EPISODE 753

[INTRODUCTION]

[0:00:00.3] JM: React, Vue, and Angular are the most popular front-end JavaScript frameworks. Each of these frameworks lets front-end developers build components. A component is a high-level visual abstraction that is used to compose a user interface. Front-end development has moved towards component-driven development.

At a typical technology company, a designer will put together a design file of different user interface elements and the front-end engineer will take those UI elements and program code that can render those designs as components. As organizations have started to reuse their components and share them across the organization, the efficiency of design and front-end engineering is improving.

User interface is gaining more of an emphasis with organizations and new tools are allowing front-end engineers and designers to work together more productively. One of these tools is Storybook, which is a system for sharing components and the code that renders those components.

Zoltan Olah joins the show to talk about Storybook today. He also talks about his company Chroma. Chroma is building tools to allow design-driven teams to work together more effectively. We talked about how the relationship of designers and front-end engineers bears some resemblance to the relationship between dev and ops before the DevOps movement.

There are some frictions in the process of moving between design and engineering implementation. In talking to Zoltan, I got an understanding for how much the UI layer could improve through better tooling. I hope you enjoyed this episode.

[SPONSOR MESSAGE]

[0:01:52.8] JM: Triplebyte fast tracks your path to a great new career. Take the Triplebyte quiz and interview and then skip straight to final interview opportunities with over 450 top tech companies, such as Dropbox, Asana and Reddit.

After you're in the Triplebyte system, you stay there saving you tons of time and energy. We ran an experiment earlier this year and software engineering daily listeners who have taken the test are three times more likely to be in their top bracket of quiz scores. Take the quiz yourself any time, even just for fun at Triplebyte.com/sedaily. It's free for engineers. As you make it through the process, Triplebyte will even cover the cost of your flights and hotels for final interviews at the hiring companies. That's pretty sweet.

Triplebyte helps engineers identify high-growth opportunities, get a foot in the door and negotiate multiple offers. I recommend checking out triplebyte.com/sedaily, because going through the hiring process is really painful and really time-consuming. Triplebyte saves you a lot of time. I'm a big fan of what they're doing over there and they're also doing a lot of research. You can check out the Triplebyte blog. You can check out some of the episodes we've done with Triplebyte founders.

It's just a fascinating company and I think they're doing something that's really useful to engineers. Check out Triplebyte, that's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte, byte as in 8 bytes. Thanks to Triplebyte and check it out.

[INTERVIEW]

[0:03:42.0] JM: Zoltan Olah, you are the CEO of Chroma. Welcome to Software Engineering Daily.

[0:03:46.1] ZO: Thanks for having me.

[0:03:47.3] JM: UI design has become a differentiator for many products. There are a lot of teams that are involved in a single UI design. You have this cross-functional relationship between product teams and design teams, the front-end engineering team, maybe the CEO gets involved at an average company if we're talking about a product company. How do designs

at a modern company, designs for products, for software products, how do they make their way from idea to implementation?

[0:04:22.8] ZO: Yeah. That's a really good question and actually something that we ask a lot ourselves when we talk to customers. I feel we have mapped that landscape out pretty well. Typically, I'll talk about the last step first a little bit, so typically designs end up in a combination of Sketch, InVision, which is a prototyping tool, Figma which is very similar to Sketch and it runs in the browser and it's sketching Figma, or tools that are basically modern versions of Photoshop tailored towards front-end teams; teams building digital applications and websites, rather than the old school suite of tools, photoshopping, one of those which is designed really for any design as print, just across various different mediums.

That's typically where everyone ends up. The process for ending up at that artifact in one of those tools is there is actually quite a lot between teams. I'd say the most popular version is the stakeholders. We'll work with first the product manager to define requirements and spec out whatever new feature, or application is being built. Then at some point, the designers get looped into that process. They share artifacts using a number of disparate tools, but let's say Google Docs is a pretty good understanding for how that typically works.

Then at some point the product manager gets less and less involved and designers take over and use various tools to get to the end result, which is an artifact, in Sketch let's just say. Usually at that point is when the front-end engineers are get involved and all of those artifacts that were generated previously, i.e. requirements and specs, Sketch artifacts, everything gets handed of the front-end person. People and then now go and do the implementation.

Some teams work together very closely in terms of physical space. They're in the same office. Usually, what we see is oftentimes there are pods around five or six people that are interdisciplinary; a handful of front-end engineers, usually one designer and one product manager.

In the case where they're all working together in the same room, this process becomes really fluid. The front-end engineers get involved earlier and everyone's talking and everyone's pretty much on the same page, where the process happens remotely, or where the team has spread

out across the organization and they aren't sitting next to each other, then a lot more digital tools get involved obviously, like Slack, Asana, artifacts get bounced around a lot more and comments added within the various tools. In general, that's the – I guess the most common union of what we see out there.

[0:07:19.9] JM: Yeah. You're illustrating that there are all these different teams involved and there are lots of different tools that different teams are using. Some of these tools might be shared between different teams. Maybe InVision is an example of a pretty flexible tool; Slack as an example of a pretty flexible tool. There are some tools that are a little bit more disjoint, like Sketch. Sketch isn't the most shareable social product, or Photoshop is not very shareable, or social.

There are tooling problems that can lead to conflicts. There are communication problems that can lead to conflicts. I think of the problems between design and engineering today as not dissimilar from some of the problems that developers and operations had prior to the DevOps movement. Do you think there is an analogy to be drawn between the world of design today and the world of operations versus software maybe five years ago?

[0:08:23.1] ZO: Yeah. I think I definitely see that now that you mentioned. It's not something that I've thought about before, but I do see the analogy. From where I'm sitting, the one thing that stands out to me is that designers have a very different job and use very different tools to front-end engineers. In that same way engineers, to infrastructure people, DevOps people had a similar split.

With front-end, maybe even more so than with that engineering to DevOps handoff. There are shared artifacts that become central to the process and those artifacts can change over time and there's a source of truth that's important. The more I talk about it out loud, the more I think, "Wow, that does sound similar to what happened between engineering and DevOps."

[0:09:13.3] JM: Now continuous delivery has helped DevOps work a lot more effectively. What about in design? Is there a continuous delivery-like process for the UI layer?

[0:09:25.5] ZO: There is and that's where things get a little bit tricky and that's where folks will often squirm in their seat a little bit, because with continuous delivery and continuous integration, the entire process from an engineering perspective basically relies on a pretty solid suite of tests and a pretty solid code review process.

Where UI is concerned, that's traditionally being really hard to test. Test of being brittle, it being difficult to maintain. A lot of UI ends up going untested. That is a contradiction to the continuous delivery methodology where you're pretty comfortable running the test, doing a code review and then merging and having the feature go directly out to production. Whereas, if that feature involves UI, then today a lot of teams just maybe cross their fingers a little bit and hope that it works, because in more in most cases, UI test coverage is just not there when compared to other code and the test coverage that teams typically would like to see before they're ready to go all in on a CD process.

[0:10:34.6] JM: Designers today think in terms of components. I think of a component as similar to the objects that we have in object-oriented programming in the software developer side of the house. How did component-driven development become popular?

[0:10:55.1] ZO: Yeah. I would say, what happened was that the rise of these modern view layers, view layer technologies like React, like Angular, what they have in common is they have this idea of modular components with very strict and very well-defined APIs. Where React in particular is concerned, the API's are typically made up of just pure JavaScript objects.

Contrast that to the previous generation of technologies like jQuery, that is not the case. These technologies are written more from the perspective of the DOM manipulation APIs that are in a browser and they're poking their tentacles into the DOM and manipulating things directly, and don't have very consistent API's, or even so much patterns for using them. What we ended up with is a movement towards a different style of technology where UI became much more modular and components became much more easily shared and dropped in to an application.

Then that has given rise to a new development paradigm around component-driven development, and that goes parallel with this trend we're seeing with design systems, because obviously, front-end engineering and design heavily influence each other.

[0:12:15.9] JM: React and Angular and Vue, these all have components, the component model varies slightly from component world to component world. Generally speaking, it's the same. I mean, we've got components in each of these languages. As components get more and more widely used, this can affect the world of design and that we can get reusability. In an ideal world, we would be reusing our components all the time. We would be sharing components like we share open-source modules. Does that happen in today's world? Are we able to effectively share components across design specs?

[0:13:03.2] ZO: That's a good question. I mean, I'd say we're still moving along on that journey. The technology is becoming standardized enough that let's say you're a team and you're working in React, you can very easily look around for a date picker component, or a tab component, or what have you and throw that into your design system and your app relatively easily.

There is still work to go on the CSS side. CSS is still more of a leaky abstraction and there are now technologies forming around CSS in JavaScript. We still haven't seen much standardization around that. I'd say that is still a work in progress, but we are moving towards a world where more components, UI components become yeah, very much drop-in in that fashion.

[0:13:56.9] JM: Storybook is a development environment for UI component. Explain what Storybook is and how Storybook is used.

[0:14:06.2] ZO: Yeah, good question. Storybook is simply an application, a very, very simple application that's installed in tandem with your main application via NPM. Once Storybook is installed, it gets access to the same components that are present in your main application just by the virtue of your checked out working copy.

At that point, you can start to write what are called stories in Storybook and these are analogous to test cases. They're basically component examples. In other words, let's say you have a simple text input component. You might make some component examples of stories for that component that represent the empty state, that represent the state of a very long piece of text in

that input box, that represent what that input box might look if there's an error, there's a validation error. You program these states and stories. They live in your source tree.

Storybook is in essence, a tool that enables you to work on your components with mock data with those components being isolated away from the rest of your app. Think of it as a workbench where you can take a component, put it on the workbench, tweak it, add some more functionality and then very easily, in fact it's already integrated into the main application. By and large, yeah, changes and the component level are very easy to make.

[0:15:30.8] JM: How does the workflow of a front-end developer, or a design team more broadly, how do the workflows of developers change with Storybook?

[0:15:42.9] ZO: What would happen, let's say you're working on a new feature that's just come over the fence from the design and the product team and you're an engineer, what you would typically do is fire up Storybook, just have a look at the design in Sketch, or Figma, understand the requirements. Try and think about how the component needs to change and you have the designs for it already. What you would do is you would typically fire up Storybook. It starts very quickly and there are no dependencies, whatsoever; there's no database, there's no staging server. Navigate to the component that you're about to change according to this spec and the new functionality.

You'd probably begin by writing some stories. Just as in test-driven development, you would write the stories, i.e. the test cases that represent the data that component is going to need, the data changes that are going to happen. Then fill in the blanks. Once you have the data, then you would create the markup, create any logic inside the component and style it visually to match the designs.

You can do most of most of your visual, your front-end development just inside Storybook without even ever having to fire up the development version of your of your application. This leads to a much faster process, because you'll notice that in this process, because the developer hasn't had to work within the main application that I haven't had to get that application into the state that's required to build out this new feature, which cannot oftentimes be a cumbersome and time-consuming process.

[0:17:24.7] JM: Let's say I join a company and I'm a front-end developer at this company that I'm joining, maybe it's a company like Airbnb. It's my job at Airbnb to make a new user interface for some new product they're building. Let's say that Airbnb is a user of Storybook. How am I going to be using Storybook in my process of building a new UI?

[0:17:51.1] ZO: Well firstly, I'd have to ask are you building – are you extending the existing UI? Are you building new components? Are they going to be specific to whatever feature you're working on? How do you envisage that? Sorry to put it back on you.

[0:18:07.6] JM: Let's say it's a completely new product. Let's say it's Airbnb golf courses. I can rent a golf course and I'm just building the thing from scratch, except maybe I'm I can reuse previous Airbnb components, if that would be useful.

[0:18:22.1] ZO: Great. Because you've said that Airbnb is already a user of Storybook, what we would probably find there is that there is a component library that represents Airbnb's look and feel, and that is just simply a design system with a bunch of let's just assume, React components for now, that would make it really easy to get started building a new product in Airbnb suite, that's going to have the same look and feel, that the rest of Airbnb does.

In this golf course's product, the engineer would start looking at the designs and the specs that are coming down and identify which of the components in those designs are already present in the company-wide Airbnb Storybook, start rubbing their hands together and saying, "Hey, I've got no work to do their at the component level. That work is already done. I just need to pull those internal off-the-shelf components into my new app and begin using them."

Now what they might also find is that there are custom components, one-off components that need to be written that don't deserve a place in the company-wide component library, but they're only used for one or two strains internally within the new app. Our developer would then proceed to fire up an application level version of Storybook, as opposed to the version of Storybook that's running at the component library level and begin building out those custom one-off components inside that local Storybook, obviously starting by writing stories first and building out components there.

There is an interesting scenario where the new app has components designed for it that may be actually useful within a company-wide component library as well. Let's say, a new style of carousel, or a new mapping component. Then it's possible – again, the developer would probably build that inside the new application first and then work with the component library team to integrate that new component back into the mainline company-wide component library.

[SPONSOR MESSAGE]

[0:20:37.8] JM: Clubhouse is a project management platform built for software development. Clubhouse has a beautiful intuitive interface that's made for simple workflows, or complex projects. Whether you are a software engineer, or a project manager, or the CEO, Clubhouse lets you collaborate with the rest of the product team.

If you're an engineering-heavy organization, you will love the integrations with Github and Slack and Sentry and the other tools that you're feeding into your issue tracking and project management. To try out Clubhouse free for two months, go to clubhouse.io/sedaily. It's easy for people on any team to focus in on their work on a specific task or project in Clubhouse, while also being able to zoom out and see how that work is contributing towards the bigger picture.

This encourages cross-functional collaboration. That's why companies like Elastic and Splice and New Bank all use Clubhouse. Those companies have all been on Software Engineering Daily, and we know they are competent engineering organizations. Stay focused on your project and stay in touch with your team. Try out Clubhouse by going to clubhouse.io/sedaily and get two months free.

Thank you Clubhouse for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:22:08.0] JM: What I found cool about looking at the Storybook technology is that – if I open up a Storybook, I scroll through it and I'm looking at all of these different components. You can

see buttons and widgets and text formatting and sliders and all the different beautiful components that go into designing a user interface. This could be useful for a designer.

There's also the fact that within the Storybook, you can find the code for all of these different UI components. If it's in Vue, then you find the Vue code. If it's in React, you can find the React code. In that, it seems quite a useful place for developers and front-end developers and designers to collaborate. Is there a degree to which the designers and the developers are collaborating in this same place?

[0:23:06.1] ZO: There is. This is an area that I think the tooling can get a lot better than what it is today. One question really that comes out of that is where is the source of truth? Designers have their source of truth, engineers have their source of truth. For engineers, it's always in the codebase. Arguably, this is the best source of truth for designers as well, because – or the code base is what is deployed to customers. That is typically the number one source of truth.

We are seeing add-ons to Storybook and techniques being developed where those components can come back out of the codebase and be exported back out into the design system that's living in Sketch, or Figma, or a tool like that. We are seeing designers get more and more comfortable with this idea that the source of truth is in the code and their design system is constantly being updated with that source of truth as automatically as possible. Then they're able to use that design system to keep designing the next feature, knowing that they're using components that are accurate and up to date.

Where that breaks down a little bit is when designers are working on brand-new components that obviously have never been codified yet. In which case, the source of truth for that time period is in the Sketch files, in the design system, until it gets implemented and then there's a version zero. Then that can kick-start that feedback loop.

[0:24:36.2] JM: There are prominent companies that are using Storybook. These are companies like Slack and Airbnb. Do you know of any examples of how it fits into their workflow?

[0:24:47.3] ZO: I mean, what we've been talking about on this conversation so far is the general workflow that they would have. In terms of specific examples, I can't think of any off the top of my head, but –

[0:25:00.7] JM: The main point is maybe it's a tool – something like Github almost. I don't know if the level of significance is at par with Github, but these tools for collaboration tend to unlock opportunities that we don't even see coming. I just look at Storybook and I'm like, this is a really, really valuable piece of design and front-end development technology.

[0:25:26.5] ZO: Well, one of those opportunities that you mentioned that we're seeing being unlocked is that stakeholders, it's very easy to take a Storybook and publish it on Netlify or somewhere else and do that, even automatically as part of the CI pipeline. Simply give folks a URL where they can see what's being worked on, what the latest state of the components and application are.

That is a really useful way to document basically the source of truth that's present in the codebase and to hand that out to designers, product people and even external stakeholders, clients, CEOs, other business people, and they can simply click that link and in their web browser see the live, or the next to be released feature, even play with it, which is typically very difficult to do at the application level. We're seeing Storybook being used as a powerful documentation tool for representing what's being worked on, in addition to what's available in the component level. Yeah.

[0:26:30.6] JM: Right. Yeah, that's okay. Storybook has been around since December 2015. The contributor base has evolved over time as a healthy open source project. Can you give me a history of how the open source project has matured?

[0:26:44.1] ZO: Yeah. Storybook was started by a guy called Arunoda way back when. He worked on it. He ran a company at the time called Kadira, and they planned to build a commercial product around Storybook at the time, which never eventuated. Arunoda moved on. He's now working at Next.js. There was a vacuum there to fill it in terms of open source maintenance.

The project was already being used at various companies and I would say it was already a thriving project. Suddenly, there was danger that it would fall into being not maintained as a lot of open source projects do.

A number of folks came along. Norbert de Lange is one of them. Michael Shilman, Tom Coleman and many others, in fact who banded together and began maintaining and improving Storybook at that time. I don't know off the top of my head when that was. Maybe early 2017, late 2016. Since then, it has become a thriving community of contributors and core maintainers who have pushed the project forward.

One of the main things that's happened since then is Storybook has gained support for a whole bunch of view layers outside of React, which was originally the only view layer that was supported. Now you have Angular, we have Vue, we have Mithril web components, etc., etc.

[0:28:12.0] JM: You're deeply familiar with the issues of developing the UI layer. The UI layer is getting easier to build, but there are still many, many developer problems there. You started a company called Chroma. Explain what Chroma does.

[0:28:30.8] ZO: Chroma builds tools for engineers. That's who we are at Chroma. We started the company, because we're seeing that there are these new view layers coming out, new paradigms for riding front-end better. There was a gap for tooling. You have companies like Adobe that exist for designers and you have companies like Github that exist for engineers in general. We believe that the time was right to form a company building tools for a specific type of engineers, front-end engineers that have quite a different job than then say an infrastructure engineer.

We founded Chroma. We're heavily involved in the open source Storybook maintenance. One of the first tools that we've built for front-end engineers is called chromatic. It's a UI testing tool that sits on top of Storybook. It's very easy to install it into Storybook and then begin getting automated regression tests across all of the stories that are in your Storybook. That addresses this need that we talked about earlier in the conversation around CICD and a lack of good test coverage at the UI level for teams that are aspiring to do a really good job with CD. Chromatic is the first of several tools that we have planned for specifically – for aimed at front-end engineers.

[0:29:56.5] JM: Can you talk in more detail about the relationship between Chroma and Storybook? Describe more detail, what are some of the tools that you're building on top of Storybook and what's your vision for how Storybook looks in concert with those tools that you're building?

[0:30:15.2] ZO: As I described, chromatic is one of those tools. It's designed to feel really familiar and comfortable to Storybook users. It's also designed to tie into the Storybook product in a way that provides the best user experience for Storybook users. At Chroma, we really believe that Storybook is essential to the new development workflow to component-driven development that is becoming the new norm for front-end teams. We're really excited to keep funding open source development for Storybook. Three of the core maintainers of Storybook work at Chroma.

Our vision is to keep pushing Storybook, the open source forward and adding more and more – some open source, some commercial, like chromatic tools on top of Storybook that leveraged this really powerful component-driven development workflow to make friend and engineers' lives easier and easier.

Another tool that we've announced that's in this vein is called Storybook loop. Storybook loop is going to make it easier to collaborate and comment on stories across teams, in a way that's currently missing from Storybook, because storybook is literally a piece of code that you install in your local application. Storybook loop is going to be a hosted service, where as I mentioned, you can collaborate and share Storybooks much more easily.

Then we have some other ideas that are just very much at the concept stage at this time. Storybook as the platform that's central on a front-end engineer's daily workflow is going to be what our tools revolve around.

[0:31:52.9] JM: UI testing is something that might sound like something – people who are listening to this might think, "Okay, UI testing, this is what I do when I hire the manual testing team, or people to go and click on – click buttons on the UI. There is a way to automate UI testing and this is some of what you're working on at Chroma. Can you split explain why UI

testing is important and what UI testing actually means from the point of view of your tools that you've built?

[0:32:24.1] ZO: Yeah, absolutely. I'll give you a little bit of a picture of the UI testing landscape as well. As you mentioned, manually navigating around an application, create an account and clicking buttons and performing actions is the very most rudimentary form of UI testing that you can do. Obviously, a human is sitting there looking at the application and making sure that it looks okay and that it works okay.

Tools have been around for years. Selenium is the most popular tool that tries to automate this very manual, very human process. It's the best that we've had, but it's not very good in the subtle changes to the application that a human would be okay with, trip up, trip selenium, right? Because it's a computer. It doesn't realize that a color of a button has changed and that's okay, or things have shifted around a little bit and that's okay.

There's a very laborious workload for engineers to keep these is very brittle, end-to-end selenium tests up today. That's the testing landscape. Selenium is central there. The other form of testing for UI that's common is it the unit testing level. That typically doesn't look at the visual appearance of UI. It tests for functionality. If I click the add an extra item to this list button, is there really an extra item in the list and that check is done programmatically and at the unit level, but again, it doesn't look at what that component looks like.

That's the status quo, the state of the art so far before chromatic. With chromatic, we have a set of what sits in the middle of end-to-end tests and unit tests. The easiest way to think about it is adding a visual layer to unit tests where the tests have already been written by engineers in the course of their development. This is during component development in Storybook, developers are writing stories, which are effectively test cases for components.

What chromatic does is it takes these test cases, it renders them and it compares the delta so the difference between the current version of your app, or of your component to the previous version, and it flags any differences. It does this at the granularity of a component. What you end up with is a process that very much slots into CICD in a neat way. That is via integrating into the code review workflow. The way that that works for chromatic is we'll badge a pool request

with any changes that happened at the component level inside as a result of that pull request and flag those changes as either.

Then reviewers who could be designers and could be product people or other engineers, can simply look through those changes and either flag visual regressions, things that don't look right, or accept changes that are correct. Also, it gives them an opportunity to validate any changes that they're not sure that they're correct with other members of the team. Typically an engineer will be working on some design. At the end of that process, they may not be sure whether their work reflects what the designer intended the way that that will surface through chromatic is a visual difference, and then that engineer can fold that – the chromatic URL, or the pull request in Github onto the designer and say, “Hey, is this right? Is this what you intended?”

A long-winded way to circle back to the beginning of the question, chromatic provides an extra level of assurance at the UI level that your app and your components are going to look and work right, without the overhead of having to maintain these complicated and brittle and tests that was the previous state of the art.

[SPONSOR MESSAGE]

[0:36:31.8] JM: MongoDB is the most popular, non-relational database and it is very easy to use. Whether you work at a startup or a Fortune 100 company, chances are that some team or someone within your company is using MongoDB for work and personal projects. Now with MongoDB Stitch, you can build secure and extend your MongoDB applications easily and reliably.

MongoDB Stitch is a serverless platform from MongoDB. It allows you to build rich interactions with your database. Use stitch triggers to react to database changes and authentication events in real-time. Automatically sync data between documents held in MongoDB mobile or in the database back-end. Use stitch functions to run functions in the cloud.

To try it out yourself today, experiment with $10 in free credit towards MongoDB’s cloud products; MongoDB Atlas, the hosted MongoDB database service and Stitch. You can get these $10 in free credits by going to mongodb.com/sedaily. Try out the MongoDB platform by going to

mongodb.com/sedaily, get $10 in free credit. It's the perfect amount to get going with that side project that you've been meaning to build.

You can try out serverless with MongoDB. Serverless is an emergent pattern. It's something that you want to get acquainted with if you're a developer, and getting that $10 and free credit to have a serverless platform right next to your Mongo database is really a great place to start. Go to mongodb.com/sedaily, claim that credit.

Thanks to MongoDB for being a sponsor of Software Engineering Daily. I love MongoDB. I use it in most of my projects. It's just the database that I want to use. Thanks to MongoDB.

[INTERVIEW CONTINUED]

[0:38:39.9] JM: Okay, so let's go a little bit deeper on this. I thought front-end code was pretty simple. It's just doing stuff like rendering a button to a page, or rendering a list of golf courses to my Airbnb, for golf course's webpage. Why is there a risk of performance issues in front-end code?

[0:39:02.5] ZO: Oh, wow. You're correct, front-end code can be very simple. In a world of web development, maybe five, maybe 10 years ago, that was more or less the case that front-end code is rendered on the server, delivered to a browser and it was mostly static and pretty basic. Now what we're seeing is very sophisticated front-end code; web applications that look and feel more like native applications.

If you look at this tool that we're using to record the podcast, you'll see the sound wave that's slowly drifting across the user interface. Now we have these very rich, these very complex UIs that are no longer just flat lists of text. They're oftentimes rendered and calculated and processed in the browser on the client-side. They become serious pieces of code. Front-end engineering is now turning into serious engineering work, where the best companies out there that are building the best apps, the Airbnbs, the Googles, the apps that we use every day and we're like "Wow, these apps look and feel amazing. They're almost magical. They're smooth. They're animated."

We have these user experiences that that are completely new in terms of what we come to expect from the web. Well, these this new state-of-the-art front-end code is no longer as simple as it used to be. That's where tooling like chromatic comes really, really essential to the modern developer in making sure that those experiences still continue to work right.

[0:40:36.7] JM: I've heard that companies have been doing this performance testing at the very top layer for a long time. You hear that classic story about how Amazon always keeps latency under, I don't know, whatever it is, a 100 milliseconds or something, because they've noticed that every request that takes longer than that, they lose profit or they lose a billion dollars a year, or something like that. What's new about what you're building today? Let's say I'm looking at the market for performance testing. What does chromatic do differently than the performance testing tools of the past?

[0:41:17.8] ZO: To be clear, chromatic today isn't a performance testing tool. It's an accuracy testing tool. We're testing to make sure that UI looks right and functions right. Not necessarily that it performs well.

[0:41:31.5] JM: Ah, okay. I see. Okay, so now I understand better, because if I recall what happens when you do a push with the code from your new design, let's say you make a design update, you push it and on the back-end, chromatic is rendering the new design and then it presents it side by side with the old design and make sure that your new code looks like you are expecting it to, is that right?

[0:41:59.5] ZO: It's exactly right. It's exactly right. Changes that it surfaces could be legitimate. They could be related to the feature that you built, or they could be unexpected regressions that you didn't realize you changed one thing over here and something broke over there that you wouldn't have found in the past. Now with chromatic, you do.

[0:42:19.6] JM: Got it. Is there also a side of this to gauging the performance issues of front-end components? Because you can't have some issues like, let's say you make a change to a UI component and it is this list of golf courses. If your list of golf courses is really, really long, or some of the golf courses have some element of their layout that takes a really long time to load, there's some leg of latency that you've added to it, you might want to know about the

performance issues that might be associated with a front-end component. Is there also a layer of performance testing here, or do I get some feedback on the latency of the UI layer? Because obviously, the UI layer in its finished form is one thing, but then there's also the UI layer in the interactive context, or the as its loading context.

[0:43:17.6] ZO: Yeah, good question. What we're seeing, what we have is when teams are working within Storybook, then they will create stories for those very – those funky edge cases for components that say you have a list with hundreds and hundreds of items for it. Well, that's very easy to create a story for in Storybook and that's exactly the perfect reason for using Storybook.

As a front-end engineer, you can develop the happy path for your component in a very simplified manner. Then you can click over to the unhappy path to the lots and lots of data path and see what happens. See if it breaks. Those performance issues at the component level are very much surface during the development process and they'll never even make it out into production.

Now for performance issues arising out of something like data latency, or infrastructure issues, then yes, those problems can happen in production and your Storybook chromatic tooling combination is not designed to surface those problems, those runtime performance issues. A lot of the time, it never gets that far, because issues are identified during development within Storybook.

[0:44:33.4] JM: Very cool. What are the other tools that you have in store for your series of products? I mean, are there other problems that you're interested in solving at the UI layer?

[0:44:46.9] ZO: Yeah. What we're really interested in is digging deeper into this design development handoff and hang back as we're starting to call it from development to design. We're exploring the nuances of that process and understanding how we can better help front-end engineers and designers, product people navigate that handoff and hand back.

Then the other thing that we're looking at is component versioning. Now that we have this proliferation of design systems and components and they're all being shared, understanding

how best to upstream changes, how best to maintain multiple versions of component or design system and work with these in a way that's optimized for the front-end development process. Those are two areas that we're really, really interested in and interested in providing better solutions.

[0:45:40.0] JM: We've done a lot of shows about go-to market strategy for developer tools. Most of it involves going after back-end developers. I feel back-end developers are a little bit closer to the world of the CIO, the world of the CEO perhaps. What's your go-to market strategy for developing sales to more of a front-end UI problem?

[0:46:08.9] ZO: Yeah. It all comes back to moving faster and ultimately saving money during the application development process. With something like chromatic, there are fewer defects, it improves the development process between designers and engineers. There's a very strong case to be made at the organizational level that it allows teams to build product faster that's higher quality and that's better for the user. The challenge can be in quantifying that as a dollar amount within organizations, but they're getting it. We're seeing them understand and accept that.

Where it's different from commercializing back-end technologies is that there, the sales pitch is in some ways a lot clearer. It's like, "Okay, you have this code and it's running in production." There's a lot of justified fear around that working well and being fast. I think that's why on the back-end, it's easier to make that case. With front-end tools, it's possible to make that value proposition in the form of workflow improvements and time to market improvements and that sort of thing. Does that answer your question?

[0:47:20.3] JM: It does. Yeah, definitely. I think time is on your side, because over time we've seen developers on the back-end become more and more able to procure things. We've seen the people on the back-end be able to buy things from AWS directly, as opposed to having to go through some procurement department. Now there's obviously cost oversights, but you see more and more that developers have control over a budget in an organization. There's no reason to think that that wouldn't continue to propagate and flow more and more to front-end developers buying more and more front-end tools.

In fact to some degree, it feels today the front-end developers don't have as much stuff that they can buy to save their time. I think that if they could buy stuff off the shelf and save more time and not spend as much time developing a front-end UI layer, things would be better. It does seem like there are opportunities for people to come in and build new tools for the front-end.

[0:48:29.9] ZO: That's exactly right. What we're seeing and hearing even in very large enterprises is that individual contributors, the developers that are on the front lines are being entrusted more and more to buy the tools that they need that will make their work better. Logical, right? Yet, years and years ago with enterprise software, it was the other way around.

Enterprises are catching on and becoming more efficient in that sense by providing a credit card that developers can use to buy relatively cheap SaaS products. Yeah, you're absolutely right. We're seeing that with our stuff as well, is that the barrier to adoption within even very large enterprises is incredibly lower for that reason.

[0:49:14.5] JM: One other canonical problem – well, sad that this is a canonical problem, but perennial problem of the front-end engineering is the disparity between the web developer and the Android developer and the iOS developer, there have been some platforms that have tried to address these in a unified fashion, like React Native or Flutter. Are you optimistic at all about these unified languages, these unified platforms for development, or do you think it's still too early?

[0:49:48.0] ZO: That's a great question actually. I've been very optimistic and very disappointed and very optimistic, almost as in riding a roller coaster over the last few years with these technologies. Because obviously, there's a big draw cut having to maintain one code base that works across everything. However in reality, what you often find is that you're maintaining one code base. It's a substandard user experience across all of the various platforms and you're spending a lot of your time chasing weird arcane bugs. It almost ends up being a whack-a-mole type situation.

We're starting to hear the same thing now about React Native, which rewind the clock two years, folks like myself were very optimistic that hey, this is maybe the model that's going to

solve it. Now we're hearing from companies that are turning away from React Native and going back to maintaining three separate code bases.

I think if I was to look at the data I'd say even though we want a technology like this to exist, time and time again, these technologies are failing. The big caveat that I'd make there is if you're really – if what matters to you is not so much a really great user experience across all platforms, but time to market and a core user experience that provides an incredible amount of value to your customers without necessarily being really polished really pretty, then I would say these technologies like React Native are really, really good. In fact, better than ever in terms of building that experience and getting a really low time to market for those type of apps.

[0:51:31.2] JM: You helped launched Apollo GraphQL. GraphQL is having an impact on the front-end, or maybe you would say middleware between back-end and front-end, front-end. What's your impression of how GraphQL is impacting the software development world today?

[0:51:52.8] ZO: It's putting a lot of power in the hands of front-end engineers. It's giving them the opportunity to move much faster than they could before. Whereas previously, they would need to go to a back-end team and have these custom endpoints written for them; oftentimes, for every little feature. Now with GraphQL, they don't necessarily need to do that, so they can get a feature built end-to-end without having to work with any other teams internally. That's really speeding up the time it takes to build features for front-end teams.

Then on the backend side, well it's also freeing up back-end people from having to write these boring, almost boilerplate endpoints that are just pulling data together, that the front-end teams need. I'd say technologies like GraphQL have made the lives of both front-end and back-end engineers a lot more pleasant in the day-to-day, and have also sped up applications for end users, because they're loading much less data that they used to and fewer requests.

[0:52:55.2] JM: Okay. Well Zoltan Olah, thank you for coming on Software Engineering Daily. It’s been great talking to you.

[0:52:58.7] ZO: Yeah, my pleasure Jeff. Any time.

[END OF INTERVIEW]

[0:53:03.4] JM: DigitalOcean is a reliable, easy-to-use cloud provider. I've used DigitalOcean for years, whenever I want to get an application off the ground quickly. I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU-optimized droplets perfect for highly active frontend servers, or CICD workloads.

Running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily. As a bonus to our listeners, you will get a $100 in credit to use over 60 days. That's a lot of money to experiment with.

You can make a $100 go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure and that includes load balancers, object storage; DigitalOcean spaces is a great new product that provides object storage and of course, computation. Get your free $100 credit at do.co/sedaily. Thanks to DigitalOcean for being a sponsor.

The co-founder of DigitalOcean, Moisey Uretsky was one of the first people I interviewed and his interview was really inspirational for me, so I've always thought of DigitalOcean as a pretty inspirational company. Thank you, DigitalOcean.

[END]