EPISODE 736

[INTRODUCTION]

[00:00:00] JM: 10 years ago there was a distinction between backend and frontend developers. A backend developer would be managing the business logic and database transactions using Ruby on Rails or Java. A frontend developer would be responsible for implementing designs and arranging buttons using raw HTML and JavaScript. Today, developers can build entire applications in JavaScript. Developers who spend their early career developing frontend JavaScript skills are finding themselves with a surprising amount of power.

With Node.js providing a backend framework and React, View or Angular on the frontend, a single JavaScript developer can write all the code for a whole application, hence, the rise of the full stack developer. At the same time, the cloud infrastructure that we use is becoming a much easier. Backend as a service, like Firebase or Netlify, simplifies the frustrations of deploying your application and standing up a database. GraphQL improves the relationship between the frontend and the backend and futuristic technologies like WebAssembly and web virtual reality are promising to make a JavaScript engineer's life even more interesting.

Adam Conrad is an engineer and a writer for Software Engineering Daily. In some fantastic recent articles, he has documented the changing nature of the frontend, including JavaScript engines, virtual reality, and how mature corporations are using React and GraphQL. He joins the show to share his perspective on what is changing in the frontend and how full stack JavaScript engineers can position themselves for future success in a quickly changing market.

We are running a listener survey to find out what we're doing wrong and what we can improve on and what you like about the show. You can fill out that survey by going to softwareengineeringdaily.com/survey. You can also sign up for our newsletter at softwareengineeringdaily.com/newsletter, and if you want to work with us or be perhaps a writer like Adam Conrad, you can check out softwareengineeringdaily.com/jobs or softwareengineeringdaily.com/write.

With that, let's get on with episode.

© 2018 Software Engineering Daily

[SPONSOR MESSAGE]

[00:02:41] JM: Today's episode of Software Engineering Daily is sponsored by Datadog, a monitoring platform for cloud scale infrastructure and applications. Datadog provides dashboarding, alerting, application performance monitoring and log management in one tightly integrated platform so that you can get end-to-end visibility quickly and integrates seamlessly with AWS so you can start monitoring EC2, RDS, ECS and all your other AWS services in minutes. Visualize key metrics, set alerts to identify anomalies and collaborate with your team to troubleshoot and fix issues fast. Try it yourself by starting a free 14-day trial today.

Listeners of this podcast will also receive a free Datadog t-shirt. Go to softwareengineeringdaily.com/datadog to get that fuzzy, comfortable t-shirt. That's softwareengineeringdaily.com/datadog.

[INTERVIEW]

[00:03:47] JM: Adam Conrad, you are a principal software engineer at Indigo and you're a writer for Software Engineering Daily. Welcome to Software Engineering Daily.

[00:03:55] AC: Thank you. Happy to be here.

[00:03:56] JM: So you've been writing some articles, and I thought we would just do a show discussing some of the topics you've been writing about and studying, and the first area is frontend engineering as it relates to React. React JS has been around for 3, 4 years at this point, and frontend engineering changed pretty significantly with React. There was finally consolidation around the frontend framework where there was previously a lot of fragmentation. What did React get right?

[00:04:29] AC: I would say the first thing they got right was having Facebook as their backer. So most frameworks don't get a lot of promotion and support because they're developed by individuals. So having Facebook back React made it a contender against an earlier framework like Angular. So that was the first thing I think they got right.

I would say the next thing that was really significant in React's trajectory was in the fact that it was component-based. With Angular, it was very focused on bringing the whole thing to the frontend, and what I mean by the whole thing is if you look at something like Ruby on Rails. Ruby on Rails is a web framework that's very soup to nuts. It includes an ORM. It includes things for views and templates and it includes a way for managing routes and controllers.

So Angular took that idea and decided to bring that to the frontend, but the problem was that it was very overloaded. I think the way that people want to develop on the frontend has shifted from being something very fully fledged, like Angular or Ember, into something a lot more lightweight and component-based. So the fact that React was lightweight from the beginning and focused only on the components made it very appealing to frontend engineers.

[00:05:45] JM: Was there something about View that took ideas that were popularized by React and accentuated them are doubled down on them?

[00:05:53] AC: So I think the big thing with View, which actually that's an amazing example of a single developer who did manage to get very popular in spite of not having corporate backing. So that would probably be the one exception to the others where you can get by and have a successful project without necessarily having corporate backing.

I would say to answer your question about View, I think I've only used it very briefly for a client project a few years ago, but what I saw as valuable for View is that it has the ease-of-use of interacting with HTML the way that Angular did. GSX is still kind of quirky and I'm not exactly – And you will find people who may not agree that GSX is the most natural way to interact between HTML and JavaScript, but View is a nice kind of blend between lightweight component frameworks and something that is easy to manipulate the way that Angular directives made it easy to manipulate HTML well with JavaScript.

[00:06:58] JM: Now you were saying that that pivotal idea of React or the most important selling point was perhaps the fact that it was backed by Facebook, and Facebook was able to provide consistent development to it. It was also able to provide a consistent playground, because Facebook is perhaps or at least at one point it was the most complex web application. So React

coming out of that is a pretty strong sign of this frontend framework actually does suit the needs of the most complex web application out there and it's performant. How does the View community and the View development world contrast with that of react?

[00:07:37] AC: I would say the biggest thing that View got right in that contrast was that they started very, very lightweight and simple. If you look at how Angular started out, it was very bloated and it included everything, and that was a pretty large project. React, also, in terms of payload, is a very large project, which is why there were others like Preact that came out that were lighter weight versions of things like React.

View on the other hand was building from the ground up didn't base itself on any previous framework and was lightweight from the very beginning. It was also very easy to use. The documentation for View is excellent, and so it makes it very easy for new developers to get on boarded on to View pretty quickly. But you do bring up a good point that another important piece of React's success was in corporate buy-in.

So one of the biggest things in evaluating a framework is, "IS this used in production? Are big companies doubling down on these frameworks?" The fault with Angular was that they weren't really using Angular all that much with Google and Google products, but Facebook was already using React internally before they did it publicly. So by having that corporate backing and saying, "We stand by this. We use this on our own applications," made people feel very comfortable that this is something they can use in their production applications. So that was more of a hurdle that View had to overcome and that's why they focused on excellent documentation and making it easy to ramp up with View.

[00:09:10] JM: Speaking of corporate adoption, Airbnb was one of the first companies to adopt React Native in a big scale use case, and we've covered this in – Well, both you and I have covered this, the story of React Native at Airbnb. Obviously, React Native is different than React, but it's a project that's closely related.

So you wrote an article about React Native at Airbnb. That's on Software Engineering Daily. What stood out for you about that case study of React Native at Airbnb?

[00:09:46] AC: I think the biggest thing that stood out for me was the fact that React Native doesn't actually support too much of the advanced feature sets from either iOS or android. I sort of came into it not really having developed much React Native, but thinking, “Okay, they're going to pour a lot of what's available on these devices into React components so that you can use them in your projects.”

But after listening to the podcast and reading up on Airbnb’s case study, I noticed that they couldn't actually use React Native for a lot of their more advanced features or even what’s seemingly seemed like rudimentary features like maps. Like they were saying that as soon as they had to do advanced features with maps or geo-location, they had to leave React Native and go directly to Objective-C code or Java code in iOS or android.

So I sort of came up with the assumption that, “Okay, if they can use React Native, they can use their JavaScript developers,” and they did, but they quickly found out that those JavaScript developers then had to start learning iOS and android development anyway. So I was kind of surprised how limited it actually is in production environments.

[00:11:02] JM: What are the frontend issues? Whether we’re talking about desktop web or mobile web or mobile apps, what are the frontend issues that a company the size of Airbnb encounter?

[00:11:16] AC: Well, I can't speak to it recently. I haven’t been in a company that large since I was at Microsoft about a decade ago. So I can't speak directly to what is happening right now, but I can infer from the issues that I've observed that the biggest thing is managing complexity. When you have a company that large, you just don’t run one application, and one application is very easy to get your head around. It’s a lot easier to just work with one application, one frontend, one backend, but as soon as you are working with a microservices architecture or you start working with multiple applications that may be running some of the same code, that complexity can be very difficult to manage.

I think that's going to continue to be an issue for Airbnb, for lots of companies, mainly because the frontend world is so fragmented right now. You're going to see so many articles coming out in the next year to where people talking about how there's too much choice, there's too many

things to wrestle about. With all this choice means issues with security, with updating all these different packages and plugins. It's really hard to just grapple around, “All right. I want to get this one thing done. Why is it that there are eight different ways to route an application? Why is it that there are eight different ways to just do deep copying of objects in JavaScript?”

I mean, if you look on MPM just for copying objects, you’re going to get a lot of results. So that's a lot to have the wrangle with when you now have more than one application that has to manage all these dependencies.

[00:12:48] JM: I mean, historically, the world of software, as I understand, as much of an historian about the world of software as I am, what I understand is that the world of software kind of fluctuates between this state of a paralysis of choice and a consolidation, and then the consolidation gets a little too cozy and somebody disrupts it a little bit and then again it spins out of control into the paralysis of choice. But what I think about is that theme of kind of going from consolidation to explosion of choice. That cycle is developed in a time where I think the world of developers was smaller, and I wonder if the trend is just eventually towards a world where you do have this paralysis of choice. Because if I just look at the AWS suite of services, there is a bajillion ways to do things even if we’re talking about the backend world, there's been consolidation around Kubernetes, sure. But there is not consolidation around how to do a backend architecture these days. There's such a variety of you can put everything on Kubernetes or do everything in a serverless fashion or use these container instances.

So quickly after there was this consolidation moment around Kubernetes, there is already this paralysis of choice. Do you think that in the limit we’re just asymptoting towards that paralysis of choice or do you think that there is going to be another moment of consolidation?

[00:14:24] AC: I think it happens in waves, and this is just my own personal experience. There are certain languages for some reason that just don't have this problem, like Ruby on Rails is one that comes to mind. That's pretty much been the dominant framework for Ruby since Ruby 1X. So I've yet to find something that is better than Ruby on Rails. Some people use Sinatra, which is like a lighter weight version of Ruby on Rails, but even if you're changing architectures from a monolith to a component-based rails architecture, people are still using Rails.

Same thing with Python. Django is pretty much the dominant web framework. Some people use Flask. Some teams use – I think I don't even know if Pylons is still used anymore. There's some variety in the Python world, but in general you have maybe one or two frameworks, and we're starting to see that even with JavaScript. It's pretty much Angular, React or View, but I think it comes in waves. A prime example of this would be jQuery. So in the late 2000's, early 2010's, jQuery was the dominant library. There weren't even really frameworks at that point for JavaScript, but having jQuery as a plug-in library was very influential on the development of the JavaScript language. So we wouldn't have things like querySelectorAll. It wasn't for jQuery.

So trying to do this in terms of having lots of choice is good, because once we start to find out who the winners are of all that choice, those are going to be the frameworks and the ideas that help move those languages forward. So I wouldn't be surprised if React goes away in the next five years, but what comes out of it is that web components actually become a thing.

So if React is the thing that drives for change for HTML, CSS and JavaScript, that's a good thing overall, and that's where the consolidation will come in where the standardization bodies will say, "Okay, these were good things to have. Let's adopt that into these primary languages." Then once that's adopted across industry, then more people will start to develop things on top of that. So React goes away. Web components rise, and then somebody finds a way to create a framework on top of web components, which is really great. Then that will start another wave of trends depending on how that gets implemented into the standards bodies. So that's kind of my guess is how I'll see it forming in the future, is that there's always going to be choice. There's always going to be variety and it's going to form in waves depending on the trends that people are seeing, how they like developing software and then using that to adopt to standards bodies so that we can develop better products year-over-year.

[00:17:13] JM: What are web components? I remember we did a show about that a while ago, but I don't really remember too much detail about it.

[00:17:20] AC: Just think of web components like XML. So with HTML, it's a standard set of tags that are adopted by the W3C standards body. XML is more flexible. It allows you to essentially define any kind of tag. Web components are similar and that you get to define your own tags in HTML, and with that comes along JavaScript so that you can create interactivity

with those tags. So it's a bit more robust than XML in terms of just being a document language and it allows you to add some interactivity to those tags and it's meant to be a web standard. It just hasn't really reached that adoption point yet.

[00:18:02] JM: And you think that could potentially supplant React?

[00:18:05] AC: I think what React is doing right now is helping shape the way standards bodies are going to implement components at a standards level, right? Because, it mean, just look at the history of pretty much any language, especially JavaScript, nothing really lasts more than five years. I mean, if you asked me in 2010, “Oh, is jQuery going to go away?” Most people would say, “Oh, I couldn't imagine a world without jQuery,” but it’s 2018 and no one really uses jQuery as the bedrock of their JavaScript development. They might sprinkle it here and there, but it's not the main thing they're using to develop with.

So in the same way, I don't see React being the main thing in five years, but I do think it's going to have so much influence on the way that we actually develop that I wouldn't be surprised if a lot of the tenants of what makes React great are going to be put into the JavaScript or the HTML standard.

[00:19:02] JM: Another case study that you and I both explored was The New York Times in their use of React. How did that usage of react at The New York Times? How did that contrast with what you saw at Airbnb?

[00:19:15] AC: So I think the big thing there was that they have so many different services to manage. What you were talking about earlier with dealing with complexity, I think The New York Times is a great example of that, because they have so many different data stores. As a publication network, they have to think about text, they have to think about pictures and video and lots of them, and they have very interactive parts of their site too.

Airbnb is a pretty straightforward marketplace. You look at listings, you buy time at a listing, that's pretty much the main thing you do there. With The New York Times, it's a far different experience. Sometimes you’re reading on your phone. Sometimes you're watching interactive

video on your desktop. They have a lot of different ways of interacting with their sites that is not just like reading a newspaper.

So managing all those different backend, managing all of those different content experiences is a really complex system and it's really great to see that GraphQL was such a useful use case for them. I'm a big believer in the right tool for the right job. GraphQL is a great case study from The New York Times, because that's exactly what it's meant to do. It's meant to make a very simple interface for frontend developers to attach themselves to many different data stores, and that's exactly what New York Times has to do. So diving into that and understanding why they use Relay and Apollo was a really great way to see this is the way you handle right tool for the right job.

[SPONSOR MESSAGE]

[00:20:57] JM: Managed cloud services save developers time and effort. Why would you build your own logging platform, or CMS, or authentication service yourself when a managed tool or API can solve the problem for you? But how do you find the right services to integrate? How do you learn to stich them together? How do you manage credentials within your teams or your products?

Manifold makes your life easier by providing a single workflow to organize your services, connect your integrations and share them with your team. You can discover the best services for your projects in the manifold marketplace or bring your own and manage them all in one dashboard. With services covering authentication, messaging, monitoring, CMS and more, Manifold will keep you on the cutting-edge so you can focus on building your project rather than focusing on problems that have already been solved. I'm a fan of Manifold because it pushes the developer to a higher level of abstraction, which I think can be really productive for allowing you to build and leverage your creativity faster.

Once you have the services that you need, you can delivery your configuration to any environment, you can deploy on any cloud, and Manifold is completely free to use. If you head over to manifold.co/sedaily, you will get a coupon code for $10, which you can use to try out any service on the Manifold marketplace.

Thanks to Manifold for being a sponsor of Software Engineering Daily, and check out manifold.co/sedaily. Get your $10 credit, shop around, look for cool services that you can use in your next product, or project. There is a lot of stuff there, and $10 can take you a long way to trying a lot of different services. Go to manifold.co/sedaily and shop around for tools to be creative. Thanks again to Manifold.

[INTERVIEW CONTINUED]

[00:23:12] JM: So GraphQL is this system for unifying the data access and it makes it simple for frontend developers to issue complex queries and then the backend can federate those queries to the different backend data services, databases, and aggregate the query response to deliver it to the frontend. If I recall, Apollo is the client side component of GraphQL and Relay is that the server-side component. Is that right?

[00:23:47] AC: So Relay and Apollo are both GraphQL clients.

[00:23:51] JM: I always get that wrong. Okay. So what is the tool? Give me the GraphQL toolchain from The New York Times.

[00:23:57] AC: Sure. So starting from the very bottom, like I said, if they have a picture, video, some content article, it's stored in a database somewhere and they have lots of databases that are storing all this different information for various reasons. They're going to have a CDN for their videos, because it's easier and faster to access from there. They’re going to have maybe a PostgreS database for the actual content of the articles or some sort of CRM.

So someone from The New York Times fronted team is going to want to access all that stuff together, because an article is definitely a composition of all those different things into one place. So they'll issue a query with the GraphQL language. So they'll say, “Pull me this query that has this title, this description and this headline video.”

So what's happening is GraphQL uses Apollo as the clients on the client side to say, “All right, take this structured query and break it down into the JSON I need to fetch from the GraphQL

server.” So Apollo will break that down into something that is readable over the wire on HTTP. It will send that to the server. GraphQL, as a server, will then deconstruct all those pieces and say, “Okay, this headline, that comes from the CRM. This video URL, this comes from the video CDN. This headline picture, this comes from the picture CDN.”

So GraphQL as the server will break those things apart, figure out which databases it needs to talk to and grab information from there. Then the databases are just living on their own servers and getting regular SQL queries as they normally do. So GraphQL and Apollo are essentially the interface between the user wanting that information and the database wanting to respond to something that actually is SQL.

[00:25:55] JM: We did a show a while ago with Prisma, and Prisma is a tool for, I believe, translating the query. If you get a GraphQL query from client, Prisma is an easy way of translating that GraphQL query into, for example, a PostgreS query, or a my SQL query, or a Mongo query. But basically you have this federation of queries to do and you don't want to write all those queries manually. So you want some tool to translate it. First of all, did I get that right? Is that the right – Is that what Prisma actually does?

[00:26:29] AC: To be honest, I haven't spent too much time with Prisma. I've mostly tackled Relay and Apollo. But you're right in the sense that what these tools do is take user input and translate that into the appropriate SQL queries for the data store. So they'll say, “All right, you're asking for this thing? Translate that into JSON so we can send that over the wire, and then GraphQL will say, “Oh, this is JSON? Cool! I'll turn this into SQL.” Essentially all these Apollo clients are doing is making it dead simple to write what would be a SQL query, but instead is just a structured JSON object.

[00:27:07] JM: Well, more broadly, how has GraphQL changed frontend development and the relationship between frontend and backend developers?

[00:27:16] AC: I’d say the first thing it's done is it's largely removed the use for frontend to really care about writing SQL. Usually if you think about, say, two tables that you have to grab information from to display in a component, you’d have to think about, “Oh, right! I need to join this thing.” If you're in Rails or you're in Django, you're thinking about table joins. Instead you

just think about, “I just need this thing from this object and this thing from this object, and I don't really care how it gets mashed up together. I just want it to be one sort of glob of data that I get back from the server.

So that's nice in the sense that you don't necessarily have to worry about optimizing SQL queries, but you still have to worry about things like fragments. So with fragments, that's one way of optimizing your queries to cache certain pieces of your GraphQL response so that you are getting only what you need and you're not repeatedly calling the same things over and over again. On the one hand, you’re not worrying about lower-level languages, but you still have to worry about things like performance and security. So in that sense, we’re just shifting responsibility from one language to the next.

I think the other thing that it does is it breaks apart the notion that everything has to obey Rest. So rest was sort of an older way of doing things. It's been popularized by things like Ruby on Rails. So with Rest, it was essentially translating your model objects into the various HTTP codes for fetching or modifying data, and the way to do that was very, very strict. There were essentially seven different ways that you could read or write data. If you are extreme inherent to the Rails rules of engagement with model objects, you would create really interesting controllers and nested calls for trying to grab the right pieces of information and it could get very complex and it also seemed kind of contrived. There are just certain times where you want to join two tables together and make them look like they’re one even though they’re stored differently in the database.

So the old way of doing things meant that you sort of have to work really closely with the backend developer to make the data work the way you want it to and display it properly. But with GraphQL, that largely goes away. You can now work with backend folks to essentially say, “Hey, I just need this data. If you can get it to me as fast as possible in any way that we can make it work, great. I don't really care about the structure. I don't really care about how ‘s t stored in the database or even what database it’s stored in. I just want to make sure that I can grab it.”

So it gives frontend developers more freedom to express the data that they want to display and it also gives backend developers more freedom to architect their applications in a way that best suits their architecture, it best suits performance and best suits the way that they want to

organize their data. I think it just really gives people more freedom to express themselves and express their code the way they want them to.

[00:30:33] JM: Are there any other trends that are pushing people to adapt GraphQL?

[00:30:38] AC: I think the trend of microservices has been a big push for this. With microservices, there's such a focus on small architectures that are all composed into one service or one platform. So when you have that many smaller services that are isolated, you're going to have more data stored in many different places. So GraphQL is great for tackling that. If you're dealing with a monolith application, may not be the best use, but if you're working in a microservices framework, which has become more popular over the last few years, GraphQL is definitely going to simplify things for you on the frontend.

[00:31:11] JM: So GraphQL and React, those are some higher-level aspects of how frontend has changed, and we've also done coverage of WebAssembly recently. WebAssembly will change frontend programming whether the developer is aware of it or not, whether the developer interacts with WebAssembly or not. Explain what WebAssembly is.

[00:31:33] AC: So WebAssembly is a way for developers to not have to write JavaScript in order to get code out into a browser, and the way that it does that is it essentially takes code, breaks it down into extremely fast and performant JavaScript and then runs that in the browser. It is a sort of middle bridge that turns what you might be developing on the desktop and turns that into something that you can run in the browser.

So a perfect example of this is if you are developing games and you're developing them on your desktop. You can use WebAssembly to create a gaming experience in the browser without having to write all of it in JavaScript.

[00:32:13] JM: I think in order to dive in to WebAssembly a little bit deeper, we can talk about the lifecycle of JavaScript code as it executes in the browser, which this was totally new to me when I started to look at WebAssembly a little bit, and then we did the shows on JavaScript engines and then you wrote about this a little bit, and it turns out that the way that JavaScript

executes in the browser is extremely complicated. Can you describe some details of the lifecycle of how JavaScript executes in the browser?

[00:32:45] AC: Sure. We'll try to make it not as complicated. I think a good way to think about it are just the idea that you have – Really, think about just a stack and a queue. Everything that happens in JavaScript is either a stack or a queue. So the stack is how you execute the main JavaScript code you write. When we say that JavaScript is a single-threaded application that runs synchronously, you're really thinking of that one stack that takes your functions and your variable assignments and it pushes them on to a stack.

So as the machine is reading JavaScript code, it says, "All right, what is this keyword?" "Oh, this is a keyword for a function? Cool! Put that function on the stack. Oh, and there're some variables inside of that function? All right, put that variable assignment on the stack." So you're building the stack up as you get deeper and deeper into the blocks of JavaScript code that you're writing.

But then you're naturally thinking, "Okay. Well, I don't just write synchronous code. There're things for async, promises exist, so how does that work?" So as we're building things on to the stack, if we find opportunities for asynchronous programming, like for example we make a call to set timeout or we're doing promises, that's where the queue comes in. So if it finds one of these keywords and it's ready to start handling asynchronous code, it's going to take that call, it's then going to put it in the queue and it's going to say, "All right, when I'm done taking care of everything on my stack, I'm going to start looking at the queue to see what's asynchronous code can I process next."

So even though it looks like things are asynchronous, you've spawned another thread to handle your promises or handle your set timeout, but everything always happens on that stack, that one stack. So we've got the code that you're writing in the browser. That gets run through the stack. So you push things on as you add more functions or blocks and we'd take things off as you execute that code. But now we've built up a queue of items that we have to process that have come from promises, or timeouts, or something that asynchronous. You then take that off, that queue, and then push those functions on to the stack.

For example, with promises, everything is essentially a callback. So you run some code and then some other code happens, and that's why they use the then keyword. Then execute some more code. When that then gets called, you then start using the stack all over again. Just like you would with regular synchronous code, we take that callback, we push that on to the stack and then start performing all the actions on the stack as you normally would. Really, if you just think about it with those two data structures, that's pretty much how JavaScript code runs.

[00:35:42] JM: That's great detailed understanding that you've just displayed there. We should also touch on JavaScript engines, which are – They have to do with garbage collection, and hot code paths, and basically making JavaScript that is going to run multiple times, execute more quickly. Tell me what a JavaScript engine does.

[00:36:05] AC: Sure. So the JavaScript engine is like the lower-level piece of what's happening in that code lifecycle. So when I say that things go on on the stack and then they get executed, but what is execution mean? That's where the JavaScript engine comes in. There're lots of different kinds of JavaScript engines. If you're running something in the browser, you're probably running V8 is an engine that resides within chrome. If you're running code in IE, which I hope you aren't, then you're running a JavaScript engine in Chakra.

So that Chakra is going to be the JavaScript engine that is running in the background for that, or if you're doing a desktop application, you're going to be running it in node. So what these engines do is they take that code and they break it down into smaller pieces so that the machine can run that code. Because with any kind of language, you have to turn what is human readable into something that is machine-readable, and the JavaScript engine is the process of breaking down that code into something that is readable.

So the three steps for that are the parser, the interpreter and the compiler. So the parser – And we should link to this in the show notes. I have a friend who actually wrote out how to actually do this in JavaScript in a few hundred lines, but effectively what happens is the parser will take keywords from your JavaScript code and it will say, "Okay, is this a special word here? Okay. That lets me know that this is the start of an expression, the start of a block, the end of a block. I'm performing some operation. I'm storing the variable," something like that. So those keywords let the language know something special is happening here.

So that parser will then create a node in a tree which then provides sort of a roadmap for how this code is going to be executed on the machine. So the parser effectively creates tokens to turn the human readable code into sort of blocks of ideas about how the machine should run this code. The interpreter then turns this into a tree. So it's going to say, “All right, now that I have all these special tokens and these special ideas here, I need to arrange them in such a way that this gets executed in the proper order and is dependent on the right properties.”

So the interpreter then interprets that parsing into something that the machine can read, and then it gets sent to the compiler, which then says, “All right, now that we have this order of execution of these special tokens, I need you to execute this code on the machine.” The compiler will then take these special – Everything that is now broken up into these tree pieces, it’s going to execute them in a certain order and it's going to take that code and say, “All right, I don't know what any of these means. I'm a machine. I need this to be spoken to me in much simpler language.” So we translate that into machine code.

We go over this morning the article how that machine code is optimized. Sometimes it's very easy for the machine to understand what's going on. Sometimes it's not very easy, and sometimes it wants to optimize it even further. But effectively that last step is basically saying, “All right, now that we have this organized in the right order with the right keywords, tell me what I'm actually doing on the machine hardware?” That runs, executes the operations and then everything gets sent back to your screen and then you see the output of that operation. That was a lot. Did that make any sense? I felt like I was just –

[00:39:31] JM: Yeah. Yeah. No, it makes sense. I think this kind of stuff is obviously hard to digest over a podcast, but I think what's useful about doing podcast about these subject is the degree to which people can have multiple avenues for exploring them. So I think if somebody really wants to understand JavaScript engines in-depth, then it's helpful to listen to a podcast and read an article about it. But more to the point, why does any of this have anything to do with WebAssembly if WebAssembly is about not writing JavaScript?

[00:40:05] AC: So the idea is that there's this compile stuff that has to get to the machine in order to execute your code. The truth is it doesn't have to be from your JavaScript that you’re

writing in the browser. It could be from something in the desktop. Maybe you like writing Rest. Maybe you like writing Closure. It shouldn't necessarily be the case that you have to write JavaScript to get dynamic functionality in the browser in theory.

So I think what WebAssembly is attempting to do is say, “It no longer matters what language you use to create interactivity in your browser. Instead, we’re just going to go ahead and say, “You know what? Use any language you want. We’ll take care of compiling it down and away.” The JavaScript engine understands the code that you’re executing.

[SPONSOR MESSAGE]

[00:41:00] JM: We are running an experiment to find out if Software Engineering Dailydaily listeners are above average engineers. At triplebyte.com/sedaily, you can take a quiz to help us gather data. I took the quiz and it covered a wide range of topics; general programming ability, a little security, a little system design. It was a nice short test to measure how my practical engineering skills have changed since I started this podcast. I will admit that though I've gotten better at talking about software engineering, I have definitely gotten worse at actually writing code and doing software engineering myself.

But if you want to check out that quiz yourself, you can help us gather data and take that quiz at triplebyte.com/sedaily. We have been running this experiment for a few weeks and I'm happy to report that Software Engineering Daily listeners are absolutely crushing it so far. Triplebyte has told me that everyone who has taken the test on average is three times more likely to be in their top bracket of quiz course.

If you're looking for a job, Triplebyte is a great place to start your search. It fast tracks you at hundreds of top tech companies. Triplebyte takes engineers seriously and does not waste their time, which is what I try to do with Software Engineering Daily myself, and I recommend checking out triplebyte.com/sedaily. That's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte, byte as in 8 bits.

Thanks to Triplebyte for being a sponsor of Software Engineering Daily. We appreciate it.

[INTERVIEW CONTINUED]

[00:42:58] JM: So is it mostly important for this ability to write code in a language that may not be JavaScript, or are there other benefits that WebAssembly will unlock to improve the applications that we have?

[00:43:14] AC: I think the big thing, and we're seeing this now with games, is that WebAssembly is letting us write performant code, and JavaScript isn't necessarily the best way to write performant code. So we've attempted that before with asm.js, and that's basically like a very strict subset of the JavaScript language to write fast code that is highly optimized, but there other languages that are better than that. Rust and C++ come to mind as languages that are very good at handling low-level systems programming. For things like games, you're going to want that performance in order to have an enriching experience.

So WebAssembly is important and that it's going to allow us to create more enriching experiences that still perform well in the browser, because the limits currently in JavaScript to make it such that it's difficult to write these programs just in JavaScript. It's a lot better to write them in Rust. Compile them down into highly-optimized machine language such that JavaScript doesn't have to be the middleman to write something great in the browser.

[00:44:24] JM: Have you looked at that WebAssembly on Ethereum stuff at all?

[00:44:29] AC: I have tried my best to stay away from blockchain stuff for the moment.

[00:44:33] JM: Okay.

[00:44:34] AC: I don't know if the opposite of HODL is, but I am not that person.

[00:44:40] JM: Right. Okay. Fair position. But speaking of games, you've written about VR, and I guess if you're dodging Bitcoin these days, that gives you plenty of room to explore the other slightly overhyped technology, or maybe not overhyped, maybe just takes a long time, maybe similar to Bitcoin in that fashion. So you've written about VR. Do you use VR regularly?

[00:45:06] AC: Yes. I don't know what the HODL is for VR, but I'm HODLing big time on VR.

[00:45:11] JM: You're HODLing VR. Okay.

[00:45:12] AC: I'm HODLing big time on VR. I think VR is the future. To be fair, also, I think blockchain is also a future, but I don't think Bitcoin is it, and I don't think Ethereum is it. This is my 30-second thing on all of that is that these are great technologies, I just don't think we've found that use case that is going to be universally adopted yet, and that's the same with VR honestly. We're not there yet with VR either, but that's sort of the cycle of a lot of big waves of technology shifts, is that –

[00:45:41] JM: But crypto – Okay, the thing with crypto is like I've really tried to delve into this, not with the depth of like a crypto entrepreneur, but like I've done a bunch of shows on it and honestly I'm like, "I don't really know." I'm okay saying, "Honestly, I do not know. Maybe it's Bitcoin. Maybe it's Ethereum. Maybe it's anything, something in the future. It's a futuristic one," but I don't really think it's one of these things where we can say, "Oh, just look at the past. This is just like Webvan, like the first iteration of these technologies. It's clearly not going to work." I think it's kind of unprecedented. Do you have actually a framework around that or is it more like a gut feel?

[00:46:23] AC: I don't know who coined this, but there's this whole lifecycle of technologies. It's like a graph and it talks about how the early – There are early adopters and then there's like the mass adoption. There's is whole cycle of like there's a lot of early interest from early adopters and people who love tinkering with new technology and then it kind of has this nadir or it dips down and everybody thinks it will never work, and then someone comes along with something that actually makes it universally adaptable and then you see a great rise in that technology. I think a lot of pivotal technology goes through that, and I think blockchain will do that as well, and that's important distinction too. I'm not necessarily bullish on cryptocurrencies, but I do think blockchain as a technology is very smart. The idea of triple point accounting I think is the real excellent piece of innovation that goes into all these things.

So that's what I mean when I say that I don't necessarily think Bitcoin will work, but I think the idea of having a neutral third-party and a universal ledger is a really smart idea. I just don't think we've found that application of that technology that has that universal adaption metric yet.

[00:47:32] JM: So you think it could be like the Amazon. They release some quantum databases. It's like a distributed ledger that is basically on Amazon. So instead of trust, like the thing – What you need a currency for is the currency incentivizes people to have that trust layer. So you do need a currency for this kind of public blockchain, but if you just say, "Okay, we're not going to trust the public. We don't care about the public. We're going to trust AWS." Then you can use Amazon's new database technology. So is that the framing in which you're saying blockchain, not Bitcoin?

[00:48:08] AC: One way to think about it is that anything where you have to keep public record on something, that's the whole idea behind triple point accounting. Right now, we live in a world of double point accounting, which is where the bank has an idea of how much money you have and you have idea how much money you have, and there's no universal way of debating that. It's basically you versus the bank.

With triple point accounting, you basically have one giant receipt of everything that's ever happened in that system that everyone can reference at any time. They just don't know who exactly traded what. They just know that, "In this pool of resources, we did X, Y, and Z." That could be abstracted to things that aren't necessarily currencies. I would imagine law would be useful for blockchain technology to have a universal ledger of everything that happened in terms of laws changing, or in terms of how people voted on things.

So that's kind of why I'm not exactly thinking about it, like cryptocurrencies, or thinking about it like the Amazon example. I'm thinking about it in terms of, "What's a way to keep track of all the things we're doing as humans?" I think blockchain is a great way of making it easier to hold people accountable for the transactions they're performing. I think that was the big reason why people clung to crypto currencies, because they felt disenfranchised by the way that the banks had handled themselves in 2008 and they wanted a way to say, "No, we want to take responsibility for the way that we handle our money."

So I think that's why cryptocurrency was the first big thing that come out of blockchain and that's why I also think that blockchain will stick around, but it may not necessarily manifest as Bitcoin crypto currencies. It may be some other cryptocurrency. It may be some other use of the ledger that isn't even related to economics. That could be something that people just go, “Oh, wow! How did we not live with this before?” So that's kind of why I think of it one higher level of abstraction from cryptocurrencies and more about blockchain technology in general.

[00:50:10] JM: Okay. Well, anyway, HODLing VR.

[00:50:12] AC: Right. Yeah. VR is a much easier case for me to get behind, and it's one of those things – I had a friend tell me one time that he said that, “You can't really see the awesomeness of VR until you actually experience it,” and it completely – I believe that to my core. I was very much skeptical of VR. I was like, “Well, I really only see this being used for games,” and the last time I did VR I use like a Game Boy VR, which was from like 90s. Yeah.

[00:50:40] JM: You mean Virtual Boy.

[00:50:41] AC: Virtual Boy. Yeah, that was it, and it was like –

[00:50:44] JM: The red thing.

[00:50:44] AC: Yeah. It was like you were in Tron, but everything was red. It was just like a projector screen. Like it wasn't – It's nothing like it is today. So I was like, “All right, I don't know if I'm so into this,” and then I actually tried an Oculus and I played this game where it's like you're in the matrix. You’re really like – People are shooting at you, and time slows down and you grab bullets and you literally throw them back at them. I was like completely blown away. Then I played this game where you’re rock climbing, and I rock climb, and I was like, “That's fine.” I get in and you’re like thousands of feet above the ground and my body, like the hairs on the back of my neck stood up, and I’m a very rational person and I was like, “What is going on?” This doesn’t make –

[00:51:28] JM: Wait. So it’s like you wear a VR headset while you're in a rock climbing gym?

[00:51:32] AC: No. No. No. No. No. You're in a rock climbing simulator game. I was just in a regular room, but that was the crazy part, was I was in a regular room, I'm a rational person who knows I'm in a regular room and yet my body is having his nervous reaction whenever I would look down and see that if I let go of my left hand, I'm falling thousands of feet. In my head I'm like, "This is fine. I'm in a room. I'm standing in a room. I can feel my feet on the ground," and yet I would instinctively like move my legs as if I were rock climbing when I was pulling my hands up in this game. It's amazing how much VR can transform your life and put you in the scenario you're in even when you're thinking to yourself, "I'm not in this subway car. I'm not on this mountaintop. I know I'm right here in a room," and yet your body is totally tricked.

If you get in the really immersive experience, it just blew me away. As soon as I did that, that's when I realized that VR, in some form or fashion, is going to transform the way that we live our lives. That's where I'm going to definitely HODL and bet. I would bet my career on it. I mean, that's why I've started focusing on VR for side projects. It's why I started developing for it. I think if you're a UI developer or you're in the frontend, that's the next major frontier, because we've already conquered the desktop screen. We've already conquered the mobile screen. The next screen is your eyes. It's going to be what you're doing in your day-to-day life. It's the goggles you're going to be wearing. We're going to be augmenting our lives with these screens so that we don't have to hold them anymore.

I wouldn't be surprised if in 20 years our kids or our grandkids are looking at us going, "Why did you have this thing in your pocket? Why don't you just look at it with your eyes?" Again, that's a very far future from now and I think companies like Magic Leap are the very first at handling augmented reality, but this is the best time as a developer to get involved, is when it's booming and it's on its way up, not when it's already been mass adopted.

So if you want to get onboard and be someone on the forefront of a burgeoning technology, VR isn't going to go away. Something like Blockchain, and it's the last I'll say on this, is there have been negative consequences that come out of that. People have lost significant money and they've lost trust in how something like these technologies work, but I don't see those kind of negative downsides with VR. I mean, you spend maybe $400 on a Rift and you play some games. That to me isn't like the end of the world scenario in terms of negatives that could come

out of VR. If anything, I find it to be a more enriching experience and a way to help us engage even better with the world around us.

[00:54:16] JM: I think it's like we don't even know how to confront the issues that we're having societally with smartphones, and I think there're some amount of fear around VR. Even me, I am like pretty much a techno liberal, techno optimist kind of person and I just – I've tried VR one time and it was just a really – One of these really stupid simulations where like you're wearing the headset and you like point and click and you move to a different point in the room. It's like really simple. I was like, "Oh, no! This is too much for me." It's overwhelming to acknowledge it, and to kind of get into the VR world would be would be to have to confront something that I was a little bit – Like I'm not sure how to deal with this. So I'm just kind of like, I guess, just biding my time a little bit.

But anyway – Okay, so what is the developer experience like for VR these days? Because I do agree with you. I think this thing is here to stay. There's going to be more and more adoption. I don't know what that adoption curve is going to look like. But what is the developer experience like today?

[00:55:19] AC: I think there're two major camps you see right now. Here are the power user hardcore VR developers who pretty much come from a gaming background. So there are frameworks like Unity 3D, and they are out there to push really immersive experiences on the desktop. So if you're developing for Oculus, if you're developing for HTC, you're developing effectively what is a 3D game and you're using pretty heavy duty rendering engines and desktop software and you're pretty much developing in C# or Java. So sort of the more traditional desktop applications with a focus on graphics and 3D math.

If you're interested in that stuff, that's pretty much where VR is that right now. But if you're a frontend developer, there are ways to get around that. React 360 and A-Frame are the two major web platforms for developing 3D applications, and these are sort of like the next level abstraction on top of the 3D graphics engines. So you have something like three.js, which is a 3D graphics library for handling 3D games in JavaScript. React and A-Frame are effectively built on top of those with the mindsets of the frontend developer. I've done demos in both A-Frame

and React and it's amazing how, basically with a few lines of code, you can put the cube in front of your face and it actually looks 3D.

So I think what's cool about these frameworks is that they have severely reduced down the barrier of entry for someone like a frontend developer to start getting involved in these things. I made a demo for a talk one time and I think I wrote five lines of HTML. I didn't even write any JavaScript, and I had a hello immersive world application to demo people on their phones and people were just blown away and they were saying, “Wow! How long did that take you to write? How much code is there?” I was like, “Well, I know this is going to sound embarrassing, but I wrote zero JavaScript for this. It was all based on the framework,” and people were blown away.

The fact that you could do so much with so little was amazing. So then think about what would happen if you actually started developing a real application? I mean, the sky is the limit for frontend development in VR. So it's definitely not something you have to have this academic graphics background and work at Pixar for 10 years as an animator in order to get involved in this kind of thing. You can be a frontend developer and start working with React 360 or A-Frame right now.

[00:57:41] JM: So you're saying that the frameworks for dealing with VR are pretty high-level and pretty approachable these days. Is there some reason why developers aren’t going at it? I guess just because the adoption is not there? What is the developer adoption like, or what's the developer community of VR like these days?

[00:58:03] AC: I wrote about this recently, and I actually think it's a great time to be involved, because it's still very weird, and what I mean by that is if you remember developing HTML pages back in the 90s, people weren’t making very finely polished websites. To the point where there's this whole aesthetic of what a 90s website looks like, it has like a marquee object that has blinking texts and it scrolls across the screen and there's a lot of cheesy looking animated gifs. There's an aesthetic that comes along with something like HTML from its heydays.

So I think that's the era we’re in right now for VR. So there are communities right now like Super Medium, that's based on a company, the two guys that developed A-Frame from Mozilla started a company that is funded by Y Combinator and they have a whole community called Super

Medium, which lets you post all of your cool 3D VR interactive applications that are from the web.

So, yeah, they're weird and they're quirky, but this is that time now where VR is about expression. It's not about serving some commercial use, and I think that's kind of how the web worked from the 90s into now. Very early on, the way that you access the web that we know today was you were on AOL 30 and you click the internet button and then you were like, "Okay, what do I do now? I don't even know what to type in," because you have like a sports tab and you had a games tab and then you had the internet tab and you're like, "Okay, this is too much choice for me. I don't know what to do."

But if you found a website, it was usually pretty weird. It was pretty interesting, and it was a means of expression. Whereas now, the web is just – Every interface looks exactly the same. It's highly refined. It's meant to be about serving a purpose, getting you to buy something, getting you to do a transaction, getting you to exchange money, whatever it is. But in its early days, it was really fun and interesting and there were no rules.

That's why VR is awesome right now is because it's not at that mass adoption phase yet. It's still very weird. We haven't really found the real-world applications for VR in terms of a business perspective outside of games. So you're finding that there're these people that are posting a lot of really creative stuff that doesn't necessarily serve a purpose, which may mean that you can't use it for work yet, but it also means that you can use it as a form of self-expression and help push the technology forward so that when it is useful to businesses, you not only have all the experience, but you have the expertise and understanding how this technology works and you've got a portfolio to back it up even if it doesn't necessarily fit a business purpose. I would argue that not everything has to fit a business purpose for technology to be interesting or useful to people.

[01:00:50] JM: All right. Well, we've run the gamut of frontend technologies, or many of them at least. What are the other frontend technologies that you're excited about, or are there any other technologies you're excited about?

[01:01:02] AC: Well, VR is definitely the one that I am most interested in pivoting towards, especially because I focus on user interfaces and the frontend. So I think that's kind of like the next interface. I'm very much a fan of interdisciplinary work. So bioinformatics I think is really interesting. There's a lot of work being done on the genome. Where I work at Indigo, like we're focused on ag-tech, so integrating technology with agriculture. I mean, this is like a whole new world of technology that I didn't even think existed, and yet it's a multitrillion dollar market.

So there're all these ways of integrating technology now. It used to be the case that technology was a thing. A business was founded on just technology, and now technologies become so ubiquitous that it's in everything. So now technology is an integration into every other business vertical. So the fact that we're seeing it complement so many other industries, that to me is really fascinating.

Integration with other business verticals I think is really, really cool. I'm really interested in the psychology of technology. One of the thing that's really cool about VR was I saw this study about how they're using virtual reality technology to help people with phobias. So if you have a fear of heights, there are psychologists who will actually put you in a VR headset and put you in scenarios. It's called immersion therapy, and the idea is that you have to immerse yourself in your fear in order to get over it with increasing amounts of exposure. So VR is a way to gently introduce your fears in a way that's controlled and easy to get out of.

So they're finding that VR is a way to actually help people get over their fears by putting them in a very safe environment to expose them to these issues. So if you want to talk about real-world applications in medicine or therapy, VR is great at that. But just the fact that it's not just a technology now, that it's actually cross-discipline with psychology and psychiatry, that I think is really, really cool. Just seeing how all these different industries and topics are blending technology is really, really cool, and everything has an interface.

So if you're a frontend developer, you can get involved with basically all of these, because you have to have some way of interacting with that software. So I think it's a great time to be a frontend developer and I think it's a great time to explore frontend in a lot of different industries and verticals.

[01:03:31] JM: Adam Conrad, thanks for going on the show. It's been really great talking to you, and also thanks for writing for Software Engineering Daily. I think you're a fantastic writer, and I look forward to doing more coverage with you.

[01:03:43] AC: Thanks, Jeff. Have a good one, and happy holidays.

[END OF INTERVIEW]

[01:03:48] JM: Software engineers can build their applications faster with the use higher-level APIs and infrastructure tools. APIs for image recognition and natural language processing let you build AI-powered applications. Serverless functions let you quickly spin up cost-efficient, short-lived infrastructure. Container platforms let you scale your long-lived infrastructure for low cost, but how do you get started with all of these amazing tools?

What are the design patterns for building these new types of application backends? IBM Developer is a hub for open source code, design patterns, articles and tutorials about how to build modern applications. IBM developer is a community of developers learning how to build entire applications with AI, containers, blockchains, serverless functions and anything else you might want to learn about.

Go to softwareengineeringdaily.com/ibm and join the IBM Developer community. Get ideas for how to solve a problem at your job. Get help with side projects, or think about how new technology can be used to build a business. A softwareengineeringdaily.com/ibm, you will find the resources and community who can help you level up and build whatever you imagine.

Thanks to IBM Developer for being a sponsor of Software Engineering Daily and for putting together a useful set of resources about building new technology. I'm always a fan of people who build new technology, who build new applications, and this is a great resource for finding some design patterns and some new ways to build and leverage these technologies, like serverless, and containers, and artificial intelligence APIs. So thanks again to IBM Developer.

[END]