EPISODE 932

[INTRODUCTION]

[00:00:00] JM: DarkLang is a programming language that is tightly integrated with the cloud. Dark takes an opinionated approach that most developers are going to want to run their applications in the cloud, and this perspectives influences how Dark looks at deployments, IDEs, exception handling and other aspects of software development.

Paul Biggar is the founder of CircleCI, which is an extremely successful company, and he ran that company for 8 years before leaving to found Dark with Ellen Chisa. Ellen is a software engineer and the CEO of Dark. Paul And Ellen join the show to give their perspective on modern software engineering and why it was time to build a new high-level language that assumes the presence of a cloud.

It is difficult to get programmers to adapt a new language and it's even harder to get those programmers to pay for products built around that language, but the timing could be perfect for Dark. Software development is undergoing tremendous change and many of these changes work to Dark's advantage, such as the growing adaption of feature flags, low code tools and sophisticated continuous delivery workflows.

Whether or not Dark is a success, it is a bold project and the team is working on something that they believe in. We also had a great discussion about AWS. Perhaps I should say debate about AWS and whether the largest cloud provider has an obligation to contribute back to the open source community commensurate with how much AWS has gained from the open source community.

If you're building a software project, post it on FindCollabs. FindCollabs is the company I'm working on. It's a place to find collaborators for your software projects. We integrate with GitHub and make it easy for you to collaborate with others on your open source projects and find people to work with who have shared interests so that you can actually build software with other people rather than building your software by yourself.

FindCollabs is not only for open source software. It's also a great place to collaborate with other people on low code or no code projects, or find a side project if you're a product manager or somebody who doesn't like to write code. Check it out at findcollabs.com.

[SPONSOR MESSAGE]

[00:02:38] JM: Looking for a job is painful, and if you are in software and you have the skillset needed to get a job in technology, it can sometimes seem very strange that it takes so long to find a job that's a good fit for you.

Vettery is an online hiring marketplace to connect highly-qualified workers with top companies. Vettery keeps the quality of workers and companies on the platform high, because Vettery vets both workers and companies access is exclusive and you can apply to find a job through Vetter by going to vetter.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily.

Once you're accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters so that you only get job opportunities that appeal to you.

No more of those recruiters sending you blind messages that say they are looking for a Java rockstar with 35 years of experience who's willing to relocate to Antarctica. We all know that there is a better way to find a job. So check out vettery.com/sedaily and get a $300 sign-up bonus if you accept a job through Vettery.

Vettery is changing the way people get hired and the way that people hire. So check outvettery.com/sedaily and get a $300 at bonus if you accept a job through Vettery. That's V-E-T-T-E-R-Y.com/sedaily.

Thank you to Vettery for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[00:04:27] JM: Ellen and Paul, welcome to Software Engineering Daily.

[00:04:30] PB: Thanks. Happy to be here.

[00:04:31] JM: Paul, you're the founder of CircleCI.

[00:04:34] PB: Yup.

[00:04:35] JM: There have been many CI providers. Circle has been especially successful. Why is continuous integration so interesting to you?

[00:04:45] PB: Oh God! I'm not sure that it particularly is. I think that the interesting thing for me is always making developers more productive, and I kind of joke that my career has been convincing investors to give me money for my pet compiler projects, and Dark is a nice continuation of that.

[00:05:02] JM: But just staying on your time with Circle or the continuous integration more broadly for a bit, because I think this ties in pretty closely to Dark.

[00:05:11] PB: Oh, for sure.

[00:05:11] JM: Do you have any lessons for – I mean, Circle was able to compete in a very crowded market and do really well in a very crowded market, and my sense is that the only way to do that is to have some kind of passion about what you're doing, like some kind of like serious level of passion.

[00:05:28] PB: I love making products that people use. While automating a particular workflow isn't necessarily the most interesting thing when you throw in customers and throw in the value that they experience and your engagements with that community, you can make almost anything fascinating. But I've always been fascinated with the idea of the productivity that you get from CI.

When CI stopped being just CI and started being CICD and it started being a lot more about just running tests, which running tests I don't think is a particularly interesting thing. But when it

becomes about this is how a developer gets their code from their machine into production and that you're making that life easier for them and then you're using the product yourself, I think that's fascinating.

[00:06:16] JM: How are modern continuous integration workflows inefficient?

[00:06:20] PB: Do you want the Dark answer or the CircleCI answer?

[00:06:23] JM: I want the holistic answer. Maybe you could give the CircleCI answer and then Ellen could give the Dark answer.

[00:06:28] PB: I think that it is challenging to represent everything that you want to represent as part of that flow in a simple process that finishes quickly. If the time that it takes to get your code into production is one of the limiting factors on development, which I believe it is, then there's an inherent tradeoff between the amount that you want to guarantee during that process and the amount of time that it takes to get that process finished.

[00:06:58] EC: I think that dovetails nicely into how we think about it at Dark.

[00:07:01] PB: It's almost like I planned it.

[00:07:01] EC: Strange, where we always think of it as we're trying to minimize the amount of time a developer is waiting for things to happen and that they're able to see the impact of their software right away. So minimizing the wall clock time that it takes to get there. From that perspective, CI is there as something which helps you be sure that what you're putting out is safe, but it's not necessarily the only way to do that.

[00:07:22] JM: So Circle was started in 2011. At this point you've been thinking about continuous integration for 8 years.

[00:07:29] PB: A bit longer than that.

[00:07:30] JM: Longer than that. You somehow arrived at the idea that in order to solve the developer workflow problems, we need a programming language.

[00:07:42] PB: Right. Exactly.

[00:07:43] JM: How did you come to that conclusion?

[00:07:45] PB: It took many years of thinking of it in many ways. I think there was probably about 7 different threads that sort of combined into a single thing and many of them came from the experience of having a startup, of managing a team, of trying to get projects built and some of it came from frustrations with the many languages I've used in the past.

Once I started running through what does it look like to solve these problems? I realized that they're all kind of the same problem and they fit into the same things. The one that's most interesting probably to your audience is the problems they had with Closure. So Closure was what we used at CircleCI, and it's an amazing language. It's a beautiful language and it has these amazing concurrency primitives and we heavily use the concurrency primitives to do things like scheduling notifications and that kind of thing.

So you just send a notification when a build is finished, when the build starts, that sort of thing. We realized that some notifications weren't getting sent, and that's because if a notification got queued and it didn't get sent and then we turned the machine off because we were in the cloud and we're spinning things up and spinning things down, then it didn't get sent.

So there is this strong impedance mismatch between the idea of Closure as a language that run on a single machine and the idea of using the cloud. Ironically, languages that had worse concurrency primitives such as Ruby ended up having much better cloud architectures because they enforced the idea of like using cues and background workers and that kind of thing. The realization I had from this was we don't necessarily need languages to run on machines anymore. We need them to run on a cloud essentially or sort of above the cloud.

[00:09:27] JM: Ellen, you got involved a few months after Paul had started DarkLang. Why did you get involved with the project?

[00:09:36] EC: Yeah. Actually, it's funny. If you look at my background, you wouldn't look at it and go, "Ellen should make a programming language." You would look at it and be like, "This person is going to start some consumer product company."

But when I met Paul and we first started talking about this, I realized I'm a pragmatist at heart and a lot of the things that were preventing me from being able to build and deliver as quickly at these consumer companies as I wanted to were actually infrastructure challenges. Where you would set up your data stores wrong and then you would have all these performance problems until the end of forever, or your staging environment wouldn't match your production environment and you would test really well in staging then it would actually break when you shipped, or you'd be like me and you'd be a PM and you would get your emulator and your environment set up so you could build small features in your free time. Two weeks later when you actually had your free time –

[00:10:16] JM: By the way, I don't know any other PMs that do that.

[00:10:19] JM: Really? I don't think so.

[00:10:22] PB: I know PMs that could

[00:10:23] JM: I know PMs that could. I don't think I've ever known a PM that does that.

[00:10:27] EC: It's basically because what happens –

[00:10:26] JM: But anyway, I think it's great.

[00:10:28] EC: Is every time you get it set up to do this, then by the time you're back to your desk to do it, everything is broken and you're like, "Am I going to go spend three hours getting someone to help me fix this environment again because of the things that have changed?" Then you got to feel like whatever value you would add by writing more code is wasted, because you've been distracted someone else to help you fix your environment.

[00:10:47] JM: Right. But still, I mean, this project is cool. It's kind of crazy. When I first saw it I was like, “Doing even pay attention to this.” Honestly, what got me to pay attention is pretty smart entrepreneurial people working on the project. Most of the time if you would've seen this kind of thing in Hacker News you’re like, “Okay. It’s going to fade away in a couple of months.” So what got you intrigued enough to actually pursue this?

[00:11:12] EC: I think that’s the thing that was unusual was I just looked at it and went, “Oh, yes. Of course, that's how it should work.” So I didn't have that mismatch that everyone else had where they thought it looked crazy. I just thought it was like, “Yes, that is where we are going.” I think that actually confused Paul at the time, because he was used to trying to convince people that it was.

[00:11:12] JM: Give me your perspective on the trends for why it makes sense for a new programming language that assumes you are in the cloud.

[00:11:37] EC: Sure. The first one is almost everything we run is in the cloud now. You might still have some batch script you run locally for your computer. But for the most part, anything you’re building that you want other people to use is going to be in the cloud. That's kind of the fundamental place we’re running things.

I think then, beyond that in terms of other trends that are going on make this the right time to do this is you’re seeing even the number of people who are either trying to major in CS and not able to, because we don't have enough faculty to train them, or the number of code schools that are popping up. There is this demand for we need to be able to write more software or we need to be able to write the software more quickly. The solutions that we’ve found so far of trying to train more people aren’t working, which means we have to fix something else.

[00:12:21] JM: What’s not working? What do you mean?

[00:12:23] EC: Oh! If you think about it in terms of like the reason the market salaries are so high is we just don't have enough people who are able to write and deliver the amount of code we need to deliver. So in terms that, things you can do, you can write less code or deliver less products, which isn't what anyone wants to do, or you can figure out how to hire more people to

apply them to that problem, which maybe works, maybe doesn't, ethical man-month and all, or you can figure out how to make the time it takes to build anything take less long.

[00:12:54] JM: So the way that I think about modern programming is its JavaScript, and APIs, and cloud services, and the only time I'm going to write code is if I'm writing JavaScript. The only time I'm going to look at another programming language is when I have performance issues. How close is that map to how you both see modern programming?

[00:13:18] EC: I think one of the great things about the JavaScript ecosystem is just how much is already available for it. So the thing that makes JavaScript nice to pick up is chances are someone has done something similar to what you're doing so you're getting a lot of high-quality answers for the problem you're going to try to solve. You're going to have a lot of packages that you're able to use right out of the box. But I think the tendency to pick up JavaScript is because of that ease-of-use and that amount work that already exists.

[00:13:42] PB: I guess your question a little bit is why do we make a new language to a certain extent.

[00:13:46] JM: Yeah. It's sort of – Well, the way that I think about Dark is it's a language that's built for like productivity. You want fast product delivery to market.

[00:13:56] PB: Yeah, that's a good way of saying it.

[00:13:59] JM: The crowd that wants to do that is the crowd that's building with just like JavaScript. They're just like stitching together or like give me their React boilerplate thing. Like let me browse some easy components to throw together, and that's my UI." Then they're like, "Okay, I got a Twilio thing and it matches to a Stripe thing." They just glue it together with JavaScript. I want to make the case that Dark can potentially provide a more productive experience.

[00:14:29] PB: Yes. I think the first place to look is why aren't things better today? If we look at how people have tried to obstruct and tried to reduce the complexity and things like using AWS Lambda, for example, or anything in the sort of serverless realm or some of the other startups

that are in this space that are focus on JavaScript. We don't believe that any of that has solved all the complexity that's out there.

When we looked at how this sort of thing needs to work, we realized that integrating directly with an existing language or with an existing ecosystem was not necessarily possible to do the things that we wanted to do. For example, if you're looking it's the live values, which is if you look at the Dark demo, you'll see that any time that you're writing code, you're shown a real production value that has happened recently in production and its effect on the code that you just wrote.

To cobble that together from the existing infrastructure from the – If you wanted to try and put hooks into the V8 interpreter to try and get that information out or the amount of work that it would take to try to shunt that into existing workflows means that we just didn't feel it's possible and still don't feel it's possible. So we needed something new, and it's not necessarily that it needed to be not JavaScript. It's also not Ruby and not Python and all these other things. It's just that it needed to be tightly connected to the editor and to the infrastructure and it needed to be designed for that purpose.

[SPONSOR MESSAGE]

[00:16:10] JM: When you listen to Spotify, or read the New York Times, or order lunch on Grubhub, you get a pretty fantastic online experience, but that's not an easy thing to pull off, because behind-the- scenes, these businesses have to handle millions of visitors. They have to update their inventory or the latest news in an instant and ward off the many scary security threats of the Internet. So how do they do it? They use Fastly.

Fastly is an edge cloud platform that powers today's best brands so that their websites and apps are faster, safer and way more scalable. Whether you need to stream live events, handle Black Friday, or simply provide a safe, reliable experience, Fastly can help. Take it for a spin tried for free by visiting fastly.com/sedaily.

Everybody needs a cloud platform to help you scale your company. Everybody needs a CDN. Check it out by visiting fastly.com/sedaily.

[INTERVIEW CONTINUED]

[00:17:23] JM: There is the modern infrastructure context and workflow context, and that's part of the motivation for why Dark is built the way it is. There's also the – Paul, I understand you have an affinity for programming languages. Could you each name how a specific programming language that is – I don't know, niche, or novel, or tell me how other programming languages inspire DarkLang.

[00:17:54] PB: It's funny that you say that I have an affinity for programing languages, because I actually feel the opposite. I feel like I hate every programming language and I just hate some less than others. So I used to describe Closure as my least hated programming language rather than my favorite, and Python is my second least hated.

There're tons of things that influenced probably by six or seven different languages directly. I think that the greatest influence I think has probably been Elm. Elm really showed us how the ML type system can make it easier to write programs and to write simple programs that are easy to understand and easy to know that you're not making any mistakes. It also showed us how good a well-designed standard library can be, which I think is a thing that many, many languages don't have.

[00:18:40] EC: I think in addition to that slightly off, we also are influenced a lot by academic work. So like, Cyrus Omar's on Hazel and thinking about how you build structured editors around blanks and how you generate them and how you see the results in each type.

There's also some work out of a university in Colorado that I think is really interesting, that when people are picking up a new language, it doesn't actually matter very much what you make the keywords. It just matters that they feel familiar and that they can learn them. One of the nice things about having the language AST-based instead of text-based is we could at some point have different flavors of Dark. So the vocabulary feels more like Elm or more like JavaScript, or in the future could be native to the language of the developer rather than always in English.

[00:19:22] JM: Well, that's actually pretty interesting what you just said there at the end. Wait. But what is AST-based versus text-based?

[00:19:28] EC: Yeah. So when you're coding in Dark, you're actually manipulating the abstract syntax tree. So say you have a variable that you've declared and then you're using it in a bunch of different places, if I was doing that in React, if I renamed my variable, I would then have to go manually rename everything and edit the text to update that. In Dark, we know that that is the usage of that variable and we update them all dynamically for you as you edit the first one.

[00:19:55] JM: The idea of a language that would – What did you say at the end? A language that is not read in English? Are all programming languages throughout the world, I mean, does everybody type in English basically?

[00:20:05] EC: Pretty much. Yeah.

[00:20:06] JM: That's kind of crazy.

[00:20:07] EC: Yeah. You're someone who speaks, say, Russian or Spanish and you're learning all of these like very specific English words that are only meaningful to you in a programming context.

[00:20:17] JM: How would an AST-based language help with that?

[00:20:20] EC: Yeah. So since there is a representation under the hood of what it is, you could know that you're using let for stance, but like maybe there is a different word that would make more sense in French for instead of let for declaring things.

[00:20:33] JM: Why can't we do that with Java?

[00:20:37] PB: The whole problem comes down to text. Parsers are a nightmare. The flavors of Java that people use have specific parsers all across the things. Consider all the places that your text lives in git. It lives on your machine. It lives in your editor. So to change it, you need a translator from the text that you have, the parser that you have that re-parses and regenerates it

in a different language and you need that to be in VS Code. You need that to be in GitHub. You need that to be in whatever other software you use. Then eventually you need to translate it back for the interpreter to use.

I mean, it's possible, but it's sort of nonsense. It's like who would do that much work when people already – I mean, tens of millions of people learn all the English words by Roche in order to write up, but they've already done that work. So who is going to do the work to like translate a language and all the standard libraries into something else in addition to all the things that it might just not be possible to cobble together all these pieces? Whereas if you are in the AST-based language or something like that, it's just a presentation layer. It's a thing that no one has to really be aware of.

[00:21:44] JM: You both mentioned Elm. Elm is an interesting example, because it's a language that is beloved and inspirational to many.

[00:21:53] PB: It's lovely.

[00:21:53] JM: I think React – Weren't parts of React inspired by –

[00:21:58] PB: Redux was based on the Elm architecture.

[00:22:00] JM: Dan Abramov is in love with Helm. Maybe that's an overstatement. But that language, as beloved as it is, I know of one company that uses it in significant production. Why is that? Why is a language like Elm that's so widely beloved –

[00:22:18] PB: I have so many opinions on this.

[00:22:20] EC: I will let Paul answer that.

[00:22:23] PB: Dark started with our editor written in Elm, and we we've moved it to BuckleScript over the – It was about a year ago.

[00:22:28] JM: What's BuckleScript?

[00:22:30] PB: ReasonML.

[00:22:30] JM: Oh! Okay. I got it. The Facebook thing.

[00:22:33] PB: Yeah, the Facebook thing. BuckleScript is just another, one of the technologies. It's the same as ReasonML basically. We switched it over because I think almost for the same reason that Elm is beloved. Elm is beloved because of how tightly integrated it is and how good the design decisions have been made in it. As a result of this, they wanted to remove a feature that we relied on. We realized that it was not actually possible for us to rewrite our code base without this feature, and that we had started to rely on this feature. So the feature is like directly being able to call JavaScript code versus going through the eventing system that Elm relies on.

We realized that we had started relying on this more and more and that we were going to continue doing so. So it's removal was the thing that we felt did not allow us to actually create the software that we were trying to create.

[00:23:27] JM: Can you extrapolate that to why Elm isn't used by people?

[00:23:31] PB: I think Elm isn't used by people because it's too weird. It's too different to how people write software, how they write JavaScript today. So it forces everything through this Elm architecture, which I think is relatively easy to understand once you get it, but I think there's quite a bit of a learning curve there.

I think perhaps Redux will push more and more people to it, because they will have learned this architecture through Redux. I think that it's hard a lot of the times for code written in Elm to interface with code that's written directly in JavaScript. That's also changing as things move more to a functional reactive paradigm. Certainly, when we were writing it, we would struggle to find libraries that we could reuse.

[00:24:13] JM: You contrast that with like Erlang, where Erlang is a weird language, but people use Erlang.

[00:24:18] PB: I think one of the things about the frontend community is how much of it is reusing other components and other things that people have written. So stuff that's written for Erlang is written in Erlang and it works with Erlang. If you want to talk to other services, you talk to it in the same way that you do in most languages, which is over HTTP. Whereas on the frontend, JavaScript is the lingua franca. So if you're doing something that's not JavaScript, if it doesn't integrate very easily, and this is one of the decisions that ReasonML made that they wanted a much nicer integration with JavaScript. But if it doesn't have that nice integration, it's hard to adapt.

[00:24:56] JM: That actually works in your favor, right? If you're talking about backend languages being friendly or people – It's like a friendly atmosphere to bringing in a new backend language, probably more friendly than like bringing in a new frontend language.

[00:25:11] PB: Yeah. One of the overwhelming trends of the lost decade is polyglot microservices. So people writing their different services in different languages. So you start with maybe Ruby or Python and then you realize that this thing might be better written in Rust, or in Go, or in Scala, and that's a decision that people make all the time and we think Dark slots in there nicely. You just have to try one new service in Dark. If you like, yeah, maybe it will be two new services in Dark.

[00:25:41] JM: Give me your perspective on how languages should handle typing, static, or dynamic, or some kind of optionality.

[00:25:53] PB: We put a blog post about this recently. I considered calling like experience report from 20 years of static and dynamic languages, but the opinion that I have on it is that dynamic languages are really awesome at some things and static languages are really awesome at some things. In particular, when you're writing new code, dynamic languages are amazing because they don't get in the way. When you have a lot of code and you want to make large scale changes to it, static languages are amazing because they have tooling that provides a lot of safety around that.

You can guess both advantages in the other community, so like Haskell has put an extraordinarily amount of work into making static types vanish at the early sort of prototyping

stage if you're an expert Haskeller. Dynamic language people will often spend an extraordinary amount of time writing unit tests to allow them have that sort of refactoring that but you get mostly, but not entirely for free in static languages. Yeah, I think both just have great use cases that they're really good at, and I personally feel that neither one is good at the other ones use case.

[00:27:00] EC: Yeah, I agree. That's generally what it found. I am much more on in the side of I would rather write things quickly, but most of my things are quick things that I write.

[00:27:09] JM: How has this influenced Dark?

[00:27:10] EC: In Dark we try to give you this hybrid best of both worlds experience. So we have functionality called the Air-Rail. So when you're writing something, what we do is we tell you, say, if you're making a data store query they might fail, might return null to you when you're writing.

As long as you're getting a response, we let you keep seeing that execution path, but we pull out the fact that you do eventually need to write the logic to handle that case. So you can keep building. Get the thing working. See it working when you know that it's like your data that you put in for a test. Later, when you want to make sure you have it handled, you can go back and easily add all of these places.

[00:27:45] JM: You write that there is some accidental complexity in many languages. What does that mean?

[00:27:53] PB: Accidental complexities is a phrase that I first heard from a paper called No Silver Bullets by Fred Brooks. The idea is that he splits up the complexity into two kinds. Essential complexity, which is the core of the problem that you're solving, the thing that your customers pay you to solve basically. Accidental complexity, which is all the other shit that you need to deal with.

So you can think of setting up your infrastructure as accidental complexity, because, sure, it matters that there is infrastructure there, but it doesn't matter what the infrastructure is. If you

magically had infrastructure, then that would make no difference to your customers versus if you spent tens of hours on it. Whereas, you wouldn't magically have your product because you want to like – That's the actual value that you provide in your opinions and your workflows and a tooling that you provide is the thing that actually matters to your customers.

[00:28:46] JM: What are the ways in which IDE users insufficient, modern IDEs?

[00:28:50] EC: I certainly think it depends on what tooling you're deciding to use. I think there've been steps recently for people designing IDEs specifically around programming languages like what Kite did with Python, or I guess in JetBrains and all of the work that they tried to do. I have a friend who's been working on an editor that's specifically set up to work well with Go. I think the challenges that people face if you're doing this multi, this polyglot microservices architecture or you're going to have one editor per language to be getting the best experience for that language. So I think for people who are primarily doing one thing, like iOS developers can tell you about both the pros and cons of Xcode and using that integrated environment. You're not necessarily going to get one thing that works when you're building backend services.

[00:29:33] JM: Anything to add, Paul?

[00:29:35] PB: In terms of IDEs sucking, there's no end to the list of IDEs sucking. I think, fundamentally, the problem is that most of our IDEs don't really understand the languages that we're in. I think the best ideas are, like Ellen says, the sort of JetBrains stuff in the world. The IntelliJs and that kind of the thing, because they have really powerful tools that are designed for the language.

I think probably the biggest step in editors and IDE is becoming really powerful is the creation of the language server protocol. It's kind of a shame that it took us maybe 50 years of software development to get to a place where go to definition consistently works in multiple languages. But once you've used an editor that really understands your language and then you go back to one that doesn't, it's completely night and day.

[00:30:29] JM: Dark does not have a compilation step or separate compiler/interpreter. That's a quote from one of your blog posts. Can you explain what that means? How do you create a programming language that doesn't have a compiler?

[00:30:42] PB: Well, it has an interpreter. Sorry. That was a bit trite. There're a bunch of different technologies that you use to create a language. So the top three are interpreters, compilers and just-in-time compilers, and I say top three, but really anything else can be re-described as one of those. The way works and Dark is you have your editor, which is directly manipulating the abstract syntax tree. The abstract syntax tree is your in-memory representation of the program. So it's got an object representing a for loop. Within that, there are objects that represent and whatever else.

So we take the program. We save it in the editor. When you make a request, we load it from the editor and we interpret it. Our interpreters is like 500 lines. Interpreters are not particularly hard to write. So interpreter in the same way that Python has an interpreter, that Ruby has an interpreter. The JavaScript interpreter is much more complex than that. So I won't say it's exactly the same as that, but it's similar in concept.

[00:31:43] JM: How would you compare DarkLang to a framework like Rails?

[00:31:50] PB: When we decompose the parts of our language, what we're calling language. So there's the infrastructure side of it. There's the language side of it. There's the editor part of it, and all of that we call DarkLang. Internally, we sort of decompose it. One part that we talk about is the framework. So the framework roughly corresponds to what Rails does in Ruby. It's the ORM, although in Dark it's not exactly an ORM. It's the middleware around HTTP. It's the things that provide a nice experience on top of the language. So there is a huge influence from Rails, and I guess most of the other web stacks that we took influence from took their influence from Rails. I think in terms of the framework part of our language, everything points back to Rails.

[00:32:40] JM: Described the hello world experience for DarkLang.

[00:32:44] EC: I think the other comparison people make with Rails is that when Rails came out, it was amazing how much faster it was to set something up. It was very pragmatic. I really enjoyed the era of time that was Ruby Rails Heroku. That was great.

[00:32:57] JM: Isn't that today?

[00:32:58] EC: It still works today, but it takes longer than it takes in Dark. So I'd moved over.

[00:33:03] JM: Oh, moved into the Dark era. The Dark ages.

[00:33:05] EC: I've moved, because me getting my hello world in Rails usually takes me like – I don't know. I have to go to my command line. I have to update things. I have to make sure all my homebrew packages are correct. Yeah. So you spend a bunch of time doing stuff in like – I don't know, maybe 30 minutes if you're fast. Later, you have your server up with your hello world. In Dark, you just open your browser. You specify the method and what you want to return as your response, and you have an API with hello world up in GIF length, like 10 seconds, 4 seconds. It's short.

[00:33:34] PB: You twitted out a GIF recently.

[00:33:35] EC: I did twit a GIF, yeah, mostly because mostly so pleased that it's that length.

[00:33:38] JM: What was the GIF?

[00:33:39] EC: It was literally making a whole the world and Dark. It fits it's in that span of time.

[00:33:45] JM: Why do I want a programming language that is tightly coupled to a cloud?

[00:33:51] EC: So many reasons. One, you don't have to do the part where you make the thing and it works on your computer and then you're like, "Oh, wait. No one else can use this. This isn't real. I now have this giant other project. Get it, like set up and put somewhere else," which I always find extremely disappointing, because I've done the fun part of building the thing and it works for me, but oh, now I have to make work for everyone else.

[00:34:10] PB: I do that the other way around. I build the infrastructure first and then I get so frustrated by that that I've lost all enthusiasm for creating whatever it is I'm trying to create.

[00:34:19] EC: Both of these results in no new software being made.

[00:34:23] JM: You guys should just quit and start a podcast. Oh, wait. That's what I did.

[00:34:27] EC: Paul has a podcast. Don't quit. Please don't quit. That would be no fun.

[00:34:31] PB: Yeah, I don't record enough of that podcast to do it.

[00:34:33] EC: But then the other thing is you get cool stuff that you don't get in other experiences. So when you're programming in Dark, because you're on your production infrastructure, you get all of your incoming requests and you can see real trace information. So there is a huge difference between writing some code and then being like, "I wonder if this works," and then shipping and seeing if it does and being able to like add one new expression. Immediately see what the effect of that expression was and have that like real-time confirmation that you're doing the right thing every step of the way.

[00:35:00] PB: I want to add more here. There's so much more. Deploy list is the thing – You can tell. We just turned from like having a conversational podcast too excited like Energizer Bunny's. So the thing that we're calling deployless this thing where you write code and then it is instantly in production and like the deploy time is like 50 milliseconds. That was like the original, the first thing that we added to Dark, and that's just like completely life changing. Anytime that I go back to the old ways of coding, I'm just like, "Oh! I can't believe I have to do this anymore."

[00:35:38] JM: You probably have both seen Repl.it, right?

[00:35:40] PB: Yeah, that's cool.

[00:35:41] EC: Yeah.

[00:35:41] JM: So Repl.it's perspective is you develop in the browser and the browser gives you this environment that's connected to the cloud and you don't even know you're connected to the cloud. I feel like Dark kind of does the same thing, but as a programming language. How would you contrast those two experiences? Do you know much about it? I haven't even used – I haven't used Repl.it. [inaudible 00:36:12] a little bit in it. I did show on it, but I don't know too much about it.

[00:36:16] EC: There's definitely I think – From what I understand of Repl.it, one of their core use cases is helping people learn. That real-time feedback aspect and why people like Repl.it in general is because you're seeing what's happening right away. Historically, can find that experience to things that are about learning or for people who are new to developments. It's like Scratch is another classic example, which is like a learning language from MIT where you see what's happening in real-time, which is mostly for children. But I think 140 million people a year use it. So it's probably not all children.

From that perspective, those tools are usually regarded as I am doing this to learn. Similarly, when you're doing any frontend developments since you're in the browser, you can open your browser tools and tweak things and, again, see it is you learn. We don't usually have that experience for building backend. So that's the exciting thing about having a language for building backends in the browser. The difference is our intent with Dark is that you actually write your real application and it keeps growing with you. It's not something where you're just playing to learn. It's certainly you can learn, but the point is actually shipping production code not playing with something or learning something new.

[00:37:18] JM: Explain what feature flagging is and why that's integrated into your language.

[00:37:23] PB: Feature flagging is basically a glorified if statements. It allows you to show different code to different people mostly from a controlled that's separate from the codebase. So it has a lot of similarities to the AB testing or to blue-green deployment, but is basically a thing that lets you show a feature to some subset of people and not all the people so that you can you get feedback from your beta testers or ship something to production that isn't quite ready but that you don't want to just live on a branch for a really long time.

The reason that it's integrated into the language is because it was important that for how we were building feature flags, it was important the once they had a semantics. What I mean by that is we needed to understand what happened when the future flag was in various different states. So what does it mean when the feature flag is not done or you've just inserted it? How can we make sure that the code continues to work until your feature flag is in the stage that you actually want it to be represented?

The reason that it's so important to have this in Dark is that feature flag replaces every other technique of conditional code deployment. Today we have our dev environments, which is sort of – Different environments. It's a different place that the code lives. We also have all the various skip branches and pull requests where we ship into media code. We also have the action of replacing code that's in production with the new code that's coming into production, and often this is integrated into stuff like Kubernetes.

All of the stuff in Dark gets replaced with a single concept, which is feature flags. The intuition there is if we're going to use feature flags to enable things to users anyway, if that's the mechanism that we're going to use. To my mind, that is the state-of-the-art mechanism that all the best companies use, then wire the other steps so important? Why is it important that there is a deployment?

Once you've used to be that when you deployed something, that was the way that you delivered something to users. But now, if you've moved that into a future flag, then all that's left in the deployment is the risk that you fuck something up and you're taking down the server. All that's left in the idea of having git branches is merge complex. That there isn't any real value to it anymore.

[00:39:52] JM: You're blowing my mind.

[00:39:53] PB: Thank you. Honestly, that's a feedback we get a lot.

[00:39:58] EC: That's I think are most used emoji. When people tweet about Dark is like the brain exploding emoji, which makes me pretty happy.

[00:40:04] JM: No. I kind of just got it, because I've been doing a lot of interviews with Facebook engineers, a lot like the earlier Facebook engineers, and many of them mention – And you're friends with Edith. You host a podcast with to be continuous. Edith, who runs LaunchDarkly.

The way that a company that where their workflow around feature flagging is mature, the way that those companies operate is in many ways a lot safer, a lot more pleasurable, just different. It's more empowering for the people who are not shipping the code directly to act like operators, because they can just turn on and off feature flags.

Feature flagging really is one of these things that's kind of I feel like sneaking up on the industry, because initially you look at it and you're like, "Okay, cool. It's an augmentation to modern continuous delivery workflows." But when you kind of see that it's just – It's like continuous delivery in the sense that like if you can leverage it significantly enough, it can completely change how your company operates.

[00:41:24] PB: It is exactly like continuous delivery, and it has – I mean, it's sort of an enabler of the two. There's a reason that Edith, the CEO of LaunchDarkly, has a podcast called To Be Continuous. It's the same concept. One is an enabler for the other and they can't exist without each other. It's all about reducing risk. They're both tools that reduce the risk of a single piece of code, like a single massive code drop, which everyone knows doesn't work into many small chunks, each of which have significantly lower risk.

[00:41:57] JM: Right. It makes the gradual ratchet up process much easier. Funny story about Facebook. This was in a recent episode, but there was just this guy I interviewed who used to do Facebook release engineering. There were some huge outage. I think I asked him like what was the biggest incident or outage that he remembered, and there were some incident where like Facebook accidentally turned on all the feature flags.

[00:42:23] PB: I've heard of this. This has been recounted.

[00:42:27] JM: I just laugh every time I think about that. Okay. So when you talk about unifying the programming language and the cloud provider, it calls into question what are you doing on

the cloud provider? What should the cloud provider be doing, or are you a cloud provider? Tell me more about how Dark presents itself as an infrastructure provider today. I guess this is more Dark the company.

[00:42:51] EC: From the perspective of using Dark, you're just programming your backend and the cloud just exists for you. You don't think about it at all. So it's completely invisible. So it's not like you write a bunch of code in dark and then there's a bunch of other settings or things you go fiddle with to make it work. It just works and that's the benefit of having something that's deployless.

From the company perspective, I guess we think of it the same way any other cloud provider would. We think of it as we're providing people, compute and storage.

[00:43:16] JM: You're on AWS at this point?

[00:43:19] EC: We're on GCP.

[00:43:20] JM: GCP. How do you evaluate those two cloud providers in the choice of usage?

[00:43:27] PB: Kind of a bunch of different ways, but one of them honestly is just like fuck AWS. It's a nightmare. I hate it there also. Very predatory company. So we picked GCP.

[00:43:37] JM: Really? What makes you say they're predatory?

[00:43:41] PB: I remember, I don't know if it was said publicly or if it was reported back, but one of the executives at AWS said that they look at their customer's traffic to decide what products to build next.

[00:43:53] JM: I mean, I'm wearing Amazon clothes most of the day.

[00:43:58] PB: I know. We're all complicit.

[00:44:00] JM: They’re 15 bucks, but it's not complicit. It's like this leaves to better quality products, right?

[00:44:05] PB: Sure, and it leads to Amazon taking all of the value of the infrastructure ecosystem.

[00:44:10] JM: Isn't that great? The infrastructure developers can go do something else.

[00:44:14] PB: Not if you want any innovation in the ecosystem. If you want companies to be able to innovate in the ecosystem, then it stands to reason that the reason that they're creating their tools isn’t necessarily so that Amazon can make more money.

[00:44:29] JM: This is what is so baffling to me, because this is what these open source companies are saying.

[00:44:35] PB: They’re right.

[00:44:36] JM: They’re billion dollar companies.

[00:44:38] PB: And they’re still right.

[00:44:39] JM: But it’s not inhibiting their innovation at all.

[00:44:43] PB: If Amazon takes all the value from them, then it is.

[00:44:46] JM: It’s not taking all the value from that. Confluent is still a multibillion-dollar. Elastic is still a multi –

[00:44:51] PB: Yeah, but Amazon has just started doing that stuff. How does look 5 years or 10 years? Are they still companies that are pumping back the revenue to innovate on the products or are they not? I mean, we don't know for sure, but certainly from the perspective of Dark, we are setting up Dark so that it is not in a way that it can be cannibalized by cloud providers.

[SPONSOR MESSAGE]

[00:45:24] JM: Every software engineer rights integration. Whether we're integrating Stripe, or Slack, or Google, or Facebook, we write code to leverage the APIs and tools of the software world. As an application gets bigger, more and more of these services exist in your app. You have Twilio, and HubSpot, Zendesk, and Salesforce. You begin to want integrations between these different services and the amount of integration code you have to write grows and grows.

Zapier can simplify the integration process between your apps and services. Zapier is an online automation tool for connecting two or more apps. For example, I can use Zapier to integrate Stripe with Google Sheets, and every time a user signs up and pays for a subscription with the Software Engineering Daily mobile apps, their Stripe email address can be put into a spreadsheet and Zapier can make that Google Sheet easily import those email addresses into our MailChimp newsletter, Software Weekly.

Then Zapier can make sure that every reply to the MailChimp newsletter sends a message to our Slack. If that newsletter subscriber is also in our Slack channel, we could send them a message and start a more real-time conversation with them. If you're looking for a single service that centralizes all these integrations into simple workflows called Zaps, Zapier is the easiest way to automate your work. Find out how Zapier can help your software integrations by going to zapier.com/sedaily to try Zapier free for 14 days. That's Z-A-P-I-E-R.com/sedaily.

There's probably a way that Zapier could make your software run more smoothly. If you are just a technical person, you probably have enough spreadsheets and Gmail accounts and social media management that Zapier could save you some time personally even if you don't have a business. So check out zapier.com/sedaily right now through November and learn how your API integrations could be managed more easily. Try Zapier for free.

Thank you to Zapier for being a sponsor of Software Engineering Daily .

[INTERVIEW CONTINUED]

[00:47:53] JM: I mean, just to stick on this point for a little bit longer just because you're a powerful voice in the space of cloud infrastructure and I really just want pushback on this, because –I mean, look, I worked at today Amazon for eight months. I love the company. I love it. I believe in their ethos and I don't think of them is predatory.

[00:48:14] PB: I think a lot of my perspective is shaped by being on the far side of the table when talking to like executives in cloud. Yeah, I'll just leave it at that.

[00:48:26] JM: Really? Come on. Expound on that.

[00:48:30] PB: I don't have any more to say on that.

[00:48:33] JM: I guess.

[00:48:35] EC: I certainly agree that there is value to us having public clouds. I think we don't want everyone to be sitting around running their and server in their basement. We don't want everyone having to maintain their own data centers. I think the industry gets a lot of value. In fact, there [inaudible 00:48:48].

[00:48:48] PB: That's the thing. Amazon did a lot of innovation of their own, like the Lambda is a thing Amazon created. It's wonderful innovation. I mean, S3, EC2, all of those are amazing innovations and stuff like Dynamo. There's a lot of amazing stuff that Amazon has done. But they also do stuff that's shitty.

[00:49:09] EC: I think that's the challenge though of how we think about them in terms of a player in the ecosystem. Clearly, there's been an entire ecosystem built around these large cloud providers. So from that perspective, that's more innovation, that's good. That wouldn't be happening if there wasn't a behemoth in this space to spawn all the other things.

But as those thing start to grow up, is there too wide of a gap to where they can be large enough to be genuine competitors as compared to something that would being acquired and it's just a feeder for Amazon innovating the acquisition rather than creating their own things or that

they get copied and then effectively destroyed. What happens there? I think we haven't seen yet will there be companies that are able to be at the same scale as major providers?

[00:49:49] PB: I mean, I think that's part of the problem that traditionally there is a sort of unwritten rule in Silicon Valley that when you compete too strongly with someone, you sort of acquire them or something like that, or you at least play nice with them to get some value – Or to allow them have some of the value that they've created. When you look at Amazon's approach of like, let's say, forking the Elasticsearch distribution, or giving a version of a clone of Mongo, but that is an old version of the protocol.

I have no doubt in their own mind what they're doing is responding to customers' demands, but they're also having incredibly negative effect on the ecosystem. When you look at these open source companies who are changing licenses, when you look at the sort of thing that the copyleft people have been saying for years about the AGPL and that sort of thing, that you start to see that there's a lot of truth to what they're talking about, that it is possible in our current ecosystem for someone like Amazon to play completely within the rules and have massive negative effects. I think that something that our industry, especially the open source side of it, did not consider when we were starting to do all that stuff.

[00:51:06] EC: I think, also, it kind of points to a bigger question where I don't think we have very good ways of talking yet about the responsibilities of a company that is a platform compared to any other company.

[00:51:15] PB: Absolutely.

[00:51:16] EC: I think you see the same thing in consumer stuff. How is Twitter's responsibility for how people are using their platform different? I think AWS, kind of there's a similar thing for the software ecosystem there and I don't think we've talked about that very much yet.

[00:51:29] PB: I mean, there's definitely a lot of people out there criticizing Amazon's ethics across the entire company and not just within AWS.

[00:51:36] JM: I think most of those people are deeply confused. This is like a discussion about generic drugs like. If a drug manufacturer creates a novel drug, there is a six-month window and then generic companies are able to produce copies of it, much like AWS produces generic versions of Elasticsearch. It's like a lower quality product. There's no way that the Elasticsearch product from Amazon is as good as the one offered by Elastic, because for Elastic, it's existential, and we know that offering search and services on top of search is an infinitely complex problem. So why are we at all worried about Elastic's viability as a company?

[00:52:20] PB: I think that's a sort of bizarre positioning to it. The issue is the sustainability for the users. If the users take their money and they pump it into something that Amazon makes, and they're going to do that, because it's convenient to be integrated with the rest of your AWS infrastructure. The stuff that comes from that that Elasticsearch cannot provide, as well as the costs and economies of scale that Amazon gets from being, frankly, a monopoly provider.

So what you have from this is that the sustainability of the ecosystem is damaged and the sustainability of that product is damaged, because Amazon is taking from that community, which it's legally entitled to do, that this is how the licenses were all set up. But do not contribute back to Elasticsearch or to Mongo. Sure, they're successful, but there are many companies that are significantly less successful, and I think Redis Labs is not a billion-dollar company. It's doing great stuff, but it has not yet achieved the size and scale of Mongo. It is equally affected by this sort of thing.

[00:53:28] JM: The irony here is that if these companies took a page from Amazon's book and figured out how to expand laterally into adjacent markets, into new markets –

[00:53:40] JM: I mean, the point of monopoly is that you can't. Only one company can do it. Once it's done, no one else can.

[00:53:45] JM: Well, but Amazon hasn't like captured the entire world. I'm advocating that –

[00:53:51] PB: 47% of the client market I think?

[00:53:55] JM: I mean, it's a massive expanding market, and Amazon created it. Their first mover, not necessarily last mover.

[00:54:01] PB: Oh! Absolutely. Think of small startups in the space, you cannot compete with Amazon's pricing power. I mean it's true across all of Amazion, but in AWS especially. Any time Amazon offers a service, what they want is compute to be used. So there they are offering it at what is your cost price. So you cannot compete even if it's a lower quality product. It's going to be it's going to be significantly cheaper. Maybe that's really good for the consumers. All I'm saying is that it stifles innovation in the space.

[00:54:31] JM: All right. This is probably a conversation for another day.

[00:54:33] PB: Right. Absolutely.

[00:54:35] JM: I will defend Amazon. I am happy to take their position. If it's any consolation, obviously, most people are on your side in this conversation, the people that I talk to.

[00:54:44] PB: This is why we're throwing a shout out for Elizabeth Warren.

[00:54:49] JM: I think it's worth a shout out to Mark Zuckerberg. I don't think he did anything wrong in that commentary. Again, we don't need to have this conversation.

[00:54:55] JM: Sure.

[00:54:57] JM: What's the go-to-market strategy?

[00:54:58] EC: This isn't going to sound terribly exciting, but Dark is really all about developers and developers writing applications. From that perspective, we're still in private beta right now, but while we are in our private alpha, what we did was spend all of our time sitting side-by-side with people who are writing services on Dark and making sure the experience was what they wanted it be. So, yup, there is not much novel in that we hopefully build a set of tooling that people love using easing way more than they have right now and helps them be more productive.

[00:55:27] JM: What kind of feedback do you get from users so far?

[00:55:30] EC: The mind blowing emoji. That's my favorite once still. Definitely, you this experience there. It takes a minute to wrap your head around the concept since it's so different to put your language or editor in your infrastructure altogether. So we usually spend a little bit of time with people kind of building up that workflow.

Specifically in Dark, one of the things that makes for a great workflow is if you've already built your frontend or a mobile application or you're building something that basically is just talking to other APIs and you don't need to build a frontend. Starting from there, and then building your API based on those requirements and then building your data stores based on the API that you need to support tends to be that preferred workflow with people.

[00:56:05] JM: Did you see the Ballerina Project?

[00:56:07] PB: Yeah.

[00:56:08] JM: My understanding of ballerina, I thought it was a very cool project or is a very cool project. It seems like they had trouble messaging it or figuring out how to get it in front of people. Do you have any perspective on why wasn't that project able to gain traction?

[00:56:24] PB: It's kind of hard to tell. One of the interesting thing about making developer tools is that developers – It's a fashion-driven industry. So the way that things are taken up and the projects that succeed are often not the best projects and there isn't a particularly strong correlation necessarily between things that succeed and things that are right ideas.

I think it's very hard to look at something and say, "This failed for this reason," or "This succeeded for this reason," apart from it managed to capture the minds of a certain set of early adopters who then championed it.

I looked at Ballerina. It's a lovely language. It made a lot of the same decisions or sort of adjacent decisions. That went with one sort of type system. We went with another. But I think

that they have a really lovely language. The place that we've made a major different decision is to run the infrastructure, and that removes the large infrastructure components, versus Ballerina sort of compiles to something that can be deployed to your infrastructure. I don't know if that's the major difference or whether it's a certain way that we talk about deployless. Certainly, our deployless blog post is the thing that really blew up and the idea that really blows people's minds. So it's kind of hard to tell what exact thing didn't work for them, but I think it's a lovely project and I think it should had done better.

[00:57:56] JM: The first company that you started, CircleCI, is still doing extremely well.

[00:58:02] PB: I would love to claim that that was the first company that I started, but there were –

[00:58:06] JM: Actually, wait. I was looking at your LinkedIn. I saw earlier companies –

[00:58:10] PB: There were two disasters before that.

[00:58:12] JM: Oh God! Can you relieve those briefly?

[00:58:16] PB: First one was company started right after college with five friends, and that was largely just disagreements of opinions, and I left and they kept it going. Then the second one was my Y Combinative startup. It was called Newslabs, where we're essentially trying to do sort of what Medium did. But to be Medium, you sort of have to be F. Williams, and we were not F. Williams. In the funding climate at the time, that was never going to succeed and we shut it down after about eight months in a real blaze of glory. It was a sort of a horrifying decent.

[00:58:50] EC: There's a very good blog post you can read on this. It's very funny.

[00:58:52] JM: Oh! Really? Okay. All right. So next time, flaming startup destruction and socialism for the cloud.

[00:59:01] PB: Yeah, perfect.

[00:59:02] JM: It sounds like the title of a great podcast. Right. But Circle is crushing it at this point.

[00:59:08] PB: Circle is doing wonderful, yeah.

[00:59:10] JM: Did do you leave the Helm to do DarkLang, or you're doing both? What's your –

[00:59:13] PB: No. left in 2015 and started Dark in 2017. It was taken over by two amazing executives, Rob Zuber and Jim Rose, who took over about six months before I left. A lot of the new – When I left Circle, it was not the sort of runaway leader in the space. It was sort of tied with another provider, and I think part of the reason that it is the runaway successes is the phenomenal job those two executives are doing.

[00:59:44] JM: I think it's cool that you left the company that was like unicorning it and started a programming language.

[00:59:51] PB: It is hard to run a company. People have certain skills, and one of my skills was not being the CEO of like a midsize company that really needed to focus on like your building out executive functions and sales team and financial models and all these sort of thing. The thing that I really enjoy is working on product. I mean, In Dark, I chose to be CTO. I did not want to be CEO.

[01:00:22] JM: Were you CEO of Circle?

[01:00:23] PB: I was CEO or Circle, but I went and I found someone who would be amazing at CEO, which Ellen absolutely is, because I did not want that to be my job. Seeing an incredibly competent CEO handling all the things that I was not good at is a humbling but delightful experience.

[01:00:45] EC: Meanwhile, it's very nice to work with someone who's been a CEO, because they can look at every single thing you do and be like, "Oh! 90% of this is correct. This 10%, I made this mistake before. Let's not make that one," which is great.

[01:00:55] PB: Thank you for phrasing that delicately, which I know I often don't.

[01:01:00] JM: What's the worst part of being a CEO? Is it like QuickBooks?

[01:01:03] EC: Now. There's actually – There's a startup called Pilot. They are great.

[01:01:06] JM: Do you use them?

[01:01:07] EC: I do. I love it.

[01:01:08] JM: I got to get that.

[01:01:09] EC: It's great. You should get it. I'll give you a referral code.

[01:01:11] JM: I'll take it. This podcast is not sponsored by Pilot yet. Use promo code SEDAILY. No. I'm kidding. You use whatever promo code she's got.

I have a bookkeeper. I mean, do I need Pilot? We've always had pilot, because they launched and a friend referred me right around the same time we were starting and they were willing to do all of our historical books back for me to the very beginning. It's great. Yeah. They like rolled it in with my first month cost. I think it wasn't that many months in advance, but it was really helpful. If you have a bookkeeper and it works well for you, maybe you don't. But I did not have a bookkeeper. So far, the solution has been fantastic.

[01:01:48] JM: We should keep talking about it. Give me a better discount.

[01:01:50] EC: Yeah.

[01:01:50] JM: Okay, as we wrap up – Okay, two more questions. I'll keep it like 10 more minutes.

[01:01:56] EC: Do you want actually me to tell you what is hard about being CEO other than bookkeeping?

[01:01:59] JM: Oh, yes! Exactly. I'm sorry. I totally derailed it. Yes. Please tell me.

[01:02:03] EC: I think one of the hardest parts about it is that when things are going well, it means you are giving away every job you've ever had. So as soon as you get good at something and it's working, it's time to teach someone else to do it and let them run with it and they might do it completely differently than you are doing it. So the hard part of the job is any time you do something well, it immediately means you don't get to do that thing anymore.

[01:02:23] JM: Interesting. That sounds like a feature and a bug.

[01:02:26] EC: Yeah. It's both.

[01:02:28] JM: What do you guys think of no code?

[01:02:31] PB: Oh! Strong opinions.

[01:02:33] JM: Or low code if you feel that there's difference.

[01:02:35] PB: Yes. I think that enabling non-developers to build applications and to be able to build the sort of things that we as developers do, I think that that's wonderful. I think that it's a real indictment on our industry that we did not do that without the creation of no code and low code.

One of what Dark is trying to do is we're doing the same thing, but we're taking a different approach to it. Our approach is that we are making coding itself so easy that people who are adjacent to coding, and then overtime more and more people will be able to use actual coding to build the kind of applications, which today they're probably only able to do in no code and low code. We think that the advantage of this is that code is a fundamentally better way of doing it, because code is flexible.

When you reach the end of the sort of wizard obstruction that you get from a no code or low code solution, you often have nowhere to go. Whereas with code, the layer above it is code.

The layer below it is code. It's easy to compose. You can build new abstractions with us, which is a thing that so far you typically have not been able to do in no code and low code environments. So we are believers in high code. You just have to make it significantly easier to build applications, and we think that's what Dark is doing. Fundamentally, that's the thing that we've designed it to do.

[01:04:03] EC: I think that's like one of the things about it and one of the challenges we see with low code, no code, is you always want to be alike look under the hood and see what's going on. So maybe you start from a higher abstraction where you're like, "Oh! I should have a sign in service." Then one day you're like, "OH! I want a new field to my sign in service or add a password reset or whatever it is." If you did have a module that is sign in service and you can't look under and see, it becomes hard to learn.

It becomes hard to get better at code. It becomes hard to understand what's going on. Whereas if you have something that's code and it's compose all the way up, you have that property by default.

[01:04:39] JM: I think Webflow does something –

[01:04:41] EC: Yeah, Webflow is really cool.

[01:04:41] PB: Yeah. I think Webflow is sort if you take low code and then you make it – I guess it's coming at the other angle. It's low code, but you make it so compose and so extendable that you can do that sort of thing. I think that that's a thing that is not traditionally done by the majority of low code and no code tools, but I think Webflow is fantastic

[01:05:04] JM: It gives you like a GraphicQL thing I think also. It gives you GraphQL –

[01:05:09] PB: I haven't seen that feature. That's cool.

[01:05:10] JM: Futuristic stuff. Okay, last question. I want each of you to give me your most ambitious prediction for how software development will be different in the next five years.

[01:05:20] EC: I think rather than everyone just buying the SaaS product that kind of sort of solves their problem, everyone will be able to build fully custom tools to match their own workflow.

[01:05:30] PB: Five years.

[01:05:30] EC: I'm ambitious.

[01:05:32] PB: And optimistic.

[01:05:33] EC: And optimistic. Two of my best and worst traits.

[01:05:36] PB: I think that we're going to see increasing bifurcation of roles. So it's the frontend-backend distinction is going to increase significantly. Those things are going to break up into many more groups. I guess backend is already broken up into a bunch of different groups. Frontend will probably break up into a bunch of groups as well.

One of the things that we're trying to do with Dark is a lot of people who don't necessarily identify as backend to do backend. So I hope that we will allow it to the rejoining of those forces, but I think if things keep going as they are, then there's going to be a lot more specialization and a lot less sort of cross-pollination of tools and ideas.

[01:06:13] JM: Paul and Ellen, thank you for coming on the show so much.

[01:06:16] PB: Thanks for having us.

[01:06:16] EC: Thank you.

[01:06:18] JM: Shout out to Sean Wang for putting this together, for pushing me to interview DarkLang. He referred this show, and it was really interesting.

[01:06:26] PB: That's Awesome. Thanks, Sean.

[01:06:26] JM: Great conversation.

[END OF INTERVIEW]

[01:06:37] JM: As a programmer, you think an object. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers and the cloud area. Millions of developers use MongoDB to power the world's most innovative products and services, from crypto currency, to online gaming, IoT and more. Try Mongo DB today with Atlas, the global cloud database service that runs on AWS, Azure and Google Cloud. Configure, deploy and connect to your database in just a few minutes. Check it out at mongodb.com/atlas. That's mongodb.com/atlas.

Thank you to MongoDB for being a sponsor of Software Engineering Daily.

[END]