EPISODE 1338

[INTRODUCTION]

[00:00:00] KP: A software engineer will make many mistakes on their career journey. In time, engineers learn to make smaller mistakes, recognize them faster, and build appropriate guardrails. The demands of delivering software in a timely and efficient fashion often forced developers to carefully optimize tradeoffs in order to deliver solutions to their problems at hand. *Software Mistakes and Tradeoffs. How to Make Good Programming Decisions* is the book by Tomasz Lelek and Jon Skeet.

In this episode, we interview Tomasz about his experiences as a software engineer, and sample the advice found in the book. Listeners interested in the copy can use the special discount code sedlelek35@manning.com.

[INTERVIEW]

[00:00:49] KP: Tomasz, welcome to the show.

[00:00:52] TL: Hi.

[00:00:55] KP: You wrote a book, *Software Mistakes and Tradeoffs, a*nd you co-wrote it with Jon Skeet. Jon has been on the show before, he is an engineer's engineer. Tell me about your relationship with Jon Skeet.

[00:01:08] TL: So, he was very professional, and we met during this book, worked on the book basically. So, it started with the mining process, and submission of the table of contents. So basically, I had an idea of the book, and like lists of 12 topics. Each topic is like about software, mistake and tradeoff that I've convinced that I was part of, or I've witnessed that as some kind of in other themes, and so on.

So, I have a list of those topics. One of those was regarding date and time. Second was regarding compatibility of data. So, backwards compatibility for guidance on how to keep it. I've

submitted this table of content to mining and they have this process to submit table of contents to other authors that were successful and so on. Jon was one of those persons that review table of contents. He liked the idea a lot and he proposed that we may co-work on this book basically. This date and time chapter was the most important from his perspective, because he is working on, at the time, C# Library. So, it was natural that he took ownership of this chapter. We agreed that he will be writing this exclusively. And also, compatibility of the data.

My first day, I was thinking about using Avro, Apache Avro as a main technology around concepts in this chapter. But since Jon has a lot of experience with gRPC, and Protobufs, he also took ownership of the stop errand. That's his work.

[00:03:19] KP: You work at DataStax. Did you listen to the recent DataStax episodes that we did?

[00:03:25] TL: No, not yet. But I plan to.

[00:03:27] KP: So, I talked to Jonathan Ellis about, oh my god, what is it, Pulsar. I talked to him about Pulsar then I talked to another guy, but another Pulsar expert. I forget his name off the top of my head. But we did two shows. I called them Pulsar Revisited, and Pulsar Re-revisited. Do you know anything about Pulsar?

[00:03:49] TL: My main expert expertise area is around Apache Kafka. So, I'm on this other edge of solutions.

[00:03:59] KP: Why is Kafka an interesting system to you?

[00:04:02] TL: I mean, when I was working on the previous company, Allegra Group, this is like biggest ecommerce in Poland. We had like 18 million of active users and the backbone of architecture and everything was built using Kafka. So, it was used as a queue between microservices. There was like hundreds of microservices and they were like using Kafka as the Main Event Bus. Even the technology and open source project called **[inaudible 00:04:36]**, we built that around Kafka to add some additional layer, API layer. So that was my first like appearance of Kafka in my projects. We started to integrate some calls to integrate with this

Kafka solution. And we will be doing streaming application for analyzing basically clickstream of end users, and it was like the divert using Kafka, and it was years ago. I suppose I was not well known by then. So, Kafka was one of the only choices, one possible choice possible that we had back then. I've started building my expertise out this technology. Then in DataStax and Kafka connector, I was involved in delivering that. So Kafka, Cassandra, integration using Kafka Connect framework.

[00:05:44] KP: Kafka, Kafka, Kafka, Kafka. Why is Kafka such an important piece of technology? Why is that tool so fundamentally useful to distributed systems?

[00:05:58] TL: Good question. Yeah. So, if I will refer to my book, this is the chapter, let me check. So, this is chapter 11, dealing with delivery semantics in distributed systems. So, Kafka is out there quite a long time. It is production proven for sure. This is very stable technology. So, if you are expecting to handle and scale – if you are expecting scalability to have delivers for your project without worrying a lot about it, then I think it's a good choice. And also, it is well documented. There's a lot of other projects around it. Also, the whole ecosystem that confluent delivers is quite rich. Integrations, like with, for example, with smart S3, and treating Avro as a first-class citizen format, for that data layer and with that data storage is also very good. It gives a lot of synergy. Because you have this versioning of data that is well incorporated in in the Kafka ecosystem.

So basically, its variant production proven. But of course, our industry is changing. So, Kafka was built years ago, and some new concepts like multitenancy, maybe not the best possible in Kafka. So then, some alternatives arrived like Pulsar.

[00:07:39] KP: When you look at the Kafka development story, you basically have a story of a technology that starts at LinkedIn, and then gets a streaming ecosystem built around it with – what's the LinkedIn streaming system, SAMSA? Gets a SAMSA system built around it, spins off into its own company, and then goes IPO. That's a lot of history. What are the biggest mistakes and like misdirection? I mean, by the way, really, really well-maintained open source product, for sure. But if we're looking at from the vantage point of your book, what are the mistakes they made?

[00:08:20] TL: I mean, in my book, I'm not trying to analyze like evolution of technologies too much for the one chapter because Kafka is here, like a one-chapter story for 1 of the 12 topics. Yeah, but for sure, it's important to understand those delivery semantics and how they impact both client and like consumer and producer size. Also, the Kafka API is quite low level for some folks. Even if you are using some API that wraps the Kafka API is crucial to understand it. So, it's not easy technology to use. It's not like when you are executing HTTP call using client library, you need to configure only timeouts and like execute it. In Kafka, you need to have this full understanding of internals of how consumer is tracking offsets, how to commit when something is processed, how to resume processing, and are you okay with duplicates or not? There is a lot of questions to ask before using this technology, and that's the one, what I'm trying to answer and explain in this Kafka dedicated chapter.

[00:09:44] KP: Did you look at Pulsar at all? Did you look at the Kafka vs Pulsar question?

[00:09:49] TL: Yeah, sure. That's a big topic here at DataStax. I mean, if you had already experts, both are experts here, I'm sure we will find something new.

[00:10:03] KP: Tomasz, you know who I think about a lot is Martin Kleppmann. Do you like Martin Kleppmann? Do you like his stuff?

[00:10:10] TL: Yeah. I love this book.

[00:10:12] KP: I mean, is he basically the best in distributed systems on the internet? Like his writing is so clear and it's so modern. Last time I talked to him, we were talking about operational transform logs and Google Docs. How did you build Google Docs that works on an airplane, like that networks over a network on an airplane? It's like a crazy thought experiment. You have a flying firewalled network on an airplane. How do you build Google Docs across that network? It's like, why are you even asking this question, man? It's kind of funny. That's what I love about Martin Kleppmann. And he wrote that book, what was the first book he wrote, like *Big Data Streaming* something or whatever? You know, I'm talking about? Lesson from Big Data or whatever it's called.

So, Martin Kleppmann's show on my podcast is I think it's our most popular episode. It's like one of our top five popular episodes. He's such a wizard. But it kind of seems like you've written something that's on the same scale, like designing data intensive applications. That is a universal concept, data intensive applications, totally universal. Kind of kike your software mistakes and tradeoffs. Do tradeoffs are this like ultimate management challenge, right?

[00:11:23] TL: Yes, yes. Well, the last data intensive application book is one of the best books I've read for sure.

[00:11:32] KP: Designing data intensive applications.

[00:11:36] TL: Yeah, I've tried to – I've read it some time ago. I will agree with you that the approach of explaining things is very – it's great there. But the difference is that his technology is quite – his book is quite technology agnostic, right? So, he tries to present concepts and it's up to reader to find which concept is implemented in which specific technology. And here I wanted to go like, and be more practice in a practical way. So, give this specific technology context as well.

For example, there is a chapter about leveraging data locality, and memory of your machines. So for sure, there is like dedicated or couple of chapters in his book about this topic, but here, I'm trying to, like, also translate it and show it in a specific technology, this is Apache Spark in this example, and how this data locality concepts partitioning, and so on in part the big data processing, using one of the most well known and most use use framework. So, that's one difference. But those concepts are, as you said, quite generic. It can be thought of like a similar book, for sure.

[00:13:06] KP: Tell me about the architecture of the book. By the way, can I just tell you something? The only reason I haven't even looked at your book is because I'm completely underwater. I'm just completely, completely underwater. At this point, I'm like waking up at 5 AM, going to bed at midnight, trying to get these companies and building off the ground. It's been pretty challenging. So, I'm sort of dropping the ball. I just want to apologize for that.

[00:13:29] TL: Sure. But on the other hand, it's I think it's also – maybe it has some benefits that you are not biased. Speaking about the book, like the first time too, I understand.

[00:13:43] KP: Thank you for understanding. I'm really curious about it, though. And by the way, I actually think that in many ways, the best way of extracting information about a source is to do a podcast about an author. Basically, I want to distill who you are and why you wrote this book, and some of the key takeaways. So, it's almost like I don't even really need to read the book to get what I'm trying to get here, which I want to understand who you are and why you wrote this book. Can you tell me that?

[00:14:11] TL: Yeah, sure. So, the story about this book started I think at the beginning of my like professional career, engineering career. I know in the software industry, there is a lot of decisions to make, like you need almost every day, you need to make some low-level decisions. Do you use this pattern in your code or other pattern? But also, higher-level decisions like once a week, once a month, when you are discussing design of a specific system or even architecture and which direction you should go, and maybe which library to pick, or how to use a specific technology or pick one over another? Those decisions, I think, everyone made some mistakes. It's not like every – there is a person that always made a good decision that was the best one. And often also, when you are considering those, you are considering two or more that are valid, that seems valid from the beginning. But then as soon as you pick one direction, and like your system is evolving, you are gaining some more insights about this decision and what you could do in a different way. On the other hand, what was as expected, and based on that, I've started to write my personal decision log and mistakes log throughout my professional career.

So, for example, when our team was speaking to the biggest scheduler librarian, or scheduler solution or to design our own, and when there was like big discussion about it, we picked one solution, and it turns out that it was not a perfect one. And then there was some after a longer period of time, you gain some perspective. Yeah, I was writing those decisions, mistakes and so on. After some time, it was like, a list of quite nice topics. After some time, I thought that maybe they are quite generic that other people also had similar problems, like when picking some technology, designing the code, and so on. So, at this point, I started to think that maybe that is a good idea to just share my findings and lessons learned some in the hard way, and share with

people to not let them do the same mistakes. Basically, each chapter is just production proven approach and and it's proven in a good way or bad way. But each chapter contains those lessons learned from going in one direction and then retrospective, what could be done in a different way.

Because for example, in a scrum and agile retrospective, it's often thought like, only regarding the smaller period of time, right? Week two or something like this. And when you make a specific decision regarding system, it can have consequences in like a month to six months, years. Because I was working at places where I was working years, like not months, but years. I've also seen the consequences of those decisions after even a couple of years. That's my goal on my book, to share those lessons.

[00:18:07] KP: In writing about this book, do you find that you can pull on your life for example? I think about investing like I think about software. I think about game design, like I think about software. I think about everything, like I think about software. Every decision I make, every micro habit I have is architected like a piece of software, like a piece of just like stringing together Linux commands or something. Do you feel that way about your life?

[00:18:36] TL: You mean, to measure everything?

[00:18:38] KP: Just to measure everything to be super methodical and programmatic about everything that you do.

[00:18:44] TL: I think that sometimes it's not easy. Sometimes I would want to be not so pragmatic. But that's the way it is. Yeah, for example, even when I'm planning some holiday during the holiday season, I am creating some kind of a spreadsheet or I think what we will be doing every day, detailed list of things. So yeah, I see what you mean.

[00:19:16] KP: So, dude, DataStax, one of the coolest companies, for sure. I feel like the core competency of DataStax is just to do really, really hard, good distributed system software. It's a strategy that's effectively unbeatable. It's effectively an unbeatable strategy. It's not necessarily the fastest strategy. It's not necessarily the best strategy, but it's a strong and almost

indestructible strategy. We just do distributed system software and we never die. We always grow.

[00:19:50] TL: But on the other hand, to reference one of my chapters as well. So, I have this chapter about consistency and atomicity in distributed system. I claimed that, right now in today's industry, almost everyone is doing distributed systems, even if you are calling an external or API or something like this, you are in a distributed environment.

[00:20:16] KP: Dude, if you're on a single node machine, building React software, you're in a distributed system.

[00:20:25] TL: Yeah, because you have multiple cores, even multi-threading, you can follow things like every core is some kind of –

[00:20:34] KP: Or even every, every virtual core, like a no JS event loop is like a virtual core, right?

[00:20:43] TL: Mm-hmm. So, I think, if you are planning to have a product that handles more than one customer, we will need to have distributed systems involved in some way.

[00:20:59] KP: So, tell me about what you work on. Maybe can you give me like – you spent 34 months at DataStax, what mistakes and tradeoffs have you made at DataStax?

[00:21:10] TL: So, for example, in the chapter two, when we are designing the Java driver, this library, so there was a discussion if we should pick at low level, if we should go into –but I think that that's a decision that a lot of people are facing. So, if you're to design your components using inheritance, or maybe you're finally through some kind of a duplicated approach, and they will evolve independently. So, that's a classic tradeoff between some kind of a duplication, but you have a loose coupling between components. And on the other hand, you have this inheritance, that will save you a lot of duplication, and maybe maintenance part will be smaller. But you are exerting tight coupling. So, this design decision is described in the second chapter.

Also, also when designing APIs and components that are used by – and client libraries that are used by a lot of other engineers, it's hard to – at the design stage and engineering stage, it's hard to get every possible use case. So, there is always a use case that may surprise you. It's it's important to go out and guard against unexpected usage of your API. So, that's also on one of the chapters, around that this is chapter – let me look. Towards chapter balancing flexibility and complexity. The more flexible your API will be, so customers will be able to inject the code everywhere. As I have an impression, you mentioned No JS, right? So, you're more from the JavaScript developer space.

[00:23:15] KP: You mean, like in terms of what shows appeal to my listeners?

[00:23:17] TL: No. What language you are most into? What technologies?

[00:23:23] KP: I mean, aren't we kind of beyond the point of language fascination? All the languages are okay, they're fine. I don't really care.

[00:23:30] TL: Sure. Yeah, I just wanted to reference like, for example, in those –

[00:23:34] KP: You know the best language is, is React. Not even React JS, it's just React.

[00:23:39] TL: But there's a lot of injection points there, right? So, you can customize a lot of stuff.

[00:23:45] KP: But injection points, or just like effectively on the fly compilation. It's just like higher level components that have sort of been compiled into UI for you.

[00:23:56] TK: But if you are providing this flexible API, it's important also to guard against those those unexpected usages. So, in the in the JV or Java world, everything can throw some exception, unexpected things can happen. It's crucial to not let the coder skills impact your event loop or your main processing and so on. It's also important in the context of multi-threading, and just like concurrency, to not not like – if you're also exposing some kind, such an API, and extension points to class, and you know what they do, it's possible that they will block in some way. For example, if you are in an event loop, every blocking call is very problematic.

It's going to block your main tracks from processing. So, it's important to detect that and just work against those usages. I think that everyone that is providing writing on some libraries that are used by other engineers will, at some point, have those problems.

[00:25:16] KP: Anyway, what else is – can I ask you a question? Here's a distributed question. Distributed systems question like – well, actually, I got a few of them. What do you think of this Databricks versus Snowflake dichotomy in terms of how they manage big data infrastructure?

[00:25:35] TL: Could you be more specific?

[00:25:38] KP: Have you ever looked at the architectures of Snowflake and Databricks?

[00:25:41] TL: Yeah, I forgot it. But some time ago, so I'm not up to date.

[00:25:47] KP: You have Spark, the Databricks ecosystem. And in there, they're doing everything in the open source. It's got this clean set of abstractions, the Spark processing engine, the Delta Lake system, which provides access semantics to essentially Data Lake information. So, you can do transactional semantics across the Data Lake. It's a profound idea. If you can run your own Data Lake and build transactional systems on top of that, that's like low cost transactionality. It's an ultimate vision. And then you compare that with Snowflake, which is kind of like proprietary Oracle style hardcore, hardcore proprietary Oracle style for data warehouse.

Both approaches seem very good, very viable. In the Snowflake approach, they can be more prescriptive about how they do the memory hierarchy. Because the memory hierarchy is not subject to open source whims. So, building a memory hierarchy system and open source is probably harder than building it and closed source. So, Snowflake arguably has an advantage at least an early advantage, but maybe not a last mover advantage. We may see Databricks be the last mover here. It's just very interesting combat that I see.

And then you have Kafka, which is sort of like, also – Kafka is also potentially a transactional data warehousing like system, but it's not exactly that. I don't know. Does that make sense? Is that entertaining to you? Or do you want to talk about something else?

[00:27:11] TL: Regarding this transactional, do you mean like, exactly once, providing exactly once in this context?

[00:27:16] KP: It doesn't have to actually be exactly once. It can mean exactly once, like, it's like some – it's just consistent. It's like, eventually consistent. We don't know of how eventual it is, but it's eventually consistent.

[00:27:28] TL: Yeah, so in the context of Kafka. If there's a way to put this effective exactly once. But in all these courses of always, this course hidden somewhere. So, it's not like, suddenly, it's not possible that HTTP for example, HTTP requests will not fail anymore, or it will not get some network partitions, and so on. So still, there will be some coordination calls. That's the customer decision to pick if they are okay, with like sacrificing something. For example, performance, to have this consistency, or not. As long as the dissolution gives you a possibility to tweak those parameters. For example, in Kafka, you can influence that by setting these acknowledgments, if you want to wait for acknowledgments for all replicas, or from one replica and all that other stuff be done in an asynchronous way.

So, as long as this is possible, then you are not like tied to a specific distributor model. Then such a solution is maybe more viable for you and more flexible. But on the other hand, if you are choosing a specialized solution, like you mentioned the Snowflake, you may be okay with this hard **[inaudible 00:28:59]** transactions model. But after some time, maybe if you will need more scalability, and you will be okay with loosening those consistency warranties, it might be not possible for you. Because it's hard to put it in the engine and it's not exposed to you.

[00:29:21] KP: Can I ask you a crazy question?

[00:29:24] TL: Sure.

[00:29:26] KP: If you had to build an infrastructure company today, what infrastructure company would you start?

[00:29:32] TL: Kubernetes is trending. So, you would need to have the Kubernetes somewhere there and maybe some automation on it.

[00:29:40] KP: Totally. I mean, that's what's amazing, right? Isn't Kubernetes pretty useful? Kubernetes is like zookeeper done right, is my perspective. You kind of want etcd. Kubernetes is kind of like the etcd company. It's like what do you do if you if you have etcd, you build Kubernetes? What do you do if you have Kubernetes, you build everything?

[00:30:03] TL: Yeah, some time ago, there was – I think, four years or five years ago, there was these two tools were like Mesos and Kubernetes as well. Do you still remember Mesos?

[00:30:14] KP: Oh, yeah. Did you look at the container orchestration wars? I covered that, like it was a war. It was a war, man. It was a bloodbath. Container orchestration, I've never seen so many failed companies.

[00:30:28] TL: But there are some companies that invest a lot in, like, for example, my previous company build everything around Mesos. And there was a lot of customization of it and so on. I don't know. I think it will be –

[00:30:40] KP: I mean, Mesos is such a brilliant system. It's such it's such a good abstraction. But they sort of lost a marketing battle, I feel. It's not really a technology battle. It's a marketing battle.

[00:30:52] TL: Yeah, for sure. Kubernetes has this Google like certification, I think.

[00:31:01] KP: It starts with a K. How many words start with K? Not very many. Kubernetes is cool, man. It's Kubernetes. That is the coolest name for any piece of software.

[00:31:13] TL: It's a bit long, to be honest. But you can sell this item.

[00:31:19] KP: It's true, but it's got one more syllable than Mesosphere. But two more than Mesos. Mesos is kind of a more peaceful name. Kubernetes feels it feels combative, almost. What do you think of service meshes?

[00:31:34] TL: If I would like to refer to my book, it's not about automation. It's more like about picking the proper automation is left out. So, I don't have strong opinions here, for sure. It's not yet production proven. So even if there are a lot of companies that go in that direction, it's like not long ago. Maybe we will need to wait a little bit to learn about more tradeoffs involved there. It's like one of the new technologies is up there and they will start using it. It might be problematic.

For example, if you are at the API level, so if you have reactive, reactive stuff, it was also trending some time ago, this streaming manifesto, and a lot of technologies that implement reactive. You have various React, or in Java, in other languages as well. And suddenly everyone started writing reactive code. But it's like it also has overheads. It's not an easy obstruction, I think for some of the engineers, and also it has tricky threading model. At the beginning, when everyone started using that API, there was not much information about those potential tradeoffs and problems. But then when it settled out, we know more than – it can be used for specialized use cases, for sure. Same regarding data.

[00:33:26] KP: Hey, Tomasz, would you have any interest in joining one of my companies as chief architect? Seriously? I've got a few companies that I raised money for.

[00:33:36] TL: I don't know, at this level. You can ask me this question again after reading the book.

[00:33:42] KP: I have to read the book? Come on, man. How about I'll read the book after you join one of my companies. Dude, we got so much stuff to build, we got so much stuff to build, we got some serious consistency issues.

[00:33:54] TL: If we could list this book as like a requirement for –

[00:34:02] KP: So, that's okay. That's kind of an okay idea. The thing is, I don't have time for books anymore. Can you give me like the 20-minute loom video version of the book, right? Can you give me like the condensed version? That's what I want. I want like the Instagram version of

your book. What's the 20-minute, 30-minute, 60-minute YouTube video that you can make out of this book?

Okay, how about this. If you join, I will make you like a short movie about this book. I'll collaborate with you. We can work with our content team, we can make a video about this book. I'm completely serious. You join Rectangle or Supercompute, and then Software Daily will produce a video for your book. Not to put the pressure on you. You can answer off air. I just want you to know I respect you a lot. I think you'd be really interesting to work with. So, we can just say that and leave this for the off-air conversation. There's no rush. But I would really like to work with you. You sound like a Martin Kleppmann kind of person.

[00:34:57] TL: Well, thank you for saying that. I'm happy where I am. But things are changing in the future.

[00:35:05] KP: Jonathan Ellis would be really upset if I could convince you to join my company. Jonathan Ellis and I played Dominion online together. Maybe I shouldn't have said that, it’s not public information. But Jonathan Ellis is really good at Dominion. He's really interesting. A very smart, technical person, and super smart distributed systems leader.

[00:35:25] TL: Regarding this 60-minute video, that sounds like a great idea. But I think it would be really hard to do this. Because often the technical books are like end do end projects. For example, I've read gRPC book one week ago, and it was like you're learning concepts from the beginning to the end regarding this specific technology, right? So, we started from the basics, how to create a simple application, and then progress to production usages, and so on so. It's easy to write, to create a 60-minute video, I think about such a book. So, you will just keep some steps, intermediate steps.

However, in our book, every chapter is separate concept. So basically, you can pick one chapter that is interesting to you and read only this one. So, for example, you have some problems about date and time in your systems, then you go to this chapter number seven that Jon has written, and yeah, you will be will be fine with it.

On the other hand, you have, for example, problem with reasoning about Kafka. So, you are going to this chapter 11. Then you need to pick an important library for your project, and you are considering, for example, dependency injection stuff, you're considering one or another. So, there's a chapter about picking third-party libraries, and there is a checklist, what tools, what to check, and what to look at, what's the most important. So then, you go to the specific chapter.

Maybe it will be possible to like present each chapter in like a 10-minute video, maybe that's feasible. Five, I don't know. But if you will multiply that by number of chapters, maybe it will be like 100 minutes, 60 minutes. But that's a great idea.

[00:37:27] KP: I sent you a short video, just now over chat. We did this video called Modern Data Infrastructure. It was very cheap to execute on. I think this is kind of the future of software media, if you look at this video that I sent you. So, it's short form video content. And we're going to do a lot more of it. But we could definitely do one for your book.

[00:37:50] TL: If you will add those graphics, that would be great.

[00:37:54] KP: The graphics that you're seeing in the Modern Data Infrastructure video, right? You like those?

[00:38:01] TL: Yeah, they look really great.

[00:38:05] KP: So, how much do you think that video cost to produce?

[00:38:07] TL: And what tool did you use for that?

[00:38:10] KP: We outsourced it. How much do you think it costs?

[00:38:14] TL: And that's the total of one minute and fifty, right? I don't know. One thousand?

[00:38:20] KP: Something like that. Something like that. But operating that, like executing on that piece of work, that's our core competency. So, it's like, the cost of goods sold is quite low. But the ultimate content is a very high margin. We're trying to figure out this video thing. We

should just make a book for you. In fact, sorry, make a video for you. I'll make a video for you for free just because I like you. And I like your book. If I can sync you with my video team, we'll make you a video. It might take us like a month or two but we'll make you something cool. Does that work? Do you think John Skeet would be okay with that?

[00:38:58] TL: Sure. It will be awesome. I think we both would be.

[00:39:05] KP: What do you want to see in software media? You see Manning, you see O'Reilly, you see Software Daily, what is software media? What should it be?

[00:39:15] TL: I mean, I think that it is different regarding from our experience. If you are a junior developer and you're starting your journey, then it's nice to take a book. Book is a good resource for you. Book about specific technology that is written by someone that needed to work with that with years and because out of experience, and give you golden rules without maybe explaining everything, and going into very low details. As you will gain more experience, then you also need a bit different content. So, you start looking – you now search for a book, how to use that specific language and some specific technology. Maybe it's enough for you to just check out a specific application, example application, experimenting with documentation and so on.

But then, maybe you like the books about more patterns, like concepts, and those also tradeoffs, right? More generic books, more like designing data intensive systems are starting to be important for you. And then, code will always tell you the truth. So also, good resources that are around practical applications are, I think, the future. Also, this very interactive way of learning things. Like for example, Graph QL and this those technologies. You have this Graph QL playground, that you can learn stuff in an interactive way.

I remember that in the Scala programming language, you have this wrap, right, this interactive command line tool that you could just write the code and you get instant feedback. So, in the Java world, it was mind blowing. How's that possible to just – you don't need to write this whole skeleton of application just to test one line of code and it was very interactive. And I think it speeds up learning a lot. As I see, a lot of a lot of learning and teaching materials are going into

that direction. In the Linux Academy, recently I took some Kubernetes courses. There was a lot of sort of interactive stuff. So, it was like, experiment with this and so on.

[00:41:51] KP: What's your next book?

[00:41:53] TL: I don't plan on it yet. I mean, right now, I'm focusing on that one, and on marketing, stuff. I think that marketing is huge amount of work, even comparing to writing an actual content, totally different kind of work. But engineers like to create some stuff and that's mainly more rewarding, and maybe more easier for engineers. But on the other hand, if you need to market your software that you create, it's totally different work and I'm focusing on that now.

[00:42:34] KP: Awesome. Well, what else do you want to talk about? Should we wrap up or anything else you want to add?

[00:42:40] TL: Maybe about this marketing. Six years ago, just after the college, with my colleague, we started to building a startup. It was called **[inaudible 00:42:51]**. It was before this crazy pandemic times and so on. But we had an idea of this people, teaching using online tools. And we based mainly that around screen hero. So, you could show your desktop, and for example, learn code, learn to code from other places in the world. So, we were engineering is application for months. There were a lot of features, and so on. This was really nice to learn this stuff. But when it comes to marketing, it was really too hard for us. We failed on that. So, creating a product was the easier part of the of the work back then. I imagine that for some products today it can be similar.

[00:43:48] KP: Alright, well, it sounds like a good place to end. Tomasz, thank you for coming on the show.

[00:43:53] TL: It’s nice to be here. Thanks for the invitation.

[00:43:55] KP: All right. Awesome. Talk to you soon.

[END]