EPISODE 1370

[INTRODUCTION]

[00:00:00] KP: The React framework has seen continuous growth of adoption since its launch. There're many reasons for that. But I think one of the reasons is how relatively easy and painless it is to use something like React Create App, or some boilerplate code, and have a fully functioning, hot reloading, live demo up and running in minutes. There is, however, a long way to go between proof of concept and enterprise application.

Paige Niedringhaus is a software engineer and creator of *The Newline Guide to Modernizing an Enterprise React App*. In this episode, we discuss best practices and strategies for structuring your React project.

[INTERVIEW]

[00:00:44] KP: Paige, welcome to Software Engineering Daily.

[00:00:48] PN: Thank you, I'm really glad to be on.

[00:00:51] KP: Well, to begin with, can you tell me about your journey to becoming a software engineer?

[00:00:55] PN: Sure. I have a very nontraditional type of journey into software engineering and development. I actually started out with a marketing degree from one of the universities here in the southeast of the United States, which is where I'm located, in Atlanta, Georgia. So I graduated with a business degree. I went into marketing and advertising for the first five years of my career. And then at that point, I was let go from the marketing firm that I was working with. They didn't have enough business to keep me employed. And I actually had the opportunity, I guess, is the best way to put it, to either go out and get another job in marketing and advertising, which was what I knew, or to take a leap of faith and go to one of the coding boot camps that had become pretty popular in my area. And this was all about five or six years ago. About five years ago now.

So when I had that opportunity, I just decided to go for it. I really didn't have a reason not to. And coding was something that I'd been kind of toying with the idea of getting into. I'd been trying out some of the online platforms like Code Academy and Free Code Camp and things like that. And I just wanted to see if I could hack it. So I went to a four-month full time coding boot camp. And at the very end of the camp, I was lucky enough to land a spot with Home Depot. They hired me on to be a contractor. And I worked with them for six months in a contract position. Then I was hired to be a full time software engineer. And I worked for two more years as a software engineer. I was promoted to a senior software engineer. And I did that for a couple more years. And just recently, about three months ago, I switched to a new company called Blues Wireless, which is an Internet of Things type of startup. And I'm now a staff software engineer. So it's been a really crazy journey. And I can't imagine me five years ago knowing where I'd be today. I really would never have expected this kind of a rise in my career from the beginnings.

[00:03:11] KP: Well, getting an introduction at a boot camp, I've heard mixed results, some very, very positive and some not quite as positive. What was great about your boot camp?

[00:03:21] PN: Well, I had kind of similar – I'd heard similar things that you had to the boot camps. So the way that I approached it was there were about three that were options in my area in Atlanta, Georgia at the time. So I went and talked to all three of them because I had the time. So I went to their open houses or basically they come and meet us, meet our students, talk to our instructors kind of thing. And really the one that I went with was the smallest and the scrappiest boot camp of the three. It was the earliest in its career progression or its beginnings. And what I really liked about it was a little bit of the scrappiness. There were two founders. There was one instructor. And when I talked to them, the feeling that I got was how much they cared about their students succeeding. That was really the thing that kind of won me over. Because the other ones that I talked to, they definitely had good pedigrees, they had good stats to back themselves up. But it also felt kind of corporate and a little bit too slick for what I was looking for.

So when I met these people and kind of saw how much effort and, I guess, care they were putting into helping their students to succeed and become working software engineers, that was really what kind of sealed the deal for me. Plus, they also offered four weeks more than the

other two boot camps were in terms of preparation and actual teaching time. And if there's anything that I've learned since going through boot camp and getting out, the biggest takeaway I think was just how much you don't know by the end of a boot camp and how much more there is to learn. So having that extra time in the class, and time to learn, and time to build projects, and just get more experience to me was a really great selling points. So that's kind of how I approached it. And that's really what won me over was how much effort it seemed like they were putting into making you not only learn the material, but then be successful with what you'd learned.

[00:05:34] KP: I know many people are intimidated about getting into software when they don't have a computer science degree. As if that's a prerequisite. What's your take on it?

[00:05:44] PN: I can understand that. A lot of the job, things that you'll see online, the job descriptions will say that they want a computer science degree or some kind of a related field, mathematics, or engineering, or things like that. That's fantastic in terms of having a good understanding of some of the underpinnings that go into computer science. But I can tell you that, from experience, I have never used any of that in my day to day activities as a software engineer. I have never traversed a binary tree. I have never written a linked list from scratch. I have never had to do any of the things that Leetcode promotes and computer science degrees talk about. It's great to have that, I guess, base level knowledge. But really, unless you're like – I don't know. I don't know what you'd have to be writing. But unless you're doing something that's really, really close to the hardware or to the metal, you're probably never going to use that. You're going to pick a great framework, like React if you're in JavaScript, or Flask if you're in Python, or who knows what. But you're going to use – Most likely, you're going to use modern web development tools because they're there and that's the whole reason that they were created was to make it easier for us to build websites and build applications. And those tools take care of all that stuff for you. You don't need to worry about the vast majority of the things that you're going to learn, the theory of with a computer science degree. So I would say don't let that deter you if that's the only thing that's trying to keep you from giving it a shot and applying.

[00:07:32] KP: What's your framework of choice?

[00:07:35] PN: My framework of choice currently is React and all flavors of React. So Next.js, which is the server-side rendered React. Gatsby, which is static site rendering. Create React App when you just need to spin up a quick React application that's just frontend only. But those are the ones that I'm most comfortable with and most familiar with and a big fan of at the moment.

[00:08:00] KP: So there's an endless number of options even within React, but we can look at other frameworks as well. As you were getting into all this, how did you sort through the myriad of options and settled in on the one that was going to work for you?

[00:08:13] PN: Well, it was a little bit of what my team was already using. So when I joined the Home Depot, my team was using Angular, which was actually a stroke of good fortune, because AngularJS was the one framework that I had learned. I should say, the one popular kind of newer framework that I had learned at boot camp. So it was really good fortune that I was able to take some of what I had already learned and apply it to our application.

And then when we had gotten about two years down the road with it, it came time that we needed to just do a complete rewrite. So I was part of the team that was looking at other options. And really, it was an evaluation of what do we have available to us. So React was a good one. That was one of our options. Newer versions of Angular were another one that we could explore. Vue hadn't really come into its own at this point. So it was really those two were the main newer frameworks that we could choose from.

I particularly was kind of a fan of React at that point, because Angular had burned us so badly with the fact that it was not backwards compatible with the newer versions. So the version that we were working on was the very first version. And after that, when Google rewrote Angular with two, and four, and so on and so forth, they did a lot better job making it backwards compatible. But with the version that we were running, there was really no way to kind of port it forward. It was going to be a rewrite from whichever direction we went in.

So when that was the option, I really appreciated the fact that React has worked so hard to make itself backwards compatible with older versions of it even while they keep pushing it forward and keep releasing new features and things to make it easier to write. So when we

looked at the options, and it was those two, I really just kind of wanted to go with React because I had a better feeling about it at least being both forwards compatible and backwards compatible. And I really just enjoyed I think the experience more than having to learn TypeScript, which was also a requirement for newer Angular versions. And I thought it was going to be quicker for our team to get spun up on as well, because none of us were writing TypeScript at that point. So it was kind of what are our team's strengths? What do we know pretty well already? And what is the future roadmap for this application or for this framework look like? And in my mind, all signs pointed towards React. So that's kind of what I went all in on.

[00:11:03] KP: Are you able to discuss what the product was you were working on?

[00:11:07] PN: Sure, it was an internal product. So you don't go to homedepot.com and get to see what I worked on is the easiest way to describe it. But the product that my team and I supported was for our internal merchants with Home Depot to actually help them manage their assortments of products in our stores. So we had 2000 stores, a million products that could be assorted to them. And our application helped to make that possible and do it in a way that made sense. Because previously, it had been very, very large Excel spreadsheets that were handling most of those types of assortments, which was not the greatest or the most fine-grained control. So we brought it into the 21st century by giving them an actual browser and user interface that they could use to manage those assortments in a way that made more sense for them.

[00:12:02] KP: Well, in addition to starting the successful career in software, you've also been writing and doing technology communication about it. Can you speak to what motivated you to take that additional step?

[00:12:14] PN: Sure, that was actually one of the best pieces of advice that I got from one of my senior coworkers after I had been with the Home Depot for a while, another one of the developers on my team. And he was encouraging me to show what I had learned and to kind of help give myself some street cred really, for lack of a better word, because I didn't have the computer science degree, like we said. And I didn't really have much of an online technical presence at the time. That just never had really been something that I had had either the time or the motivation to do up until that point.

But one of the things that he told me was that it would be something for me to point to an artifact, really, to show to potential employers, to team members to other people in the industry that I actually knew what I was talking about, which is definitely an important part when you're either on the job search or even if you're just trying to show that you know what you're doing. So after he had encouraged me to do that, I really took it to heart and I started blogging on Medium, because it was easy. It didn't require me to set up my own website. I didn't have to worry about CMS and things like that. I just could write and share what I knew. And really, it opened up so many opportunities for me, I guess, is the best way to put it. It didn't take off immediately. It probably took about a year of me writing maybe every couple of weeks and just writing a 5-minute or a 10-minute piece on something that I had learned either at work or at side projects. And then some publications started reaching out to me on Medium and asking if I wanted to publish with them. And suddenly more people were following me. And then other companies were reaching out to me asking me if I wanted to write for them, and they would pay me for it. So I was doing some technical authoring that way. Then a couple of my pieces for those other companies were actually picked up by some of the newsletters like Frontend Weekly and React. And I was being invited on podcasts. And it amazing how much happened because I started doing that and because I started doing it consistently.

And actually one of those podcasts that I went on then turned into a full time podcast hosting opportunity. So I do that now. And I'm a weekly host of the React Round Up podcast. And then that snowballed into being invited to another podcast that one of the hosts also ran. I met people that I would not have otherwise come into contact with. And one of those people actually turned into a referral for the job that I'm working at now. So it's really amazing what has happened because of just putting myself out there in a way that I guess a lot of people don't typically do or don't do as much of.

[00:15:33] KP: Let's talk a little bit about React Round Up. What kind of discussions do you get into there?

[00:15:39] PN: Well, it's a really fun podcast. It's very conversational kind of like this one is. But we usually have a panelist guest, or a panelist list of maybe three to four hosts. And we'll have a guest on who will be an expert in something. And it's typically React related, but not always. But

they'll come on. Like last week, one of our guests was talking about how he's a storybook maintainer. So he talked about some of the new features that are happening with Storybook JS. Some of the cool things that people might not know that it can do, as well as other things that he's working on either with his company or on a side projects. So it's very interesting to get into the nitty-gritty of some of the tools that people might be using, but might not know all the fun tricks and features of.

And we also tend to have episodes where we'll talk more about kind of the soft skills that are a little bit less touted, but also are incredibly important when you're a developer, things like how to work with a team, or how to approach maybe refactoring an older application, and getting buy-in from your business partners and your user experience designers and the rest of the dev team and things like that. So it's a good mixture of both technical stuff as well as more soft skills and tips to level-up and become a better developer. So it's a really interesting mix of people that we get to talk to on a regular basis, which is really fun.

[00:17:15] KP: Sounds great. Well, I'm curious, with all the writing you were doing and sticking to a good schedule, do you ever get writer's block? Or conversely, how do you decide what to write about?

[00:17:26] PN: Writer's block is definitely a thing. But one thing that I've found helps me, and it's very, very low tech, is to actually just keep a little note on my phone. And when I would learn something new, either at work or on an on a side project, something that I thought was interesting or kind of challenging to figure out or just a little bit different than what I had seen before, I would just make a note to myself to be able to reference that when I was looking for new material to go back and write about. So writer's block hasn't really been as much of an issue for me, as I know that it can be for some people, because we're solving interesting problems all the time as developers. And even though it may not be something that you would necessarily think was very difficult or it might seem very obvious to you, it's probably something that somebody else will get hung up on. And we'll be searching for something similar. So that's kind of how I started to choose things that I thought would be interesting for other people. They were usually problems that I had solved myself that I hadn't immediately been able to find an answer for. Or sometimes maybe a cool feature that I had built that I was particularly proud of and wanted to kind of show off. So that's usually how I did it.

And then one thing that I relied upon a little bit was I did get paid for some of the articles that I started writing towards the end of my tenure on Medium. I don't write for it very often right now. But I would actually come up with a list of topics and then send it over to the company that was paying me. And then they would tell me which ones they liked the sound of. And it was just like one or two lines about a particular post that I was thinking of. And they would kind of give me the green light on if they thought that would resonate well with their audience or not. So really, it's just kind of trying to solve or share my solutions to problems that other people are probably going to encounter in their own coding journeys.

[00:19:36] KP: Gotcha. Well, can we talk a little bit about the course you're working on? You've recently released *The Newline Guide to Modernizing an Enterprise React App*. Can you give a synopsis?

[00:19:48] PN: I would love to. So *The Newline Guide to Modernizing Enterprise React Application* is a labor of love that's been in – I've been building it now for the past eight months at this point. And it has been the greatest undertaking that I have done outside of work. So that's one thing all on its own. But really, it's a look at what it takes to build an enterprise level application. Because especially for large companies like Home Depot, like Coca Cola, like Delta, Facebook, things like that, it's not necessarily just about the code that's running in production. It's also about all of the underlying things that make it possible to be successful. So it's things like tooling and configurations. It's code linting. Its code formatting, refactoring and upgrading on a regular cadence to keep it fresh and up to date with what's happening in the industry. It's testing. It's using external libraries, like design system libraries. There's just so much more to enterprise applications than just what you see in development or in production. And that's really what I was trying to show. Because I haven't been able to find any good examples of how to take an outdated application, like an older version of React, for instance, and kind of bring it up to today's standards and use the latest and greatest that the framework now offers. So that's really what I was trying to create was kind of a guide almost for the things that go into enterprise applications, and then also how to take an existing one, which is a scenario that a lot of developers will encounter in their lives, and bring it up to those standards of today in a consistent and easy to do manner. So I'm really proud of it. And it just came out a couple days ago. So I'm thrilled that it's live and available for anybody who is interested in it.

And I hope that it helps people and gives them something to reference when they're stuck or when they're trying to figure out how to go about this in an organized manner.

[00:22:24] KP: Well, if we think about the persona of the engineer who has just inherited a software application, maybe something that came out of a startup. So not necessarily an enterprise application, but now it's their task to get it there. Would you describe that as technical debt? Or is this just a normal evolution of software?

[00:22:42] PN: I think it's a little bit of both. Typically, when you're first building an application, it's really hard to know what the future is going to hold in terms of how long that application will be around or even what the best architecture is for that application. So when you're inheriting that, there's a whole lot of just history behind an application that you'll never probably know why it was built the way it was built. But the idea is that you would be able to come in, see that codebase, and immediately get a feel for what is missing and what could be improved.

So for example, when I joined my startup company here, Blues Wireless, I came in, I looked at our code base, one of which is a React code base. And I was able to immediately identify that we don't have a lot of end-to-end unit tests. And so that's something that I want to improve because I've seen so much value from having those kinds of things. Not only does it increase the – It lessens the chances of us adding bugs to production. But it also gives us more confidence that the code that we're writing is also not necessarily breaking any existing code. So those are the kinds of things that I want you to be able to more quickly at a glance be able to understand about a code base, and then be able to make some suggestions for how to improve that and make it better going forward. That's really the biggest thing I think.

[00:24:28] KP: Can you speak to the importance of some of these tools, things like lint that you'd mentioned, where to some developers maybe they feel optional or like a chore to install? Why is it critical for these to be part of an enterprise build?

[00:24:41] PN: Well, things like linting have really been coming into their own in JavaScript in the past few years, but they've been around in other languages, especially the compiled ones for years before this. And really it just helps improve the code quality, if nothing else. Because things like Airbnb, which has a fantastic JavaScript and React linting setup, those are just things

that will make your code more resilient, more stable. It will make it less prone to break in production when edge cases are reached or unforeseen errors happen.

So having these things in there doesn't take a whole lot of setup today. There're some really great packages and just a few lines of code and you can have a linter setup and linting your project. You can even set it up so that it will fix some errors itself, which is fantastic. You don't even have to lift a finger. But it just really helps to give you a better feeling about your project. And for me, honestly, linting has made me a better developer. Like I see some of the same issues coming up in my code again and again, the fact that I'm able to look at what the linting error is and then see the suggested recommendations for how to fix it helps me to know for the next time that this is probably the way that I should be writing the code instead. That will make the linter happy and make my code higher quality. So really, to me it's about a trade off of it might take a little bit more time in the beginning. And it might feel not very fun to see how angry the linter gets about particular code scenarios. But in the end, it'll make your quality better, which is really what we all want is good applications that are not going to crash or do strange things when users hit errors and scenarios that we haven't thought of. So it does take a little bit of extra work. But I think that the tradeoff is well worth it in the end.

[00:26:51] KP: Well, I'm familiar with some backend projects that pride themselves on having this very high percentage of test code coverage and really kind of weaponizing the whole source code with unit tests over everything. I see that is less common in React maybe because it's so easy to get things right and to have hot reloading and just kind of fix things as you go. Can you speak to the importance of testing?

[00:27:18] PN: Oh, testing is huge. To me, testing is one of the things that we probably don't spend enough time talking about. But it really is one of the things that has become almost in my mind a nonnegotiable, especially when you're talking about mission critical applications, like the ones that we're now building on mass. So testing, I can't over stress how important it is. It is the way to, like I said, not only feel confident that the code that you've written is working in the expected ways and also handling errors in the expected ways. But it's a way to feel more confident that what you have written is also not breaking another part of the application. My unit tests have saved me many times from accidentally adding bugs into our code, because while I was fixing one thing or adding a new feature, something else was breaking that was also relying

on a particular function, or a component, or a piece of code. So it's just a way to communicate not only to yourself and your team how strong your application is, and how stable it is, but also to convey that to the outside world.

Now when we're on GitHub looking for new repos or when we're downloading new libraries from NPM, I do check if they have tests to see just how much code is covered in those tests and if this library is really a solid player that I can depend on. And that used to never be something that entered people's minds or that people even cared about. But today, it's just a thing. And although your team might not practice TDD, which is test-driven development, being able to say as a developer that you know how to use testing library, Jest, Cypress or Selenium, is a big deal. Because like I said, it's not really a nice to have anymore. It's almost a necessity or a requirement of applications to be able to prove that they work the way that you expect them to work or the way that product managers dictate that they work.

[00:29:41] KP: There's testing that goes on in a lot of levels. Like you'd mentioned Selenium, I think of that is this – I don't know. I don't think its higher level, but somehow higher level than a unit test to me in some way. Do you have any thoughts on where I should spend my time for testing like that? Should it be, I don't know, evenly distributed, or 50%, 25-25 kind of thing? Where do you focus on testing?

[00:30:03] PN: Well, if you are familiar with the testing pyramid that came out about six or seven years ago from Google, their pyramid went like this. The smallest number of tests should be the end-to-end tests, which are things like Selenium or Cypress would be running. And those are the full user flow, where a user goes from logging into an application, to doing a bunch of different things, to making changes, saving it, and then exiting the application again. Those should be the smallest amount of tests because they can do the most in one flow. And they usually also take the longest to run, because there has to be some sort of a headless browser to actually interact with the DOM in the same way that a user would.

Followed by that, in the testing library, it would be integration tests, which are kind of testing the functions of multiple pieces of an application. So let's say you have a checkout flow. You would test a couple of different components to maybe say, “When I add this item to my cart, then it does appear in my cart, and it has a price, and it has a description, and it has a name.” So those

are the kind of integration tests that you would write more of. And then the lowest level of the testing pyramid is the largest, and that is unit tests. So that's taking individual components and testing that those individual pieces do what they're expected to do. So if you click this button, this function fires. If you put this data in, it renders in this table, things like that. That was kind of the prescribed way to do it. And it has been for, I don't know, the past three or four years.

But when React Testing Library, and Testing Library in general, came on the scene with Kent C. Dodds in about 2018, it really kind of took a new look at the whole testing pyramid and made it into more of what Kant termed a testing trophy. So it still had end-to-end tests at the top, the smallest amount of tests that you would write. But it took a much larger look at integration tests, because the way that Kent described it was users don't really care that a particular function is firing, or some data is being sent to the backend, or that these individual pieces are working as expected if they don't work together, which makes a lot of sense to me. It just makes more sense. You don't, as a user, care what's happening on the back end, so long as when I put an item in my shopping cart, that that item will eventually check out and will show up at my doorstep.

So what React Testing Library and the testing trophy kind of proposed was a lot more focus on integration tests and making sure that those different components worked together in the way that they were supposed to. And then sure, you can have some unit tests, but not as much. And then it also recommended static tests, which were things like using prop types, and making sure that the particulars of something being passed into a function where the expected inputs and outputs and things like that. So there was also this kind of additional focus on just making sure with things like TypeScript or prop types, things like that, to just kind of help ensure that what you were using was what the system was expecting to have put into it. So it's really now a lot more about pieces working together to form the whole, I think, and less about is this individual piece calling exactly what I think or doing exactly what I think it's doing? Which, to me, makes a lot more sense, and I've definitely seen the benefits of it myself.

[00:34:04] KP: Well, for a listener interested in *The Newline Guide to Modernizing an Enterprise React App*, what sort of prerequisites or background should they bring to the table before starting?

[00:34:14] PN: Well, I would recommend that you have at least an intermediate understanding of JavaScript. This is definitely not going to be React for beginners or even I have a little bit of experience with React. This is for people who have been working with it, but they may not have been working with the latest version. So maybe they know about class-based components, but really haven't had a chance to try out hooks yet. Or people who have been building applications but just haven't seen all of the different things that go into a large enterprise app versus maybe a smaller app that you're doing as a side project, or just on your own, or for a small company. So really, anybody who has some familiar already with React and with some of the testing libraries that we've talked a little bit about today, as long as you kind of have that as a basic understanding, I think that you could get a lot out of this not only to improve your knowledge of some of the newer features of React, but also just to get a better understanding of all the extra stuff that kind of goes around an enterprise application to really make it that level of stability and continued functionality, I guess, is the best way to put it.

[00:35:34] KP: Well, if I have the timeline, right, I suspect you begin learning React before hooks were around and then had to learn all about them and maybe migrate. To earlier point, React's been pretty good about backwards compatibility. But that's also kind of a big mind shift. Can you speak to the experience of – I don't know if you expected that, as you got into software, this idea that everything could change all at once or how it's been to keep up with stuff.

[00:35:58] PN: It's definitely been an interesting journey. And it's something that I am sometimes more motivated to learn more about than other times. There's a little bit of burnout that happens from time to time. But yes, I did start learning React when classes were the main way to do everything and then had the opportunity and a team that was forward-focused enough that we made the decision as we were building new functionality and new features that we would start using hooks. So I had the opportunity to kind of keep the classes where we already had them and then also build hooks in and get a feel for how those worked and get better at them as I went. Definitely, it took a little bit of getting used to, I will say that. It was very weird going from some of the component-based lifecycle methods to suddenly having multiple use effects that we're all dependent on different things and fire at different times and learning the rules of hooks, like you have to have them all at the high-level so that they'll all fire and then trying to wrap my head around custom hooks, which are still a little bit of a mystery at times when you're writing applications.

But it was exciting at the same time when you finally figure out how to use those new features and you get it right, and you see either how much less code it takes to be able to write something or how much more modularized you can make the code by doing it in a particular fashion. Like, hooks and functional components really, I think, brought React to where it's supposed to be, which is functions first and no reliance really on classes. Unless you have classes already, it won't break because of those. But it's really cool to see how much less code needs to be there to write functional-based applications and state now. But it's definitely a thing that just takes time and experience, I would say is the best way to go about it.

And so I've had the best luck personally with building applications along with tutorials online. That's really how I learned is by writing some code, following somebody who knows what they're doing. And then once the lesson is over, breaking the code and trying to do something that they either didn't show or kind of go in a different direction and see how far I can get before I get stuck and need to kind of go back to the prescribed path. But those are really, I think, the best ways and how a lot of developers probably learned is by building and breaking and trying again and Googling errors and throwing some StackOverflow code in there and seeing what works. So that's really my thing is to just kind of try and keep abreast of what's happening in the industry, whether that's through podcasts like this one, or newsletters, or just following particular people on Twitter who you know are thought leaders in your particular area of interest. And just trying it out, and seeing what you like, and seeing what works, and what doesn't, and put putting the code on the page. That's the biggest thing.

[00:39:32] KP: Well, good advice all around. Paige, where can people keep up with you online?

[00:39:38] PN: Probably the place where I'm most active is on Twitter. So we'll put a link in the show notes for that. But you can find me @pniedri, and I wish I could change that, but Twitter is forever. And so I can't change my username now. But it's a combination of Paige Niedringhaus. So there's only one of me. If you look for my name online, you'll be able to find me in all sorts of different forms.

[00:40:07] KP: Well, sounds good. Paige, thank you so much for taking the time to come on Software Engineering Daily.

[00:40:12] PN: Oh, absolutely. Thanks for having me. It's been a lot of fun.

[END]

