TFTC 465

**Marty:** [00:00:00] Yeah, it was fun. I got out of the cage, though. I've been up for 12 hours already. It's only 2. 30 in the afternoon. Good luck. Jocko

**Will:** Willick would be proud. Yeah, yeah. Did you run six miles? No. No? No, I was just up. You were just awake dealing with babies?

**Marty:** That's the first part. Just getting up. I'll eventually get to, uh, taking a picture of my watch at 4.

30.

**Will:** Yeah. Saying good. I can't do it. Anything before 5, can't do it.

**Marty:** I, I can't do anything before like 6. 30. Yeah. Which is

**Will:** bad, I'm trying to wake up earlier. Yeah, it's good for you, I think. Or it's bad for you. Then don't do it.

**Marty:** I'm the opposite, I don't get inspiration to write until like 10pm at night. I stay up.

**Will:** I'm trying to get inspiration to write. That's why we're here, sir. I know. Your writing's pretty good. This is my, this is my accountability, uh, way to make sure that I hit the publish buttons. I'm gonna talk about it [00:01:00] here, and then I'll have no choice but to publish. Yeah, TFTC,

**Marty:** we like being accountability buddies here.

Yeah, yeah, exactly. At the show, in the studio, within the commons. You missed Sean Baker last

**Will:** night. I did, well, it's someone's fault, but

**Marty:** He gave me a ten minute

**Will:** warning. Someone else got a notification. I didn't get one. Turns out.

**Marty:** Cause I, I saw that he had just left. Yeah, yeah. Parker. Yeah. I said, get your ass back

**Will:** here.

No, I love Sean. I mean, you remember when we first started talking, like we were in his orbit. At the very beginning. Yes. Because we were really hot on the keto side of things. And then we were like, Who are these guys that are cooler than us over here? They've taken it one step further. Who's

**Marty:** this jacked monster that's only eating steaks?

He is a monster. He crushed my hand last night. Did he? I went to shake his hand to say goodbye and it literally, audible knuckle crack on

**Will:** my end. You know he's a UT alum. I did not know that. Yeah, yeah. So he, that's how Bitstein got in touch with him when he was a, [00:02:00] Not a student, but like, through that connection.

Through the alumni network? Yeah, or something. I mean, maybe not official, the official channels, but I think that was the initial spark. Interesting, I didn't know that. Yeah.

**Marty:** He was in California,

**Will:** now he's up in the Pacific Northwest. You're making me think that I'm wrong on this, and I could be. I think I have a memory that he went to UT for like, grad school or something like that.

Well, Sean, if you're listening.

**Marty:** Yeah. Confirm or deny. Yeah, yeah. The allegations of you being a Longhorn.

**Will:** Yeah. Yeah. It's a proud thing to be right now. Ten and one. Who did they lose to? They lost to OU. Yeah, it's bad. Yeah. It's not that bad. Like, it's a rivalry game. If you see

**Marty:** OU coming out of the tunnel last weekend, it's pretty bad.

It's pretty embarrassing. You lost to that team.

**Will:** Yeah. Yeah. I mean, both my teams lost to OU. SMU lost to them too, so I'm, uh, I am embarrassed. I went to OU. I went to Norman. Not a great place. No, you didn't go to school there. You went to watch the game. I went to watch a game there this year. SMU played there because Texas fans never go there.

They always play in [00:03:00] Dallas. I went there this year for SMU. The stadium's like an architectural disaster. And I've never seen so many people limping,

**Marty:** limping, a lot of limps, gout, limps,

**Will:** something limps. Like there's just, I, it was like, it was notable. Like I took a mental note of the fact that so many people were limping.

This is just an odd. Thing I'm not making any judgments

**Marty:** Boomer sooner Sooner's

**Will:** limping Living in they were pretty happy walking out. Yeah, they were limping in. Yeah,

**Marty:** the football culture down here is fascinating

**Will:** Yeah, I mean, I mean high school's nuts. But yeah, I

**Marty:** need to get to a Wesley came So, I

**Will:** mean Pennsylvania's like that too, though.

I

**Marty:** mean my high

**Will:** school is pretty good. Yeah Small Catholic school, but Westlake. Yeah That's small Catholic school, too. My brother played with, uh, Drew Brees for a bit. Drew Brees is pretty good. He's, he's

**Marty:** okay. Pretty

**Will:** good. Is he in the Hall of Fame yet? Probably. [00:04:00] Yeah, Justin Tucker's also from there. Best kicker of all time.

He's a Hall of Famer. We have

**Marty:** two alumni on the Eagles. One of them scored a touchdown. The other one set up a touchdown. Yeah. Epic comeback.

**Will:** Oh, actually, Eagles starting quarterback, uh, uh, championship winning quarterback, also from Westlake. Jalen? Oh, Nick Foles, yeah. Nick Foles, yeah. They've had a lot of them.

**Marty:** We will build a statue of Nick Foles in Philadelphia one day. Calling a shot at the Philly Philly play. He's so

**Will:** good. Thank

**Marty:** you, Nick. Thank you, Westlake. Yeah. That was the sports ball section of

**Will:** the TFTC podcast. Is that typical? I mean, I don't think you talk sports too much here. Matt and I talked about the Eagles yesterday.

**Marty:** That's true, yeah. He's an Eagles fan. Big win. He was wearing an Eagles hat. It was hard not to touch on the subject. Yep. Which is a good segue into this. I mean,

**Will:** Matt Yeah. So I worked with Matt a lot on the stuff that I'm writing about now. Um, [00:05:00] which, uh, just to catch everyone up, um, I'm trying to get some writing out on Basically product development, building software, and what I've learned over the past, shoot, 18 years of doing it.

Um, mostly, mostly as a product manager, or something close to it, founder of a couple startups. Um, but, uh, I'm doing it not just to get it out, but, uh, because One of the things that helped me a lot were all those people that, you know, because you were into the product development world for a long time, it's like the stuff I was reading in from 2004 to 2008 or so was so foundational and getting me started and giving me an opinion on how things are supposed to be done.

But all those guys aren't doing it anymore. Um, and because so many of the Bitcoin companies that I interact with are started by, you know, usually younger people, I want to start getting some of that stuff out that, um, that, uh, [00:06:00] I feel like I got a very good education on this. Well,

**Marty:** I mean, I think anybody in the world who knows your background would agree because you worked very closely with Joel Spolsky.

Yeah, yeah. Considered one of the OGs of The internet age, if

**Will:** you will. Sure. I mean, he's, you know, in some cases, he's kind of like the original tech blogger. You know, before there were really, you know, blogs and people who were just setting up their own webpages, um, Jolan Software was like the only place you could go to read about someone's opinions on how to build software.

Uh, him and Coding Horror, you know, started Stack Overflow together. Yeah.

**Marty:** I mean, Stack Overflow is probably one of the most successful. Ahem. Engineer.

**Will:** How would you describe it? Stack Overflow was an amazing community. Uh, it's a, it's a forum, right? And, uh, the origin stories of it, you know, I wasn't there, uh, at the very beginning.

Uh, but the origin story is [00:07:00] basically that they were on a mission to, uh, kill Experts Exchange, which was the, the Q& A forum for programmers, but you had to pay to get answers and so Joel and Jeff Atwood had started it. I mean obviously there was a bigger mission than just killing one company, but this idea Very early on that this should be out there for free And that we would find ways to make that's when I was brought in was to find ways to make money later once it was Successful as a forum.

And, um, but what was interesting about the, those two guys coming together is that they had this huge install base of users because they were probably the two most prominent or widely followed, uh, bloggers. about software development. I think we were selling

**Marty:** them. Sure, they're not just bloggers. They also are software

**Will:** developers.

Sure, yeah, yeah, yeah. Well, and Joel, that's kind of the funny thing is that, like, I didn't know you could be something, um, other than a programmer [00:08:00] and build software. And Joel was a program manager at Microsoft, worked on Excel and all sorts of other, I think he's largely responsible for getting, um, like making Excel programmable and getting, uh, basic.

Inside of Excel. Great, great product. Um, and, uh, but it wasn't until I started reading Joel that I was like, Oh, I don't have to be a programmer to do this because I'd programmed my whole childhood. My dad was a programmer. My brother, my uncle, like everyone around me were programmers. And so rebelling in my family was not being a programmer.

And then finding out through Joel and through his writing that you could do something else And still build software was actually pretty exciting for me. But, um, yeah, I mean, I ended up joining stack overflow and, uh, Joel had been this prominent writer. He had a lot of opinions on how to build software, but, uh, Joel never, you know, as a stack overflow started getting, you know, bigger, we hit 20 people, 40 people, a hundred people, 200 people up to 300 at one point.

[00:09:00] Um. Most of his stuff outside of Microsoft had been very small, small teams. And, uh, there was a point around 2017 or so where we decided, um, that we had to just reinvent everything. Like, we didn't like We'd read all the books on Agile development and, uh, I'd, I'd done all the things that, you know, all the, all the sort of cliche things you do when you're going through like Scrum and all that.

And we decided that we were going to sort of build our own process that was really methodology, um, agnostic. Meaning, like, you could throw Kanban or Scrum or forms of Agile on top of it, even Waterfall. If you wanted to, uh, but that there seemed to be these core things that every time we were successful, we were doing very specific things.

And every time we were failing, it was because we didn't do one of those things that made us successful before. And we just weren't. Consistent and repeating our successes. [00:10:00] And so we spent a long time on it. Um, the other person that, that was, uh, a huge contributor and like the early forms of this was, uh, the VP of engineering at the time, uh, David Fullerton.

And so it was kind of our ass on the line. It's like, we got to get better. We got to ship, we got to ship things that work. Uh, customers need to be happy. We need to make more money. You know, there were a lot of problems that needed to be solved. Um, and so we just banged our head against the wall for about.

You know, three months, uh, coming up with this and then really deployed this process into, uh, that organization. And then over the years, you know, really it started, that was like the genesis point for it. Um, but, uh, when I moved on to Unchained, you know, I was starting to install this as we started growing as well.

We got up over a hundred people. Um, uh, as we brought on people like Matt McManus, who you had on here, uh, he was really helpful. helping me change it up a bit. Um, for instance, you know, at Stack Overflow, you [00:11:00] can imagine, you know, it's a forum full of programmers trying to get answers to their questions. Uh, now that's chat GPT, of course.

Um, but, uh, it's been rough. Um, but, we didn't really care that much about quality assurance, for instance. Not only was it, you know, the stakes, you know, not only were the stakes lower, but our customers, our users actually liked finding problems with our site. It was like a fun thing for them to go onto the meta sites and like tell us what was wrong.

And then as you can also imagine, when you get to a place like Unchained, Our customers don't like finding problems, right? The stakes are much, much higher when you're securing people's funds. And so we had to put in a lot more accountability and checks, not just into the quality of the software, but into the, um, sort of unintended consequences of dealing with, um, you know, disparate parts of the code base.

Um, [00:12:00] so, uh, it, it morphed a little bit, you know, through that. And then now at Zapwrite, I'm working with John who has a, you know, very different skill set than a lot of founders. He's a designer, um, which is a freaking godsend for all of you that have started companies before. You'll know that like, if one of the founders is not a designer, design usually comes pretty late in the process.

Um, and uh, it's been such a Privilege for me to work with him where, um, that part of the product, it, you know, before I was there was, was fantastic and is only getting better. And so anyway, getting back to the sort of origins of this, um, is that, uh, we came up with a system, we just called it our stack overflow product development process that became the unchained product development process.

There's a lot of people that contributed to it other than me, but, uh, I do feel the need to kind of get this out. And, um, really the way I'm trying to describe this to people is that [00:13:00] there are invariants for product development, things that you always have to do, no matter what, right? And there's only five of them, right?

And if you do them consistently, you'll ship better software. And especially for like, you know, the Bitcoin audience out there that are younger, um, it can be kind of overwhelming to see these, you know, huge books. I've never read a good product development book, ever. Well, sorry, that's not entirely true. Uh, Ryan Singer's Shape Up is phenomenal, uh, from 37 Signals.

Mm hmm. Um, they made Basecamp and Basecamp, you know, uh, few, uh Trello, did they? No, Trello was us. Uh, that was Fog Creek and Stack Overflow people. Yeah. Um, but yeah, I mean, uh, Most of these books are kind of nonsense, to be honest. Uh, most of them could be blog posts, that's why Joel and Software were so good.

And, um, but this, this idea of invariance, what it, what I'm trying to say is that, um, the most important thing in software development is that you finish the job of solving problems at higher [00:14:00] levels of abstraction before you move to lower levels of abstraction. An example of that would be strategy. Like setting the strategy for your product team or for your company and then lower down would be something like what I call discovery which is where you design and specify very detailed things of what you're going to build and then later on when you're building something your Actual lines of code.

So each one of those represents a descending level of abstraction you're setting strategy You're writing specifications and making designs and talking to customers and getting feedback on ideas. And then you better know what the fuck you're doing because you're writing code, right? Less abstract, less abstract, less abstract.

This concept. Sorry, this way of working is very familiar to programmers, right? They have to think about data models first. Abstract, bigger, have a lot of consequences, right? Then they might be defining classes, [00:15:00] and then they're writing lines of code, right? So they deal in these layers of abstraction sort of just naturally in their jobs.

But the other people involved in building software typically don't think this way. Uh, at least in my experience, most of the people I worked with that aren't programmers don't think this way. And when you're building software, I think it behooves you to think like a programmer.

**Marty:** And so. Let's back up and start with the highest level of abstraction, which is strategy.

What do you mean by strategy? Let's use Unchained as an example. Let's walk through one sprint that you guys did and go through these levels of

**Will:** abstraction. Yeah, so this is great. So, even a sprint, right, is a lower level of abstraction, right? It's strategy, strategy documents and, and the process of creating strategy is usually longer lived, right?

So it, it, it will live past multiple sprints, maybe 10, 20, you know, or something like that. I tend to think that like a good strategy will last about a year, right, in the software world. Maybe it's [00:16:00] updated a couple times, you learn a few things, but like, again, at that level of abstraction, you should have put in the work where your strategy can work for about a year.

What that really is, right, is it's not prescriptive. It's not a design. It's not a, uh, it's not a specification, but there's just enough requirements to narrow the field of possibilities of what you could build that you could do. Get something acceptable at the end. So a good example of done chained would be, we had a strategy, uh, that we wanted to build a new monetizable business line and that, that, and that we had made a decision on what that business line was going to be, that was going to be trading right now.

There's a lot of things you could do when you build. A trading desk. And what we decided to do was do it manually at first, talk to those customers, learn everything that they wanted, right? And then gradually build it into like a, uh, uh, automated process at the strategy [00:17:00] level, right? What we were actually saying is something like customers and unchanged have Bitcoin being stored in multi sig all that Bitcoin was bought somewhere.

Right? So this is its resting place. Why isn't it their starting place? Right? So we're going to allow people to buy Bitcoin directly into cold storage with keys that they control. It's a mouthful, but it is a strategy, right? Now, just saying that. There's still a number of ways to actually implement this.

And so there's a lot of creativity and autonomy that goes to the teams that are executing on that strategy. But you know, this is also one of the things I see missing in organizations is that someone needs to know what the fuck. Is going on and what, what, what, what you're supposed to do, right? Um, how many times have you walked into like an organization and you're talking to one of the programmers or to one of the PMs and you're like, I don't know what we're doing, right?

Or like, you know, I'm working on this. It doesn't seem to matter very much, right? That's a lack of strategic [00:18:00] thinking. And there are people that are supposed to set strategy and organizations. Usually it's a founder or an exec or, you know, a senior PM or a CTO or something like that. And, um, it seems that a lot of people, uh, You know, don't, don't follow through on that part of their job.

I think, I think it's also very, very hard, right? And, uh, at that, at that high level of abstraction, you might ask, like, Well, how are you supposed to know what to do? Like, is it just some guy? Like, is this top down, you know, command and control, you know, Style of building shouldn't we have more bottom up and it's like it is bottom up even though it's my responsibility at Unchained at the time to set product strategy, right?

I would be a fool if everyone's fingerprints on my team weren't all over that thing, right? The job is, uh, of the person setting strategy is to be in an information advantage to earn the right to, to author that strategy, right? [00:19:00] So if you're not in an information advantage, you have no business in setting the strategy for, for the company or for a product team or for, you know, whatever.

So how do you get to an information advantage? Well, you have to talk to customers. That's the number one thing. If you're not in regular contact with customers, there's no way you can possibly set a strategy. Two, you have to have a different type of relationship than just a managing relationship with people on your team, right?

You have to actually know what they know. Learn what they've learned about the product, about the usage, about the anecdotes and the data that are coming through. You have to, um, have a You know, domain expertise and what you're doing. I think domain expertise gets more and more overrated. The further down you get, uh, in the layers of abstraction, right?

So setting strategy, you got to know your industry, right? Writing code for it. It helps, but it's not as important, right? A good programmer can be a good programmer for Microsoft or Unchained Capital. [00:20:00] Yeah.

**Marty:** And so what it seems like you're getting at here needs to be a lot of communication. Uh, Between the engineering team amongst, uh, members of that team.

Internally and then externally to the rest of the company. Yeah. Make sure that you're striving towards that same strategy and everybody knows where you are, uh, in terms of fulfilling the goals

**Will:** of that strategy. Sure. Yeah. I mean, like, you know, the input is. Customers, people on your team, the engineers, everything you know, inside the company, compe, competitive knowledge, all that the output is.

Now, it ends up, I mean, being a document, right? You know, you create a document, it says, you know, unchained trading strategy, right? And you're writing out why this matters to the customer, what the business case is, how you're gonna. Measure whether you win or lose. Right? And then the really, the hardest part is, [00:21:00] is defining the requirements for it, right?

Because at a strategic level, like what you don't wanna do is end up writing a bunch of product specifications. Those aren't very memorable. You know, strategies should be short. Uh, everyone on the team needs to remember. What it says, right? They don't have to reference the doc over and over and over again, right?

It needs to be, it should be funny. It should be, you know, engaging. It should be good writing. Right. Um, but memorable is probably the best thing, uh, that a strategy document should be and how you get there might be. You know, brainstorm sessions and whiteboards and all this stuff. But at the end, you need this artifact, this like, this thing that everyone is going to be driving towards.

And it gets more important the more people you have in an organization. Like at ZapWrite right now, it's easy, right? There's four of us. You know, we can, we can get together, we can do it, we can write the doc, everyone remembers, we're all in the same room every single day, but at Unchained or at Stack Overflow or at Microsoft, right, you, you get [00:22:00] from dozens to hundreds to thousands of people that have to understand what this thing is, the more memorable you can make it, the better, right?

**Marty:** Yeah, that was one thing you referenced towards the end of the blog that you haven't pressed publish on yet, but at the end, you asked, uh, Basically your readers, like, can you go to your team and ask for three, three pieces of information.

**Will:** Yeah. The fact you just described. Yeah. And then two other pieces. So I say, at first, it's like, say you're an engineer.

Uh, uh, can you show me the specification that you're working off of? Right? It's a one write down what you're supposed to be doing. Um, and it can't be a to do list, right? People, that's one of the, the tragedies in how people have interpreted agile development over time is they take a Kanban board or, you know, Trello board or whatever, and they put a, you know, a title on it and a description of what they're talking about.

And that's their spec, right? That's not going to cut it for the most [00:23:00] part. Like you haven't done your job as a product person or as a engineering manager. If that's what, what. Someone's working off of right? Um, so you take those first of all, do you even have that document? Right? Then is there a strategy document that that's in service of, right?

And then just go ask someone who's tangentially related to the product development process, not the Holy Trinity of designers, PMs, and, and engineers, but go ask someone in marketing cause they have a role to play in this process. Right? Um, Towards, towards the end of delivering this project, this, this software, this bit of software to the public, ask them, what are the most important things being worked on?

Most places you can't get all three of those. Yeah.

**Marty:** And on the last piece there, it's always a funny ongoing battle is usually between sales and marketing versus engineering. Like sales and marketing guys are going out and pitching things to customers and then coming back to engineering and be like, yeah, the customer wants us to go [00:24:00] build it.

And so is. Yeah, setting up this, these invariances and internalizing them in the company culture. Does it prevent that type of problem?

**Will:** Yeah, I mean, it's a good process should prevent that type of problem, right? So I've always found this problem to be the most hilarious because it's such a cliche, but it's cliche because it's true, right?

It's like the salespeople are selling products that don't exist and they hate their engineering team because they're too slow and they don't know what they're doing at any time and they know everything that the customer wants and the engineering team doesn't know anything. And I always say like, that's the whole reason you have a product manager.

If that conflict didn't exist. Then there would be no reason for there to be product managers. I tell this to PMs all the time is like, if you don't have an engineer, you can't build software. So you have to have them. If you don't have salespeople, no one's going to buy it. Right. Uh, if you don't have designers, it's going to look like shit.

You're the only person who's expendable here. Your job is to make [00:25:00] every single one of those people better. And if you can't, then you're bad at your job, right? And the only way you can do that consistently is by having some process in place to make sure that those inevitable, you know, conflicts arise, because Again, it sounds, it sounds so silly when I'm saying it out loud, but like you're on the same team, right?

You know, um, if the defense is mad at the offense, like, um, it's not going to be a very good culture and clubhouse, right? You're going to have trouble hiring people. You're gonna have trouble, you know, your coaches are going to get fired and that the PM's job is basically to make all of those people better.

And they do it. By writing things down. It's that simple? Well, things that they write down is important. Right? Uh, I talked about the inputs and being at an information advantage. At a lower level of abstraction from strategy, say in discovery, right? Is where you're actually saying, Okay, we need to build a trading product.

Now I'm going to Write in very specific detail what that trading product is going to do. Because [00:26:00] in the strategy it didn't necessarily state that you had to be able to reach final settlement in 24 hours. You might decide that the only way we're going to win in the marketplace is if we can settle to Unchains Vault faster than you could settle to Unchains Vault if you bought through Coinbase.

So all of a sudden you have a new requirement in there. This has to happen and it has to happen within three hours. I don't know. Right? Why? Well, because you can buy it somewhere else and you can send that Bitcoin to an unchained vault and if that's faster, then people will just continue to do that.

Right? It'll be at its final resting place quicker, which is what people want. So That's where you're writing these very specific things about what the requirements are. That's not necessarily in the strategy document, right? And the strategy document should really, again, it should narrow the playing field of what the acceptable options are for, um, for the software to do, right?

You, you don't [00:27:00] want to say like, We're going to do trading and then someone goes out and builds a custodial wallet that says, okay, people can trade into this now and you're like, no, that's not, so it has to have some specificity. But, um, again, you have talented engineers, talented PMs, talented designers, um, working on this stuff.

You want to make sure that you're giving them just enough so that they can go be creative and make really important decisions on the implementation layer. Right? And I feel like I should back up real fast. I actually named these invariants, right? So the first invariant is not wallet, uh, aligns with the strategy sort of atomic unit of product development.

Um, I call it think and then do Right? But thinking first is very important. And I've been surprised how few people actually think things through before they get going. What are

**Marty:** some examples of people not

**Will:** thinking things through? I'll give you examples on my, uh, of me, right? Is, uh, let's see, uh, Stack Overflow.[00:28:00]

Uh, we'll start there. Um, I was really convinced. That, uh, we could launch something that stood next to Q and a, and it wasn't just me, it was obviously lots of people, but we as a team didn't think this through that we could launch, uh, something that we call documentation, right? That we had. A bunch of programmers writing questions and answers, just putting stuff on the internet.

Creating these nice little artifacts that, um, that people could use. I thought that we could get them to create another type of artifact, which is long form documentation. And, um, before we You know, did the things that I know you're supposed to do. We just started building it. We started writing code, right?

This was what the best documentation product would look like. We took it to market and it flopped like bad. And when we went back and looked at our product development process, we realized we didn't do the things you're supposed to do to be successful. Not that we wouldn't have spent time on this, right?

But. [00:29:00] Very early on because when it was failing, we start talking to people. Why aren't you doing this? You have a bunch of rep on stack overflow. Why aren't you also getting rep over here? What we found is that, um, people like being prompted, right? The types of people that, you know, the answers. Are very important, right?

And you know, there's more answers than there are questions, right? We need a lot of people competing over getting good answers because that's how you get good answers. Um, and, uh, those questions are very important because it's prompting the, the person with knowledge to say, hey, come give us some of your best stuff, right?

You know how to solve this. Documentation is very open ended. How does this API work? It's like, well, it's not a super motivating thing. It's a, it's a nameless person asking how it works, like, how do you even know to get started? Why, why do you think you're the person to do this, right? And we never ask those questions to begin with, like, at all.

Right? Um, so [00:30:00] You know, not thinking before you do in this case, uh, where in writing strategy is like, there is no primitive before it. Right. So the only thing you can do is think. And then you can start doing stuff. So in this

**Marty:** particular case, you guys skip

**Will:** discovery. Well, yeah. In this particular case, we kind of pretended we had a strategy without validating it.

Right. And that's an important thing also with each, with each one of these faces. And let me just say it real fast. Cause I feel like I need to set, set this up a little bit better. I think there are five sort of proto atomic units that all teams. Need to do in order to ship quality software consistently and that's have a strategy Run a process called discovery, which is basically the design specking You know user feedback loops a build phase where you're actually writing production code Right a quality assurance phase where you're making sure that that code does what you want it to do in a delivery phase Where you're taking that to market right five things right and each of [00:31:00] these each of these proto units I'm Giving you an invariant name to so that strategy phase thinking then do the discovery phase is don't design without a strategy right and Again, I want to repeat this because it is the most important part is that you have to finish the work at a previous Layer of abstraction at a higher level of abstraction before you start on the next one or else you run into these problems We hadn't finished in the documentation case the the the important work in the strategy phase before we were just Jumping into discovery and seeing this is the Best documentation thing we could build.

This is what it looks like. These are all the little tools. This is how it's going to integrate into our reputation system. We just got really, really excited over this thing, but we hadn't asked the most important questions. We hadn't solved the problems at the higher abstraction level.

**Marty:** So what would. The important questions have

**Will:** been to ask.

Ah, those are the good ones, right? So like, [00:32:00] it's really like, what is the output of a strategy doc? Like, what problems have you solved? What decisions have you made? You've usually done about four or five things just depending on the product, right? Uh, or what you're building. You've usually made a business case for it.

How's the business going to get better? Right? Are we going to make more money or less money? Is this a loss leader? Is this something that's supposed to make us profitable? Do we get more traffic from it? What's the business case? How does this help the business? Why do the customers want this? Right? And usually you have data and anecdotes leading to a thing that says, like, we know there's a subset of people that want to do this or a lot of people that want to do this or a new group of people that want to do this Don't answer questions, right?

Then you want to say, What are the minimal requirements here? Like, if it, in a good metric for a good strategy is that it's so bare bones on the requirements that it [00:33:00] does not work if one of the requirements is missing. Like, there's nothing, there's no fat on the bone at all. If you take away one thing, the whole thing falls apart.

That's the right level of abstraction there. And then finally, you have to have some sort of idea of how are we going to tell if we're winning or losing. Right? Those are metrics, KPIs, things like that. How do we know whether or not this software, this feature, this new product line, is doing what we thought it would do?

If you can say those things, right, and you have a compelling case, not only, you know, can you say them, but when you show them to your team, they get excited, they believe you, right? You've been persuasive, right? Um, then, you have a strategy. If you say all those things, and you're getting a lot of pushback, Did users really say this?

Did you talk to anyone? Um, have you looked at this data about how many words go into an answer versus how many words there are in document in a well documented API, like, um, [00:34:00] You know, these aren't the same things, right? The motivations are very different, right? If you get that type of pushback, then you second guess yourself.

Maybe you're not ready. Maybe you haven't solved the problems at that higher abstraction level that you haven't earned your way down to writing those specifications that you get really excited about. Yeah.

**Marty:** And so it was a documentation product comparing it to the pure Q and a, the answers were.

Typically shorter documentation is way longer than a quick answer.

**Will:** Way longer, it's unprompted. Um, and uh, that there were relatively, right, relatively few people that were interested in doing it. Also, another weird thing is that it turns out it's someone's job, some team's job, in really big places like Microsoft.

To write documentation, right? There are people whose entire job it is to do that when they release a new product, API, integration, whatever it is, [00:35:00] webhooks, and um, really, if we wanted to be successful we would have talked to them first because they write all the documentation in the world, right? You talk to them first and say, would you do this?

Right. What would they have said? I'm not doing this. No, I'm not doing this twice. Right. How much are you going to pay me? Yeah. Right. Um, you know, it was fun. Look, I'm, I'm, I'm picking on this one example. It's not unique. I mean like, but one other thing that I, I, I think I stole this, I'm going to look it up after this.

I think I stole this from Peter Thiel, but like the stakes are higher at the higher levels of abstraction too. Right. So you make a mistake in strategy. And you fail, no one gets better. It's a tragedy, every single time. You've just wasted everyone's time. Money, treasure, whatever it is, right? When you fail at lower levels of abstraction, the consequences get less and less, right?

You make a bad decision in discovery, right? You start writing code, you have to fall [00:36:00] back there. It's less expensive than having to fall back all the way to strategy. You write a bug in the code, right? And it gets out. Well, you can go back and fix it. Right. It's a lot easier to fix words on a page than it is a data model.

Right. Um, uh, and so, like, you know, catching those problems early on is way cheaper because the stakes and the stakes are higher. So it's like you have to put in the work there. Yeah.

**Marty:** So in terms of timeline, I mean, you referenced earlier strategy. And software development world typically on a year cadence.

And how do you break that year up in terms of like how much time you spend on discovery? How much time do you spend writing code? Is that predefined once you have your strategy set? Yeah, or

**Will:** that's what I kind of like about these like atomic units. And then like the, the, the invariants here is that, um, you could get through all five invariants.

In half an hour or an [00:37:00] hour for something small, or it could take you months to get through every single, uh, stage of this, right? Typically speaking, it really depends on what role you're playing. I have this little breakout. If you're a product manager. You're going to spend more time at the higher levels of abstraction and strategy and discovery than you are at the lower levels of delivery and building, right?

If you're a marketer, you're going to spend more time at the end of the, at the lower levels of abstraction than you are at the higher. Uh, If you're an SRE, you know, you know, you know, it differs, but typically speaking, it's like when I say that the strategy document needs to be a long lived document, it's, um, it's because if things are, are changing a lot at that level of abstraction, then you don't really know what to do.

Right. And so how long does it take you to know what to do? I don't, I don't know the answer to that because it is different in everything. Right. But for like trading, for example.

[00:38:00] It probably took us a couple weeks to convince ourselves that we really wanted to do this. And then we spent an inordinate amount of time in discovery. A long time in discovery because we didn't feel the need to immediately start writing code. We were like, well, We can hire a trader and they can just start and we can tell our customers, you call the trader, they'll execute trades and put it into your vault.

And we did that for a very long time. Now it's not necessary that you do that, but we were just really building up that validation. That was the case. And also it allowed us to change some things in our code base that would really allow us to build the automated solution. But. I'll go back and just say like, you know, the strategy process in general, if, if the person authoring the strategy document is good at their job, they've already, they're already setting up the right meetings with their teams that are talking to customers, doing all this, and you need to inflict a new passion upon your team, um, then yeah, usually a couple of weeks, right?[00:39:00]

Discovery is, is the, The part where like the designers and the PMs are playing the most role, uh, the biggest role. And really it just depends on like the complexity, right? Um, are you inventing something new or is there a comp somewhere out there in the marketplace that you really like? Or are there three that you want to use together?

An example of this would be when I first joined. One of the things we knew was that our messaging system, uh, where, uh, for our recruiting site, um, uh, where you hire programmers from, from like the stack overflow, uh, user base, um, the messaging system was really janky and bad. And, um, we used a little trick that Joel and I talked about a lot.

Uh, and David, uh, which is like core versus context is like, is messaging core to our business? Like, is it going to help us win in the marketplace or is it context? It's going to, uh, it's just something that needs to be there, but we don't have to like be the best [00:40:00] at it to win in the market. Right. And so something core you want to spend a lot of time on.

And something that's context, you want to spend as little time on as possible. And so if you use that framing, you can say, well, you know, if we're going to in the discovery phase, like if we're going to do this right, better do it fast. Cause we have more important things to do, right. That are going to help us win in the market, even though this is a necessary component for the, for the, um, product to work.

**Marty:** And then, I mean, you referenced scrum earlier, and so you have the spectrum of what I would define as like liberal approach to product management and conservative.

Waterfall on the

**Will:** conservative side. Yeah more command control on one side. Yes waterfall more bottom up more chaotic on the side, yeah,

**Marty:** and So on the agile side like that's where you get scrum masters weekly stand ups point. I'm trying to get at is throughout this whole [00:41:00] Process as you're going down the lower let levels of abstraction Mm hmm like one thing that people talk about a lot is like have his Few amount of meetings as possible.

Sure. Like when do you reach checkpoints at every level of traction when you feel the need to be like All right, when you get everybody on the same page here where we are.

**Will:** Yeah, you know what saves meetings? Emails? Writing well. Yeah, right and having artifacts that are traceable Findable, you know, that relate to the other things at the higher levels of abstraction.

So if you write a spec, right, you just put a little link into it and saying like, Oh, by the way, we're solving this part of the strategy, right? When those things are traceable, you need a lot fewer meetings, right? When we talk about like the waterfall to, to agile spectrum scrum, and I like scrum actually quite a bit.

Um, I don't necessarily agree with like the titles people get and things like that, but especially in very large organizations, scrum can be. Awesome. And you can use it on top of our process, [00:42:00] right? Um, the invariants are not a methodology, right? You can do waterfall through it. You can do scrum and agile through it.

You could do stage gate through it. You could do all sorts of things. Um, but on that continuum, it's like, yeah, if you want to have fewer meetings. It's, um, the people that are solving the problems at the higher level of abstraction, if they've written those things down very, very well, then maybe they have to talk to two people, and those two people have to talk to ten people, and those ten people have to talk to fifty people, in a typical organization, right?

But You, right, by writing that have made everyone's job much, much easier. Yeah. Good writing. Well, and this is, you know, the, you know, I call them the agilistas in, in, in my, uh, in my blog posts is like, you know, I've read the agile manifesto. It doesn't say not to write things down, right? You know, somehow between, you know, 2000 and 2010 or something, uh, the agilistas of the world decided that, you know, not only is writing things.

Down dumb, [00:43:00] but you know changing code is fun and easy right and we should just do everything in code and Again, I've read the Agile Manifesto It says nothing of the sort that gets perverted into this like we can never know what we want to do And so let's just start writing code and hopefully something good comes out of it.

It's in fucking insane, right? There's this guy Jeff Patton. He writes a III think about it Every week. Uh, he wrote a blog post in 2008. Uh, you can tell when I was like coming of age in this, uh, in this, uh, industry, but he wrote a, he wrote a blog post in 2008 called, um, uh, I don't want, I, I, I don't know what, what, I, sorry, let me start over.

It's called, I don't know what I want, but I know how to get it. And he's explaining the difference between iterating through a problem and and incrementing through a problem. And he has this beautiful, uh, diagram, or, uh, sort of imagery set up, where it's, uh, the difference of someone thinking about painting the Mona Lisa.

And the [00:44:00] first person, the only thought they have is a woman in a pastoral setting. That's the only idea they have. A woman in a pastoral setting, and they're supposed to make the Mona Lisa. So how do you get the Mona Lisa from that idea? Well, you iterate through it. You don't really know what you want, but you have this process to get it.

And so you start sketching with pencil, and you're like being very, very careful about things, and you're erasing stuff, and you're, you know, changing it up a bunch. That's iterating, right? And most people in agile teams are iterating through almost every problem. And then you have incrementing. And this person, when they're thinking about painting the Mona Lisa, they have an image.

of the Mona Lisa, right? So what is this person going to do? This person's going to paint the Mona Lisa, right? That's incrementing. They might paint this corner first, but they're gonna paint it very, very well, and they're gonna paint over here, and they're gonna do this, and yeah, they still might do some sketches, but they're going to have a much more direct way to painting the Mona Lisa.[00:45:00]

Instead of the person who had and so iterating and incrementing, right? They're both valid, right? You can imagine if you have the picture of the Mona Lisa in your head right now Say it takes 20 people to paint something because you're gonna paint it on a mural or something, right? You're gonna be very command and control and how you Get from point A to point B from idea to to solution if you have the idea of a woman in a pastoral setting then It doesn't do you much good right to just be command and control You're gonna have to sort of tell a lot of people have this image of a woman in pastoral setting when we try ten different Things of what you think that means, and then I'll pick the best one and then we'll go on that, you know, a little bit more in the software world.

Um, they're both valid approaches to building things right? What I found is that there's been this time where everyone thinks every problem should be solved through iteration, which is again, [00:46:00] sometimes valid, but what's the onus on the person who's supposed to know what? You're trying to do right. Is there any accountability?

Why do you have execs? Why do you have VPs of product and CTOs and things like that? If no one ever knows can picture the Mona Lisa in their head and then tell you what you want to do What they want to do, right? I think it's a abdication of responsibility Now it's not to say that that every organization needs to be top down Command and control waterfall style, right?

But the fact that like that's been completely rejected in many circles is insane Yeah, I

**Marty:** mean I remember when I was transitioning from finance to attempt to foray into product management. I did the, uh, UX, UI, front end development, design bootcamp, and Waterfall was completely like Voldemort. Like, not to be mentioned.

Cannot [00:47:00] do it. Yeah. It was, it was in 2014, so it was like the height of Agile scrum.

**Will:** Absolutely. Yeah. That, that, in that time, if it is a dirty word to be called, uh, waterfalls, uh, it's a shop that does waterfall. You don't want to work there. Yeah.

**Marty:** Yeah. Yeah. But that's what, uh, I really like the analogy you use where it's, it doesn't have to be either, or you can combine both and what you need to do as a product manager is be a salmon.

**Will:** Yeah, yeah, we can swim upstream, right? So if you have these like these, uh, these, um, you know, atomic units, the, the strategy delivery build QA and delivery, right? It might sound very water folly and you have to finish all the work up here before you can go here. It's like, okay, but, um, if we go down one layer of abstraction and we're trying to write a spec and it's not coming out very well, yeah.

Then we just go back to the strategy and we iterate through that loop. And then, once you have the spec and you're getting into the code base and you're saying, well, [00:48:00] actually, what we didn't know is that this is going to cause us to write this gnarly migration, you're right, that's going to be an absolute nightmare and keep us up all night, you know, on the day of launch.

Maybe we can do this a little bit better. Uh, you iterate through that problem, right? It's not a linear process of, of strategy down to shipping the software. It's this constant back and forth, right? And almost every. Almost every single project you'll ever do will acquire some form of fallback, right? The important thing is to know what mode are you in, right?

Are you iterating or are you incrementing? Because you'll work differently, uh, in, in both those stages. And two is, how do you know to fall back, right? How do you know to fall back and how quickly can you get those cycles to go? Right. Um, and then when you get really big, right, where you have 10 different product teams doing 10 different things, you want to pipeline these things, right?

You want to be, you know, some teams are going to be at the strategy stage of their [00:49:00] process and another team is going to be at the discovery phase and you want different people going back and forth, um, you know, pipelining through that process, but it doesn't have to be waterfall. could be waterfall. Um, but if you look at scrum, it's very similar, right?

There's, it's, it's, it's narrower because even they typically don't get the strategy phase, right? Because it's more about like just a sprint. What are we doing during the sprint? What is, what is the input into the sprint? You know, prioritization, blah, blah, blah. Okay. Uh, Right. Right. Specs for two days, and then we're gonna code for a week and a half, and then we're gonna test it.

Right. And, um, and they have all the different names for the people that do all this stuff. Um, but even they understand that there has to be some, like, structure to how do you graduate from one phase to another? How do you know you're ready to write code even in scrum, right? You're not writing code on day one.

of a sprint, uh, typically. Uh, you could, but it's not typical. Right? You have planning sessions, you have specking sessions, and then you have coding sessions. [00:50:00] Um, so yeah, it's, it's, you know, we can swim upstream. We're salmon here. We can go from delivery all the way back to, to strategy. Now that's gonna be more expensive, and you have to do a good job at each stage, but, uh, it doesn't necessitate that.

You're right that, A lot of times when I've presented this to people, their first question is like, is this just waterfall? And I was like, if you want it to be, it can be right. Um, but it doesn't have to be. No.

**Marty:** And so do you think, how many teams out there in the world do you think are applying your strategy right now?

What are examples of team shipping good software?

**Will:** So I've, I've interacted with a lot that, you know, everything I'm talking about here is like derivative of something else I've learned. Right. And so there are teams doing variations of this, you know, for all I know, stack overflow is still doing this. A lot of the people that PMs and things that I hired, they're still [00:51:00] there.

Um, unchained still doing this sap writes, you know, starting to do this, although we're four people, so it's a little bit different. Right. I know, uh, I've talked a lot to the cash app team in the past. Um, they have an excellent. Product organization. Really excellent product organization. It's small, it's tight knit, and they get shit done.

They do something very similar to this. Um, it's a little bit different in some cases, but like, but like, they check the boxes on the big five, right? They just get through it differently than, than I Um, Microsoft, uh, has certain parts of the organization that will do this. It's too big, you know, the product teams are too disparate and all over the world, but there are certain teams there, um, especially on the Azure team that do stuff like this.

Um, uh, I've talked to people at Facebook. They're not doing this. Not even close.

**Marty:** They spent, what, 40 billion on Meta? Yeah. Is that the worst? Strategy phase failure

**Will:** of all time. [00:52:00] It's, it's really confusing one. I'd like to know what inputs the people that set, you know, Mark Zuckerberg and his colleagues that set that strategy.

Like, because a lot of the strategy is you have to have some idea of what this is going to cost you both in treasure and in time, right? Did anyone start that process and say like, we're going to spend 40 billion on this. And everyone was like, yep, that's a good idea. Right. I don't think so. Right. And it could end up being a perfectly fine product, but if it costs 10 X what it was supposed to cost, is it a good product?

You know, it's like the product might be good. Customers might like it, but it wrecked the company. Right. And now you can't share, you know, photos on Instagram or something because you can't afford it anymore. I don't know. Right. Um, uh, similarly, I mean, like, you know, okay. It's, it's kind of like you, you can't avoid disasters all the time.

You have to have some tolerance for risks. I don't believe in like zero tolerance, right? [00:53:00] However, it's about consistency, right? And it's about being able to repeat a process over and over again and get reasonable results. So, you know, one of the, one of the ways that I would, you know, check the process on this is that like, um, you do I don't know, a monthly demo to teams, right?

To show them what's what's about to come out, right? And in that demo you tell people, okay, we had decided to do this three weeks ago, right? We said it was going to take three weeks. Today we're demoing the final version of it and tomorrow we're shipping, right? You have this accountability set into like some sort of ritual that you go through, right?

And, uh, part of that ritual has to be at the very beginning of saying like, How much is it going to cost us? How much time? How much is it going to cost us in time and in money? Right. And if you don't have a concept of that, it's really hard to hold yourself accountable.

**Marty:** Yeah. Yeah. And you touched on it. Mm hmm.

Particularly larger [00:54:00] organizations like creating those pipes across teams that are different levels of the abstraction layers of These strategies that they're executing on. But I think we should dive into this a bit more, particularly for the companies in the Bitcoin space that are reaching maturity and looking to scale.

Like, what is your advice to product teams that are reaching that, that point in their company life site cycle where they're going from 50 to 100 to maybe 500 employees? How do you manage that growth as

**Will:** a PM? Yeah, it's tough, you know, but, um, the. Uh, I'd say there's probably three things that I would advise.

One is even if you're not a command and control style of company, right? Hierarchy does matter and accountability matters, right? Um, and it really should flow upstream a failure in strategy. I say there's always a tragedy, but it's always the fault of the people at the top head should roll if your strategy fails at the top, right?

There has to be consequences for this, right? And it's not the programmer. [00:55:00] Less, you know, it was the CTO that came up with it, you know, that's going to pay the price for that. Just like when there's bugs in the code base or there's sloppy, uh, you know, specifications or bad design. You know who to blame.

Like, when strategy fails, you know who to blame. Do you know who to blame? Right? Who's setting the strategy at your company? Are the founders doing it? Are some early key hires doing it? Um, knowing who's responsible for what as you get bigger. Is a pretty big fucking deal, right? And a lot of times, uh, what early startups that are graduating into that 20 to 40 to 50 people is that You've had people that are kind of responsible for everything, right?

And now all of a sudden, there's a lot of people, and you can't be responsible for everything anymore, and you have 10, 000 customers, and, and, uh, so what are you responsible for, and what are they responsible for? And then you hold each other accountable. And the only way to do that is to have some sort of process that you go through in order to say, like, I'm gonna say I'm gonna do this, and then I'm gonna do it, and you'll know I did it because X happened, [00:56:00] right?

So you solve that through process and you solve that through, through, um, sort of ownership over certain things. The second thing I would say is that, um, uh, don't trick yourself into thinking that like writing things down is a waste of time. A lot of early startups are started by engineers, right? Or people, designers and engineers and product people, you know, together.

And they can say, well, it's going to take me four hours to write this down, and then, uh, it's going to take us two hours to code it, so why don't we just code it? And just do it, right? And then you run into problems because you haven't documented that, and you have a FAC, and you have all these things, but if you had written the spec, you could have just taken that out and put it in the FAC and Everything would have been fine, is that, uh, documenting these things is an important ritual to start even very, very early.

Like, uh, you know, John and, and Parker and I at Zapwrite, like, you know, we write things down, we wrote down a strategy. We, we write down specifications for payment links that came out, right? [00:57:00] And, uh, sometimes our programmers, who are fucking great, uh, can, I'll finish writing something, I'll send it, and I'll wake up in the morning, and they're like, done.

I'm like, well, that took me three days, you know, to like, to like do, but at the same time it was worth it. They wouldn't have been done that fast with that level of quality if I hadn't written it down the way I did. Right. Um, so writing things down is worth it and creating checklists. I love checklists, but you know, having a Kanban board with just a couple of things saying I need to build a login system.

It's not writing things down, right? That's a to do list, right? To do lists are not specifications. Um, and uh, that the earlier you exercise this muscle, the way less problems you're going to have building like a reasonable culture of shipping quality software later on in the business. And of course, you're starting this business because you think it's going to have 50, people, right?

Make sure you're a part of that or make sure that you're setting [00:58:00] yourself up to be a part of it as it succeeds Because when it does it can be really fun or it can be rather tragic when everyone hates what they're doing

**Marty:** And I can imagine a world which maybe this is the world in which we live in it doesn't need to be imagined But using zap writing as an example start with a four person team writing all these things down You guys are going to be wildly successful, get to a 50, 100 person team.

Absolutely. You can use those artifacts from the early days as sort of like an onboarding thing that you hand to Oh, yeah. new employees, where it's like, hey, here's how we've done things historically. You can see how we built every part

**Will:** of this product up to this point. Yeah, I can give you two examples of that, actually, which is, um, when we first did this at Stackover, before we formalized this, we didn't have much, right, in terms of, like, What's the stack overflow view of how we build software, right?

We had some, some blog posts from Joel from 10 years prior, right? But we didn't have much of an identity [00:59:00] ourselves. Once we did, and we had that documentation, you're right. That goes into onboarding docs, right? And so. You know, we did all this before I had hired a product team, right? And then we hired, you know, 20 people on the hire, on the product team.

And, um, onboarding those people, every single person would come back to me and say like, this was the best freaking onboarding I've ever had. Day one, you had, you had all the information I needed to learn how to do this job very, very well. Right? Same thing happened in Unchained, is that, you know, I'd already had that experience, and so when I joined Unchained, it was fairly small, you know, um, you know, 13, 14 people, uh, probably, total, um, you know, Four or five engineers, uh, no designers.

Um, and so, you know, I didn't come in like, you know, like a jerk and say like, Hey, things have changed. But like over time we, some of that process and a lot of it looked a lot similar to what we did at Unchained. Some of it changed quite a bit. [01:00:00] Then when I was onboarding a QA team, right. For instance, or, uh, You know, we hired 15, 20 engineers.

Um, it was very easy to say, like, this is what's expected of you, right? This is how we do things here, right? And a lot of product people are very scared to do that to engineers or to designers and say, like, I'm giving you, I'm putting handcuffs on you, right? We're a stodgy, you know, business. It's exact. I can guarantee you in 80, 90 percent of the cases, the engineers are going to be like, thank God, because they've all worked at some shitty company before where someone comes in and they're usually working in a panic because they don't know what to do, but the results aren't going well.

And they're saying, fix it. To an engineer, fix it, build, build a product that makes us money. You know, build a product that converts better, really is converts better. And a spec is that a strategy, you know, and then they go in and they're like, okay, what's going to convert better? Less clicks. And it turns out less clicks is worse than more clicks.

And they have no [01:01:00] fucking idea what's going on. And then they're sitting idle for three weeks because they don't know what to do, right? There's no strategy and. Like if you can show someone when they're coming on board the first day, we have a way of doing things here, right? You don't even have to do it my way, right?

I'm just doing it as a way that's been successful to me. But if you have an opinion on what is going to work and how we build software at this organization, you have an identity around it, then yeah, onboarding people. Breeze and you get complimented on it and makes you want to do it more, right? Like you put some effort into that and, uh, you know, someone that's coming up is going to, um, you know, they're going to make a commit on their first day at work.

**Marty:** It makes you more effective as a

**Will:** company trying to scale. Absolutely. Yeah. And that, and really that, that moment of like somewhere between 10 and 50, right? It's like. Most of those aren't gradual. Usually you're like 10, 12, 15 people. And then all of a sudden you're 50 people. Cause you raised 5 million, 10 million, right.

In the, in the VC software world, [01:02:00] at least, um, not in the bootstrap world. That would be more gradual, but like, you know, it's all of a sudden you go from 10 people to 50 people, and then all of a sudden you're at a hundred and you're like, holy shit, that it's been eight months, right? That's happened to me twice now.

Um, and I think it's a fairly common thing, especially in this easy money world where VCs are writing 10 million checks off of, you know, PowerPoint slides is that it's going to be the experience of a lot of people. Uh, That are, uh, in the Bitcoin space, starting companies today, you know, in 12 months, you're going to be surrounded by 50 people.

Yeah,

**Marty:** it's funny thinking of all this. I write every single day. Yeah, you mentioned writer. Thank you. I'm gonna shit on myself here. Because I like it's my goal Tftc to bootstrap like I just have this fascination with bootstrapping the business. Yeah bootstrap to date and I really want to Make this not a large company, but hopefully we get to like 10 to 20 employees at some [01:03:00] point in the next decade.

Um, the fact being I write every day and I've never read, written any of the strategy stuff that you're talking about. It's something we only have myself, Logan, Trevor, and one other person, but. Something I tried to do. I have to notion a few weeks ago to put some like documentation down. I just yeah, I just fucking

**Will:** glossed over it Yeah, it's a different type of writing, right?

Yes. No, I'm not good at the type of writing you're doing That's what I'm trying to get into is more public, you know writing for more public audience I'm very good at writing for you know, our company or you know, what is our mission? What is our strategy? You know specs, you know anything I can write all that very very well my advice to you Would be like, because I talk to you a lot, you know, we work 10 feet away from each other is, um, is I know you have a vision, you've told it to me many, many times for what we want TFTC to, uh, turn into is that like you just put yourself like, you know, one of those lock yourself in a room and I'm not coming out [01:04:00] because you could write your strategy in four hours.

I know. I just gotta do it. Yeah. It's tough. And then it's gonna help every single person that you work with. Like, immeasurably. It's gonna save you so many meetings. It's

**Marty:** probably the biggest problem between Logan and I is my, uh,

**Will:** portent. You feelin me,

**Marty:** Logan? Logan, what do you, what do you think about my communication skills?

When are we doing RHR this week? Uh, probably 1pm tomorrow. Or, Thursday.

**Will:** We'll see, what's tomorrow? Yeah. I don't even know what month it is, so. It might be December.

**Marty:** It's tough. It's a bit, eh. Um, it's tough though, like juggling all these balls, like trying to do the content, run the business side. Um, just thinking of like my business and cause I do want to have more tech aspects to what we're doing, but it is, you've convinced me and I just need to do it, but like lock myself in a room and do it and then that'll have profound knock on effects.

**Will:** Do you ever listen to Theo Vaughn? [01:05:00] Yeah. Okay. So I was, I was driving back from Dallas. Um, A couple days ago and um, uh, I was listening to his podcast. I can't, can't remember who the guest was, but, um, he's explaining this exact same problem. He's like, man, you know, all of a sudden, you know, I'm the boss and you know, I know what I want to do with this thing, but like no one else knows what they want to do and it's not their fault.

It's my fault. I don't know how to do this, you know, like I'm a standup comic, but now I have a business and I don't know how to run a business, you know? And what he's basically saying is like. I know what this needs to be. He's the guy who has the picture of the Mona Lisa in his head, right? And what they're doing in the podcast is they're pretending as if it's a woman in a pastoral setting because he hasn't been able to articulate that yet.

Right. And so. You know, look, you know, I, I say the strategy document is, is, is a document that's well formed that answers all these questions about the business case and everything. But oftentimes the things that come before [01:06:00] that are just as valuable, right? Is getting your team in a room and saying, locking each other in a room and saying like, okay, this whiteboard by the end of this is going to be full of stuff.

Describes our mission. Describes what we're building. Describes how it's going to make us money. Describes how it's going to get our, our viewership up and all this stuff. And that blackboard is going to look like a fucking mess at the end. But everyone in that room is going to understand it. And then it's just translating that into a written word that's understandable.

For anyone that wasn't in that room when you did it. Right? Um. I mentioned Ryan Singer before. At a lower level of abstraction, maybe this will help, is that he wrote something in 2004 called, he's still writing a lot, he wrote ShapeUp like two years ago, but I like his older stuff as well, called, uh, uh, something, something, something, design patterns.

It's about building design patterns, where he says like, okay, you want to write a spec, you know, you want to build a new dashboard for ZapWrite, you know, [01:07:00] um, Just start with, uh, first thing column bits. What's a bit a bit is, uh, a module that says how much Bitcoin you've earned. Okay. Bit number two, a module that says how many.

Um, how many dollars you've earned next thing, a module that says how many invoices you've sent out a module that says how many payment links you have live, you know, you're just listing bits, no priority, nothing, just literally a bit is a piece of UI that's on the screen. How many pieces of UI do you have on the screen?

Right. You just list them out in random order. And then it's like next stage group, the bits, right? Take these bits. Are they logically aligned in any way? Well, the, the total Bitcoin and the total dollars, those kind of sound similar things. We're going to put those together. Bit one, bit two are related to each other.

Bit four and bit seven are related to each other. So now you have the bits and you have them grouped and then you say, okay, [01:08:00] The groups, right? So A is the most important thing. B is necessary. C is nice to have, right? You group those up, and then it says, last one is, sketch the groups. And you just draw little boxes.

This is roughly where it goes. Okay, I can do anything. I can do any web app, single page web app or anything, that process, 20 minutes. Right. It's just, it's, it's, it's the way to get out over writer's block. I'll do them for strategy docs too. List the bits, you know, they're not pieces of UI, but they're points I'm trying to make, you know, and like just anything you have.

Some people use note cards, some people do, you know, whatever. But like this, like design pattern thing that Ryan Singer put together, uh, you know, I hope he hears this at some point, because I think he'll think it's hilarious that someone still uses that strategy from, you know, 2004 that he wrote. Almost 20 years later.

Yeah. But I. Every time I give this to like a PM or a designer when we're doing something It's like [01:09:00] let's just do a design pattern for 20 minutes together. Okay, let's list this out. Let's group them. Let's prioritize them Let's do it. We're 20 minutes in and someone's looking at me and they're like I got it from here It's like it's amazing how much like that level of organization which is like barely any at all will completely change someone's trajectory on a project right and You can use something like that.

You know, I use that mostly at the discovery phase of, um, of this particular product development process, but it can be used in other phases as well, like just to get you going. Yeah,

**Marty:** it's fascinating. Process is key.

**Will:** It all seems so simple sometimes. I feel really stupid saying it sometimes, but when I show it to people who have been doing this for a long time, they'll say like, I've never done it that way.

Like, oh my gosh, that actually is very helpful.

**Marty:** Yeah, I think it's the time aspect of actually writing. It's funny. I wanted to say this earlier, but now's a good time to bring it up too. Like the bent part of the reason it started because I was getting inundated with emails and [01:10:00] texts like what's going on with Bitcoin.

Yeah. The price was running and the other part was I was trying to get a product manager job at the time. Oh cool. Having never been in industry and the feedback I got when I kept getting rejected from jobs was Alright, if you've never built anything, like, prove that you can write about something. And so, like, the newsletter was multifaceted, where I could teach.

My friends and family were asking me about Bitcoin, about Bitcoin, and then prove that I could write about a particular product, which

**Will:** was Bitcoin. Sure. Yeah, yeah. Um, think about what, uh, I mean, like, I mean, that's really interesting. Think about, like, um, what Amir Ataki did. For the core protocol, right? With BIPs, right?

I think I'm pretty sure he invented the BIP process. It was the first time outside of like, you know, the mailing list or something that like people were actually writing down with specificity. Like, what are we, what am I trying? To do next, right now, it's necessarily different than what you're doing with TFTC or what I'm doing at zap.

Right. But like, even that alone [01:11:00] was a huge, huge change in Bitcoin's development. A little bit of structure when a huge, uh, took took the whole project much, much further. Yeah, create a process of Bitcoin improvement process. Yeah, it's great. Yeah, um, I think it's a fantastic thing, yeah. Yeah, Bips. What's your favorite Bip right now?

My favorite Bip right now? I don't know, what are they, Calm and Lightning again? I'm watching Bolt 12. It's changed a lot. I was, I was enticed. Yeah, it's just Bolt, right? Yeah. I don't know what it stands for, but, um, I don't know. I worked in multisig for a long time. I'm not a lightning expert yet. I'm catching up.

Um, but, uh, no, uh, Bolt 12, I'm keeping tabs on, you know, uh, right now is it's, My favorite thing, uh, to, to read about, uh, also, uh, I like the, uh, I've been [01:12:00] implementing, uh, the LN address spec and the LN URL verify part of that, um, as well. And then sharing it with people that we're integrating with right now.

That's how we did the GitAlbi, um, integration. Turns out there are a lot of wallets that have done, that have followed very well the LN address spec, but have not done the LN URL verify part, which is that right we need so that when a transaction is. Sent to one of those, we get a message back saying it's, it's been received, like it's been paid.

Um, so, uh, I've been reading these a lot and like, yeah, the fact that there are specifications for this, like you can't have an open protocol without specific. I know that there's no, there's no spec for Bitcoin or anything like that, but there are specs for Bitcoin, right? Like there's not a spec for Bitcoin, but there are specs about.

Aspects of Bitcoin. Yeah, like standards. Yeah, standards and, and BIPs and BOLTs and, you know, all of these. It's, it's very important to get, um, you know, conformity around this. Like, you need interoperability [01:13:00] between the protocol and industry and industry and Other industry, right? And, uh, having those is super helpful and we're at a point in Lightning in particular right now where there's some very attractive, um, there's a lot of very attractive reasons to be, uh, very native to Lightning instead of doing proprietary APIs on chain.

There's still some problems with that, but like in the lightning world, it's like most of these integrations, you don't have to do anything proprietary at all to be interoperable with other wallets, other companies and things like that. It's very, it's very nice. Yes. It's very encouraging. Oh yeah, it's huge.

On chain will always have some problems with that. Just because of the different address structures? It's because of the final settlement and the way it works. And like, you know, for instance, like the LN URL verify stuff, like, um, we could just, you know, Build an API between us and every single lightning wallet there is.

And saying like, well, you send us invoices and then, and then [01:14:00] we'll show them and then they get paid and then they get deposited in your wallet. And then you have to tell us, you know, back in like, we could do that on a proprietary basis. And a few years ago we may have had to, right. However, now we can use the lnaddress, uh, spec and the lnurlverify spec and we can do the same thing on a non proprietary level that can be repeated with anyone that we work with, right?

And implementing it just so happens in this case to be very, very, I won't say trivial, but simple in comparison to other things. While on Bitcoin, there's just, there's other drawbacks. Uh, you can do it in a native way, but, uh, You run into problems like, uh, there's no communication layer on the main chain to say, like, Hey, start at, uh, this index, uh, for addresses, and, you know, you know, Oh, they're getting, they're getting other invoices paid over here, right?

You need to know about that so you [01:15:00] don't try to use one of those addresses again. Like, there's like little things like that that you just can't really solve for. And so proprietary APIs make more sense. I've experienced this with

**Marty:** btcpayserver when you have a number. Yes, of on chain invoices that don't get paid in your wallet.

You get big gaps. You get big gaps. You got a gap forward and

**Will:** Sparrow Yeah, and then you look on your Trezor on your own. You don't see it. You're like what the fuck and you're freaking out But it's there. Yeah. Yeah, you just got

**Marty:** to go gap forward a thousand addresses

**Will:** to find it Yeah, yeah, exactly. But like, so yeah, there's going to be some like drawbacks there.

Um, uh, so proprietary API APIs do make a little bit more sense, but I mean, there are native ways to do it. Just give me some X bubs and I can increment addresses. Right. Um, it's just when lots of people are trying, you know. If that vault or that wallet is not specific to this use case, then we can't know what's going on outside of it.

And you run into these types of problems. Lightning, you can completely avoid that. Like, it's not an issue at all. [01:16:00]

**Marty:** You feel like you're on the cutting edge again, coming from Stack Overflow into Bitcoin?

**Will:** It was an interesting process for me because, uh, very much like in very early in my career, I was like, well, if I'm not writing software, like I'll go be an eye banker, you know, cause there's nothing for me in the software development world until I found that product management was a thing.

Um, and that I was good at it. Um, Similarly for Bitcoin, it was like, well, if I'm not contributing to the core protocol, there's nothing for me to do here. I'm not that interested in building, you know, an exchange, you know, a casino. And so like, there's not much for me, which is a really weird way for me to think, uh, it was, uh, Parker Lewis, actually, as, as we were talking probably in 2016, 2017, that started to convince me that he was like, well, you know, if Bitcoin's money.

What's the product of money? I was like, I don't know. He said financial services, of course, right? [01:17:00] Bitcoin itself can replace, you know, the centralized, uh, Fed and central banks, right? But there's still going to be a need for products out there that solve the problems that financial services do now. And they're going to look a lot different, right?

And even if they look familiar and similar, they're going to be based on something that's very, very different, right? Uh, um. And so it was that realization that made me think like, Oh, I do have something to offer here. And it took me a little bit of time cause I loved my job at stack overflow, like loved it.

I loved working with Joel and David and everyone there. Um, so I wasn't like super motivated to leave, but in 2019, uh, uh, it would, it became. I'll say untenable there. Uh, Joel had moved on, uh, just a few months prior. I wasn't enjoying it as much without him around. Um, and, uh, you know, you know, typical Silicon Valley, New York, startup type stuff.

I didn't fit in too well anymore. And, uh, so. [01:18:00] Bitcoin needs people that like this and like, you know, there's so many talented engineers there. Um, they're starting to get a lot of talented product people. Uh, although it was, it wasn't the case even four years ago when I started on it. Well, let's end up

**Marty:** with a pitch to product people out there.

It might be on the edge. Like what? What kind of mark can they leave on the world if they were to come over and build

**Will:** on Bitcoin? Yeah. Actually, yeah, I like this. This is a good way to end it. Is, um, is, if you're talented in the software development world, if you're working at Microsoft and you did something great, if you're working at Google or if you're working at Meta or if you're working at any of these places, right?

But you think Bitcoin matters. Right. Is there are companies that need your expertise. We're building software here. And while Bitcoin core protocol is necessarily very different than the types of software you would build at Microsoft, right? The companies that are, you know, the industry building on top of that, the unchains of the world, the zap writes of the world, the TFT sees of the world, all of us river, all the great Bitcoin companies.[01:19:00]

It's the same thing. You're, you're just building software, right? And so we need talented people that have learned from the best to be coming over. And it's like, if you don't think that you have something to offer, you're just wrong. Um, and I, I, I, I was in that sort of mindset and I was just wrong. Um, is that all these companies need you, um, that they're well financed, um, that they have, you know, Actually, I find typically that, um, Bitcoin has, uh, Bitcoin companies that I've engaged with have, uh, incredibly well suited founders for the stage company that they're at.

Meaning that, um, these aren't just people trying to build a new marketplace, a new SaaS thing that they're going to flip for 200 million dollars in a couple years. It's that they have passion and they have vision for what they want to do. And if you're joining something, nothing, nothing, can be more important than knowing what the vision is.

Um, if the [01:20:00] vision is to go work in San Francisco and to flip your company in two years to Oracle, right? Fine. I mean, there's nothing wrong with that, but it doesn't really get me excited, and I bet you these people listening right now that are working those types of jobs aren't very excited. It's very exciting to work on something where the vision is something you're actually passionate about.

So I would encourage you to, uh, know, one, that your skill set is needed, and two, that, uh, the conditions in which you'll be coming into are a hell of a lot more fun. Jacked up. Yeah,

**Marty:** I wasn't expecting this conversation when you approached me a couple weeks ago, but I'm very happy we

**Will:** had it. Yeah, I, I'm, I'm, I'm excited to actually get this published.

I'm gonna publish it, um, on a new ghost site that I'm setting up, and then also on the Zap write, uh, blog. 'cause you know, zap write is going to work. Um, it is one of those things where I'm sure I'm gonna blink and we're gonna be surrounded by 50 people. Um, and, um, and, uh, yeah, so we'll be cross-posting it on, uh, zap Write and yeah, I've paid [01:21:00] two

**Marty:** Zite invoices this month, so.

**Will:** Yeah, it is working. Yeah. It's really funny. Actually. I found, um, I found a lot of weird, people I wasn't expecting, dentists, lawyers, you know, using Zapwrite for invoices. And then on the payment link side, it's a little bit more standard, it's just, you know, people that sell things and want Bitcoin. And we know a lot of the people that do that.

But, um, yeah, no, it's been really encouraging to see, you know, John's initial, you know, version of it with, with some of the new stuff on here. It's like, it's, it's been, it's been a fun last few months, uh, since BitBlockBoom when we launched payment links. Yeah.

**Marty:** It's been great. I mean, see the team that you guys have gathered.

It's fun watching you, Parker and John do the damn thing. Are we going to win?

**Will:** We're going to win. Yeah. You have people like John starting companies like this. Like he has a vision for what he wants, right? Passionate about it. Like we don't struggle with a lot of the things that typical software teams struggle with.

So yeah, we're going to win. [01:22:00] Team of killers. We're going

**Marty:** to win. We're going to win. You heard it here first. It's always a pleasure, sir. Thanks Marty. Peace of love.