EPISODE 936

[INTRODUCTION]

[00:00:00] JM: Jeff Rothschild was one of the earliest engineers to join Facebook. In the 1990s, Jeff had founded and sold Veritas Software to Symantec and worked on several other of his own companies. He was working with Excel Partners on investments when he started to learn about Facebook. Excel was an investor in Facebook having made a large investment in 2005. After that investment, Jeff was helping the company with hiring.

As Facebook built out its senior engineering leadership, Jeff realized that he would enjoy working with the company in a more active role. For the next 10 years, Jeff Rothschild worked as a VP of infrastructure software at Facebook. He helped scale the company's infrastructure and architect the technical strategy.

By 2005, Jeff's entrepreneurial background included software companies ranging from storage management to gaming. His diverse skillset was useful for setting product direction as well as solving novel engineering problems.

Jeff joins the show to discuss the early days of Facebook and his philosophy of entrepreneurship.

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

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

[SPONSOR MESSAGE]

[00:02:02] JM: When you listen to Spotify, or read the New York Times, or order lunch on Grubhub, you get a pretty fantastic online experience, but that's not an easy thing to pull off, because behind-the- scenes, these businesses have to handle millions of visitors. They have to update their inventory or the latest news in an instant and ward off the many scary security threats of the Internet. So how do they do it? They use Fastly.

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

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

[INTERVIEW]

[00:03:16] JM: Jeff, welcome to Software Engineering Daily.

[00:03:18] JR: Well, thank you very much.

[00:03:20] JM: You joined Facebook back in 2005, and some of the Facebook engineers who I've interviewed who were present around that time said that when they've joined the company, they found some things surprising and almost alien to the way that the engineering and the product worked. What did you find surprising about the way that engineering at Facebook worked when you joined?

[00:03:47] JR: I joined in June of 2005. Facebook was a very small company. Most of the code was written by Mark and his roommate, Dustin Moskovitz. From a software engineering standpoint, what would be surprising, well, there was no source code control. This was a dorm room project initially. I mean, of course, that's the mythology around Facebook. But guess what? It's true. Mark was writing this and he and his roommates worked together and they could easily discuss what do you check in, what should be in the release and who can edit what module.

Yes, when I started, we had no source code control. We had no deployment process. We had no auditability in our code. No standardized methodology for logging, for error reporting of metrics. Is that surprising? For a small project, not necessarily. Everyone starts that way. You don't know how large things will be until you get underway and then your process and methodology needs to catch up to you.

[00:04:49] JM: One of the core things about Facebook that makes it different than some of the other applications that had scaled by 2005 is that it's multiuser. You look at something like Google, and for the most part, a search engine is a single user application. Same thing you could say for Amazon. How does the multiuser nature of Facebook give it unique scalability and infrastructure challenges?

[00:05:18] JR: I think that's exactly it. That is the attribute of the product that creates the scalability challenges. Remember, you're not looking at your own data. Facebook isn't a place to go and see your photos, your posts, to remember who your friends are. Facebook is a place to see what your friends have posted. To see your friends photos. To surface what is potentially interesting to you within your network. That's inherently more complex problem and presented some of the more significant scalability challenges for the company, because as your network grew, then the cost of being able to surface and evaluate that data and determine what is going to be shown became a more complex problem.

[00:06:04] JM: As the company matured, were these multiuser problems, were they still canonical or did you find particular patterns for solving multiuser problems?

[00:06:16] JR: Over time, patterns evolved that were successful. Other patterns that were less successful fell by the wayside. One of the earlier observations was is that you couldn't pre-calculate what a user would find interesting at the moment.

For example, they would see their homepage, rather their newsfeed. So the observation was is that this should be done dynamically, and therefore our focus would be on being able to, in real-time, evaluate potential content and then make a determination as to what will be the most engaging and valuable content to surface to the user. We went through a number of models, but

there are certain paradigms today such as doing these things like this on a dynamic basis that have clearly emerged as most successful.

[00:07:04] JM: Can you give me a specific engineering problem that you recall around this idea of the real-time responsive kind of calculation?

[00:07:19] JR: Sure. In the first version of newsfeed, we took more of a supercomputer approach, which was we installed some very high-performance disk systems and figured, well, the way to build the newsfeed would be to scan through all of the users, determine in some abstract form the collection of stories that would be most relevant to them and then wait for them to come into the site. Of course, you take a step back and you say, “Well, what if they didn't come in?” Then you just expended the cost of building this context and then never had the opportunity to use it.

Of course, the next time you do the scan, if you end up replacing it, that work served no value. Those were cycles, and time, and heat that were expended without a payback. That was a very clear case, a recognition of doing this on demand and to focus on computer organization and software architectures that would support real-time generation of relevancy and context.

[00:08:20] JM: There's this classic computer science law you’ve’ probably heard of called the Conway's Law, where the organization supposedly – The communication structures and the organization supposedly reflect I think the software architecture, the systems architecture, of the problem that you’re actually solving. Do you think that applied to Facebook?

[00:08:43] JR: Over time, I would say yes. Of course, early on, it’s one team. Everyone knows all systems at all code. As a project becomes more complex, the surface area of systems and code because greater than any individual can comprehend, then it becomes necessary to have specialization and to have not just specialization in terms of code familiarity but also in terms of skillset and methodology that matches the problem. Then organizational structure necessarily follows from that. That is, I would say, was reflected as Facebook developed a product team and an infrastructure team over time.

[00:09:28] JM: Tell me more about the engineering processes that developed within Facebook during your time there.

[00:09:34] JR: As I alluded to, there weren't a lot of processes when I showed up. I should say that this was a LAMP Stack. A term that we don't really use any longer, but Linux, Apache, MySQL, PHP was a common formula for building a website in 2005. Recognition that the company wouldn't have been able to have moved quickly and developed this product without using open source software, I think that that recognition of the value of open source and the obligation to the open source community to return the favor is still evident today.

Engineering processes evolve and they don't just grow. You replace what you're doing. One of the traps that a larger company falls into is that every time there is an incident, every time there is a failure, some new process gets layered on the development team. But all that does is have the consequence of slowing people down, and that's the best it can do. The worst is, is that it steals from potentially higher return processes that already were in existence at the time that the new process was added.

One of my outages is that engineering process, frequently, it's a case of understanding where you have a return on investment and doing your best to make that highly effective and resist the temptation to always be layering on new process.

In my opinion, most engineering teams will find that code review is possibly the most valuable thing they can do, and I don't think there is no contention around the notion that code review is valuable. However, if you truly believe in the value of code review, then you also should think about, “How do I make my code review more effective?” because you're going to invest time and it. It's going to take a certain amount of time whether it's an unstructured walk in the woods or a very structured game of golf.

I like to see code reviews that are preceded by the developer, the one who is submitting the code for review, writing down their assumptions. This is what I believe could go wrong with this code. Here are the things it could impact. Maybe a bullet list is all that's needed. No prose is needed here. Just a list of things. I could break the following things with this check-in.

Another list, let's say parallel list to that, of here are the things – Here's how I verify that it didn't. So it's effectively this is my test methodology for ensuring that each one of these things that I can envision didn't actually take place. Now, for the code reviewer, they now have a reasonable understanding of your assumptions. They know that they can look at the code and say, "I see that this could break an additional feature on the site."

If you didn't write down what your assumptions were, then that reviewer could say, "Well, of course they do. It was going to break that feature." So I don't even need to look into that further. But if you've created a list of 10 things that this could impact and they think of an 11th, then that's a red flag. They know, "I now need to look at that and I need to call that out because it hasn't been addressed by a corresponding test action."

Also by showing how it is you tested, you will give the reviewer the opportunity to question whether the efficacy of those tests. So rather than responding to an incident, or responding to a quality problem, or an availability problem with new process, maybe the answer is get more out of the process you're already doing. Sort of the other side of that is, is we have a certain capacity for doing things other than building our product.

When you ask people to do new process, you're actually taking time away from old process. If I had to bet, the first thing you decided to do from a process standpoint was probably more important than the new step you're asking for. So less is more. More is very frequently less.

[00:13:58] JM: Does less is more, more is very frequently less, apply to volume of tests?

[00:14:06] JR: No. To a degree that you are able to capture tests and roll them forward and then potentially replace those that are relevant. That is like working with a safety net. Every developer tests, and if you're able to capture that work product in a way that it can form your regression and potentially capture the next error, that's a good thing.

I think that the investment in test infrastructure typically has a very real payback. I know that in many organizations is a debate in terms of is this worth it? The right answer is possibly all over the map and its project dependent. But in general, I think that using unit test as the basis for

building regression is generally a good idea. That's sort of an abstract way of describing what I'm trying to capture, which is that the testing that a developer does in order to assure that what they wrote is correct to the degree that you can capture that and have that serve as a part of your test foundation is generally a good thing.

[00:15:15] JM: What about in a highly complex application where you can't test a large percentage of the surface area of that application? What should your approach be to testing?

[00:15:29] JR: Very much context dependent. I can give you an example of that from Veritas Software company that I had worked at and cofounded many years back. We were building kernel code for the logical volume manager and file system of UNIX, and our customers were computer vendors. Computer vendors who had teams of people who prior to being approached by Veritas owned that functional area of the operating system. Not surprisingly, there were a few people who might've resisted the notion of their company buying the piece of software that potentially defined their career.

So we had to do our job well, and there were very little opportunities to get it wrong, which of course results in corrupted data or system instability and still be able to maintain these particular system vendors as customers.

So we made a very rigorous investment in a testing methodology where we could encapsulate both kernel level codes, so device driver code and application code, and exercise every potential API return back into that code. If you're calling some system call and there were seven different error returns over the course of testing, our test framework would insert those errors back into your code so that you could then exercise all of the code paths that would be very difficult to exercise in a live system, such as the kernels out of memory buffers. Well, if a persistent kernel is out of memory buffers, whatever instrumentation you're using isn't going to tell you anything. So you need to be able to fake that response as best you can in order so that you can measure it in a controlled manner.

Complementing this was a tool that would actually look at every code module and find every unique path through that module. You enter here, and here are the ways you can leave, but here are the intermediate branches that you could execute and it would build a database of

every unique path. Then as tests were run, it would annotate which paths have been covered in which paths are left uncovered, and that's where these emulation libraries came into place, because since most paths are going to be error paths, you're not going to be able to cover them. You're not able to validate, but you can execute those paths without the ability to push the code through those error conditions.

At priory, we set a path coverage target for our code. I think 70% of the paths were covered for application code. Maybe it was 90% for kernel level code and you use the test methodology to drive ourselves to those points and which case we could declare victory and ship the product. That was inappropriate methodology for a company that would've had a difficult time surviving significant data loss or introduction of instability in a customer system. In a different environment, of course, that type of that level of rigor is probably the wrong tradeoff. So it's very much context dependent.

[00:18:46] JM: Were there any ways in which that kind of simulative testing was useful for Facebook and exploring the kinds of paths that would be hard to enumerate and testing infrastructure?

[00:18:58] JR: It's dangerous when you say, “Well, how does engineering work at Facebook?” Facebook is a big company. Facebook has kernel developers and their life and their day is spent very differently than someone who's working on the analytics team and it’s going to be very different from someone who's working in product or is building mobile code. It's not going to be a one size fits all methodology. Yes, what I was describing, would be appropriate for kernel developer and wouldn't surprise me if kernel developers at Facebook and other organizations today adopt similar levels of rigor.

[00:19:38] JM: You alluded to your experience with Veritas, you've started several companies prior to joining Facebook. When you joined Facebook, did you have a feeling of kind of adult supervision, “I've been through this rodeo before. Here's how you do the entrepreneurship thing,” or was it more of a beginner's mindset?

[00:20:01] JR: Some of both. It's a terrible mistake to assume that what is ahead of you looks just like the past. In fact, if that were the case, it would give up today, because it's not going to

be interesting. What attracted me to Facebook was that this was a challenge that I personally hadn't seen before and also was a product that people cared about.

When I first showed up, and I was only going to spend maybe two weeks helping Mark and the team recruit some engineering leaders and operations team folks. I wasn't really planning to work at the company. But after a few days, I took a look at the mailbox, the inbound email, and I think there were 70,000 messages that queued up. Again, this was a very small team and had a 2 million users at that point. The ratio of people to users, or users to people is very high.

But what impressed me was the sentiments of people using the product to the degree to which they felt that it made a positive impact in their life in terms of their ability to integrate into a new environment. You had people show up at college and they don't know anyone there, but Facebook served as a tool for them to be able to form these new relationships and discover who was around them. It also helped them maintain connectivity back to people who've gone off to different schools. They had friends who were at other universities. Again, we were just a college product at that point in time.

But it was a very clear message of this is enriching people's lives and made people happy. There are companies that spend a lot of money trying to convince you that they make you happy. Think of a soda company who believes that they are flavored water. They might delay by spend hundreds of millions of dollars a year trying to convince people that their brand or sugar water will make your life go better than another brand. But they don't really make people happier. This did, and that really meant something to me.

Veritas was a successful company. We had $1.5 billion dollars of revenue, a little bit more. We had 5,000 customers. This was a successful firm, but I never got a letter from a system administrator who said that they were happier because of the Veritas volume manager. Just never got that letter. Here was a whole mailbox full of people who said this made a difference in their lives, and I was just intrigued by that. It made me think, “Maybe I don't want to just do this for two weeks. Let me try to understand what it was about.” I took a little bit of time to really understand what Mark was trying to do and made that commitment to stay on.

[SPONSOR MESSAGE]

[00:22:56] JM: Better.com is a software startup with the goal of reinventing the mortgage industry. Mortgages are a $13 trillion industry that still operates as if the Internet doesn't exist, and better.com is looking for engineers to join the team and build a better mortgage experience.

The engineers at better.com are attacking this industry by bringing a startup approach into an industry filled with legacy incumbents. Better.com automates the very complex process of getting a mortgage by bringing it online and removing the traditional commission structure, which means that consumers can get a mortgage faster, easier and end up paying substantially less.

Better.com has a modern software stack consisting of Node.js, Python, React, TypeScript, Kubernetes and AWS. They iterate quickly and ship code to production 50 to 100 times every day. Better.com is one of the fastest growing startups in New York and has just announced a series C that brings the total funding to $254 million.

If you're interested in joining a growing team, check out better.com/sedaily. Better.com is a fast-growing startup in a gigantic industry. They're looking for full stack engineers, front-end engineers, data scientists. They're looking for great engineers to join their quickly growing team. For more information, you can visit better.com/sedaily.

[INTERVIEW CONTINUED]

[00:24:34] JM: My sense is that your sense of intrigue and curiosity has led you down a number of valleys, some of them blind alleys, some of them very useful detours to your life that wound up being successful. There's a term that I've heard you use called collateral knowledge. Can you explain what collateral knowledge is and how that fits into your framework for how somebody should go about gathering the useful aspects of the world?

[00:25:15] JR: Well, it's interesting, because I'm not 100% sure that I agree with the concept myself, but I still mention it because I can draw such a distinction from the world I was in when I started as a software engineer 40 years ago and today. Back in the 1970s, we had stacks of documentation. If you were to work on a given operating system, you had a documentation set

that was probably 2 feet wide, and there was a book for everything. I mean, for the machines language of the system, the individual system calls, the language runtimes, the job control language. So many different aspects of what defines a system environment.

If you needed to know something, there was only one way you could get it, which was a somewhat linear search through this documentation stack. You might get to the right book, but after that, your journey to the right answer involved a lot of stops.

I found that I learned a lot on those journeys. When on the advent of the Internet or shortly thereafter, when people started to build hypertext documentation, I found that you could get to that right answer so much faster. Stack exchange, I mean, these types of Stack Overflow, these types of services really enable us to of very quickly hone in on someone who's done exactly what we have done and we can copy what they've done. We can restrict our learning to their learning except they wouldn't have probably been able to comment on it unless they themselves had gone through somewhat more circuitous path as well.

My observation is simply that highly targeted information retrieval obviously saves time, but there is a cost associated with it and we should simply be aware of that cost and choose sometimes to take the long route and potentially learned a few other things on the way to what we were looking for so that we're always developing a broader footprint of knowledge and a deeper understanding of the abstractions we use and the environments in which we program.

[00:27:29] JM: Isn't the net time spent and the net knowledge acquired the same if you're going a circuitous route to find one piece of knowledge versus a direct route to finding as many pieces of knowledge as you would find during the process of going to that circuitous route?

[00:27:53] JR: You might think, but I don't believe that's true, because you're not searching for information the whole time that you're working on a problem. You're mostly just doing direct implementation and debugging. The points in time where you go out to find something new, those are relatively infrequent relative to how you're spending the rest to your day.

The difference would be between, let's say, you need to know how does this – Some behavior, some side effect of a system call, and people have different responses to sort of lacking this

knowledge. One is you lean over to the person working next to you and say, “How does this work, or have you use this before?”

Another might be to go read the source code or maybe write some test code if you can't read the source code for what it is you're concerned with. I would contend that the person who reads the source code or writes the test case is going to understand the behavior of that system and their knowledge footprint will be significantly broader than the person who leaned over and asked their friend for some help.

While they may take longer to get to a particular problem, overtime, their efficacy as an engineer simply will eclipse the one who initially asks very, very clearly. I'm hesitant to say that this is always the desired behavior. It clearly isn't. There is a time to ask. There's a time to go to Stack Overflow, but there is also a time to say, “Let's learn a little bit more about this.”

[00:29:37] JM: If we map this to entrepreneurship, it arguably indicates that there are plenty of entrepreneurial journeys that people go on. I think you've been on some of these, where by the time you get to the end of it, you think, “God! This was probably a waste of time.” At least from a profit-making standpoint, it ends up this was a circuitous journey towards nowhere, or I made some mistake along the way, or something like that. But then that circuitous journey ends up factoring into some brilliant insight that you're able to have in a future success.

Do you have any particular examples of how one of those circuitous journeys has turned into a successful insight?

[00:30:28] JR: Well, rather than to take a stab at that one, I mean, there certainly many examples in the industry of products that were not the intended product of a company. So they set out to do A, and in the process they had to do B as part of, let’s say, a tool to get there and found that that tool was the product

At Veritas, we actually spun out a separate company to build and productize the toolset that I described what we never would've set out as that as the objective. But what I probably would say is that you can't always be right. If you write about every product hunch you have, then you're probably behind the curve. Even within an organization that has a product, there's going

to be features that you're going to build, and those features end up falling flat. You could view those as mutations.

Let's just use an analogy to biology. If you don't have any mutations, there is no evolution. If there is no evolution, then the organism is eventually going to not be adapted to its environment and fall extinct. The parallel back to a product is pretty obvious. A company that doesn't innovate its product eventually loses its market.

If those innovations to the product are all successful, then it probably means that they're iterating the product too slowly and too carefully. I like to see some number of those product iterations, those product changes fail, because it just as in nature. Most mutations are not beneficial. Most of them end up not working and maybe it might be a bit too much to say most of the things you do with your product should fail, but certainly some of them should, or else you're probably being too conservative.

[00:32:26] JM: You went to this evolution as an individual where you initially wanted your career to be as a psychologist, and your advancement in the direction of computers only happened near the tail end of your undergrad. Eventually, it seems like your prime passion has become entrepreneurship, and I think entrepreneurship is it's more in the water today than it was probably when you were an undergrad and when you were in grad school. I guess I'd like to know how your focus evolved to become entrepreneurship and how you learned that that was really the foundational thing that drove your passion.

[00:33:16] JR: Well, it's interesting. I don't think that it was a cautious decision. Just an observation that I needed something to get me up out of bed in the morning. It's a bit like seeing a movie. If you've already read the book, if you've maybe even already seen the movie, then watching a movie doesn't provide the same level of engagement and excitement than it would if you didn't know the outcome of the story.

Well, if you go to work and you know the outcome because you've gone to work at a big company, a big successful company, then there's probably – Unless that company is facing some existential threat, for me personally, it was less exciting. I think I needed to have that fear of going out of business, that fear of – Or that challenge of everything is stacked against us.

Let's see if we can make it work. That's what provided excitement for me and that motivation to say, “Okay, let's get up and do this again.”

I worked at some larger companies when I left school and I found that that was missing and asking myself, “Okay. Why is this not as exciting as I thought it was going to be?” Led me to, “Okay, let's try tilting windmills a little bit. Let's try something that has some risk in it.”

Now, in the observation of how I got into computer science, I did have a background in psychology and I didn't set out to be an engineer. What happened for me was a psychology professor asked me to write some statistical software. I went ahead and took a class in FORTRAN and learned to code, and then wrote this cluster analysis or whatever it was I was doing at the time and had an observation. The observation was this was a lot of fun. I had an idea for something in my head. I envisioned a solution, and through these skills I had recently learned, I was able to realize that solution and then I was able to put it to work to create some good. People recognized that. That entire process brought me joy. That was just a fun thing to do and if I had not been asked to do it by a professor, I still – If I had known how much fun it was, I would've done it anyway.

That was what made me say, “Well, then this is what I’m going to do for a living. I want to experience that joy every day of my life rather than sit in some office selling commodities or figuring out insurance actuarial. I want to go through this joyful process.”

But observation has helped me in managing teams, because I actually believe that the experience I had is pretty universal. I think most people in the field of software engineering had that experience of creating something and taking something that initially existed just in their head and bringing that to life and they found joy in that process and said, “Okay. This is what I’m going to major in. Here we go. I found a path.”

Now they go to work at a company, and the company has a mission. The company has a goal it’s trying to achieve. Immediately, there's this tension between I have ideas of my own and I want to realize them, and I'm now part of a team where everybody needs to row the same speed in the same direction and stay in cadence. It isn't necessarily a slave galley. Maybe it’s

not quite that grim an analogy, but it isn't the same thing you experienced when you were starting out.

In fact, I would say that I probably have interviewed thousands of people over the last 40 years. I still am waiting for someone to tell me in the interview that the reason I chose computer science as a career is because I'm excited about the idea of implementing other people's ideas. Nobody has ever said that to me. Obviously, nobody ever will.

However, it raises a bit of a paradox. How can people be – How do you make people happy in a job where there actually is going to be some constraint and some discipline when the thing that brought them into this actually was the freedom of being able to work on what interests you and being able to exercise this creative process?

I believe this is where good management comes into play. I believe that recognition of this tension is what separates an effective manager from one who is simply stumbling through the job. Unfortunately, most new managers, people who are trying their first-hand management either because they started the company themselves and suddenly it's not just them anymore and they need to be directing other folks, or their boss tapped them on the shoulder and said, “I'm transitioning you to management.”

The tendency is to feel a little bit insecure about this new role, and people frequently deal with that insecurity by trying to impress upon their team that they are a great engineer, and they’re still a great engineer. That's why they deserve being in this management role, but of course that's probably the worst thing they could ever do, because if you walk into a meeting and you say, “Okay, here's the problem we’re going to solve. By the way, here's how I think we should solve it.” Because you want the team to know that you're really a leader and you’re not just a guy or gal who's pushing papers and doing reviews. You’re still an engineer.

You've sucked the joy out of that engineering process. The real effective manager walks into the room and says, “Here's the problem that we need to solve,” and then shuts up.” Even if they have a strong idea in their mind of how that problem should be solved, just keep quiet. Let somebody else put the ideas on the table. If you think that the idea being presented isn't as good as what you would suggest – That’s what you were thinking, then the right way to bridge

that gap is ask some questions. How will it deal with adding these features in a later release? How will it address integration with this other product?

You just might be surprised that the person who put that idea forward has a really good answer for it and – But you get a good job in keeping quiet. That was an effective approach. You also may find that someone scratches their head and say, “Well, I didn't think about that. So here's a 10° course correction.” Then they change what they're proposing. But at the end of that, it's still theirs. It's still there idea. They have the intellectual ownership of that work.

As a manager, your objective is to leave that room with the largest number of people feeling that they have intellectual ownership of the work product of the team, because by addressing people's emotional engagement with their work, all of the other things that you need to do as a manager suddenly become easier; schedules, quality, commitment, communication. All of these other dimensions that we think our job, our primary job, actually are a lot easier to achieve if the team feels, “There’s something of me in this. I'm doing it because it's my idea. I have a stake and its outcome and not just a financial stake due to the success of the company,” because as an organization grows, you lose that. By the time you're at a few hundred people in a company, most people will realize that their own contribution is probably not going to be directly affecting the outcome.

So as a manager, you need to have different tools, and I think that recognition of why people got into software engineering and focusing on emotional engagement as your primary metric of success as a manager is going to lead you to success with your team and success of the people on your team.

[00:41:39] JM: That approach works well for a manager that can go to their team and say, “We’re building X. You have the freedom to do it. Go.” The team can get riled up about and excited and passionate about solving X however they want to.

[00:41:59] JR: I wouldn't just say go. I think that it's tell me what you're going to do. I strongly believe in managing through questions to the degree that you can avoid definite assertions about an architecture or design. The stronger you will be as a manager if you ask questions

about the behavior of a proposed system. You haven't transferred ownership of the idea to yourself. So it's not just a question of go, but it's knowing how to interact with your team.

You are a manager for a reason. I mean, it is that you have this insight. It is that you are able to assure of the efficacy of the work product from that team. It's just how you go about it makes the difference in terms of whether the team owns that work or whether you own that work.

[00:42:40] JM: Certainly. But if I'm the engineer that says, "I am not passionate about X. I don't want to work on X. I want to go and invent Y, or I want to go and work on team Z." Is there a way to reliably retain and please those engineers or are some of those engineers just going to be the ones that end up leaving the company do their own thing?

[00:43:08] JR: Probably a little bit of both. Certainly, if the company is at a state and a scale where you can accommodate their interest in Z and that actually is aligned with an objective of the company, then that's probably the right answer.

In other cases, they may be interested in doing something that doesn't align well with the interests and objectives of the company, in which case there may not be a place for them there. They may not have the maturity to be able to say, "Okay, I need to focus my attention on this problem set and then find some joy in contributing solutions in that domain." If they're not able to do that, then possibly they don't have the emotional skillset needed to be successful on your team.

It's similar to the notion of achieving consensus. One of the mistakes that I've made as a manager is I have tried too hard to get everybody on the same page. I want everyone to agree that the path forward that we're taking is the right path. But the truth is I'd gone out and hired a lot of really smart people. Smart people don't necessarily see the world in the same way. We're all victims of our own experience. We all have come to the conclusions we have because we've been exposed to different solutions and seeing different success models.

There will be a diversity of opinion, but a good rule of thumb is to say, "Well, what would you need? What would you expect of any organization you worked in?" I think that you can draw a real distinction between your opinion being valued and respected in getting your way. People

who need to get their way probably are going to be toxic to a team, because they're not always going to get their way and they're discontent, and that discontent, it can be contagious. It can lessen the experience for those around them. But you do you expect that your ideas are valued.

So is a manager, your goal relative to this issue of consensus is solicit everyone's opinion, especially someone who should have an opinion in the area, because it's close to the domain in which they are working or have worked in the past. Value people's opinion. Make a decision. Make that decision as openly as possible, "Here is why we made the decision this way." Acknowledge that input that you're not following. Joan may have had an idea that is really interesting. It may have been the right answer. In fact, it may actually be the right answer, but we're going to go down this other path. If we're going to keep measuring it and we'll keep an eye on it, because if it isn't right, then we're probably going to want to go back to the idea that this other individual had submitted.

By acknowledging and valuing as many of the opinions of a team as you can do gracefully and sincerely, you create an environment where people are comfortable putting ideas on the table. It doesn't have to be the winning decision. If the only path that is ever valued is the one that was the consensus opinion or was the eventual outcome, then people may be reluctant to speak their mind. But if you value and acknowledge those ideas that you don't follow, people are more likely to put them forward. But more importantly, as a team grows, you're going to have a diversity of opinion and not everyone's going to get their way. So don't create the expectation of consensus. Create the expectation of we're going to pick something and we're going to explain the reasons why we picked it and we are going to value and recognize those which we don't follow.

[SPONSOR MESSAGE]

[00:47:01] JM: As a programmer, you think an object. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers and the cloud area. Millions of developers use MongoDB to power the world's most innovative products and services, from crypto currency, to online gaming, IoT and more. Try Mongo DB today with Atlas, the global cloud database service that runs on AWS, Azure and

Google Cloud. Configure, deploy and connect to your database in just a few minutes. Check it out at mongodb.com/atlas. That's mongodb.com/atlas.

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

[INTERVIEW CONTINUED]

[00:47:57] JM: Facebook was started before the cloud became present or popular. As a result of that, Facebook scaled and developed a ton of platform infrastructure that is unique to Facebook. My understanding is that an engineer who's building product inside of Facebook has an experience that is not unlike somebody who has access to a cloud provider with a wide breadth of tools available. But it's not completely like somebody using the cloud providers that people are familiar with. It has its own has its own unique characteristics.

Now I know you left the company. You're gone by 2015. The platform infrastructure is no doubt changed a ton since then. But do you have a sense of how Facebook's platform infrastructure and platform engineering is completely unique from anything else that you've seen in the industry?

[00:49:06] JR: I would say that Facebook has traditionally valued open-source. I mean, we wouldn't have gotten started as easily without open-source being available. Facebook has contributed quite a bit back to the open source community. Many of the tools and systems in use at Facebook are found outside the company, things like React, Cassandra, improvements to MemCache, lots of infrastructure components were initially started there.

It's probably not accurate to say that it is unlike what you find outside the company, but there's going to be other systems that don't really offer, wouldn't make sense is an open source product. They're too tightly integrated to the product itself to that as the Facebook experience or to other internal systems and they're not good candidates for open sourcing.

I think that the dimension that is probably more interesting in terms of the tension between having standardized infrastructure and then letting everyone do their own thing and encouraging

people to do their own thing is similar to the idea I mentioned earlier about products needing to evolve or potentially go extinct.

If you centralize architecture or you've centralized infrastructure and you use common services for everything you do, you certainly can achieve a great deal of symmetry in your software, commonality in the way reporting and logging are done and that metrics are serviced. There's a lot of efficiency that comes from that. But then you're going to lose that opportunity potentially be using a best-of-breed surface that might more accurately match the new demand of whatever problem it is you're working on.

The trick is finding the balance between always using the existing infrastructure that is sort of horizontally deployed and common across projects and building new things. But if you go too far to just using the common infrastructure, then you're not going to be evolving. You won't be exposed to new open-source projects that somebody wants to try out or a new architecture that someone envisions and wants to develop.

It's a bit of a fine art, and I can't say that any manager is ever going to get this 100% right, but you need to strike a balance. Obviously, having every new system have its own infrastructure stack is not the right answer, but never doing it is also very clearly not the right answer. I think Facebook has, from what I can see standing on the outside of the company today, has struck a pretty good balance between making sure there's always a new stream of innovative projects that are some may be successful in some they may find are not successful. But if you don't have these mutations, let's say ,then you're not going to ever have the adaptive mutation. But also trying to achieve efficiency and development efficiency by having teams that are building common resources that are somewhat akin to a cloud provider.

Back in 2005, to answer the other part of your question. No. There were no cloud providers. If they were, if they had existed, we would probably have done a number things differently. There would've been problems we didn't have to solve. So we could have then focused more on those things which were truly unique and unique to the Facebook product, but we didn't have that luxury. So we had to build some infrastructure services that today a small company would not ever consider building themselves and only would address when they are at sufficient scale to whether they would either realize a product or efficiency benefit.

[00:53:03] JM: What are the parallels between psychology and computer science?

[00:53:08] JR: That was an easy one, which is that computer science doesn't occur in a vacuum. Most interesting problems are solved by a group of people who are collaborating. Most interesting problems involve how people interact within relate to either other people, the case of a social product, like Facebook, or to systems as product which provides a direct service to someone, like interacting with your bank account.

Psychology involves an understanding of people, how people respond both to other people and to stimuli around them and hopefully gives people insight into writing, let's say, better software. I don't think that they're that different. I would say that the connections, they're for a complementary. Even looking at it from another dimension, as I mentioned, most projects, interesting projects are not done by one person. How do people collaborate? How do people share ideas? How do they work towards a common goal? How do they achieve recognition as a group for that goal? I think that insights that you gain through a study of psychology have a lot of bearing on successful software engineering.

[00:54:30] JM: As I understand, the reason that you moved from starting companies originally was because you wanted to make an oath that you wouldn't be working hundred-hour work weeks. Then when you kind of felt the pole, the magnetism of Facebook, you took the VP of engineering role. Were you able to modulate your work-life balance and like assert a 40-hour workweek or did you find yourself just going back into the hundred-hour work weeks?

[00:55:01] JR: I'll answer the second question first. No. I failed miserably at that. I can look back at that and say, "Well, yeah there are things I should have done differently in terms of delegation, creating other roles earlier." But this first part of it, I'd say that it wasn't exactly that I didn't want to work intently. It was early in my career when I worked the hundred-hour weeks on a project that didn't mean much to me. That was really my observation.

I went to my manager and said, "I don't mind working this hard." It's just I want to do it on something that I really care about." That was when I made the decision to branch off and start a

company, start a number of companies, most of which lost in the dustbin of history. I mean, they were not successful. Occasionally, one was.

[00:55:58] JM: Do you have a count by the way?

[00:55:59] JR: How many things?

[00:56:00] JM: Yeah.

[00:56:00] JR: It's difficult because a lot of them, I killed them in their infancy, and fail fast is a very important lesson. You don't want to waste a lot of time on something that's not going to succeed. Persistence is it is an interesting attribute, but the truth is it also has a downside. So you need to know this isn't going anywhere. There's an opportunity cost for continuing it. Let me try something else. So there were countless projects that I started up on my own and they never saw the light of day nor did I ever involve somebody else in it, because I decided that there was no there there early on.

But I think that's true for most of us. We have more ideas than we have good ideas. The truth is, is that ideas really aren't what build a company. It's execution. You need to do something which is going to be important to some audience. One of the mistakes that I find that budding entrepreneurs often make is that they're trying to provide a little bit of value to a lot of people, and that's a really hard thing to do in starting a company. It's hard to convince somebody to use your product.

Sales is incredibly tough. Marketing, distribution, creating awareness of what you're doing, it's a real challenge. If you have the choice between creating compelling value for a very small group of people and modest value for a much larger group, always go for the compelling value, because that will be your beachhead, and you will be able to reach those people and they will look at what you're doing and say, "My God, that's for me. You are addressing my concerns."

You may be able to efficiently reach them through a marketing campaign that you can actually afford because there's an identifiable subset. The sales process isn't going to take you nine months, because the value proposition to that customer is so high and they'll overlook the fact

that you’re a young company. There’re all of these reasons why the compelling value, even if it's to a very restricted audience, is a much better forward than more modest value. Modest value, how do you get people's attention?

Remember this precious idea that you've been working on for the last three years and then you write someone a letter. It's your brainchild, but it's their junk mail. When people read their mailbox, they're executing an algorithm, and the algorithm typically is how can I delete this? How much of this do I need to read before I can make the decision to delete it? They're not looking for something that's going to improve their life, or improve their business, or improve their processes. What they're looking for is to clear out their inbox.

It might be your shining apple. It's their burden to get through this message. They'll look for something that can categorize it as to something that they know they don't need so they can hit the delete button. They’ll look for the keyword that lets them say, “I know what that is.” It's gone. If you have compelling value for some audience, that audience is much more likely to say, “Wow! I need to talk to these people.” Probably that's the number one mistake now we make as new entrepreneurs. I certainly did that enough times.

Obviously, if you can provide compelling value for everyone, well, then your Google, your Facebook, your LinkedIn, but that's a tall order. I think that your choice generally is that compelling value for a smaller audience versus modest value for a larger one, and my prejudice is to go with the smaller one.

Another aspect of being an entrepreneur is that you need to be able to tell a story. If you can't tell a story around your product that shows this is life without your product and this is life with your product and you can't make that compelling argument that the second scenario is better in some meaningful way, then you probably shouldn't be building the product.

It's essential that you’d be able to tell that story, and inability to tell that story is probably a reason not to do it. Maybe it's a bit of an oversimplification, but I like to say that simple always wins, that what your product should be simpler than what people do today on some dimension may be simpler to use perhaps is just simpler to understand, maybe simpler to buy.

I mean, it's just an easier way to be able to buy it or simpler for your sales organization. The whole sales process is easier, but there's going to be some dimension of simple and hopefully more than one that you can claim as yours. Without that, it's not to say that you don't have the business, but it's a good opportunity to take a step back and say, "Is there an easier way to do this? Is that what we should be doing or is a likely somebody else is going to do it?" Because simple tends to come out on top.

[01:01:03] JM: Describe your approach to building a diverse team.

[01:01:08] JR: That's a great question, because there's a lot of sort of built-in tendencies that we have as managers where that lead us away from diversity. One of those is we don't like to be wrong. We want to make safe hires. We want the largest percentage of the people we hire to hit the ground, quickly become effective and be a long-term success within the company.

If we're looking to really create diversity within the company, we're probably going to have to take a step beyond that and recognize that not every person we hire is going to hit the ground and be successful. They may need some additional support. Possibly, they're not going to succeed. But if you don't give them the opportunity, you're not helping anyone. How does this play into diversity?

Well, look at what an HR team dose. They get resumes and they filter those resumes, "This looks to be the background of the people who've been successful here. So let's forward them forward," or they're from a top 10 engineering school, let's forward that to the hiring manager. They're certainly not going to question my knowledge and ability as a recruiter if I'm forwarding them an engineer who went to MIT, but that's going to create a certain sameness among the team.

You need to sometimes go out there, take an additional step, take some risk. I would contend that organizations that are of averse to hiring failures probably have the greatest challenges in having diversity in education, cultural diversity, gender diversity. Just, "No. This person hasn't done it before." Well, there may be reasons they haven't done it before because maybe the last manager said, "You hadn't done it before." Take a chance. Give them an opportunity. Sometimes you'll be right. Sometimes you will be wrong, but you're always wrong if you just say, "I'm only

going to hire the person who was previously successful at this job or has a background that matches the people who were.

[01:03:21] JM: I thought the approach of Facebook, and I could be totally wrong about this, was you don't want the false positives. You'd rather have an excess of false negatives. Meaning like you'd rather have not hired a person who actually would've been a good fit than hire the person who's going to maybe end up being a bad fit, because the bad fits have an excess of downside to them.

[01:03:50] JR: I'm not really speaking from a position of saying this is how it's done at Facebook. I'm really addressing really across the industry why is it that we have had such a hard time breaking out of the mold. If you just look at the dimension of where people went to school, people coming out of a small list of universities get a disproportionate amount of interviews, of recruiting activity, because they're safe. The schools themselves are viewed as filters and they're viewed as sort of a sorting hat. Now, the hiring manager is able to say, "Well, this is a safe choice. It's a defensible choice. After all, I hire somebody from this background, and by all means they should be able to do this job."

If you want to get out of that, if you want to get someone who's coming out of a nontraditional background, then you need to be able to accept a bit more risk in your hiring, and it may be that this isn't the job that they end up eventually doing at the company. It might be they move into a different role. It may be they move into no role, but it's just something I wish more people were trying.

Not really making a comment about Facebook. Facebook did a really good job of recognizing that it's easy to hire people who are like yourself. I'm guilty of this as well. I happen to have a background in file systems and I have noted that in a disproportionate number of interviews I asked people to trace an I/O through the file system. Why do I do it? Because it's easy for me. I'm on solid footing to know whether they're bluffing or whether they're giving me a correct answer and in some other domains. I wouldn't be as certain.

What happens? Well, I've hired a probably a few more file system people than I should have. But I think we all are guilty of this. We all are comfortable working in hiring within the area we

know. I've heard some people use the term meritocracy to define the outcome. It's easier for us to hire people who are like ourselves, and unless we're cautious of that, unless were intentionally going outside the box, that will happen.

Another aspect of this is the cultural interview. This is something that many companies do. Of course, by cultural interview, they're referring to what somebody's work methodology. What's their approach towards work? What are the skills that they value? Can you evidence that through an interview and use that as part of the decision process?

I would suggest people take a step back and say, "Well, if you have a strong culture within your company, if you have a way of doing things, if there are shared values about how somebody works, then in all likelihood, someone joining your team is going to adapt largely to those values. Then to some degree, they may be bringing a different way of working in that might be foreign to folks on your team, but that might be additive to your environment as well.

I don't think it's too much of a stretch to say that this is a bit like some of the debate around immigration. You have a fear of other cultures. Why are these people coming in? They're different than us. When in fact, if you have a belief in your society, if you have a strong society, most people coming in are largely going to adapt to the values and traditions of that society and then they're going to add something as well and they're going to enrich the environment. I don't think a company is that different.

Just as we would like to see overall society be more welcoming of diverse backgrounds, I think that the cultural interview may serve the purpose that is not complementary to achieving our broad diversity goals within engineering organizations. I'm discouraging.

[01:07:42] JM: Last question. Do you have a vision for what the Facebook product will look like in 10 years?

[01:07:47] JR: No. If I did, it would be wrong. That would be outside of my wheelhouse.

[01:07:55] JM: Jeff, thanks for going on the show. It's been really great talking to you.

[01:07:57] JR: I've enjoyed this as well.

[END OF INTERVIEW]

[01:08:08] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[END]

