EPISODE 625

[INTRODUCTION]

[0:00:00.7] JM: Flutter is a project from Google that is rebuilding user interface engineering from the ground up. Today, most engineering teams have dedicated engineering teams have dedicated engineering resources for web, iOS and Android. These different platforms have their own design constraints, their own toolset and their own programming languages. But each platform is merely building a user interface.

Why should development across these three user surfaces be so different? This was the question that Eric Seidel was asking himself three years ago when he cofounded the Flutter Project. The Flutter Project had a few rough starts as the team tried to figure out exactly what layer of abstraction they were trying to provide.

Around that time, React JS and React Native were growing in popularity. Seeing the React projects provided some data points and some inspiration. But Flutter ended up taking a lower level approach to cross platform app development by presenting a rendering layer and a runtime API that are interfacing with the hardware in the same way that Open GL does.

In today's show, Eric joins the show to explain how the Flutter Project came to life and his lessons from starting an ambitious project that took several years to pick up steam. I enjoyed this episode because Flutter could have massive improvements for how quickly we can build applications.

I also enjoyed it because Eric is a serious software engineer and there are so many insights in this episode about computer science, software engineering and project management.

[SPONSOR MESSAGE]

[0:01:27.1] JM: Before we get started, I want to mention that we're hiring. The jobs we're hiring for include writers, researchers, a videographer and you can find all those jobs along with several other jobs at softwareengineeringdaily.com/jobs. Some of these jobs are part time,

some are full time. If you are hiring, you can also post on our job board, it's easy and it's free, you can just go to softwareengineeringdaily.com/jobs and you can see how to post a job. We'll be sharing some of those job postings with our listeners. Let's get on with the episode.

Raygun provides full stack error crash and performance monitoring for tech teams. Whether you're a software engineer looking to diagnose and resolve issues with greater speed and accuracy or you're a product manager drowning in bug reports or you're just concerned you're losing customers to poor quality online experiences.

Raygun can provide you with the answers. Get full stack error and performance monitoring in one place. The next time you're struggling to replicate errors and performance issues in your code base, think of Raygun. Head over to softwareengineeringdaily.com/raygun. Get up and running within minutes and dramatically improve the online experiences of your users. Thank you to Raygun for being a sponsor of Software Engineering Daily and if you want to support the show while also checking out Raygun, go to softwareengineeringdaily.com/raygun.

[INTERVIEW]

[0:03:13.4] JM: All right, Eric Seidel, you are an engineering manager on the Flutter team at Google, thanks for coming on Software Engineering Daily.

[0:03:19.4] ES: thanks for having me.

[0:03:19.9] JM: Before you worked on Flutter, you worked on the Chrome team, you worked on Blink which is a browser engine, I think you could describe it as, similar to Web Kit. How did that experience with Chrome shape your beliefs about web development and the future of the web?

[0:03:39.7] ES: Yeah, I worked on Blink and Web Kit for about a decade before getting involved with Flutter, before starting Flutter. How did it change my beliefs about the future of web development? I guess, I'm a big believer in the web. I like the web a lot, it just moves really slowly. Eventually that sort of slowness caught up with me and you know, it's really hard to change something when it has you know, trillions of applications running on it. We went off to do a different thing.

[0:04:05.8] JM: How would you contrast the advancement of the web to the advancement of mobile application platforms?

[0:04:13.0] ES: Again, it just moves slowly. The last, say five years, I worked on the web where we all focused on mobile and making the web amazing on mobile and yet I think, still today, there's a lot of desktop focus on the web. That's just hard, whereas mobile platforms typically only work on mobile. They only care about devices that have a touch screen and have a battery and yeah, just a different set of constraints.

[0:04:36.6] JM: What led you to founding or cofounding the Flutter Project.

[0:04:41.9] ES: It's all kind of an experiment. Back when myself and a few of the other leads from Blink decided to take a somewhat radical experiment approach and slice and dice Chrome down and see how fast we could make it go.

We allowed ourselves, I think it was a week or two and we just tore Chrome apart and see how fast we could move on our benchmarks. Allowing ourselves to break compatibility to the web. The end result was we were 20 times faster on some of the benchmarks that we cared about and obviously, the end product wasn't the web, it didn't render all web pages, but it taught us that there was a lot of potential to do things better, faster, if we worked on something that was beyond the web.

[0:05:24.3] JM: How did that train of thought lead to making a cross platform mobile development experience?

[0:05:32.5] ES: Cross platform wasn't necessarily really the goal, so much as maybe portable or just being able to do beautiful, fast, experiences, wherever you know, Google needed those experiences. We, I think started with maybe that in mind and then the cross platform was just sort of a practical application of this. Right now, where are all the developers? All the developers are on iOS and Android and so we brought the system to iOS and Android because we saw problems to solve there.

[0:06:00.6] JM: Was the train of thought this, "We want to see how fast we can make some parts of chrome because the cool thing about chrome is that browsers run anywhere and that doesn't seem to be a trend that's going to change anytime soon. Maybe you get an augmented reality interface, some sort of browser is still going to run there. Maybe if we're trying to make a user interface and an application platform, let's just start with the web."

[0:06:29.6] ES: Yeah, I think we started with the web because that was what we knew. There was also I think, I had invested a decade of my life at this point in the web. I had made a lot of, I thought, really nice pieces of engineering that sometimes felt compromised or held back by the constraints. You know, some of the things that we found when we were digging was that the web pays enormous penalties for compatibility with 30-year-old web pages.

It's death by a thousand cuts, I remember one piece that I took out – there's this really deep down in the web's rendering system, there's this If Statement that only exist for like a few Israeli banks that use this encoding system that was like deprecated in 1989 and that kind of, it still has to render this page that was created two decades ago.

You know, that one If Statement cost like 2% of text rendering speed. You could imagine you remove thousands of those and you get something really fast. I think it was, we wanted to dig out the gold, we felt we had invested in the web platform over the previous decade of work.

[0:07:32.6] JM: This eventually led to the creation of Flutter and I think people who are familiar with Flutter think of it as a way to make cross platform mobile applications, the way you're describing it is something bigger. I'd like to get into the bigger vision eventually but let's just frame it right now as something that allows you to make iOS and Android apps from a single development experience.

We've had these different cross platform ways of doing software development because of the differences between iOS and Android. What are those differences between iOS and Android that are so distinctive that cause us to have to make these somewhat hack-y cross platform development experiences?

[0:08:21.8] ES: Well, there's lots of differences, sort of at every level at the maybe simplest layer, the UI toolkits, the way that you render pixels on the screen are simply written in different languages. One on iOS it's written in Objective C or now some on Swift I think. On Android, it's written in Java. You then have to create an abstraction layer on top of that if you want to use those, that's one of the things that React Native, a big player in the space has done and we just took a fundamentally different approach. We wrote a new UI toolkit that renders straight to the GL buffer. Just a very different approach.

[0:08:56.8] JM: What is the GL buffer?

[0:08:58.7] ES: Maybe that's imprecise terms, we render straight to open GL which is a way to talk to GPU's, we also can render the Vulcan which is a new modern way to render GPU's. Basically, a way to flip the pixels on your device.

[0:09:10.3] JM: Do open GL libraries run very performantly on both iOS and Android?

[0:09:16.2] ES: Yeah, but you typically have to write to GL in C++ so like, the way that many mobile games are written is straight to open GL for example. Then if those games are written, largely in C++ or largely in C, they are then portable between iOS and Android. You see this already.

[0:09:34.3] JM: Okay, for those games that are written in Open GL for example, they do have a right once run anywhere type of experience for the developer it sounds like?

[0:09:45.1] ES: Correct, they do, the compromise there is that you are in the games world so when you launch the game, you don't have any buttons or at least they don't look like iOS buttons and you don't have a drawer on Android. You have whatever the games UI system is, and you know, the game's quality of text which may or may not be good. Whereas we took sort of a game like architecture but then we applied that to writing a user interface library.

[0:10:10.6] JM: This is a more holistic question but why is that important? Why is it important to convey the operating system level design decisions to the apps that run on that operating system?

[0:10:25.0] ES: Some would argue that it's not important but in the end, it's all really about matching user expectations, right? When I'm a developer on my application, a lot of my job is to provide value to the user and some of that value to some set of users is looking and feeling like they expect.

[0:10:42.5] JM: I think I was watching some presentation that you gave about Flutter and the thing that you emphasized in the first demo that you gave was that the physics of just scrolling is something that is subtly different in iOS and Android and when you break the physics of how scrolling on a mobile device works within an app. If you make an iOS app and you somehow break the scrolling experience either because you're in a game developed environment or a cross platform developed environment. Something feels wrong, something feels broken.

[0:11:19.2] ES: Exactly. That's exactly the existing – when you're trying to not have to write your application twice, your choices are typically, write your own physics, you know, write your own whole stack in a game perspective or sit on top of the existing iOS and Android stacks you know? Possibly through an abstraction but writing your own physics is hard.

[0:11:40.9] JM: The cross platform tools that are historically represented, whether we're talking about React Native or these things like Accelerator, other cross platform tools. Most of the time, the ones that I've seen use a Java Script bridge which is basically the cross platform elements of the application are written in Java Script.

Can you describe the pros and cons of a Java Script bridge maybe, describe what a Java Script bridge is?

[0:12:09.3] ES: Yeah, this isn't really my entire area of expertise but my general understanding is that a Java Script bridge is about exposing objects from one language into Java Script. For example, because iOS is mostly written in Objective C and say you would want to interface with some Objective C object. Say, even just like the view, like a button.

You would need to expose that up into Java Script in some way. The way that that's done is via this bridge I believe. Same thing for Android but there are some downsides to having to deal

with these cross languages. There's some slowness that can come as a result of having to bridge between languages.

[0:12:50.5] JM: In contrast, your cross platform findings are at the lower level as you said that this C++ rendering engine that's at the same level as Open GL?

[0:13:03.0] ES: IN any of these solutions, you're going to need to have some way to talk from your shared common language that you're sharing between platforms, to the languages that the platforms are running themselves. You're going to need some bridging strategy, it's a question of where do you put it?

For Flutter, we put these bridging strategy in sort of – we push it out to the edges where it's less in the critical path, whereas some other solutions where you're crossing the bridge, many times per frame, many times in a short span of milliseconds. Where as in Flutter, we try to push that to the edges where you wouldn't have to cross these bridges as often.

[0:13:40.2] JM: It kind of reminds me of Web Assembly where you have a really low level interpreter and I guess, they just use LLVM which is literally the Low Level Virtual Machine. Did you take any inspiration from the Web Assembly project?

[0:13:58.8] ES: Web Assembly is about how to run other languages on the web safely. We don't run inside the web's container. Flutter shares no code with the web, we started from a web code base but we – these days, don't share any code with the web. We don't have that sort of security boundary; we don't have to like worry about running arbitrary code safely or anything like that.

Web Assembly is largely an orthogonal project. I do think that Web Assembly is interesting direction for the web and we'll see where that plays out over the next coming years as browsers continue to adopt it.

[0:14:34.5] JM: Yeah, okay, focusing on the Flutter world. I'm a little curious about this rendering engine that you had to build at the lower level between some Flutter representation

and a graphics representation. What was the engineering requirement set for that really low level bridge that you're describing?

[0:14:56.0] ES: Yeah, our run time, our lowest levels that are all built in C++. Again, stemmed from initially from Chrome and the web, we just cut away until we had them minimum set of things we needed and again now are wholly separate, but our run time consists of really only three pieces. One is we have a CPU abstraction, the way that we run code on the central processing unit of your phone and that's Dart.

Then we have another which is the GPU abstraction for the graphical processing unit and that is SKIA, that's the same graphics library that's used by Chrome and Android and then the third part is we have a small bit of C++ code that is really hard to write and needs to run really fast and that is the text layout code for like laying out lines of text and that we took from Android and that also exist in this bundle of code.

Then we have a few other little odds and ends but that's basically it. A part of the principle was to put his little code here in that layer so then the rest of the whole system is all in the same language that you write your application and thus under your control.

[0:16:02.9] JM: Dart specifically, what is the benefit of using the Dart language?

[0:16:09.6] ES: There's a lot of nice properties of Dart, we talk some about this on our website at our frequently asked questions. One of the nice properties of Dart is that it has a really fast garbage collector, one of the choices that we made in Flutter was to have this reactive style system where it's very common to allocate thousands of objects in a – if not, tens of thousands of objects in a single frame.

In a span of a few milliseconds and then immediately let go of those objects again. There's a variety of ways to do that but Dart, having a generational garbage collector, can handle large volumes of short lived objects very quickly. Dart also has some really nice performance characteristics, it has an ahead of time compiled back end which allows us to compile straight to native arm code, allow us to achieve really fast startup on really consistent performance and Dart also has a nice focus on developer experience which really got along well with my team.

[0:17:08.4] JM: Right, I've seen from your talks that developer experience is quite important to you in how your architecting the toolset and different ways that you can interact with code as a Flutter developer. The generational garbage collector that you mentioned that's important because of the short lived objects.

Is there something about modern application development, perhaps in the context of mobile development or maybe it extends to the web as well where you have large sets of short lived objects or is this something that you feel has always existed in applications?

[0:17:41.6] ES: It's very new. Well, I'm sure the computer science theorist would tell me that it dates back decades but the pattern, seeing broad application that at least came to my attention through this React JS pattern of popularity and Flutter definitely was inspired early on by React JS.

In that paradigm, you do tend to allocate lots and lots off short lived objects and you need an efficient way to do that and Java Script has a garbage collector that can do this and Dart does as well.

[SPONSOR MESSAGE]

[0:18:24.3] JM: Stop wasting engineering time and cycles on fixing security holes way too late in the software development life cycle. Start with a secure foundation before coding starts. Active State gives your engineers a way to bake security into your language's run time. Ensure security and compliance at run time, a snap shot off information about your application is sent to the Active State platform.

Packaged names, aversions and licenses and snapshot is sent each time the application is run or a new package is loaded. So that you identify security vulnerabilities and out of date packages and restrictive licenses such as the GPL or the LPGL license. You identify those things before it becomes a problem.

You can get more information at activestate.com/sedaily. You want to make sure that your application is secure and compliant and Active State goes a long way at helping prevent those kinds of troublesome issues from emerging too late in the software development process.

Check it out at activestate.com/sedaily, if you think you might be having issues with security or compliance. Thank you to Active State for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:19:54.4] JM: Okay. These objects are getting defined and – this is like in the React JS world, it's in the representation of the virtual DOM tree and that DOM tree is constantly getting refreshed and rebuilt and things are getting pruned and changed and I guess these are the objects that we're talking about in the React JS context?

[0:20:17.6] ES: Well, I can speak more intelligently about the Flutter context, in Flutter, the things that are short lived are the widget layer. Things below the widget layer like the rendering layer have long lived objects which is more typical of what you say, see on iOS or Android where you would instantiate a view on iOS or a view on Android and that view would live for as long as that view is on screen.

The difficulty with that approach is the data flow is harder to reason about. In a reactive style system, the data always flows one way. You always are regenerating your entire view from your data. Whereas in one of these older systems, these object observed pattern systems, you have to both explain during creation time of your view, how to set up your view as well as to update time like every time your view changes, and those two data paths, it can be just one more thing to keep in sync.

People like this reactive pattern where you have this simple data flow but it does create some complexity for the framework authors and the run time authors to make something like that fast.

[0:21:24.0] JM: Let's go through these layers of Flutter, we've pretty much laid out what the lowest layer is like, this is the run time layer, the run time API has an interface with the operating system and as I understand, it's through this rendering layer that is compatible with Open GL.

I don't know, is there anything more you want to say about the interface between the runtime API and the operating system?

[0:21:54.1] ES: How about I just sort of lay you through an oral version of our layer diagram?

[0:21:58.2] JM: Sure, please.

[0:21:59.9] ES: At the lowest layer of Flutter, there is this ball of C++ that we've talked a little bit about now, that the run time. It does need to talk out to the operating system and it does that through some pretty limited ways. Its' mostly about putting graphics on the screen so it needs to talk out open GL and knows how to talk to some accessibility API's and knows how to talk to a little bit of file system IO and some network IO, out the bottom.

Out the top, it provides an API to the Dart code to access those system. Then the whole rest of the Flutter system is all written up in Dart. It too is then layered. There's this very low level layer which is called the bindings layer which mostly is about sort of initial setup et cetera and then on top of that, there's – I know I'm probably skipping around a little bit here but then there's a rendering layer which is important thing to think about in the rendering layer is like a typical view model, these are long lived objects that the system generally manages for you.

On top of that, then is maybe the widget layer. The widget layer is where we have this sort of reactive pattern, the widgets are typically how you interface with Flutter, so you would just create a whole tree of widgets for the one frame that you're trying to create and every time you're putting up a new frame in your application you're creating a whole new tree of widgets.

Then on top of that widgets layer, we have then a variety of more opinionated layers that exist to express someone's design aesthetic. Such as say, material design, we have a material layer which is Google's design aesthetic and then we also have Cupertino layer for expressing the iOS design aesthetic.

[0:23:26.8] JM: Should the rendering layer, this is I believe you said this is where the objects are repeatedly created and thrown out in every frame, is that right? The widget objects?

[0:23:37.7] ES: Sorry, the widgets sit on top of the rendering and those are short lived objects. You can think of them like templates. “Make my view look like this.”

[0:23:46.4] JM: Okay, the rendering layer is below the widget layer, the rendering layer is for longer lived objects, right?

[0:23:53.2] ES: Correct. The rendering layer handles things like layout and painting. There’s also other things like, text editing is implemented up in here, gestures, animations, the rendering layer is a typical UI framework, it’s just then we also have then an additional layer on top of that typical UI framework.

[0:24:11.1] JM: The widget layer, the short lived ephemeral objects that’s on top of this rendering layer. This is a composition based way of doing object management, you encourage extreme composition over inheritance. Explain what composition is?

[0:24:30.2] ES: Yeah, I could explain it maybe through an example. Okay. It’s very typical in systems to have a container class, on the web this is a dev or on iOS I guess it’s an NS view, or an Android view. Sort of, in this sort of container class typically, the way you draw a box and if you want to participate in the system, you inherit from one of these containers. On the web, you would maybe, if you’re doing custom elements, you might need a subclass div and along with it, you get all of the various things that you can do with a div.

That’s not typically how we design things in Flutter. It’s sort of the other way around. If you look at container in Flutter, it is itself composed of 10 different seller classes. Instead of inheriting from container, you would just use one of the smaller classes or you can even use container yourself but it’s this – that you compose these various objects together as supposed to you being one of these objects.

[0:25:31.0] JM: Can you describe a few more examples of widgets that we might define an application?

[0:25:34.0] ES: One example that we give sometimes is that say, your designer wants to have a custom button and so you might implement your own custom button to implement your own designer's look and feel or their custom animations. That's a very normal thing to do in Flutter.

You also of course can use the buttons that we provide out of the box but it's a very normal thing to compose a new button out of the various pieces that build up other buttons.

[0:25:58.9] JM: All of these different layers that we've talked about, the opinion layer at the top and then the widget layer and then the rendering layer, then the runtime API at the bottom. All of these are designed to be configurable and tinkerable. Would the average developer want to tinker with all of these different layers, where is the average developer working?

[0:26:20.4] ES: Yeah, I don't know exactly who the average developer is but I think that many developers would just work at the highest opinionated layers and would just take a material button and put it next to a material text box and you know, use a gesture detector to detect a swipe, add a scaffold and an app bar. But it's a design choice of the system that you can change any aspect.

You can access any layer and you can fork the whole thing and then here gain through some of the magic of Dart. Dart has what's called a Tree Shaking Compiler where when dark goes to compile your application, it only includes the bits that you use. All of our massive framework is not included in your final application. Only the bits that you actually touched.

[0:27:08.7] JM: Let's zoom out from the layers to the developer experience. If I'm a developer that's actually just – I just want to write a Flutter application. I think your launch application or the application that you've presented as an example in some of your talks which is the Hamilton application like the Hamilton musical. There's an app for that and it's created in Flutter and I think this is a great example of something that people would want to make cross platform, this is not the type of app that people should have to write two distinctive apps for.

I think it's a good example but if we're trying to make an app like that, the developer experience on Flutter is optimized. One aspect of Flutter that you've emphasized is Hot Reload. What is Hot Reload and why is that important in cross platform development?

[0:27:59.0] ES: One of our founding ideas is just trying to make the whole developer experience better and Hot Reload is a part of that. A Hot Reload is one of the technologies we developed along with working with the Dart team where the end effect for user is that you type your code and your Flutter app, you hit save and then a second later and sometimes less than the new running code is updated on your application running on your phone and this provides for a very nice developer experience.

You can as some say paint your app to life that you just sort of tweak with a little bit of here and you tweak with a little bit there and this is really a visual thing and I would encourage you to go to our Flutter.io website and see it for yourself but it changes the way you do development when your cycle times are not measured in minutes or tens of minutes from when you make a change to compile the C on your phone but rather in milliseconds.

[0:28:58.5] JM: You talked about Hot Reload as being a hard thing to build. What are the technical difficulties or the technical tradeoffs that you make in developing a Hot Reload system?

[0:29:08.1] ES: Yeah, I think a lot of what makes many of the things that Flutter does, but including Hot Reload, hard is the coordination between all of the various systems and layers. One of the things that's neat about Flutter is that we wrote the whole system. One team wrote the whole system which means that to make a really nice Hot Reload experience, it is not just about having a runtime that supports injecting new code which we do and that itself was hard.

But you also then have to then teach the rest of the system that a thing called Hop Reload exists and that there needs to be some way to communicate to the rest of the system, "Oh hey don't throw away the state of the application, just redraw the entire screen. Don't change where it is navigating, don't change where you are in the application just repaint where you are even though you are running wholly new code."

And so the complexities around I think many parts of Flutter but Hot Reload included is around the coordination between all of the pieces of the system.

[0:30:06.6] JM: Let's say I am developing the Hamilton app or some other simple app for a movie, what's the tool chain? What's the developer experience for working with Flutter?

[0:30:19.2] ES: We find that most folks working with Flutter work in one of our two or three supported IDE's. The big ones being Android Studio, IntelliJ and Visual Studio Code and the IDE's provide nice menu items and hooks and buttons to do all of the common tasks. The developer experience sort of once you get going, once you've had the template generated for you once you hit the run button. As you've described you just edit the code, you hit save and it just appears.

On your emulator or on your phone and you can of course debug from these environments. Others choose a different developer experience like many of the folks that I work with here at Google. They choose to use VIM or EMAX, something that doesn't have a gooey interface and they use Flutter's command line tools which also provide a bunch of nice developer experience affordances. Where we try to abstract a way the differences between Android and IOS between Android Studio and X Code. Where we have nice tools to help with install, we have nice tools to help with launching emulators, etcetera.

[0:31:17.4] JM: If I am writing my app in Flutter, do I ever need to duck down into the native IOS or Android code?

[0:31:25.6] ES: So it is very much a founding principle of Flutter that anything the hardware can do you should be able to do. So it always should be possible. We find that many people say the average developer doesn't need to do that because as our community as our ecosystem continuous to grow, there's lots and lots of plugins, our packages site where you can download and someone has already written nice Dart wrappers around say firebase or century crash reporting.

Or around the accelerometer or around the camera or around playing video but it always is and always should be possible for you to ask your iOS dev or put on your IOS hat and dip in and write some objective C or dip in and write some Swift if you need to.

[0:32:10.7] JM: And if I do write some Swift in the iOS world, where is the point at which the Swift code gets inserted and how is it interfacing with the Flutter side of things?

[0:32:24.6] ES: So Flutter runs all of its code on a separate thread from the OS from the rest of the UI system and so anytime you're talking in Swift code, it is typically on a different thread and so the way you communicate then is through this simple JSON although you can send any kind of serialization you want but simple typically JSON protocol between say the Flutter code in Dart and Swift code. It is like talking to a web backend is a typical way to do it.

[0:32:55.7] JM: That's pretty cool, what are the advantages of that approach or maybe you could describe a simple message that I might need to send for my Flutter code to a native iOS piece?

[0:33:06.6] ES: Yeah, so we have a bunch of examples again on our website about how to write plugins and how to bundle them up into packages but a very simple one might just be to read the accelerometer on the device and so you would write a small bit of IOS code. We have some systems in place to help automatically compile for you and automatically include in your app for you but you write a small amount of iOS code that would know how to read the accelerometer.

And then know how to respond to the incoming JSON message and then reply with a JSON response as though you are writing again a little web server in objective C that then talked to the other side, maybe the client that it's in Dart.

[0:33:49.7] JM: How do you handle – maybe this isn't a hard problem but the fragmentation here because obviously iOS operating systems are not exactly the same as Android operating systems. So there is going to be specific functionality to the specific platforms that you have switch statements or is there some awareness within the app of what the operating system is running on and some ability to switch between functionality based on the operating system? How do you handle the cross platformism in the application you are writing itself?

[0:34:26.4] ES: Well certainly. So we certainly expose the platform that you are running on and it is something that the framework already leverages some for you in terms of doing things like the physics we are talking about earlier, right? To know to use the Cupertino physics, the iOS

physics or the Mountain View physics, the Android physics for you and similarly for some of the other aspects of the look and feel but you also can access that yourself and change your behavior accordingly.

In terms of accessing very divergent API's that's typically done again at the plugin level and we see people who write plugins that are sometimes only designed for one platform, like there is a popular plugin called Android Intense that just knows how to send Android Intense and obviously you would only call that plugin from the Android side of your application. Similarly, though there is a share plugin where the author of the share plugin took the time to write.

Sort of a least common denominator abstraction around the various shared API's. They didn't have to do that but that was the choice that they made and so I guess our approach here is to provide you the tools and help the plugin ecosystem to grow to answer all of the needs. But we don't legislate a only least common denominator approach nor do we only provide bindings for one system.

[0:35:42.5] JM: What are some other tools in the developer experience?

[0:35:46.6] ES: Some of my favorites are we had some help with installation. It can be somewhat complicated to get all of Android Studio and all of X Code and all of the various dependencies installed and so we have a Flutter doctor, which will help diagnose your system and help you install any missing dependencies. I think that has worked pretty nicely. We also again abstract a way on top of Android and Android studio and X Code particularly for the building and the communicating with the device.

Including we recently added an emulator's functionality for helping with launching emulators from the command line. I should note that all of our tooling at any layer again follows this layering paradigm and so the command line tool is sort of the lowest layer of our tooling and all of our other tools sit on top of it. So that allows us to have all of the tools in all of the various supported editors.

[SPONSOR MESSAGE]

[0:36:42.7] JM: Nobody becomes a developer to solve bugs. We like to develop software because we like to be creative. We like to build new things but debugging is an unavoidable part of most developer's lives. So you might as well do it as best as you can. You might as well debug as efficiently as you can and now you can drastically cut the time that it takes you to debug. Rookout Rapid Production Debugging allows developers to track down issues in production without any additional coding.

Any redeployment, you don't have to restart your app. Classic debuggers can be difficult to set up and with a debugger you often aren't testing the code in a production environment. You're testing it in your own machine or in a staging server. Rookout lets you debug issues as they are occurring in production. Rookout is modern debugging. You can insert Rookout non-breaking break points to immediately collect any piece of data from your live code and pipeline it anywhere.

Even if you never thought about it before or you didn't create instrumentation to collect it, you can insert these non-breaking break points on the fly. Go to Rookout.com/sedaily to start a free trial and see how Rookout works. See how much debugging time you can save with this futuristic debugging tool. Rookout integrates with modern tools like Slack, Data Dog, Century and New Relic. Try the debugger of the future, try Rookout at Rookout.com/sedaily.

Thanks to Rookout for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:38:50.1] JM: This is an ambitious project and I really want to know how you managed it because it has gone on for about three years. You got it started, you got it off the ground, you scaled it up to a big enough team to really get it rolling and now to the point where it's got adoption and you've got a user base and a high volume open source project. What has been your biggest set of learnings around how to manage a team like this from the three years that you have been building the Flutter Project?

[0:39:25.0] ES: Wow, big question. I would say two things that have stood out for me. One is the reaction. Prior to coming to Google, I had a small startup that failed. One of the reasons why

we failed was that we didn't talk to our customers and so one of the things that has been very engrained in the Flutter Project from the very, very beginning was providing value to users immediately, yesterday, and to talk to our customers.

So there is that aspect to it and I think another part is maybe embedded in Google's culture but also I try in Flutter's culture to just set people up to succeed. There is a lot of really smart people in this world and just make a space where they can do important things and be set up to succeed.

[0:40:08.1] JM: What is the responsibility of a manager in setting up people for success, setting up people that are working under your management?

[0:40:15.7] ES: There is some amount of shielding people from distractions. There is just the administrative bits of make sure you take care of them having the right desks and getting enough funds and there's that aspect to it. There is checking in with people regularly making sure that they're happy and that they feel aligned and feel communicated with and that they have a voice. Yeah, it is just a lot of taking care of people.

[0:40:40.3] JM: So this project was started around the time when Angular One was out I think like in terms of other frameworks although I guess Angular One is a web framework. What were the other teams that you were interfacing with in the early days when you are architecting the project and trying to figure out what exactly are you working on?

[0:41:01.6] ES: So I think one of the earliest teams we had contact with was the Fuchsia team. They were a very early consumer of Flutter which was very helpful to us. We did have some amount of head down, helmets on, just trying to find a set of technology that really worked. There was this long phase of technological exploration where we restarted the project basically three times and then I think once we got out of that and realized that we were building something cross platform.

We had it working on iOS and then there was this long phase of talking to customers and making sure that we had built something that provided them value and this caused us to rotate to focusing internally on Google, for maybe a year to find that first Google customer. Then also

to work a ton on developer experience because we found very early on in talking to developers that that was something that provided them a lot of value more than the cross platform, more than the custom UI just simply making their iteration so nice.

[0:42:02.5] JM: Why did you have to restart the project three times?

[0:42:05.0] ES: So it was a technological restart not really a people restart. It was we again started from Chrome's code base and we started with something that looked a lot like HTML and we were just trying to make a fast HTML and then we threw that out and made something that started from Java Script directly and was building a custom elements like solution and manipulating the dom and then we threw away the dom and tried to render everything straight through the canvass basically element.

And then we kept ripping things apart and then we eventually had to change languages for a variety of reasons and change away from Java Script. So that was another restart and then around that same time, we changed off of being an object observed style pattern to a reactive style pattern. So yeah, it was just we did a bunch of technological learning. We had this gold that we knew was buried inside this system and just how to put it together in a way that would provide value to others that was along with the challenge.

[0:43:05.3] JM: Yeah, how did you keep your focus and your sense of confidence in the project as you were shifting between these different framings of what you were working on? Because it sounds like early on you had – there is a set of things you knew and there was also a set of things that you knew that you didn't know and you were really trying to figure out like, "What am I working on and who am I building it for?" You have said, you were talking to customers early on but it also sounds like you weren't exactly sure what your target customer would even look like.

[0:43:35.5] ES: Yeah, we saw so much value for a better way to put pixels on the screen. Again, across a wider context than just iOS and Android but again, that's where we ended up. How did we maintain focus? We just knew there was a value here because we've worked on that for a decade and we had a very small team for a long time and that helped us to stay focused and to

remain agile as they say because we could rewrite the codebase because it was only a handful of people.

[0:44:04.5] JM: And so it sound like this was really a fundamentals perspective of 'How do we build the best set of cross platform user interfaces?" That was the focus. It was not how do we build a good set of user interfaces for the common context right now like based on the Android iOS duality. It is more like, what are the fundamentals of making a user interface?

[0:44:32.2] ES: Exactly. I would remove the word cross platform from your original statement but exactly like we were, "How do you build the best way to develop?" And we focused on mobile. So you know for mobile, whether it is for one platform, whether it is for many platforms, whether it's for today's platforms or platforms 10 years from now. We try to have a fundamental shift in development and took a very long view.

I mean the tech lead for our project, his previous last decade was on redesigning the web for HTML 5 and so like he takes a very long view, I take a very long view, and this is a very long targeted project.

[0:45:05.6] JM: And you had to be ready for the world of augmented reality and the world of virtual reality as well, where there some foundational conversations with people who are more familiar with the bottle necks and those development processes and the kinds of processors that those kinds of systems are going to need when you were trying to architect what the system would look like and what the insertion point was for you runtime API, for example?

[0:45:33.5] ES: We haven't spent a lot of time looking or talking about VR or AR. I guess it's just based on the fundamental idea that there are lots of surfaces in the world that need 2D graphics and my experience from working on the web is that all of these many, many surfaces, they have really poor choices for how to put those graphics up. They have existing cross platform frameworks of variety which does not include for example React Native.

Because some of these systems, these devices, these refrigerators or whatever, they don't have a native UI toolkit. They need a UI toolkit. So I think that was the original guiding light and again,

we pivoted from that to providing value to developers today but again, as we talked about the larger vision is that like, “Why not completely reboot development for any screen?”

[0:46:24.6] JM: I don’t know how much you could talk about Fuchsia but is that what Fuchsia trying to do for the operating system world? They’re trying to think of, “What happens if we just rethink?” like even throughout Linux. Let us just rethink how the operating system works.

[0:46:38.3] ES: I can’t really talk about Fuchsia. It is not really mine to talk about. I do know there is a lot of code that’s in the public and there is a lot of news articles. So I would direct your listeners there.

[0:46:48.3] JM: Oh yeah, I have looked at those news article and I have looked a little bit at the code. It’s been like walking through an alien civilization and trying to decipher what the aliens are working on but yeah, I don’t know, maybe, somebody in the audience or if you know somebody on the Fuchsia team who could talk to me at their limited province there. Whatever, we could talk about that. What’s in the future? What are you excited that is being worked on in the Flutter Project right now?

[0:47:12.9] ES: Oh man so much. So our focus right now is on getting to our first stable release. We have been used inside Google for production applications and obviously outside Google for production applications for quite some time now, but we haven’t done all of the polish that we would like to do and so we are really focused on that last stretch of polish to get to our first stable release. So that we are releasing stable releases on a regular cadence just like we release betas today.

There is some big stuff that we are working on that, as we continue to always work on performance because that is very important to us. We are working a bunch on making it really easy to add Flutter to your existing app because we know that there are millions of devs literally who have existing apps and they don’t necessarily want to do a full rewrite, but they might want to work with Flutter. So we are working on that a bunch and we are also working to continue to improve our viability to iOS focused devs.

We have had a lot of Android focused devs what we've worked with inside Google for many years and you see that I think in our system that we have done a lot to build out material design and we have done less work to build out the iOS specific affordances and we are talking to a lot of devs right now to do that.

[0:48:22.4] JM: The surface area of Flutter today and correct me if I am wrong about this but it doesn't extend to the web right? I mean Flutter developers developing for mobile applications, is that right?

[0:48:35.9] ES: That is correct. We focus on mobile. Flutter is written in Dart and Dart as a language is very flexible and compiles to many formats including Java Script. You can transpile Dart to Java Script. That is actually how Dart has been widely used inside Google for major projects like ad words and ad sense for many years but Flutter itself does not transpile to the web right now and there had been a variety of people publicly making explorations with that but nothing that my team is working on.

[0:49:04.9] JM: Yeah, so I guess you just wanted to solve the harder bit first right? Because it sounds like going to the web is probably going to be easier.

[0:49:12.8] ES: Well you bring me back or that reminds me then to your earlier question about management. I do think that maybe there is a third thing which was about focus and we spend a lot of time talking about focus and every time we bring up something new to do, several from my team are very good are saying, "Okay if we do that what are we not going to do?" because the reality is the fact is on the ground. The truth is that you just only have so many people and you only have so much time and so, we try real hard to focus.

[0:49:40.7] JM: Totally and that has been an overriding theme of people that come on the show whether they are working on a company or they are working on a team or they are working on a project by themselves, that ability to focus, it seems to be a characteristic of engineers or creative types that it is just the expense of things that you could work on that feel adjacent to what you are working on, is so great but you must focus on that one thing or that small set of things.

So, you do have an open source community to I guess leverage or throw out ideas to and if you throw out an idea hopefully maybe somebody runs with it. What is the trick to the right balance between maintaining focus and also presenting to the world that you have a maintained sense of focus but also conveying the ambition and presenting those ideas and being able to say, “Hey here is a thing. If somebody out there wants to work on it you could. You could go for it.”

[0:50:42.9] ES: Yeah, I don’t know if there is a one quick trick. We do very much have an open source culture. Many of my engineers sit on our public chat. Many engineers, answer questions regularly on Stack Overflow, we very much try to engage with our community. We have what I would describe as a relatively early stage open source project. We have a couple hundred contributors total in terms of all time. I would like to see that grow significantly.

Again, my biases come from having worked on Chrome which had thousands of people contribute over the years. So yeah, I think that as we expand to a larger community, we will have to do more around communicating the longer term vision to that community and making sure that people agree. But right now, it mostly consists of people at Google where we have those conversations on a regular basis about what we are focused on.

[0:51:35.2] JM: Okay last question just to return to this management point for a little bit more, the management of the project has changed I’m sure as you’ve scaled up. How has the team structure changed and how do you allocate tasks to different teams or is it self-managing and you’ve got these different layers. How do you allocate the engineering resources?

[0:51:57.8] ES: And so Google has a long very flat management structure which I had liked. So most contributors of Flutter report directly to me. There is that reality of Google also being a distributed company. There are a variety of teams in other offices and so there are other managers on the Flutter Project. But yeah, it is mostly as you described, sort of a self-management. You know my job is to make sure that the vision and the direction is clear.

And the communication is happening throughout the organization but then who exactly does what is more of self-management. One of the things that we do, we did institute very early on as part of being a distributed team, is that we have daily stand-ups that exists for exactly five minutes. It is a hard stop and as the team has grown that room has gotten very full, but we still

managed to get through every one. I think that most people on the team have found those valuable. In terms of just a high level communication. It is optional, not everybody comes but a lot of people do.

[0:53:01.1] JM: That's a new one. That is a cool trick, hard stop five minutes.

[0:53:05.1] ES: Yeah, I think again it is always the context of where things come from right? But I think some of that came because there is a large team who had stand ups nearby us and that seemed to last for about an hour and it just didn't make any sense. So we instituted a hard stop.

[0:53:20.2] JM: I've been at that standup before. Eric thanks for coming on the show. It's been really great talking to you and I am inspired by Flutter. I am inspired by your long term vision and your ability to execute on it with considerable focus and upfront work for three years and I think you are reaping the fruits of your labor right now. So congratulations and I look forward to covering it more.

[0:53:44.5] ES: Thanks for having me on.

[END OF INTERVIEW]

[0:53:48.1] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you are interested. With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers.

I know that the listeners of Software Engineering Daily are great engineers because I talk to them all the time. I hear from CTO's, CEO's, directors of engineering who listen to the show regularly. I also hear about many newer hungry software engineers who are looking to level up quickly and improve themselves and to find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineeringdaily.com and if you are a listener to the show, thank you so much for supporting it through your audienceship.

That is quite enough but if you're interested in taking your support of the show to the next level then look at sponsoring the show through your company. So send me an email at jeff@softwareengineeringdaily.com. Thank you.

[END]