EPISODE 485

[INTRODUCTION]

[0:00:00.3] JM: "Management is about human beings. Its task is to make people capable of joint performance, to make their strengths effective and their weaknesses irrelevant." That quote is from Peter Drucker.

It's one of the many useful quotes collected in Ron Lichty's book, *Managing the Unmanageable.* It illustrates why we work in teams. When we collaborate with each other, we make each other's strengths effective and our weaknesses become irrelevant.

To collaborate effectively, we need leaders, we need management. Ron Lichty spent six years managing engineers at Apple and many more years in management and director roles elsewhere. In his book. Ron lays out the lessons he learned in 30 years of these engineering management roles. He also describes concrete strategies for how to manage engineers productively.

An engineer who becomes a manager needs to learn new skills. The hardest skills to master have nothing to do with technology. Prioritizing the right projects, allocating engineering resources, making architectural decisions, all of these skills are important. But the art of relationships, the art of diplomacy and language, it's harder to learn than any technical skill.

How do you motivate an engineer to do something that is boring? How do you have a difficult conversation with an engineer who needs to improve? When a conflict between engineers come up, do you confront the conflict head on, or do you wait for those engineers to resolve it among themselves? These questions do not have easy answers. The best way to learn how to react to these situations is to live through them unfortunately.

The second best way to learn is to read and listen to people who have seen so much of the management dynamic that they can distill it into anecdotes and aphorisms. In today's show, Ron shares several stories that change how I think about management.

Ron and I did not have time to discuss everything I wanted to, and I recommend checking out his podcast episode on Software Engineering Radio for more detail. That link is in the show notes for this episode.

I also recommend checking out his book, *Managing the Unmanageable,* which is also linked too in the show notes. Thanks for listening and thanks to Ron for coming on the show.

[SPONSOR MESSAGE]

[0:02:31.2] JM: Women 2.0 is a company with a vision of gender equality in the tech world. Women 2.0 is a community, a media company and a jobs platform that connects top female talent with engineering jobs around the world.

At the new Women 2.0 jobs platform, find vetted jobs for women engineers, data scientists and product managers. To find a job that is right for you, go to women2.com/sedaily. If you're an engineering company, you can connect with top female talent on Women 2.0.

Companies like Twitter, MongoDB and Craigslist use Women 2.0 to find new hires. Go to women2.com/sedailly to find out how to post your company's jobs to Women 2.0.

Thanks to Women 2.0 for being a new sponsor of Software Engineering Daily. Check it out at women2.com/sedaily.

Thanks for listening and let's get back to the show.

[INTERVIEW]

[0:03:39.7] JM: Ron Lichty is the author of *Managing the Unmanageable.* Ron, welcome to Software Engineering Daily.

[0:03:44.7] RL: Hey, thank you.

[0:03:46.1] JM: I have a bunch of questions for you about management, particular managing software engineers, which you have a ton of experience in. The questions that I want to start with are mostly related to experiences I have had personally being managed by engineers, engineering managers, and as well as some experience managing engineers myself.

The first question I have is something that is very close to my heart, because I would say it was the number one frustration/difficulty I had as an engineer. That is the question of boring work. If you're managing engineers and the engineers are doing work that is frankly boring, I mean this is a frequent thing that's going to come up. How do you keep engineers engaged when the work that they have to do is boring or otherwise uninteresting to them?

[0:04:43.6] RL: Well, I think the first think you need to think about is what's the contribution that work is going to make to the product, to the company, to the customers, to the team and to the world? One of the fundamental roles that managers that have is to connect the dots basically between the work that any individual programmer is doing and the contribution to all those places, to the team, to the product, to the company and to the world.

What it's really helpful to work in a company that's got a company mission that people can get behind. Fundamentally, Jeff I imagine this is true for you. It's certainly true for me. Most of us got into engineering, because we wanted to make a difference in the world. Knowing that and even boring work can be really meaningful, knowing that it's going to make a difference to somebody if we do it. I think that's the biggest piece for engineers, for developers, for testers, for those of us who are technical, it's just knowing we're making a difference in people's lives.

[0:06:00.5] JM: That was the argument that was levied at me when I was at Amazon. I spent a little bit of time at Amazon. That was the gist that I would get would be like, "Hey, I know you're working on the service that you're not supper passionate about," but think of the scale that yes, this is just a system that calculates taxes on items that come in. But it's calculating taxes on every item that comes into Amazon. Isn't that exciting?

I've heard other engineers who work at big companies and they say that this is a similar tactic that is used by their managers where their manager says, "Listen. I know this is not a super

exciting project where you're refactoring this UI component. It's going to take a month. But think of the impact. Think of the number of people that will be affected by you."

There are engineers who just hear that and they're like, "Yeah, I still don't care. I don't care if a million people are going to be seeing the new icon. It just doesn't – still doesn't motivate me." Is that a sign that such an engineer is just in the wrong place? Is that just going to be a non-starter? Or is there some sort of way to recover that situation?

[0:07:14.1] RL: Well, maybe. Or one of the other opportunities you hear from managers talking to developers is it may be boring to do that work, but it's maybe significant from a technical standpoint. One of the things that I've done with teams is to make sure that as a team, every engineer on the team is making a contribution to that team and making sure that every one of them is sharing what it is that they're doing with the rest of the team from an architecture standpoint.

It may be that there's a lot of piece work to be out where a lot of work seems insignificant, but that lot of work is it can be – that lot of work can be communicated to the rest of the team in ways that explain how the whole system works. You want to get engineers whiteboarding what their solutions are for the rest of their teams.

There is a technical contribution and there's a customer contribution and there is a product contribution. All of those things are wrapped up and key for managers is to understand what motivates every one of their engineers. It may be that you've got an engineer that is just not motivated to be in this team. That happens and we need to help them find somewhere else to work.

[0:08:50.6] JM: In a big organization, they often have these kinds of things where you can't switch teams unless you've been at a specific role for N months, some period of time before you can switch. Are you a fan of that policy, or does it just vary from company to company? Is that just a policy that you have to have when a company is at a certain scale? Otherwise, you just have too many moving pieces?

[0:09:17.3] RL: Good question. It does vary from place to place. It varies totally from place to place. What I see is that is a general – I think it's useful as a general rule, or a general principle. But it's also – it's where it's effective is when managers and HR can work together to customize it to the individuals that we're talking about. I think saying that there's some one universal rule that has to work for everybody is not terribly useful, but it's good as a guideline.

[0:09:58.7] JM: Part of the reason I start off with these kinds of questions is because your book is called *Managing the Unmanageable.* When I imagine an unmanageable person, I imagine these kinds of issues where you have an engineer who simply is in a role that they do not like. They're clearly talented. They made it through the interview process. They have some set of skills on their resume, or some set of side projects that is clear evidence that they are talented and they're capable of getting stuff, and yet they're in a role where it seems that they just simply are not productive.

These things happen so often. I think they definitely happen in software engineer, particular because engineers have so much latitude. I think in some case you could say engineers even get spoiled, because we have such a plentiful set of opportunities. We are definitely in a position to take an unmanageable attitude. Let's talk about conflict.

[0:10:57.5] RL: Yeah, before we move into conflict let me just – let me just suggest, Reid Hoffman who was the founder of LinkedIn, before he founded LinkedIn, Reid and I worked together at Fujitsu.

[0:11:10.0] JM: Wow.

[0:11:11.3] RL: He was one of our product managers and I was leading engineering on an online animated virtual world product. Reid wrote a book in –

[0:11:21.1] JM: Is this social net?

[0:11:22.6] RL: No, it was before social net. It was at Fujitsu. It was a product called Worlds Away, which was the – it was the third iteration of an online animated virtual world that was

originally called Habitat, that originally ran on the Commodore 64 ages ago. It was the original online animated virtual world.

We were doing the third generation of that and Reid came from Apple. When he left Fujitsu then started social net, then after that started LinkedIn. Reid has written a book on hiring and recruiting and getting people into jobs that I think it's got a really valuable concept in it, which is this notion of terms of duty.

His realization that he puts out there in his book, and I forgot what the name of the particular book is, but the concept he puts out there is this concept of terms of duty, which is we have this – we conjure up this idea that people are going to come work for us forever, and that we're going to employ them forever. In both of those cases, neither of those is true.

Rather than doing that, let's identify a term of duty. It might be two years or three years or five years, and look at what do you the employee, what do you the programmer want to get out of that term of duty? What do you want to learn? How do you want to grow? What skillset do you want to have at the end of the term of duty that you want to take on to your next role? Whether that's role is in our company or somewhere else. What do we want to get out of it from your having been here?

I think that's a really valuable, really important conversation to have in the hiring process, whether it's engineer, or as a manager, or an executive is coming into a company, what it is that they want to get out of having been at our company and what it is we want to get from there having been here? Having that very serious conversation about the contributions we're going to make to each other I think really sets up a different relationship and a different way of working.

[0:13:41.1] JM: Yeah, I can't remember if that – I read Reid Hoffman's – the books that he wrote. He has a co-author that I think he wrote both of his books with, or at least one of them. At least one of them is called *The Start-Up of You*, which is a cheesy title. It gets at this idea where it's basically you as a person can operate like a startup in the sense that you have a long-term vision for where you want to get to and it's an ambitious vision.

If you think about in terms of *The Start-Up of You,* maybe that vision is – can have different constraints than the type of vision you would have a startup of a business, because the startup of you, you could say like, "I want to scale into a person that can manage people. I want to scale into a person that has good understanding of business finances. I want to scale into a person that has great ability to deploy code in a high-scale organization."

I think the whole idea of that – is tour of duty, I think is – might have been the term that they used. Reid Hoffman talks about this idea, like if you're managing particularly an unmanageable person you say, "Look, I get it. You don't want to be here long-term. Let's be honest, you want to be here one to three years, and then you want to go on your way and do something else that's higher upside, higher impact, more closely aligned with your artistic interests, whatever it is.

For those one to three years, you're going to have to contribute to the company and we need to find a way that our interest align during that tour of duty. Alluding back to myself is just a personal example. That's something I probably should've done more effectively when I was at Amazon. Instead, I was a little more impatient about doing something that just impacted me positively.

If I would've taken a step back and said, "Okay, how can I evaluate this tour of duty in a way where I can grow really effectively and then have a good exit from Amazon after about a year and a half or something, one of these more respectable tours of duty, rather than the 8-month employee that looks terrible on your resume and so on."

I probably would've had a more effective experience there. I think from the point of view of the manager, just keeping in mind that your employees are probably not going to stay – they're not going to be lifers anymore. They're going to be do tours of duty. This is not a life or business anymore. Does that resonate with you?

[0:16:17.0] RL: Yeah. I don't think it's Start-Up of You. I think it was this next book.

[0:16:21.9] JM: Okay. What was that other one? What was this next book? Are you looking it up right now?

[0:16:28.1] RL: Yeah, I'm looking it up right now.

[0:16:30.2] JM: *The Alliance.* That's what it is. Okay. Reid Hoffman's book is –

[0:16:32.9] RL: Managing Talent in the Network Age is the subtitle. *The Alliance,* and *The Alliance* is a great title for that book, because what we're looking for in a tour of duty is what am I going to contribute to you? What are you going to contribute to me? What is the contribution this company is going to make to your career and what's the contribution you while you're here are going to make to this company?

It's essentially creating an alliance for that tour of duty period. Not saying that we're going to employ you forever, because that never – that almost never happens. It's really rare these days. Also not saying that you're going to stay here forever, because that also rarely happens these days. We're really looking for how do we form an alliance for some period of time.

At the end of that alliance, let's be honest with each other and look at is there an opportunity here for your next tour of duty, or do we need to help you find the next place? If you made the kind of contribution that our alliance asked you to make, I'm going to be overjoyed to try to find you the right place, whether it's in our company or another one.

[SPONSOR MESSAGE]

[0:17:53.7] JM: Indeed Prime flips the typical model of job search and makes it easy to apply to multiple jobs and get multiple offers. Indeed Prime simplifies your job search and helps you land that ideal software engineering position, from companies like Facebook or Uber or Dropbox.

Candidates get immediate exposure to top companies with just one simple application to Indeed Prime. The companies on Prime's exclusive platform message the candidates with salary and equity upfront. Indeed Prime is a 100% free for candidates. There are no strings attached.

Sign up now and help support software engineering daily by going to indeed.com/sedaily. That's indeed.com/sedaily if you're looking for a job and want a simpler job search experience.

You can also put money in your pocket by referring your friends and colleagues. Refer a software engineer to the platform and get $200 when they get contacted by a company and $2,000 when they accept a job through Prime.

You can learn more about this at indeed.com/prime/referral. That's indeed.com/prime/referral for the Indeed referral program. Thanks to Indeed for being a continued sponsor of Software Engineering Daily. If I ever leave the podcasting world and need to find a job, once again Indeed Prime will be my first stop.

[INTERVIEW CONTINUED]

[0:19:34.2] JM: Yeah, absolutely. Conflicts occur within a engineering organization. Whether those conflicts are between a manager and employee, or between two employees it is oftentimes the job of the manager to resolve a conflict. What are the circumstances that create conflict and how do you avoid conflict?

Then once a conflict occurs, how do you resolve it? Do you resolve it head on? Do you just avoid the conflict and try to find a way around it indirectly? Tell me your philosophy around conflicts.

[0:20:10.5] RL: Yeah. I don't think that avoiding conflict is the right way to think about it. I do think that we need to set up a culture of collaboration and a culture of cooperation and a culture in which we work together and which we find solutions together. That said, having – conflict arises out of two different programmers having entirely different approaches to – or entirely different thinking about, or entirely different architectures, or entirely imagining a solution entirely differently, as well as the petty stuff.

I don't like the clicking you make when you're thinking. One of the goals as for managers is to set up a culture of collaboration and of communication, because fundamentally software developments of team sport. We need to not have jerks in our teams fundamentally. One of the learnings that many of us have along the way is we put up with a jerk too long who we should have helped out of the organization or out of the team.

[0:21:27.6] JM: Often a brilliant jerk.

[0:21:29.0] RL: Often a brilliant jerk, which is why we put up with them. If they weren't brilliant, we probably would've helped them out of the team earlier. Almost to a one – managers I've talked to, my own experience has been managing jerks out earlier is always better.

[0:21:47.9] JM: Yeah. I agree with that. Do you have any stories about managing a conflict improperly, like when you made a mistake? Like things that just stick out in terms of conflict management, or either you were too soft, or too hard, or you move too quickly, or you move too slowly. What are the top mistakes in terms of anecdotes that come to mind?

[0:22:13.1] RL: Yeah. The top mistakes are always moving too slowly in dealing with conflict that's unhealthy for the organization or for the team. I really lucked out as a manager. I began my managing career at Apple. I was hired into Apple to create and manage a product management group, having been a programmer for seven years I did that for a year and a half and done immediately when Apple blew up my group, when Apple re-org'd, I immediately headed back into engineering again and started managing software and started being a programmer and then managing software developers, and then being a programmer and managing software developers.

I went back and forth several times before I embraced management as a career. As I said, I really lucked out in the sense that I didn't have that kind of conflict. Maybe I lucked out, maybe I created the culture that I wanted to create. But a couple of companies later, I walked into a culture that the person who had been the team lead was pretty toxic to the team. He had just been demoted before I arrived to just a member of the team.

He continued to be toxic and I let that go for – I let that go, because I didn't want to deal with it. I think most of us who are managers don't like dealing with conflict. Most of us who are people don't like dealing with conflict. It's a people issue more than a manager conflict, more than a manager issue.

It's probably endemic to programmers. Most of us who are programmers are – the people who do Myers Briggs have told us that most of us who are programmers all have the same

personality type, something like 70 plus percent of us are INTJs, and the I stands for Introverted.

We tend to not want to deal with interaction generally, stepping up into managing as a big change in thinking about interaction, and dealing with conflict is the worst end of that. I had the benefit in that very first conflict situation that I had a coach who had written a book on handling problem employees, and I truly had a problem employee at that point.

He coached me through handling that employee in what I thought was a very humane. In a way that was very – I don't want to say friendly, because it wasn't friendly, but it was a very human – it was a very human way of dealing with a problem employee by pointing out the impact that that person's behavior was having on the rest of the team, and devising a plan going forward that would – where that would result in change, that would result in change from that employee, result in change for that team.

[0:25:26.3] JM: Now when you sit down with somebody who you have a conflict with, how important is the preparation before that conversation? Because my experience is that it's super important and you almost have to map out in your head the decision tree where you say, "Okay, here is the thing I'm going to start with. Here is how I'm going to phrase it in a way that will not be super offensive, but will get my point across. Here are the different ways where he or she might respond, here are the responses I will give to that."

It sounds like over-engineering a human interaction, but it's one of those things where it's like planning – what is it? Plans or nothing, planning is everything. Where you do want to do this, because your impulse, the improvisation that you will have during conflict will almost always be incorrect.

[0:26:15.2] RL: Yeah, the planning. That very first one, so that coach who took me through it, the planning was days of planning for that meeting.

[0:26:27.3] JM: Yeah. For a single conversation.

[0:26:28.3] RL: For a single conversation. I should say that when I've done these and I've probably done a half dozen of them in my career where I've got somebody who's – who either has to change or has to leave. I want to have that conversation before I get to the HR plan. I want to have a conversation with HR and tell them I'm doing this.

There is this HR plan thing and this is really an intervention that I'm trying to create before I get to the HR plan, my coach was a guy named Marty Brownstein. Yeah, Marty had written a book called *Handling the Difficult Employee.* I've got a copy of Marty's book. I've never read my copy of Marty's book, even though I've recommend it to other people, because I had Marty.

Marty walked me through the planning for this. This is a meeting and this is how that first meeting went at that company; I set up a meeting with that employee for either an hour, or hour and a half, but I reserved the room for half a day. That meeting lasted for almost half a day. I spent probably two days planning that meeting.
That two days planning that meeting had to do with identifying the – first identifying the behaviors that I needed to be changed, but then being able to address those behaviors in terms of impacts, so the impact of those behaviors – some of the impacts of those behaviors included the fact that I had just spent two days planning a meeting with this employee to deal with these behaviors.

They were impact on the whole rest of the team, there was impacts on the team's morale, there were impacts on the quality of the code, there were impacts on how other people deliver the quality – other people in the team deliver the quality of the code, there were impacts on customers, there were impacts on – I literally had a list of impacts that the behavior resulted in.

The interesting thing that Marty taught me was that – or the interesting realization of what Marty taught me was that while an employee may argue with your perception of their behavior, it's pretty hard to argue with impacts. It's pretty hard to argue with the fact that I've just spent two days planning for this meeting. It's pretty hard to argue with the morale of the – with an observation about the morale of the team. It's pretty hard to argue about the quality of the code and the results of the behavior on the quality of the code.

By talking less about behaviors and more about impacts, now we're talking about what it is – how those impacts need to be changed and what it is – the past is represented what the future needs to represent.

This was a meeting in which it started out with my presenting impacts and then Marty's – I'm going to say in Marty's words, although this is so long ago that I don't know what his words were. But he basically said, "You're going to put out the impacts, and then your employee is going to let out whatever they need to let out. They are going to deny it. They're going to go into rationalizations, they will do any amount of talking they want and your job is to listen until they're done."

The interesting thing – you're saying, "Do you plan for all of the possible ways this conversation could go?" "No." I had no idea what he was going to say at that point. To Marty's point, what my job was, was to listen reflectively until he was done, to just hear him out, and to listen reflectively to really hear him not just let it come out of me, but to really hear him and then to come back to, "Okay, but we've got these impacts. We have to change. We need to come up with a plan that we're going to change them and that plan is going to mean that we have to have a plan going forward with check-ins that might be every day, or might be every three days, or might be every week, or might be every two weeks."

There's a huge commitment on the part of a manager to that turnaround. Marty's claim at the time, and this has been my experience with one exception that an employee who has that really honest conversing with whom you engage and that really honest conversation about the impacts have to change and coming up together with a plan that will make that change, that employee will either stay and make that change, or that employee will leave.

That gets out of the whole HR plan, it gets out of the whole firing thing, it gets out of all of that aftermath is over. You're getting it done in this meeting. You're getting it done in this meeting with the plan that has to go forward in the commitment you have to make the to the employee, to meet with them and to follow through.

[0:31:53.3] JM: You layout objective facts, so that this is not an emotional conversation, because you can keep referring back to objective facts. You're not going to get into some

debatable loop. You can say, “Hey, look. You left out a semicolon in this line of code and you left that a unit test in this highly sensitive matter. You did these eight things. Then here are anonymized comments that your teammates have given you. You’re unreliable. Another one of your teammate said you didn’t show up to five meetings in a row.” You just have these objective things. It’s not an emotional –

[0:32:30.4] RL: It’s not the words are unreliable. It’s the words, “When you didn’t show up to those five meetings, your teammates were unable to move forward.” It’s cast not in terms of their behavior, but cast in terms of their impact. That’s a really lot of work for a manager to – this is not on my brain thought. I’m not sure it’s how my brain thinks now, but it would definitely was not how my brain thought 20 years ago when I did that first one.

[0:33:00.4] JM: Yeah. I mean, this whole defusing of a problematic, unmanageable employee seems like one of the pillars of becoming a good manager, because if you do it correctly you’re going to be viewed as human, you’re going to recover difficult to recover situations, and you’re going to avoid giant HR issues.

If you do it incorrectly, then you’re going to be seared in the employee’s memory as that horrible manager that I had, because these are the types of conversations and interactions – I mean, I’m probably referring to something that happened to you 20 or 30 years ago and it sounds like it’s crystal clear in your head and you aren’t even the one being criticized. I’m sure for that employee, that whole half day, probably his entire day that day is seared into his memory. Probably seared into his memory is a vision of you as a more sympathetic, reasonable person than could have otherwise happened if you would’ve had less preparation.

[0:34:04.9] RL: I’m sure that’s true. Yes, I’m sure that’s true. I’m going to reflect back on an earlier one, the very first performance review I gave. The very first performance review, all of us walked into managing with role models. The role models most of us have for managers, in my experience and in talking to a lot of managers and a lot of employees, managers of programming tend to either be micromanagers, or they tend to throw their employees into the deep end and see whether they sink or swim.

They don't have a lot of – Human interactivity and human factor – human interactivity skills are not strengths in technical people generally. Partly, because we don't have a lot of role models. All of my role models were managers who had thrown me at the deep end to see if I could sink or swim. I thought that was how this manager thing worked until, and because I had the advantage of managing at Apple initially – Apply had Apple University. Apple had a set of course work internally for managers.

There was nothing in Apple's curricula that was specific to managing programmers, but there were a number of classes on managing in general and on becoming a better manager. The very first one I took was managers in the law. My manager told me I was going to take that and it was a half-day class, and I came back with my jaw having dropped for things I had not known about managing and the law that were really valuable.

I thought, "Okay, so I need to keep taking classes, because I clearly – there's clearly a lot to learn here." I think the second was situational leadership, in which you need to take into account the juniorness, or seniorness or a particular employee combined with their knowledge and experience and wisdom in the particular – for a programmer, the particular code they're working in, and take into account where that employee's at in terms of how much of a deep end you can throw them into.
The very first employee I had was a kid out of college, and I had thrown him into the deep end. It was now two months into his tenure when I took the situational leadership class. I came back from class, sat him down and said, "I think that I've really aired on your part. I'm sorry. I've throw you in the deep end. I'm actually now going to provide you considerably more support in what you're doing, and we're going to have – I'm here for you in a different way." He breathed this audible sigh of relief.

I think that we as – I've talked to a number of managers about this and all of us, almost to a one are apologetic to the first people who worked for us, because that transitioned from being an individual contributor to being a first-time manager. It is a hard one for us, but it's a hard one for those first employees. I'm starting to talk about that performance review.

When I gave my first performance review on this notion of I'm telling you what's wrong with you. I was really uncomfortable. I really hated doing it. I disliked it enormously. As I fought through

that process, as I fought through that interaction over the next few weeks I realized that the next performance review I wanted to give – I wanted to give it in an entirely different way, and I did give it. I totally changed how I gave performance reviews from the first one to the second one.

The second one, what I envisioned was sitting down on a park bench with my employee, and it's sort of the – it's Reid Hoffman's book *The Alliance* so resonated with me, was I imagined sitting down on a park bench with my employee, looking out at the horizon and saying, "Tell me about where you want to go. Let's envision out there on the horizon where you want your career to go. Let's now talk about what's standing in your way and what's going to get you to where you want to go."

That's a really different conversation for a performance review and one that's enormously helpful to the people that work for you, and one that's enormously helpful to you getting what you want from – the contribution you want from the employees who work for you.

[0:39:22.2] JM: Yeah. Because it doesn't assume that they're going to be lifers. It assumes that may be people who might leave the organization and they want to develop skills that are agnostic of the organization.

[0:39:34.3] RL: It also gives you the opportunity to say – not to say you're really screwing up, but to say, "Here is what's standing in the way of your reaching where it is you want to go."

[SPONSOR MESSAGE]

[0:39:54.0] JM: You've got a bacon delivery service and you need to notify your customers when their bacon has arrived at their doorstep. Twilio helps you make sure your customers get the bacon while it's hot.

Twilio's programmable API lets you build SMS or voice alert easily in the programming language of your choice, all in under five minutes with only a few lines of code. Now your customers get a text or a call the instant their bacon is ready.

If your customers want to see the bacon frying on a hot pan, Twilio has video APIs and SD case for the platforms that you know and love. Learn more at go.twilio.com/podcast and get an additional $10 when you sign up and upgrade your account. That's go.twilio.com/podcast. You will only pay for what you use and it costs less than a penny to send a text.

Get started at go.twilio.com/podcast, get your bacon delivery service cooking with Twilio's APIs for voice, SMS and video.

[INTERVIEW CONTINUED]

[0:41:13.7] JM: Yes. Yes, agreed. I want to shift to talking about something – something a little more day-to-day, not necessarily managing conflict or doing performance reviews or something, but just a management idea that I read – I've read in a couple difference places, but the place that sticks out is Peter Thiel saying about his management days at Paypal, and Paypal is often regarded as one of the best managed companies and one of the most uniquely managed companies. You can see that by the Paypal mafia and how successful the people that came out of Paypal have been.

One of the management tips that Peter Thiel says from that event and people have varying beliefs on whether they agree with Peter Thiel on certain things or not, but you can't dispute that he is an effective engineer – I'm sorry, effective manager.

What he says is that to succeed as a manager, you have to give every person exactly one thing to work on, and that there is massive downside when you assign multiple things to one person. How much do you agree with that? Should we be assigning multiple tasks to a single person? Should we assign a chain of tasks to them? What do you think about multi-tasking versus single-tasking?

[0:42:37.8] RL: Multi-tasking. Every single study that's been done shows that human beings do not multi-task, we context-switch. Context switching is enormously costly. It's especially enormously costly I think to programmers. It's one of the things that – what I've done for the last five years is I switched my career from being a fulltime manager to consulting as interim. VP of engineering interim, VP of products interim, CTO, interspersed with training teams in agile.

One of the things that I tried to do in training teams in agile is to get product managers, their product owners, their business side of the team to recognize that multi-tasking – the interrupting programmers has enormous cost.

One of the questions that I'll ask developers is if you leave work on Friday afternoon, you put a bookmark into the code that you're writing, so that you can on Monday morning pick it back up again. We're going to imagine that you don't work over the weekend and you don't think about your code over the weekend, and I know that's hard for some of you to imagine, but we're going to imagine that.

When you come in on Monday morning and, okay so you're going to look at your e-mail for a little while and figure out where things are. Now you're sitting down in front of your code, you're in your IDE and you're about ready to write the next line of code, how long does it take you between sitting down and actually writing that first line of code?

I'll get answers that range from, "Well, it would probably take me 10 minutes to, it will probably take me to half hour to, it will probably take me two hours." Then we'll walk through that a little bit and people will describe the process of resetting up all of the context to write the next line of code, because that next line of code has implications for other lines of code that are – well some of which are right in front of you and others of which are in other parts of the code and other files and other modules and other components and other objects.
You're forming this mental model that has almost certainly 10, or 12, or sometimes 20 or 30 different mental pieces in order to get this all set up. For those teams that say, "Well, it will take me 15 minutes to be ready to write my first line of code." I'll then say, well so what happens when you're interrupted on Monday every 14 minutes? How many lines of code do you write during the day? They'll look back at me and they'll say, "Zero."

I'll point out that that's true, except that there is this other way of doing it. I've never met a programmer who said, "I came to this company to write bugs." Jeff, have you ever met a programmer who said, "I came to this company to write bugs"?

[0:45:59.8] JM: No. But I have come to a company and been asked to fix bugs all the time.

[0:46:04.7] RL: Yeah. Where do bugs come from if programmers don't come here to write bugs? I think one of the places that comes from is it takes 15 minutes to set up all that context, at 14 minutes you see one of your colleagues, or you see the product manager, or you see your manager, or you see somebody coming your way and you think, "I've got enough. I've got enough context to write this line of code. If I don't write this line of code right now, I may never get any lines of code written today."

You write the line of code, but you are one minute short of having enough context to write that line of code correctly. That's the cost of multi – that's the cost of interruptions, but it's also the cost of multi-tasking.

[0:46:50.6] JM: Yeah. Is the implication here that as a manager, you should literally be thinking of each person – in an ideal world doing one thing at a time?

[0:47:01.0] RL: Yes. I've actually come across in a really interesting study that actually Mike Cone has it in one of his books. In his book succeeding with agile, he's got a chart, a study that shows number of projects that people are working on. This is not programmer specifically, but it's a study that shows the number of projects that people – that information workers are working on and their productivity.

It actually shows productivity going up with two projects, and then it pretty much falls precipitously with three and four and five and six projects if you do that. The interesting thing is – I've asked this question to lots and lots of product teams, "Why do you think that productivity rises with two projects?"

Pretty much to the one, the two big answers, one of them is, "Well, I get bored with something and I want to work on something else." The other is, "I get blacked on one. To be productive, I need to work on something else."

The interesting thing is that study was done in 1993, or 1991 maybe. It was the early 90s and it was – that was the time when intercompany e-mail was just arriving and it was a time before all

of the Slack and IRC and texting and mobile phones and all of the million channels that we've got of electronic communication had happened.

There are folks who have contended that that first project is already taken with e-mail. When I sit down on Monday morning to write code and programmers will say this all the time, "Well, the first thing I do is sit down and look at my e-mail. When I'm coding, if I get blacked the first thing I go back to is go back and look at my inbox, or go back and look at Slack, or go back and look at the electronic communications that have come my way," because darn it, those people who e-mail me expect that I actually read that stuff and respond to it.

What that means is that the first project is already taken, and the project that I'm working on is the second one and productivity falls precipitously after that first project. Because when I fall back on is e-mail and electronic communications.

[0:49:41.6] JM: Well, that makes sense. I guess, the 20% time idea are you a fan of that, or not a fan of that?

[0:49:49.4] RL: I'm a fan of organizations figuring out what works for both the Individuals on the team, the teams and the organization. I'm a fan of 20% time, I am a fan of Atlassian's FedEx days, I'm a fan of lots of – I'm a fan of organizations that figure out how to make work be fun and productive and interesting and useful and making a difference.

[0:50:27.1] JM: Right, right. Yeah, maybe you have the 20% time in one giant batch. Maybe it's one day per week. Maybe it's in the form of people just have a day off occasionally. I think the 20% time is commonly used as a way to allow an engineer to decompress. Many engineers decompress by doing more engineering, but it's more relaxing engineering, more free-thinking, more creative, less constrained to fixing bugs, but to some organizations it might be more just taking time off and decompressing completely. I think that's what the implication of the 20% time is.

[0:51:06.3] RL: Yeah. I think there are other alternatives. I think that life balance is really useful and important. If you look at what mental innovations is doing in Ann Arbor, if you look at that its founder Richard and he's written about in his book *Joy Inc*., if you look at what Pivotal Labs

does in training teams to use – actually both of those companies are using stream programming, they are – code is only checked in when both members of pairs are there. They only pair a program.

Programmers do not work when they're not pairing, and they tend to work eight or fewer hours a day, because peer programming is so intense. Then people go home. It's yet another way. Again, I'm really a fan of organizations figuring out ways that do not make programming – that do not attempt to have people running at 100-yard dash speeds for what is fundamentally a marathon.

[0:52:20.6] JM: You have in your book *Managing the Unmanageable* this section where you've got a large collection of quotes. Many of these quotes are useful aphorisms. I think one way of understanding management is through a series of aphorisms.

[0:52:37.8] RL: We refer to them as rules of thumb and nuggets of wisdom.

[0:52:41.5] JM: Yes, exactly. These are things like save early, save often. Managers must manage. Leading by example occurs whether you like it or not. I picked out some of my favorite ones.

[0:52:55.5] RL: I think the one in programming that both my co-author and I both collected very, very early Brook's law, adding manpower to a late project will make it later.

[0:53:09.0] JM: The one I want to ask you about is actually leading by example. If you're a manager and you're not writing code, how are you setting an example for your engineers?

[0:53:20.8] RL: I think you're setting an example by the culture you create. You're setting an example of being a collaborative person and creating a collaborative team. I think that agile teams have this notion of self-organizing teams. That notion is one in which everybody on the team needs to be a leader and needs to contribute their unique skills, insights, ideas, observations to the team in a way that in a lot of ways is very similar to what you see in acting within product groups, or in music with jazz groups, or maybe in sports with the Warriors who won the NBA championship last year.

I am not a sports guy, but when Ed Schwab – I had a management coach who thrust a book into my hands by a guy named Phil Jackson, who it turned out, I had no idea at the time because I'm really not a sports guy, but it turns out Phil Jackson was the – at the time the coach of the Chicago Bulls. He had inherited the only basketball player whose name I knew, which is Michael Jordan, who was setting all kinds of shooting records, at the point at which Phil Jackson arrived and the team was not winning.

Phil Jackson's job as coach was to create a winning team. Phil Jackson did not do that by going out and playing basketball. Phil Jackson did that by coaching the team. The Warriors when they won the championship last year, at the end of the third game in which the Warriors had been up by 20 points or something like that in the first quarter and the second quarter and then went down by almost that number of points, until the last minute in the half of the game, at which point they came back and won.

The ESPN reporter came up to Kevin Durant at the end of the game and said, "How is it? What is it that makes your team so selfless?" Kevin Durant said, "We practice it every day. Coach talks to us about it every day." That's the kind of leadership that managers need to provide to teams to get them to form that self-organizing team, a great programming team.

A great programming team is like a great basketball team, if that basketball – at least if that basketball team is the Warriors, or the Chicago Bulls under Phil Jackson, or the LA Lakers under Phil Jackson. Phil Jackson said, "When my team is firing –" In one of his books he said, "If my team is firing, they're like a jazz group." I thought, "Oh, they're like a programming team. That's it. Got it."

[0:56:32.9] JM: From the management perspective, what I take away from that is that there are traits that extend from managers to the team. For example, reliability, or communication, or cleanliness in conduct. If you're a manager, you follow-up with e-mail in a timely manner. If you don't, then your – your programmers, programmers are perceptive. They are going to see that you take three days to reply to something and they're going to say, "Well, I guess that's excusable."

It's certainly easier to reply to something in three days than to reply immediately, and they'll copy your behavior and your organization will immediately backslide. You got to do that stuff responsively and people are going to follow suite.

Another quote, actually this one comes from you specifically. Is that you cannot overcommunicate. I found this one interesting. The idea that it is impossible to overcommunicate in an engineering organization, because there are places in life where overcommunication can be pretty bad.

If you're negotiating over the price of a car, for example. You don't always want to communicate everything that is going on in your life. Why is engineering management a place where overcommunication is a good thing?

[0:57:59.1] RL: Yeah. I think that fundamentally – I'm going to go back to the example I was giving, which is jazz groups and improv groups and basketball teams. Software development is almost to an organization very similar. It's a team sport. You're calling out a number of attributes, Jeff of that we want to be and expect of our teams.

The two that when I get – when I ask developers to call out, to think about the best, the very best team they've ever been on and to call out characteristics of those teams, two of the characteristics that almost universally come out are trust and respect. Almost all of the attributes that you are listing actually fall under those two things. If you get trust and respect, you are going to show up for meetings on time, you are going to actually listen to each other, you are going to communicate with each other.

Fundamentally, software development is short of an application that you are writing and we can be the only person writing it, and we can create it. There are some mobile apps that are small enough that we can still – I think that was the amazing thing that happened when mobile applications came out on IOS and on Android was that suddenly, there were again applications small enough that a single programmer could write one of those applications.

Most applications that most of us work on are require more than one person and when we've got more than one person working on an application, it's fundamentally a team sport, and we really need to talk with each other.

The notion that we cannot overcommunicate, I think that in sales people are going to overcommunicate. I think in marketing, people are going to overcommunicate. I don't think in software development we can overcommunicate. We tend to be introverts, we tend to not talk with each other enough, we tend to not be terribly good at talking with each other, and we just need to learn that stuff and become really good at collaboration and communicating.

I'll go to that example you were giving. We've got a nugget of wisdom, and I just looked it up, the nugget of wisdom from Tim Swihart, who is an engineering director at Apple who said, "Have your annual reviews done on time. Nothing undermines your credibility as a manager more completely and impounding on your team all year to get their work done on time and then telling them that you don't have the reviews done, because you were busy. Whatever you're busy with likely wasn't managing your people, so you've just proven to them that they don't matter. Good luck motivating them next year."

[1:00:57.7] JM: Yes, indeed. I want to begin to wrap-up. We touched earlier on this anecdote of you doing this difficult conversation with an employee. What are some other lessons that were very hard to learn, that were like touching a hot stove, or you had to learn these lessons from negative experience? Any other anecdotes would be fantastic.

[1:01:22.0] RL: Well, I'll give you a couple. One of the rules of thumb about moving up the ladder, whether it's moving from programmer to manager, or moving from manager to director, or director to VP of engineering, and presumably from VP of engineering to a CEO role is that the things that make you successful at one level often get in your way at the next level.

This was not obvious to me anyway. I don't know. Maybe it's obvious to somebody else, but it was not obvious to me. It's really clear when you think about moving from being a programmer to being a manager, as a programmer the thing that makes you successful is the ability to shut out the world. It's the ability to climb into the microprocessor and listen to the gates open and

close. It's the ability to become one with your code, one with the program you're writing, one with the computer and shut out all of the distractions.

When you become a manager on the other hand, you really not only need to put a welcome mat out in front of your door, but to invite interruptions; interruptions from your team, all those people who work for you, interruptions from your peers who are going to be coming to you with concerns and well, with all kinds of things, and interruptions from your boss.

That is just a dramatic change and it's a change that if you don't see it coming, it is painful. The change happens again when you become a director and it happens again when you become a VP of engineering and there are levels in between in large organizations that that's true of as well.

For example, the realization when you become an executive that the CEO, or the CIO, or the SVP of engineering is operating in bullets. They expect to be communicated to in PowerPoints in really short-piffy communications. It doesn't have to do with their inability to deal with detail. It has to do with the fact that the detail is not relevant at that level, and that you've got to communicate it at a different level.

I think that growth through the latter is pretty universal. There is at least one really good book that's helpful in seeing what it is you need to learn at each level and what it is that you need to cast off at that level. I think that's one of those.

[1:04:23.6] JM: Okay. Well, Ron it's been great talking to you. Time has flown by and I enjoyed your interview on Software Engineering Radio. I enjoyed your book. I expect to consult it in the near future as I become more of a manager.

[1:04:39.7] RL: Okay.

[END OF INTERVIEW]

[1:04:42.3] JM: Ready to build your own stunning website? Go to wix.com and start for free. With Wix, you can choose from hundreds of beautiful designer-made templates. Simply drag and drop to customize anything and everything. Add your text, images, videos and more.

Wix makes it easy to get your stunning website looking exactly the way that you want. Plus your site is mobile-optimized, so you'll look amazing on any device. Whatever you need your website for, Wix has you covered.

Showcase your talents. Start that dev blog detailing your latest projects. Grow your network with Wix apps made to work seamlessly with your site, or simply explore and share new ideas. You decide.

Over 100 million people choose Wix to create their website. What are you waiting for? Make yours happen today. It's easy and free. When you're ready to upgrade, use the promo code SE Daily listener discount. Terms and conditions apply.

For more details, go to www.wix.com/wix/lp/sedaily. Create your stunning website today with wix.com. That's W-I-X.com

[END]