EPISODE 612

[INTRODUCTION]

[0:00:00.3] JM: When you write a front-end application in JavaScript, you assume that it will run on the user's browser, but different browsers are compatible with different versions of JavaScript. If you write an application in the most recent version of JavaScript, you might take advantage of new language features. A user who's running a old browser can only interpret the older features of JavaScript. To solve this problem, JavaScript developers use a tool called Babel. Babel transpiles one version of JavaScript into another version of JavaScript, so that the code can run on a target browser that would otherwise be incompatible.

Henry Zhu is a core maintainer of Babel and he joins the show to describe how Babel works. Henry works on Babel full-time and he's supported through Patreon and OpenCollective. This is a newer model for employment and Henry describes in this episode how he ended up working on open source full-time, as well as the costs and the benefits of living that open-source lifestyle.

Before I start the show, we're looking for a videographer. We're also looking for writers and a couple other jobs. If you're interested in working with us, go to softwareengineeringdaily.com/jobs. If you're interested in a lower commitment way of interacting with the Software Engineering Daily community, we have the open-source ecosystem, which is available at github.com/softwareengineeringdaily. We have open source apps for Android, iOS, as well as the web. The iOS app just got a major improvement with some added features of downloads and saving episodes offline, as well as a news feed within the app, so do check out that Software Engineering Daily app for iOS. If you like the podcast, you might like the app, and we'd love to have you as part of the Software Engineering Daily open source community. We hope you get involved.

[SPONSOR MESSAGE]

[0:02:09.1] JM: At Software Engineering Daily, we have a web app, we have an iOS app, an Android app and a back-end that serves all of these frontends. Our code has a lot of surface

area and we need visibility into problems that occur across all of these different surfaces. When a user's mobile app crashes while playing a podcast, or reading an article, Airbrake alerts us in real-time and gives us the diagnostics that let us identify and fix the problem in minutes, instead of hours.

Check out airbrake.io/sedaily to start monitoring your apps free for 30 days. Setup takes only a few minutes. There's no complicated configuration needed. Airbrake integrates with all of your communication tools, from Slack, to Github, to Jira and it enhances your current workflow rather than disrupting it. You can try out Airbrake today at airbrake.io/sedaily. If you want to monitor and get visibility into the problems that may be occurring across your application, check out Airbrake at airbrake.io/sedaily.

Thank you to Airbrake.

[INTERVIEW]

[0:03:28.6] JM: Henry Zhu is a core maintainer of Babel. Maybe I should say the core maintainer. Henry, welcome to Software Engineering Daily.

[0:03:35.4] HZ: Thank you for having me.

[0:03:36.4] JM: Babel is sometimes difficult to explain, but I think the place to start is the fact that different browsers are compatible with different versions of JavaScript, why is that and why is it important?

[0:03:51.8] HZ: Yeah, I think the browser landscape is pretty unique. There are multiple implementations of the JavaScript, or what we say ECMAScripts specification. Unlike, say more of a server-side lane, or even though we have node now like Python or C, everyone uses that thing, even though there might be multiple compilers, but with the browsers, each vendor they try to implement the spec the best they can, but they have different teams, sizes, different release schedules. You can't really know what they're supporting, unless you look at their sites. More importantly, you don't know what browser your users are using.

If you support all the major browsers, or even IE 11 or less, then you can't assume what they're allowed to do. Babel is this tool that lets you support the widest range of browsers, but also lets you write the latest syntax.

[0:04:47.8] JM: When you think about different versions of Java for example, I think many of the advances that come with a newer version of Java, at least we talked about like from Java 7 to Java 8, I think were in the flavour of syntactic sugar. Java 8 was comfortably backwards compatible with Java 7 and the versions that came before it. I don't think in the Java world, you need something like Babel, where you have these transpiled different versions of Java that can port to different places. Why is it important to have a transpiler, because that's what Babel is? It allows different JavaScript versions to be compatible with another, and you can transpile one – a newer version of JavaScript into an older version of JavaScript. Why is that important?

[0:05:41.2] HZ: Yeah, I think there are a few reasons. With Java, it's like on your end to upgrade. If your company's on Java 7, they want to go to 8 or 9, you just need to update all of your code to do that. With JavaScript, you have no idea what browsers your users are doing, so you can update your own code to the latest version, but it doesn't mean that it will run on your users' browser. In order to do that, you would have to either wait until somehow you know that, but you can't really guarantee. A tool like Babel lets you standardize and not have to think about what browsers you're targeting.

Also importantly, the reason why we have transpilers is there is – it's twofold, one is this backwards compatibility story, but the other is that transponders can actually help influence the language itself and I think that's the second more important role of Babel is that unlike other languages, where you – there's a committee of people, which there are for JavaScript as well, but instead of waiting for the implementers to do all this work in C++ land in the browsers and then show developers and then make it, at that point it's almost too late for real feedback. With Babel, it's like we can implement a plugin, because Babel is written in JavaScript and users can test our ideas, give feedback, like this is not even a good approach at all, or it's not intuitive. That way, we can more get the community's feedback at a faster pace or even just make sure you're on the right track.

[0:07:18.4] JM: Yeah, and we'll talk about that role of Babel as a way for developers and standards bodies to interact with each other. Just to talk a little bit more about the developer side of things, so Babel is a project that transforms newer JavaScript syntax into backwards compatible code. If I'm a JavaScript developer, how does Babel fit into my workflow?

[0:07:45.4] HZ: Yeah, so for most at this point, a lot of it is the fact that most people are already using Babel, and a lot of times, people don't know what the project is because it's so – it's infrastructure level now. It's like, all the tools that you use are already using it, so you might not even know that you're using it, because it's already built into some framework or tooling that you're using. If you use React, you might use something called create react app, or if you're using Vue, there's vueCLI, or emberCLI, all these tools already have Babel built in.

One of the issues we have as a project is that it's almost too ubiquitous, too "successful" in a way. People don't realize they exist, and then it also means that they might assume that it's like a company, or a lot of people are working on it when it's just a few volunteers and crowdfunding.

[0:08:39.3] JM: Yeah, so if I'm writing a react application, I get to use the nice world of react components and building in the react developer ergonomics that that people love. What developers don't often know is that part of the reason that that's such a nice experience is that under the hood, one of the things that the react compilation to JavaScript, when because react itself is not JavaScript, it's you're writing stuff in JSX and then it gets translated into HTML and JavaScript, but one thing that's happening is that it's taking care of that cross-browser compatibility for you out of the box and it's doing that through Babel.

[0:09:24.9] HZ: Right, exactly. JSX is not part of JavaScript and yeah, Babel will convert that into a function call. You could not do it, but the point is that instead of doing it manually or not writing it at all, it lets you have this nice way of writing react, yeah. It's not completely necessary. It's just people are used to that and it's a nice way of doing it, and it does it for you automatically. You're also already going to compile yes, I think, so it's like, you get the JSX part for free.

[0:09:53.5] JM: Babel generates the right version for the right browser at the right time, depending on what browser is consuming a given application and what version of the browser is running on the user's device. How does Babel do that? Does it translate the code into a bunch

of different versions and then deploys all those versions, or it somehow senses what the user agent is and gives the user the right version? How does Babel generate the right version of JavaScript for the right browser at the right time?

[0:10:26.6] HZ: Yeah, so that thing is, it doesn't actually do that out of the box. It's more low-level. You can specify what browsers you want to target and it will compile into that. It's like, think of a spreadsheet of every possible browser version and also every different syntax. If it's arrow function, is this supported by browser XYZ, or version XYZ, or classes, or different kinds of syntax.

Then it will do these common denominator of what you support. If you want to do what you were describing earlier, you need a lot more tooling on top of all that. I'm not sure if a lot of people are attempting this, but it's definitely been suggested and talked about a lot about should we read the user agent? They used to do some weird try catch and an eval, where you try to see if the browser supports that syntax. Or do you do something completely different, where you just, okay I'm just going to assume that the baseline is IE and then you ship as much code as possible. People are realizing, yeah, we shouldn't have to do that if we're able to detect certain things.

The problem with the whole user agent thing is well, it's hard to know if that is actually accurate. Even if you are, creating a different bundle for every single version can be difficult in the sense, now your builds might be longer, because you support 10 browsers, you're going to build your website 10 times and then testing 10 times. Then if you have a bug, then it might be harder to realize where is that coming from. Is it from Babel itself? Is it because you wrote this weird code? Is it compiled differently? That just gets a little more complex. Maybe some people have a better way of doing that, but that's my thoughts on that.

[0:12:12.7] JM: Babel transpiles JavaScript from one version to another. That term transpiler, how does that differ from the term compiler?

[0:12:22.2] HZ: Yeah. I think there's a lot of bikeshedding on that, and a lot of people get into arguments over whether we should use –

[0:12:29.1] JM: Sorry, bikeshedding?

[0:12:30.6] HZ: Yeah. Just like whether should Babel be called a compiler or not. We actually changed all the documentation, I think a while ago so that it just says Babel is a JavaScript compiler, because it gets confusing, because it's a newer word anyway. People would say source-to-source compiler, or it's in the same language so that's why we call it a transpiler. Right now I just say it's a compiler. It just happens to compile to JavaScript, instead of a lower level language. Yeah, I don't think it's not necessarily that big a deal whether we say transpiler or not. Yeah, it's still – it changes a form of a code into another form.

[0:13:11.5] JM: Right. Okay, so this transformation, or compilation, or transpilation, or whatever we want to call it, transforming, converting one piece of code into another piece of code. This takes three stages; there's the parsing stage, the transforming stage and the generation stage. Describe these three steps.

[0:13:31.8] HZ: Yeah. I think, you put it well that there's in high-level, there's only three steps. A lot of people have a hard time grasping what a compiler is, or what it does. I never took a compiler class, or studied it formally or anything. I just learned everything.

[0:13:47.6] JM: You should be thankful.

[0:13:48.4] HZ: Yeah. Maybe that helped, right?

[0:13:50.2] JM: At least my programming languages class, oh my. I can't believe I had to pay money for that. It was the most – I'm sorry. I hope the professor that taught me that out there is not listening, but it was just abhorrent and so boring and so unapplied. I think compilers is one of this – it is why I'm glad to have you on the show is because, compilers I think has a reputation as being super academic, super dry, super uninteresting. When in fact, it's a very vibrant and highly applied area of computer science.

[0:14:24.6] HZ: No, I totally agree. I almost purposely didn't choose to do computer science as my major in college, because of that, and knowing that it's going to be so boring. Even though, I

had a lot of interest in computer science and programming, but I just wanted to do visualizations, or just games, stuff like – things that seem fun.

[0:14:42.8] JM: What you study?

[0:14:44.0] HZ: I studied, it's called the industrial and systems engineering. It's more like – more math, statistics, queueing, transportation. Very different, but also related to program in a way.

[0:14:59.4] JM: Oh, yeah. I know a lot of super creative engineering leaders actually that did industrial engineering. It's one of those majors that you don't really hear about as much, but you find all the time. It's just you find people in power who were philosophy majors. It's just like, "Okay." That surprises you sometimes, but it sounds like a pretty interesting major. There's a lot of cross-disciplinary coursework in that thing.

[0:15:28.3] HZ: Yeah. You were able to take – there were a few programming courses, specifically you could take like, whether it was databases or machine learning, but yeah, I didn't have to – it was fine, because I was able to just choose electives that I liked just for fun outside of that. Yeah, I didn't – I hated the idea of wanting to learn about compilers, and now ironically, I work on one and I like it.

[0:15:51.9] JM: Okay, so parsing, transforming and generation; the three stages of the compiler of Babel.

[0:15:58.1] HZ: Yeah. The way I like to think about it is when you want to transform your code to do something, you could do a simple find/replace, right? Say I want to turn all of the variable names like A into B, you could go through your editor of choice and then do like control+F and then find all the As and change them to Bs. The problem is that, you don't know if the A that you're looking for is a variable, or is it in a string, or is it in a comment. The point is that you want to semantically know if the A that you're using right now is a variable, or a comment, or a string, or something else.

What we do is we use intermediate representation of the code. Instead of a string, we use what we call an abstract syntax tree, an AST. An abstract syntax tree is just a separate form that lets

you know what part of the code you're in. The way I think about it is like in English, or languages we have, like that whole – your sentence grammar, tree structure where it's like, “Oh, this is a verb and this is a noun, like parts of speech and all that stuff.” In the same way, we have that with code. It's just defined more formally.

In the three stages, you're just trying to convert the string into an AST. The parser is the piece of the tool that converts your first – your file that reads the string, it turns it into an AST. Then the second step is the transformation phase. It just is the part where you have that AST and you change the AST around. Then the last step is the generator, which is you can just think of as a printer. It takes the AST and then it spits it back into a string.

Before with the find and replace thing, you're just taking a string and then making a string again, but we have this intermediate step where we change the form of it into an AST, change that and then print it out again. Also, an AST it sounds really crazy abstract syntax tree, but the way I think about it is just JSON. A lot of us use JSON to send data around. We know that there are nodes and there's properties on these nodes, and all we're doing is just moving the data in there, because it's easier to manipulate.

[SPONSOR MESSAGE]

[0:18:24.5] JM: In today's fast-paced world, you have to be able to build the skills that you need when you need them. With Pluralsight’s learning platform, you can level up your skills in cutting-edge technology, like machine learning, cloud infrastructure, mobile development, DevOps and blockchain. Find out where your skills stand with Pluralsight IQ and then jump into expert-led courses organized into curated learning paths.

Pluralsight is a personalized learning experience that helps you keep pace. Get ahead by visiting pluralsight.com/sedaily for a free 10-day trial. If you're leading a team, discover how your organization can move faster with plans for enterprises. Pluralsight has helped thousands of organizations innovate, including Adobe, AT&T, VMware and Tableau.

Go to pluralsight.com/sedaily to get a free 10-day trial and dive into the platform. When you sign up, you also get 50% off of your first month. If you want to commit, you can get $50 off an

annual subscription. Get access to all three; the 10-day free trial, 50% off your first month and $50 off a yearly subscription at pluralsight.com/sedaily.

Thank you to Pluralsight for being a new sponsor of Software Engineering Daily. To check it out while supporting Software Engineering Daily, go to pluralsight.com/sedaily.

[INTERVIEW CONTINUED]

[0:20:03.6] JM: That term abstract syntax tree, I think we could break that down into what it is. Abstract, it's a higher-level, easier to interpret from both a human point of view and the computer's point of view representation of computer code as it's going to be evaluated; that's abstract. The syntax is you're referring to something that is related to language, that's a lingual structure and it's a tree, that's a data structure. You could represent this as a non-tree structure. You could probably represent it as a hash map, or a graph if you wanted to, but this happens to be in a tree format. When you think about those terms; abstract syntax tree, it's actually not the most complicated thing in the world. It's just a data structure that represents programming language code.

[0:20:52.3] HZ: Yeah. I think that's a great summary for sure.

[0:20:55.0] JM: Okay. The abstract syntax tree gets built. When it gets built, is there a place where the human needs to look at that abstract syntax tree and do something with it to manipulate it – as the programmer, if I'm a programmer that's building something that required, let's say I'm a react programmer and I'm interfacing with the point in the react compilation code that uses Babel. Do I need to know how to manipulate that abstract syntax tree?

[0:21:27.4] HZ: Well, if you're just writing a web app, then I guess in some sense you shouldn't need to know how any of it works.

[0:21:33.4] JM: Sorry, what I meant is – let's say I'm a react core developer that is building code for that cross-browser compatibility.

[0:21:41.3] HZ: Actually, even if you're a library author, you wouldn't need to know it. If you're trying to make your own Babel plug-in, or change how it works, then yes, you definitely need to understand that part. The react, like JAS as itself has its own spec and whatnot. It has different AS nodes and different names for all these parts. If you have a JSX element, like a button, right? The capital B button, and you have there like your less than sign and button and greater than sign. Then that itself has a name, right? There's JSX element and inside, there's JSX opening tag, JSX closing tag, and then there's JSX attribute. There's different names for these things.

When you're talking about what these things are, it's a good way to at least use those terms. If you're trying to make the code that actually transforms that into JS, then yeah, you'll need to understand those terms.

[0:22:39.0] JM: What's a Babel plug-in?

[0:22:40.1] HZ: Yeah. A Babel plug-in I would just say is one of the individual steps in the second part that we were talking about, where we have the parser and then we have the transforms in the middle, and then we have the generator. The plugins are just individual transforms. A plugin is just the fact that Babel is – it doesn't really do anything by default and every plugin, or every transformation of syntax can be its own plugin that you can even write yourself.

Even the core plugins are not any different from a plug-in that you would write yourself. It's just the ones that we give to you that you can add, but you can fork them and you can make your own plugins, you can change whatever you need. It's just JavaScript file that finds – yeah, it will just convert syntax. The way it actually works too, we use this thing called a visitor pattern, which is a common thing that compilers do. Basically, when you're writing your plugin, it's like – I think of it like jQuery actually. jQuery, when you're using it you have the dollar sign and you put stuff inside of that function call. What you do is you select elements on the page and the dom, and in the same way, we do that with Babel, but with the nodes in the AST.

When you're using jQuery you might like, “Oh, I want all the A tags, or all the H ones and I'm going to do something to that, right?” In Babel, it's like, “Oh, I want to find all the functions, or all

the classes or all the arrow functions specifically and I want to do something." In the same way you're trying to find something; you "visit" a node, and then you either change it, or you just analyze it, or you can throw an error, stuff like that.

[0:24:27.2] JM: You need to do this changing to the AST in the event that may be an old browser is going to react to – or I should say respond to disambiguate. In case an old browser is going to respond to some portion of my code in an unfriendly way, I want to be able to handle that response in a specific way, is that right?

[0:24:51.3] HZ: Yes. For the core reason that people use Babel, yes. If you sent an arrow function to an older version of IE, it would just throw an error and be like, "I don't know what the syntax is." We would find all the arrow functions and then turn them into regular functions. Also, I would say that you can use Babel for things that are not really about supporting all browsers. There are a lot of community plugins that people write, or that you can do a lot of – you can do anything you want, right? You can change and then transform the code.

[0:25:24.7] JM: Actually, let's go through a couple examples. In case somebody was a little bit confused by the Babel plugin example, give maybe two examples I want a more a conventional example of the Babel usage, where I'm a developer, I want to write a plugin that manipulates the abstract syntax tree in a specific way. Then maybe give an example of somebody who's doing something off the beaten path. I don't know if it's static analysis, or something else that's not just transforming it for cross-browser compatibility.

[0:25:56.0] HZ: Yeah. For a cross-browser thing, we can use the example of a class. In Java, there's classes, and JavaScript now there's also classes, and the syntax is just class space, like A and then open curly, close curly. That won't work in older browsers. It will just throw an error like, "What's class?" If you use Babel, then it will actually convert it into the equivalent, and the equivalent is a function call. We find all the classes in the AST and then we just convert that into functions. That's the, I guess straightforward thing that Babel does.

If you're trying to – you can do a lot of things with your own plugin. One example could be when you use react, it converts JSX into what we call a react.create element. If you use that JSX, you actually have, because it converts – using JSX actually means you're using a react. That means

you have to require that at the top of your file. Then someone wrote a plugin where it's like, every time I use JSX, it automatically adds import react, from react at the top of your file for you, and that's just – it doesn't mean that everyone has to do that. It's just something you could, just because they thought of it.

The way I think about it is do you want to do something at build time that you might have wanted done at runtime, or something that's more of a yeah, a developer-friendly thing? Maybe I want – a lot of people write their own Babel plugin console things, where it's like, maybe they put a comment above a function and they automatically add a console.log that this function got called, or it better debugging experience. Only in development environment maybe output more comments, or make the code say something differently. Those can be really helpful.

There are plugins that help you do code coverage. There's a lodash plugin that I don't know if you need to use anymore, but it used to allow you to – all right, it still allows you to import from the lodash namespace like package, but then when you use it, only it transforms it so it only uses a specific function that you use. If you only use pick, then it will transform the regular import, like lodash import, pick from lodash to import, pick from lodash/function/pick or whatever the namespaces.

You can also do stuff like in production and maybe you don't need all these extra code, so strip that away, or maybe optimize certain things in production. Yeah, I think there's a lot of different things. You can just have fun too. Sebastian, he's the creator [inaudible 0:28:34.2]. He made a plugin a while ago called Babel plugin, emoji fly or something, and it turned all the variable names into emojis, just because he can, so you can do a lot of stuff.

[0:28:44.7] JM: I want to give people a chance to catch up for again, if they missed some of the ideas of what Babel was that you explained. Let's take again, the example of I've got a react application that I've written. At some point, that react application is going to turn into raw JavaScript and HTML. It's written in JSX, it's going to get translated into JavaScript and HTML. I believe, it's been a while since I wrote a Java – a react application, or also since I did a show about anything react related I think, but I think there's this notion of you can have server side rendering, as well as client side rendering so you can just ship the react code to the client browser and have react be imported to the client's browser and then the client's browser is

responsible for translating the react into – from JSX into HTML and JavaScript and CSS, and then it gets rendered on the page, but that can be an expensive process.

Therefore, you can also do this server-side rendering, where you ship – the react application gets rendered on the server, and then you ship just the raw JavaScript in the HTML and the CSS directly to the client, so the client doesn't have to take care of that transformation process. If you take those two approaches, where in those approaches does the Babel process take place?

[0:30:10.9] HZ: All right, that's a really good question. I would say that – I mean, the recommended thing is that you always run Babel as a build time step. Before you do any of those things, you run – you write all your code, you run it through Babel and then you send that to the client. The only reason I can think of at the moment for wanting to transpile on the client side is for a use case, like our REPL. On a Babel REPL, we – it lets you test out syntax, there's like two panes, there's a left and a right and then you can type whatever you want on the left and it automatically compiles it on the right for you. If it's for your production app, there's no reason for you to have to do it on the client. Unless, you do the whole – we try to figure out what browser it is and then we do it on the fly.

Then you have to load all of Babel in the browser, unless you do it in a web worker, or something. Even if you're doing server-side rendering, you would run Babel and then you would – and then you would send it to the client.

[0:31:10.9] JM: There's a few terms that I want to explore to help describe what Babel does. Can you define the term polyfill?

[0:31:20.7] HZ: Yeah. Yeah, this is actually a pretty big source of confusion. Polyfill is you can just say, it's a piece of code that yeah, I guess they fill, or emulates, or implements a certain functionality that will be in the browser, or already is in the language. What I would say is that it's different from syntax. A polyfill would be like there's a – that you can use a capital P promise, or a symbol, or all these different – we call them built-ins, things – you can think of it as a standard library basically of JavaScript. If you don't know if a browser supports it, then you would write – you could use a polyfill instead.

A lot of people, what they'll do is like this – they could check if it's supported in the browser. Then if it's not, then they require their own thing. This is different from what Babel does, because Babel converts syntax. It doesn't convert promise into a different piece of code. It could, but that would be really awkward, because that means every single time you use promise, it would implement the whole promise, or the whole function every time. When instead, you could just import the polyfill once and then it's a – it would be there every time you use it. I don't know if that helps.

[0:32:49.1] JM: Well, is an example of that like, if I have an arrow function in my code and I want to make the arrow function compatible with older browsers, I might write a polyfill to satisfy the functionality of the arrow function in different ways on different browsers?

[0:33:05.8] HZ: That would be an example of the syntax.

[0:33:09.2] JM: I see.

[0:33:10.1] HZ: There's no way for you to polyfill per se for the browser, because it would just air at the parse level, because the browser itself is trying to read through your code and it's like, "Oh, I saw this parentheses and this other parentheses. I don't know what that is." It just completely errored out. If you use Babel, then it'll convert it to a function and knows what that is. When I said that example of a promise, when you use promise you have to do new promise, or something like that. If I typed in promise, it's not going to error in the browser, because it's just a variable name. It will just know – It will just think it's undefined, because if you type promise in the browser and the console, it'll just be like, "Oh, I don't know what that is." Then what you can do is say like promise equals function, blah, blah, implementation.

[0:33:56.2] JM: Okay, right. Because the problem is it would error silently. You want to – I guess, is that the main difference?

[0:34:03.6] HZ: Yeah. With the syntax error, it will tell you immediately like this doesn't – I have no idea what this is. If it's a – what I'm saying is a built-in, or standard library function, you're

free to implement it yourself, even though you probably don't want to do that, because you're probably going to have bugs.

[0:34:19.3] JM: Another term, source map. What is a source map?

[0:34:22.7] HZ: Yeah. I think it explains it in some way. The thing is that if you use Babel, it doesn't even have to be Babel. You can use a minifier, like Babel minify, uglify, or even simply bundling or concatenating your code together. When you open it and try to view the source, or go to the element or the source inspector in your browser dev tools, it won't be the same thing as what you're writing. The source files are different from what you're actually shipping to the browser. Sometimes and most of time, it's really hard to understand what's going on.

Maybe you have 10 files and then use something like web pack, and then you bundle them to one file, when you're trying to look through debugging, or just looking through the code, it's all going to be one file, and then you're going to have to look through and figure out like, “Oh, this file was like a.js was this part of the giant file.” The same thing with syntax. You're writing your arrow functions, but now they're regular functions and you don't know exactly where that is. Maybe the variables got renamed, maybe if you minify that you have no, idea because you wrote this really awesome name that's long, right? Now they're all ABCD.

Source maps were created so that yeah, it's literally a mapping between every – there is different forms of it, but this character, or this line and column corresponds to this other line and column. In the browser, it shows you, it recreates the original file for you so you can read it.

[0:35:57.4] JM: Babel allows people to try out new browser features more aggressively, because they can – as a developer, they can try out that feature and then in their production code and then write a Babel translation that makes it backwards compatible. How does this affect web development?

[0:36:19.8] HZ: I would say that it allows people. There's two things. One is if you're using things that are not in the language and things that are already in language. I'm going to talk about things that are already in the language, like finalize in the spec. It lets people not have to wait until all the users that they support are able to – you can use that and state a syntax

natively. In some sense, you don't ever really know if all your users support it, unless you start dropping browsers or just airing like that. In some sense, you could say that maybe you always want to use a transpiler like Babel, because you can't really know.

Unless you're doing an internal thing, or you know you only support the latest version of a browser. Yeah, it just lets you not have to think about it. This shouldn't be something you had to think about in the first place.

[0:37:10.6] JM: You alluded earlier to the fact that Babel is actually able to be a medium of conversation between the ECMAScript standards body, this is TC39. These are the people who make standards that JavaScript adheres to. ECMAScript is a standard that JavaScript adheres to. There are other implementations of ECMAScript, JavaScript is by far the most popular so you could just think of ECMAScript as the JavaScript standards body, I believe, you could correct me if I'm wrong. How do ECMAScript standards get defined and why is Babel an important place for discussions between developers in the standards body to take place?

[0:37:54.4] HZ: Yeah. TC39 is the committee, technical committee 39. I don't know about the other committees. I don't even know if there are 38 other ones. Yeah, so they meet every two months, and they – There are companies that can join the committee and they can send representatives from those – for those companies. Examples of those companies could be all the browser vendors, so whether it's Google, so Mozilla, Apple, Microsoft, and also other implementers. Then there's also the – so represents from them. They could also be developers that work on those companies, or the implementers of those JavaScript engines themselves, so like people working on V8, SpiderMonkey, Chakra, so the people writing the C++ code.

Then also, there are more programming language experts where they are the ones that they have a lot of history in the background of programming in general. They have a lot of experience in making languages, and then there's also just developers that are – but are more involved in the tooling and the spec side of things. I think you can also talk about the history of how this – the committee has evolved over time.

Maybe initially, it was just Brandon and then a few people and most of the people are super involved in languages, and then some implementers, and then they had more and more

implementers of browsers. Then eventually, we have more and more developers that can advocate for the developers that aren't using language, because the people implementing are the programming language experts. They might not be the ones using JavaScript in the day-to-day, right? I think it's – that's why it's been in the committee's interest to get more and more people involved, so that we have a better sense of what the community needs, what they want, and what are the problems in the language. Not just from maybe an academic point of view, but from what – this is what we need, because we're dealing with this issue.

Maybe people are implementing libraries, or polyfills for themselves and we can implement that in a language, or they're writing code in such a weird way that maybe we should have a syntax on top of that, or we want to just change part of the language to do things that were never possible in the first place. There's so many different ways that people can shape the language, and I don't know if we could say that any one person or group is like, “Oh, I want a language – the JavaScript to be an object-oriented language, or I want to be functional, or I want to do XYZ.” There's a lot of players and we want to take that into account. That's why maybe it seems a little crazy at times, because it's the biggest language and all of the web and IoT and all these things, it's hard to really grasp what that looks like.

[SPONSOR MESSAGE]

[0:40:48.2] JM: Citus Data can scale your PostgresSQL database horizontally. For many of you, your PostgresSQL database is the heart of your application. You chose PostgresSQL because you trust it. After all, PostgresSQL is battle-tested, trustworthy database software.

Are you spending more and more time dealing with scalability issues? Citus distributes your data and your queries across multiple nodes. Are your queries getting slow? Citus can parallelize your SQL queries across multiple nodes, dramatically speeding them up and giving you much lower latency. Are you worried about hitting the limits of single node PostgresSQL and not being able to grow your app, or having to spend your time on database infrastructure instead of creating new features for your application? Available as open source, as a database, as a service and as enterprise software, Citus makes it simple to shard PostgresSQL.

Go to citusdata.com/sedaily to learn more about how Citus transforms PostgresSQL into a distributed database. That's C-I-T-U-S-D-A-T-A.com/sedaily, citusdata.com/sedaily. Get back the time that you're spending on database operations. Companies like Algolia, Prosperworks and Cisco are all using Citus, so they no longer have to worry about scaling their database. Try it yourself at citusdata.com/sedaily. That's citusdata.com/sedaily.

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

[INTERVIEW CONTINUED]

[0:42:33.4] JM: Is there dispute between the different browser manufacturers over things in JavaScript that relate to the businesses of the different browser manufacturers? For example, I can think of Safari and Chrome having different ideas for what level of advertisement should be in a browser. I can imagine that potentially percolating down into decisions in JavaScript. Does that happen?

[0:43:07.4] HZ: I haven't been on the committee for that long and I'm not – I'm actually still a guest and that's its own conversation, but I guess, that could be part of it. I don't really know if that comes up that often, but you could say that in the end, every representative does represent their company and they have certain interests. Maybe just the way they do their business, they might want a certain feature that other people don't want. No one's going to like – also the way the committee works is by consensus, so if someone says no, then it's not going to go forward. You need to actually be a champion and convince everyone this is a good idea, and that can be pretty difficult to do. You can't just be like, "Oh, I need this and this is what we need," and that's it.

[0:43:51.5] JM: Okay. What has been your interaction with the browser manufacturers since getting involved in TC39?

[0:44:00.0] HZ: Well, I guess the thing is that I didn't really have that much interaction with people until I decided to go. Even now, it's – we're trying to figure out what that looks like. It's weird still that I work on this project that's seems to be pretty important to the committee and even our ecosystem, but we're not – it's not anything about funding per se. It doesn't mean that

people have to pay me, or the project. It's just we don't even have enough resources, or people to work on the project itself. It's like, I don't have to be the one working on it. It's just, we all expect all this work to be done, but we stop thinking who's the one doing that. It's like, well if no one's going to do it, you can pay to do it. It's like, yeah, we should get more resources.

A very simple example of this and I'm saying this more, which is good, is company getting involved in the project by the fact that if they're the ones championing a new proposal, a new feature or whatever it is, they can be the ones that implement the Babel plug-in and that will help move the proposal forward, because it's like, "Oh, I already implemented it." Then hopefully after we release that, then you have feedback from the users and developers and they have a better sense of the confidence that this is a good idea. That's a really simple straightforward way of how TC39 Babel can work together. It's actually moving that more formally into the process, instead of just like, "Oh, Babel will implement it." It's just like a – it's just there, black box thing.

[0:45:36.9] JM: You do work solo now. You're supported by Patreon, and what's the other thing?

[0:45:44.0] HZ: OpenCollective.

[0:45:44.9] JM: OpenCollective, which is like Patreon, but looks more like – I guess you can support a group of people, rather than –

[0:45:53.6] HZ: Right.

[0:45:54.3] JM: - one person. If somebody wants to contribute to you specifically, they can contribute to Patreon. If they want to contribute to the Babel project in aggregate, then they can contribute to OpenCollective. In any case, you're not part of any specific company. You're doing this solo. What are the pros and cons to that approach?

[0:46:17.7] HZ: Yeah. I would say I am solo in the sense yeah, I'm not at a company. We do have a lot of other people on the team, so I'm not working on Babel by myself, so I just wanted to make that clear. Yeah, so the pros and cons. Yeah, I guess I'm qualified to talk about this. Before, I was just doing this for fun in my free time. At the time, I was working at Adobe on the Behance team in New York. I was doing all that and I realized we're using Babel at work and

being a maintainer, it didn't make sense to me that I would – we would have issues, or we want things to be in the project, but then why would we wait for me to do it in my free time when I could just get paid to do it if we actually need it?

I asked my boss if I could work on it at work, they actually graciously gave me half time; 50% of my time doing it at work. The pros of that is that I don't have to stress out about working and then going home and then – most people why would you even want to do more code when you go home, because you're tired and you have other things to do, but I just happen to like it a lot. I had that privilege. Being able to working at work, it lets me, I don't know, not have to go crazy with doing that outside of work.

There are some issues. One is the fact that in the end, you have to compromise maybe in the sense of the company wants certain things and that you might want certain things and you have to find things that align for both. Maybe that's just a discussion you have to have. It's fine. That's good. There's also this personal sense of almost guilt. That's not a fault of anyone there. It's just the culture where it's like, well if I'm the only one doing – it doesn't even have to be the I'm the only one doing open-source, but if I'm the only one doing half my time on this project, no one else is really involved. Everyone's working on things for work, for deadlines and meetings and everything that we're talking about is about Adobe or Behance stuff. I'm always going to feel a sense of doing my own thing, which may be freeing, but also I'm not a part of the greater community, of the culture of the company.

It's not emphasized enough, so it's like, even if it's your job, you're always going to feel like, am I doing the right thing, or should I be spending more time on work stuff? That part is difficult, right? Unless, you built a team and you're talking about it and you're always going to feel it's different, right? Then also just that maybe over time, you either find out you don't really want to do open source that much, unless you're okay with just like, I want to take a – it's different where you're just like, I want to take a break from my regular work. It's just a side thing, and that's totally fine.

For me, I guess I realized over time I want to be more invested in this and I believe in what the vision of this project is, to the point where it's like, "Yeah, I actually do want to do this full-time. I don't want to work on other things." It helped me to think about what that would look like, and if I

need to explore other options if they're not willing to do that, because if I work on our product team, I don't know if it makes sense for someone to differ them to pay someone to work on open source full-time and there's no tooling team, even if you're at a big company. I didn't find any company that would be willing to do that. That's why I ended up trying with this experiment with doing on my own.

[0:49:50.9] JM: Wait, are you saying there wasn't any company you could have gone to, to work for – work on Babel full-time?

[0:49:56.6] HZ: Yeah, I didn't find any company that would be doing that.

[0:49:59.0] JM: Really? Facebook would not be able, for example, would not be able to hire you to work on Babel full-time?

[0:50:04.6] HZ: No.

[0:50:05.3] JM: That's so surprising to me. I remember, maybe it's – I guess, it's different. The OpenSSL thing I remember, where there was some vulnerability a while ago in open SSL and people were like, “How did this happen?” It was like, “Well, there was one or two people that was supporting it and they were open-source developers and they weren't really supported in any particular way.” It was like, “Well, oops.” As a community was a tragedy of the commons. I guess, there was never really a post-mortem and figuring out of what to do to support open-source people. Do you remember the OpenSSL case? Was any precedent set there?

[0:50:48.3] HZ: I think because it was so bad that – It’s like, “Why are we waiting for a catastrophe to happen for this thing to happen?” It’s like, “The fact I'm still working on it now, maybe it's bad because then no one knows what the current situation is.” It's almost that – the same issue with node, where they didn't have a release for a whole year and that's when people are like, “Oh, something's gone wrong,” and then suddenly all this thing happened. For us it's like, “Well, I'm still doing it, and all these people are still involved, and it's still working for people, so there's no issue.”

I don't want to have to step away almost on purpose, just to make people feel the pain thing. I guess morally, but maybe I should just so that people realize it. Yeah, for that issue, it was I think, a bunch of companies got together and they were able to fund. Even then it's like, are you only funding for that one year? What's going to happen next year when there is no apparent issue? It's like, you have to maintain this thing forever for a long time. You can't just pay for that one year. Maybe that person doesn't even want to work on it and maybe they need to retire, or they have kids, or all these different things. Just because you added in money for that one time isn't really going to solve the problem at all. That's just for their project, right? What about all these other projects that haven't had that catastrophe happen?

The ironic thing is that that already happened with Babel in a different way, with left pad, which I don't want to have to bring that up again. It's funny, because that's my username now. With that, that was a good example of not about – anything other than the fact that Babel is used by so many projects, or whatever ecosystem it is. Babel was the main project that used left pad as a dependency. Once that got unpublished, everyone started like, “Whoa, what happened to my build and all that?” I think, that was a great example of that. It's so ubiquitous that just because that one project wasn't able to get downloaded, everyone started complaining about everything.

That was two years ago. It's like, we're still trying to figure things out and you have to deal with – people talk about issues in poorer class, but that's just the project, right? There's also the ecosystem, there's all the products that are using Babel too and how we get them to upgrade to be on the latest thing, making sure they're using the best practices. Then also the language and moving that forward. There's all these different things that could be multiple full-time people maybe, and yet, we have a few people that are doing it and trying to figure things out.

At the same time, you have people that are new. It sucks, because they don't know how it works and they don't know who's working on it. I don't I blame them for saying like, “Oh, it's complicated, or why doesn't this work?” Then on the maintainer side, you're like, “Well, you don't understand what I'm dealing with.” Then there's all this entitlement and complaining on both sides, and it's no wonder that all these people get burnt out, right? It’s just a weird situation.

[0:53:51.6] JM: Is it working out for you, or are you feeling you're not making enough at this point and it's a frustrated experiment?

[0:53:59.4] HZ: Oh, no. For me right now, no I feel really good. I mean, yeah, so there's stress in many different ways whether it's money or not knowing enough about business, or sales, or insurance, or taxes and all those things, but I'm doing this because I believe in a certain – a vision of what the project is, but also what open source is. Not because I think it's all going to be okay in the sense like, I know what I'm going to do, but I like working on this issue not just for Babel, but for open source in general and I would like to see it change. I would to see the culture of not how we use open source, but how we care about open source, how we maintain open source, how we steward it. I want that to change.

Not just from a developer point of view, but a company point of view, and from people outside looking in, like what's this programming thing about? What's open source about? I don't want it to be about making all this money and VCs and that thing. It's like, there's another side of open source, where it's about community and helping people and serving. I think that's really important.

Also to a little bit different is just, that you shouldn't have to sacrifice everything to do open source. You shouldn't have to compromise where it's like, “Oh, if I want to do open source, I have to decide to not make any money, right?” Or that in order to open source, I have to have all this free time, or be younger and all that stuff. It's like, no. Anyone can get involved. It doesn't mean everyone has to, or should, but changing the culture around that. Maybe the best thing we could be doing is just the fact that companies should allow people to do open-source during work.

[0:55:44.8] JM: There's also the futurist capitalist notion, that the solution to this problem is to have open-source protocols do an ICO and make their protocol into something that can be publicly traded. It sounds like that's not really the issue for you at this point. It sounds like, it's not you need some new medium of people paying for it, it's fairly straightforward. If you want to invest in the protocol, support Henry Zhu through Patreon, or OpenCollective, it's really that simple and it's more a question of are there enough individuals? Are there enough corporations? Or is there a single corporation who could just say, “Okay, we're going to stand up here and contribute or virtue signal,” or whatever is the value to them for supporting you.

Somebody needs to do it. Otherwise, it's a tragedy of the commons. The conversation needs to progress. I mean, well, I think the conversation is progressing, but if it doesn't progress from discussion into action, then we'll see more open SSLs, we'll see more left pads, we will see more crises.

[0:57:00.3] HZ: Yeah. It's really just, it's not normal, because we haven't done it before, right? People just expect open source to work. They just use it, it's there, no one thinks about who's working on it, especially if you're new, right? If you're not new, it's just like if something works, there's nothing to complain about, but you forget that the maintenance cost is way higher than the initial cost. We need companies, or people to step up for the first time. Once one company starts donating, then other people are going to start doing. They're going to start wondering, "Why are we doing it too?" It's almost just like competing, but – competing in teams and donating and trying to change that attitude.

Instead of like, "Why are we donating? This is a waste of money." It's like, "Why aren't we donating?" Because this is something we rely on and we need this as a community to take ownership of the things that you're using and to realize like, "Yeah, this is important to us and we're doing this together." Not like, "Oh, this is not my responsibility, or this is not my issue." It is your issue if you're using it.

[0:58:03.9] JM: Does it need to be a philanthropic line-item for these corporations, so that if they give to Patreon open source, maybe they can do a tax write-off or something?

[0:58:18.4] HZ: I mean, that would be nice, because that would be more in-line with, "Oh, it's a non-profit," and then they can get – Yeah, they can write it off all that. That's definitely a way to explore it, but there's also just like, it's literally in their interests if they want to sustain their business to do this. They've built their businesses on top of open-source. There is this goodwill thing, but that's more of a marketing thing. It's like, "Oh, we're supporting open-source," so then now we make people see them better, or in a different light, or that you have better recruiting, because we contribute to open source or we employ people in open source. That's definitely a thing, that's why people work at companies, and that's definitely an incentive.

Just yeah, changing the perspective around how people are using it, it's interesting. I think another issue is people – and usually companies would rather donate to the OpenCollective. This my experience, rather than at Patreon. They would rather donate to the project, rather than the people. That intuitively makes sense. They're like, “Oh, we need this project. We don't care about the person,” even though maybe that person is the person that works on the project, but what happens when that person leaves? What happens when that person doesn't want to work anymore?

We can think idealistically that there's this money in a pot and then random people will show up and they're going to take the money and make the project better, but it's like, do they really care about the project? Are they just trying to make money? How does that actually when that work out? How do we divvy up the money? There's a lot of issues with just paying people as well. I can’t say that just because we have money, suddenly everything's going to get better either. There's a community issue, there's an issue with how to do that.

Maybe that's why a lot of people are like, “Oh, we need to just employ people.” I think, this doesn't mean there's one way of doing this. There are a lot of different ways, there are a lot of different projects. Not every project is as big as Babel. It is simple, but it's also pretty complicated too. Just a lot of moving parts to all these, right? There's a trust issue too, right? It's like, a company decides why are we going to give money to this random person? I would say that you're already trusting random people right now to use their code. You're trusting me right now to maintain this thing, but yet, you're not willing to pay, because for some reason people are like, “Oh, you're going to use the money in the wrong way.”

You were giving about lots of money to people that are doing startups that are unproven too. It's funny, open-source is like, if you told an investor we have this tool or project or company where 90% of the world is using it, or whatever it is, but we don't have – we don't make any money, they'd be really willing to give you money. We just can't give the money back. Yeah, it's a really interesting social problem, where it's like, yeah, in the end maybe it's not really about programming per se. It's why it's really cool to be able to talk with friends that aren't in tech at all and they have – they're really fascinated that this is even a thing.

[1:01:19.0] JM: All right, well let's close on an aspirational note, because you mentioned that you have a vision for what this looks like. For anybody that's using Babel under the hood, they probably feel like, “This is a solved problem. I don't really need to contribute to this. It takes care of my problems,” but it sounds like you have a real vision. To entice people to go to Patreon, or to go to OpenCollective and give to what Henry Zhu is going for here. Explain your vision. What are you trying to do with Babel? What's the big goal?

[1:01:52.7] HZ: Yeah. People could sponsor just for the fact that it takes a lot of time to maintain a project, right? Just to keep the status quo. Babel is a little unique, unlike other projects that just might go away, because people don't want to use it anymore, or it's not necessary. Inherently, Babel moves with the language. If you want JavaScript to improve, then you'll want to support the efforts that we're doing with Babel. As long as you're using JavaScript and you want JavaScript to be better. Even if you don't, it is the most popular language right now. A lot of people are using it for the first time. It's like, how do we create better language for new people, people that are more experienced, and different use cases, whether it's on the web or not.

I would like to figure out how the project can be more of an educational resource too, whether it's teaching people how it's compiling things, that's – I think this is probably one of the most accessible compilers there are, just because it's for JavaScript, it's written in JavaScript. You can write your own plugin. It’s like, how do we get people that want to be involved with programming languages and compilers, which seem to be very unapproachable things and guide them through that process? Doesn't mean everyone has to become a compiler engineer, or even contribute to the core code, but you can contribute to our website, the docs, the REPL. There are a lot of really cool tools out there that are not consolidated that we could use help in.

Things that people haven't even created yet. It's like, people like to ask me like, “Oh, I want to get involved in all the source, tell me what to do.”

I don't know what interests people. I want to find out what interests you, and to help you guide through that process. Unfortunately, I'm just that one person, so talking to a hundred people, a thousand people, or even just 10 people it's difficult. Yeah, I want to encourage people to get involved and know that it is a process, it will take time. It's fine to just do open source for a day or two, but there's also a cool thing in just working on something for a long time, to be

committed to some yeah, vision or purpose for what this thing represents in terms of like, “Oh, this is a feature of how languages can work.”

Hopefully, our project is a good representation of what the open-source can be in terms of a community and trying to build on that. I think I'm trying to figure out open-source in general, but then through this specific project. How do we make a community that's not just about code? Bringing people that are good at documentation, or writing, or videos, or art, there are a lot of things that we can be doing. A lot of the work I'm doing now is not coding at all. Whether it's fundraising, or talking to people, or doing meetups, stuff like that, there's a lot that can be done, and a lot of things that we're not even thinking about, so I'd like to encourage people to get involved and yeah, find what interests you, what inspires you.

[1:04:53.8] JM: Henry Zhu, thanks for coming on Software Engineering Daily. It's been great talking to you.

[1:04:56.6] HZ: Thank you.

[END OF INTERVIEW]

[1:05:01.1] JM: Every team has its own software and every team has specific questions about that internal software. Stack Overflow for Teams is a private secure home for your teams’ questions and answers. No more digging through stale wiki's and lost e-mails. Give your team back the time it needs to build better products.

Your engineering team already knows and loves Stack Overflow. They don't need another tool that they won't use. Get everything that 50 million people already love about Stack Overflow in a private secure environment with Stack Overflow for Teams. Try it today with your first 14 days free. Go to s.tk/daily.

Stack Overflow for Teams gives your team the answers they need to be productive, with the same interface that Stack Overflow users are familiar with. Go to s.tk/daily to try it today with your first 14 days free. Thank You Stack Overflow for Teams.

[END]

