a change. I want to very quickly deploy it into production and get feedback about that change. And ideally, every developer is deploying at least once per day and getting very fast feedback.

Lukas Egger: I love what you're saying because essentially, from my perspective, what you're advocating for is no matter the architecture, what you really want to encourage is the ability to create feedback loops. Whatever the organization is working on in terms of like functionality that creates value to the user, you should be able to put that out quickly and in a way that limits the interaction between teams. So, it does not slow down each individual team.

Chris Richardson: Yeah, so that, I guess my thoughts that's sort of coming out in a slightly random order here, but so let me tie this back to the monolith versus microservices debate, right? With the monolith, there's a single code base and a single deployment pipeline. And if you have a small number of teams, then probably you are getting fast feedback. Right? The single deployment pipeline can keep up with the rate of changes that are flowing out of the teams. And so, the feedback loop is quite short. But as you add more and more teams, ultimately, that single deployment pipeline will get overloaded. Becomes a bottleneck, and the feedback loop just lengthens. Right? Whereas with the microservice architecture, in multiple deployment pipelines, it basically scales as you add more teams, and you continually get fast feedback. And so, once again, this is a key differentiator between the two. You can get fast feedback at scale with a microservice architecture, whereas with the monolithic architecture, eventually things slow down.

Lukas Egger: Now, that makes perfect sense in terms of like describing the symptoms. And I love that you mentioned, hey, there are already patterns or practices that we know that will help you. It's DevOps, it's loosely coupled teams, and its good architecture, right? I kind of now want to go into a little bit like how can one evaluate, especially from the outside or from the business perspective, whether the organization is at a level that is sufficient for being set up for the future. Because I think a lot of the leaders out there currently think, hey, the world will only get, in your words, crazier, right? Volatility, uncertainty, all these things will only go up or accelerate. Now, what kind of metrics or what kind of symptoms do I need to look at in order to make sure that we're set up for success? So, for instance, for DevOps, we do know that, for instance, Google is very successfully pushing like Dora metrics, right? Where they have like a certain set of KPIs where they want to make sure that, hey, if something goes wrong, how quickly can we recover and so forth. Now, architecture often feels like when you talk to people that there's a style to it, right?

There is like an aesthetical sense to what good looks like. Can you maybe help, especially for people who are maybe not developers, like what are maybe the ideas or the KPIs or the questions someone could have in order to make sure that the organization and the software practices is set up for success?

Chris Richardson: Yeah. Well, I think, you know, a good start, I think there's sort of three categories with metrics. And one category for sure is the Dora metrics. So, lead time, the time from commits to deploy. In a traditional sense, that could be, you know, once a month, once every six months. Right? Because the change I work on to today, if you're only deploying once every six months. Well, that's a lead time of, you know, could be as much as six months. Whereas good metrics are on the order of, I mean, just pick a number. But let's just say under an hour, if not even shorter than that. And then the other one is deployment frequency, which is essentially how many deploys per developer per day. And in a traditional organization, that might be not many, right? Once a week, once a month, so on and so forth. But in a high-performing organization. It's of the order of at least one deployment per developer per day. And then I'm sure people will go, well, that's crazy. How can you test software, right? We have to test it really thoroughly and go much more slowly. But then it turns out that if you are delivering software in this way with proper automated test suites, you actually are doing it in a much more reliable fashion. So it's like moving fast and not breaking things actually go together there are the other reliability metrics which is change failure rate like how often do you have a production outage which is really low in high performing organizations and also mean time to recover if there is a production outage how quickly can you fix it well if you have a super-fast deployment pipeline, you can very readily get a new fix into production. Assuming you put in place good monitoring, which is another kind of architectural property that you need. You can quickly detect the problem. Localize it to a small part of the system, quickly fix it, push, run it through the deployment pipeline. Deploy it into production.

Lukas Egger: And I think it's important also to mention that just because you can doesn't mean you have to. So as an organization, if your customers are maybe not accustomed to change, you don't overburden them with an interface change every day, but you need to have the capability to effectuate change at that frequency. Right, it just means you have the capability does not mean that your software that you're delivering needs to be like constantly moving and shifting around it's more like the capability in the back that ensures that you can effect a change not that you are pushing the change every day to your customers using your interfaces.

Chris Richardson: I would say as a whole well first of all there's a good good way to think about this to separate deployment the concept of deployment which is having the software running in production to releasing it which is making it available or making it visible to the end users so i think it makes sense to just deploy continuously but then releasing you can hide those changes behind a feature flag which is essentially a switch if the flag is off then show the old ui else if it's on show the new Ui right so you're actually changing things behind the scenes but they're not visible until you flip the switch, Just as a data point, by the way, Amazon, for example, is rumored to be deploying every 0.6 seconds, Amazon.com, right? Obviously, massive organization. And it's like, what are they doing? Because the UI always seems to be the same to me. But obviously, it's a massive, complex e-commerce, logistics, and everything organization. But it does show that you can have constant change with incredible reliability.

Lukas Egger: Yeah, that's definitely what the target should look like. Now, when you engage with a company or teams, how do you determine whether the people, processes, code, architecture, like where really the constraints are and what needs to be fixed, right? Because we've been talking about architecture, but it's not obvious that that always translates to codes, right? Because you also mentioned, hey, it's not just a code, it's your DevOps, your ability to deploy code, run it and give the ability to the coder to create functionality and release it or at least push it into the systems, right? And then loosely coupled teams, which is a lot about team topology, hierarchies, incentive structures, all of that. It's really not all code. So how do you suss out where your talents are needed and maybe more generally how to think about like, what is the most principal problem?

Chris Richardson: Well, I suppose in a sense. It's almost always a people problem. Right because you know I mean in a way right because the root of everything is what are people the culture the attitudes that what they actually do and then that leads to well we organize correctly are we working the correct way and do we have the right deployment pipeline and, architecture more generally so i guess i just have a series of conversations that are really looking at all of these different aspects of the success triangle like yeah what is your architecture what is your organization what are your development practices.

Lukas Egger: And then, where would you say one needs to start effectuating change are you starting with devops with teams or with architecture typically if you want to improve overall and get to what good looks like let's say the ability to change code every second.

Chris Richardson: In a sense, I guess it's like an incremental improvement of all of these specs. I feel like they are kind of intertwined in a way. Certainly, you could say that, you know, one distinct area to focus on is the deployment pipeline. What is the path from laptop to production. Right? Because quite often that will involve manual steps. And approvals so then the focus is well how can we fully automate this bake all of the approvals into the deployment pipeline as an automated mechanism.

Lukas Egger: Now, as a first North Star, the frequency of deployment and getting a reality check from your organization, that can be like a very strong indicator where you are on your journey. Lately, a lot of people are like, in a way, are not talking so much about architecture and whether it's microservices or cloud native or not. Like the flavor of the month is AI and vibe coding. We had a conversation with Sebastian Baltas in episode 14, where we talked about the future of development and development itself is changing quite a bit. How do you feel about more people coding and vibe coding and what's going on with AI? Because so far it feels like vibe coding and all of that. but it does not really connect with architecture. It's more like, just say what you want, and it will magically appear.

Chris Richardson: Well, okay, gosh. By the way, I just want to point out that the Dory metrics are not the only metrics you should measure. There's also what are known as the developer experience metrics, direct metrics. And I think those are worth tracking and that's a whole other conversation. And then also tracking architectural metrics, analyze what's going on to make sure that your teams truly are loosely coupled and that's something we could talk about a lot later but yeah let's talk about ai actually though you know James gosling came up with a great suggestion that we should stop calling it ai and instead call it advanced statistical methods. ASMs, right? Because I feel like we're confusing ourselves by, if we say, anthropomorphizing the technology. First off, it is not intelligent in any sense related to human intelligence. It doesn't know anything. Right? It really doesn't. And it appears to, but it doesn't. And then also, it cannot reason. It appears to reason, but it's all basically next, should we say, it's simply a non-deterministic next-token predictor that provides the illusion of knowledge, reasoning, and I guess intelligence. Right? And I think if we could reprogram our brains to use different words for those things, I feel like we could have a much more sensible debate or discussion about it.

Lukas Egger: But it's certainly regardless of what the technology is at the bottom, right? It certainly stirs a lot of hope. There's tons of companies, especially in the development side of, let's say, AI or advanced statistical methods, that somehow follow this promise of every person can be a developer and software development will radically change in the next whatever, like one, two, three. To an extent that you can just wish what you want and almost like magic, it will materialize. That's pretty much the state of excitement in the industry right now, right?

Chris Richardson: Yeah. Well, I mean, it's funny. I feel like if you've been in this industry, every so often someone will come up with an idea that will be the end of coding. Right? Or the end of developers. And maybe the first attempt was COBOL. Oh, we're just going to write stuff in English. And was that a 60s thing or a 50s thing? And then, I mean, at some point it was visual programming, so on and so forth. And obviously, famous last words. You know, I apologize to our future AI overlords if I've got this wrong. Please do not kill me. But I feel like with, say, complex business applications, you basically need a, especially if it's in a novel domain. Right? And it's like, if we're not building software in a novel domain, why are we building it? Why are we not just using a library if we're just reinventing the wheel? So, if we're doing kind of innovative things, I would argue that it is still fundamentally a human activity. Because there is this idea which is like. Whoa, I can just go hire up a swarm of agents and they go build some software, right? But given that, well, I'll call them AIs, actually hallucinate and they are non-deterministic, I feel like the odds are pretty slim that they can actually produce toads that you can never look at. I mean, people make the analogy between a compiler for a high-level language and assembler. Where it's like the generated code is like assembler, and you never really look at it. But to me, compilers are these deterministic algorithms, and if they get things wrong, it's a bug, and it needs to be fixed. And as a result, we never need to look. But with AI, I think that you actually do need to look under the covers and you need to make sure that every line of code that's written is good. They just generate junk sometimes. Like it's not even junior developer grade stuff. It's actually college. Well, I don't know. It's just badly written code. It's a terrible error. I mean, I've been experiencing this week. I started a Bromley project a week ago with called code and the code that it generated was just junk. And I had to teach it how to write good code with good error handling and a whole bunch of patterns and then once I taught it, it could actually copy it. It could kind of clone it and did a whole lot better, but it required careful review. So, I'm of the school of thought that, properly leverage AI for development, you actually need to be a better developer, right? And a more disciplined developer. And so, there's myself included, there's a few people who think that like baby steps and specifically test-driven development. Which is where what's driving development is you write a test. The test fails, you then write the code to make the test pass, right and then you refactor it to clean up the code and you constantly go around this loop and it's a human directed loop and that actually requires discipline to actually do test-driven development but I mean just to give you a data point like I've actually i started writing this project a week ago it was like Friday afternoon and i was kind of bored and it was like oh let's just give Claude a go on. So, I just said, I told it, I just started building this new sign. I'm using TDD, so test-driven development. I think I have used it to produce over a thousand lines of code a day.

Lukas Egger: Which, for reference, is more than anyone can just quickly pass over and read. So, it's like a pretty substantial amount of code.

Chris Richardson: Yeah, I mean, I don't know how much I could... I wasn't even working on it full-time. So, I feel like it's actually boosted my productivity. But I actually felt quite good because I just started off by using ChatGPT to create a specification, turned the specification into a set of tests. And then told Claude to work through those tests one by one. And I was pretty much reviewing everything. And interestingly, it was almost overwhelming because it was like the cognitive overload with reviewing this stream of changes and interacting with the AI just exhausted me, actually.

Lukas Egger: Hmm. So, you're in a sense saying that at this point, the lure of this magical technology that can solve development problems is like any other panacea before it. It doesn't work at this point. And what you really need is still good taste and good architecture and the fundamentals. And then you can hope to improve productivity.

Chris Richardson: Well, but on the other hand, it did work, but it required a lot of I mean I feel like overall it was successful but it really was implementing things one test at a time with code review and then teaching it how to do better and then telling it to go clean up the code so that it was conformant with the revised guidelines so I think in a way it was kind of successful but at the same time I felt like I'm still the boss, Like it can't be trusted to do things by itself and it needs human overview. And so, the notion of, oh, I'm going to get rid of all of my developers and why code this production quality app? I feel like that's probably a fantasy for real world applications. Right?

Lukas Egger: I love how you debunk this because what I'm actually hearing is we need to invest into the metrics that you mentioned into good aesthetics, good architecture, and understanding of what makes architecture and organization and its code agile. And this goes through the human oversight of very well-trained, well-equipped in terms of like architecture tools and understanding people and employees. So, this idea of just getting rid of developers and getting stuff done with AI essentially will just create a giant heap of technical debt. Whereas if you empower your developers and maybe even invest more in their understanding of architecture and how to be writing good code, then you can actually reap rewards and maybe get a productivity boost.

Chris Richardson: I mean, I would say that it boosted my productivity. Because as I said. This was kind of a part-time thing. But I managed to write a lot of code, actually. Yeah, then there were times when it was super confused and couldn't figure stuff out. And I had to really dive in and so on. It was kind of a win, but it required guidance.

Lukas Egger: And I hear you're saying it required less of the writing and more of the supervision and the feedback and the understanding of where the organization and you as a developer want to ultimately go with the project, right? So, it may be alleviate some of the pain with writing the lines of code, but it will fail if you don't have a strong understanding and good aesthetics about where you want to go. Yeah, yeah. Like the taste buds need to get better.

Chris Richardson: Yeah. You know, it's funny. I'm talking about feedback loops, right? So, with this, you know, TDD, there's a major feedback loop, which is basically continually running tests. And so, one aspect, in order to speed all of this out. You need to structure the code base in a way that accelerates the test execution time. In a nutshell, you need the right architecture to accelerate this sort of micro feedback loop.

Lukas Egger: I love that because it harkens back to what you said at the beginning, that ultimately the good architecture is always serving like better feedback loops because it creates less dependencies, it speeds up and it gives the people who are working on a problem and whether they're working with agents or not, it gives them a faster feedback loop to what they're building. So I think that's a very strong and very tangible position that can serve as a really awesome North Star in evaluating these things. Now, moving on from AI, the question we always ask, if you could redesign one thing, like if you could change one business process, if you could magically change one thing and let it not be AI, what would it be and why?

Chris Richardson: Yeah, that made me laugh when you mentioned you were going to ask this. Well, so I'm a one-person business corporation. Right? And let me just say the least joyful part of my day is I love working with big companies because they always have interesting architectural problems. But the engagement is sort of, I should say, bookended by two things. One is the procurement process. Particularly the legal aspect of it, which is quite overwhelming for a small business. And sometimes the other end is accounts payable.

Lukas Egger: Yeah, we're definitely feel called out here. Yeah.

Chris Richardson: And yeah, sometimes it's all kinds of feats that you have to jump through and iterations on the contracts. Which are in some cases. The amount of legal fees would have exceeded the. It was like for a one-hour talk would have exceeded the fee for the talk. Right. That's an extreme example. It's just sort of like, if those things could just be simplified and shrink-wrapped contracts that were fair to both parties and so on. And payment processes that recognized a computer that documents aren't transported by mule from one department to another.

Lukas Egger: You know. i think a lot of our listeners will have intimate knowledge about what you're describing that is unfortunately a common occurrence right yeah Chris thank you so much i think you have put your finger onto something that's truly important the world will only get crazier in order to adapt and be successful we need to care about the feedback loops and some of the most important, if not the most important, go all the way down to where code resides and architecture plays out. There are a couple of metrics and good practices in between that thought leaders and businesses who really care about their future prospects can look at. You mentioned a couple of them, but in terms of first principle thinking, it was extremely helpful to get your take, whether whether it's on AI, whether it's architecture, there's a clear understanding of what needs to be done. And I think that's the best we can hope for is getting more clarity in a world that's getting crazier and crazier. So, thank you so much for helping with that. And thanks for being on the show.

Chris Richardson: Well, thank you. It was a fun discussion. I hope your listeners find it useful.

Lukas Egger: I'm sure, 100%. And with that, thanks for listening to another episode of Process Transformers. This podcast is brought to you by the dedicated efforts and the hard work of our entire team. So, a heartfelt thank you to Beyza Kartal, Jahanzeb Khan, Reagan Nyandoro, Erica Davis, Cecilia Sarquis, Fouzi Murad, and Julian Thevenod. If you have questions or comments, email us at processtransformers@sap.com. And until next time for another transformative conversation.

End of transcript. www.sap.com.