Will Corcoran - **00:00**

Documentation, security testing and hardening. And there's a number of things that.

Will Corcoran - **00:07**

We can start with, but I thought.

Will Corcoran - **00:11**

Maybe we would just start with kind.

Will Corcoran - **00:13**

Of a temperature check.

Will Corcoran - **00:16**

And this is just meant to be.

Will Corcoran - **00:17**

Like, a fun conversation starter to jump in. And if you were to take, like.

Will Corcoran - **00:26**

The blockchain code base and the SBTC code base, and you had these sort of ends of the spectrum dumpster fire.

Will Corcoran - **00:34**

And Sistine Chapel, something that is, I.

Will Corcoran - **00:39**

Guess, known for its simplicity and elegance, some beauty. Where in there do people feel that.

Will Corcoran - **00:50**

Our code bases line up colony?

Jesse Wiley - **00:56**

Because you're right in my Kona vision.

Brice Dobry - **01:00**

All right, well, that's at least I.

Brice Dobry - **01:03**

Get to say the easy things before anybody else does.

Brice Dobry - **01:08**

So for the blockchain code base, I'd.

Brice Dobry - **01:11**

Say it's a mixed bag because I.

Brice Dobry - **01:13**

Think that the code itself is pretty solid.

Brice Dobry - **01:20**

I'm happy with the code, but the documentation is severely lacking, including comments in the code.

Brice Dobry - **01:28**

So it makes it difficult to learn.

Brice Dobry - **01:32**

As I've been doing stuff, I really have found that I have to dig through the code to find things and trace through, like, oh, there's a thread doing this, and this is how these two are communicating. So that's the place that's really lacking.

Brice Dobry - **01:47**

So as far as documentation, I'd give.

Brice Dobry - **01:50**

It like, maybe a two, but for the code itself, I would give it like an eight or something like that. The SBTC code base, I don't have a good feel on, so I don't.

Brice Dobry - **02:04**

Have an opinion on that one.

Will Corcoran - **02:05**

Yeah.

Will Corcoran - **02:09**

I would assume that the SBTC.

Will Corcoran - **02:11**

Code base, it hasn't collected all of.

Will Corcoran - **02:16**

The iterations that the stacks blockchain code.

Will Corcoran - **02:21**

Base has over the years, and being.

Will Corcoran - **02:25**

A bit fresher and a newer vintage, if you will, it's probably a bit.

Will Corcoran - **02:32**

Leaner, I would guess. I don't know that for sure.

Will Corcoran - **02:37**

And to your point about documentation, there's a section that we're going to have next, I think, talking about documentation, and Kenny Rogers on the call really asked to be here. He's always invited, but made a point, hey, I really want to be involved and get a better sense of what's going on in these calls. Documentation is part of one of the things that he does with developer advocacy. So I think trying to make an easy onboarding experience for anyone that's coming in and integrating themselves into the code.

Will Corcoran - **03:18**

Base is like a life and death.

Will Corcoran - **03:22**

Matter for us because of our ability to scale, our ability to onboard new people in a timely manner, an ability to get code contributions from folks that as a way to just try out, essentially make themselves interested in joining core engineering or one of the orgs is a really, I think, successful way of finding talented, interested and passionate engineers.

Sarala B - **03:59**

I would actually split the documentation part into two parts. One, yes, the documentation that we're talking about that is external to the code ray that's supposed to be helping onboarding new users new developers, that's an aid. But the documentation that Bryce is also highlighting is in code documentation where even for established engineers like Aaron Bryce, everyone else on this call just figuring out why something was handled or designed the way it was. That kind of self contained documentation within code is also something that we could really work on. Leave it better than found.

Brice Dobry - **04:46**

It kind of yeah.

Will Corcoran - **04:50**

What's like an actionable step that people could take with that? Are there examples, like within our own code base or another code base that you've seen that say this is a good pattern to start by following?

Sarala B - **05:09**

We've seen a lot of new improvements around this thanks to efforts by Aaron and Bryce, again around the time from 2.1 to 2.4. Most of the fixes that went in, if you look at the code now, there's clear documentation of why those decisions are made. At least a one liner, inline code comments. I think any bug fix, any functionality that we add that you find that it's important for others to know, just documenting that. I would love to actually Bryce and Aaron on that, maybe because I'm outside looking in and I'm not as familiar with the Rust code base itself. I don't know if that resonates with the rest of the engineering team.

Aaron Blankstein - **06:00**

Aaron yeah, I would say it definitely all resonates.

Kenny Rogers - **06:05**

I think that.

Aaron Blankstein - **06:10**

Just speaking on comments, I think that sort of the in code comments that we're missing the most.

Kenny Rogers - **06:19**

Are not so much the what is this code doing?

Brice Dobry - **06:25**

Because I think that for the most.

Aaron Blankstein - **06:27**

Part you can read the code in the code base and see what the code is supposed to be doing, but the more the whys. And I think that there are some places in our code where we have Rust docs that sort of explain the interfaces that different units are trying to provide, but especially when we try to tie this into more advanced testing regimes and things like that. I think that's something that would be very helpful is that if every code unit in the code base describes sort of exactly what properties it was trying to achieve, what its invariance are like, what it was expecting from inputs and what its expected outputs are supposed to.

Brice Dobry - **07:12**

Be, I think that could be.

Aaron Blankstein - **07:17**

Thought of as very bureaucratic. But I think that the code base is at this point sort of mature enough, big enough and widely deployed enough that we should be engaged in that. You're asking for a numeric value. On how I would rate the stacked blockchain. I would go with a three. I would say that we're halfway between a pure dumpster fire and a dumpster fire cysteine chapel hybrid.

Will Corcoran - **07:51**

Is that due to bloat or is that due to just being opaque?

Aaron Blankstein - **08:00**

I think that there's like any number of areas I could point to. I think that one of the biggest things is that the bulk of development in the stacks blockchain code base was done by, say, fewer than five people working, generally speaking, under time constraints with the feeling that it's better to get something that works than nothing at all. And that's reflected sort of all up and down different points in the stack.

Kenny Rogers - **08:43**

You could look at just our build system.

Aaron Blankstein - **08:46**

Like our cargo tunnel layout is a pure dumpster fire. You could pick any random Rust code base from GitHub and we would probably be in the bottom 5% of Sensical Cargo Tommel files. And you could do that all over the place in our code base. The directory structure. Exactly. The fact that we have this thing called the neon node that is like 90% of functionality of the code but has a name that few, I think, could explain.

Jesse Wiley - **09:32**

There we are.

Will Corcoran - **09:35**

Jacinta, if you have your hand up.

Jacinta Ferrant - **09:38**

Yeah, this is a bit of a as an outsider looking in at the code base, it's very hard to decipher and just similar to what Aaron said with the why. I'm sure anyone who's written it will have the context as to why it's laid out, the way it's laid out, but from not having any of that context, it's quite impossible to follow. And one of the things I was wondering, I know that this is a tech debt thing, but something I found incredibly helpful in general, and maybe it's not possible with the stacks blockchain, but maybe for any new additions, even just enabling Rust, clippies auto deny anything that's public that doesn't have docs associated with it. It just forces you when you code to ensure that it's documented within the code. So if you run rust docs, it's there. So things like that are like low hanging fruit, but in terms of fixing past code base, it's a lot harder.

Jacinta Ferrant - **10:42**

But just was a comment I wanted to make because that was something that helped me a lot at my old work when we had the issue with documentation. Is anything new clippy deny if no documentation there. Highly recommend. Yeah, that was it.

Jesse Wiley - **10:57**

Great.

Will Corcoran - **10:59**

Martin, I'd love to hear your perspective. You're also coming anew with a first set of eyes, working on another code base, but obviously having to interact a lot with the blockchain code base.

Mårten - **11:15**

Yeah, because I've been contributing to the stacks node a bit in relatively isolated tasks. You can still find your way around it, but yeah, I give it a 2.5 because it sounds like people are self aware of it. And I think it's quite evident when you're working with it that it will sort of build together. Another low hanging fruit, for example, would be just breaking it up, breaking up the whole code base into smaller crates just for compile times. You can argue about the structure of things, but yeah, it's not very idiomatic. You often see some over engineered pieces, some things that can be very simplified, even very simple things that like yeah, I mean, when I joined we had a lot of compiler warning, so the compiler would tell you can do this is better than that. And there's still a lot of clippy errors where an automated system can say that this is wrong, this is something we can improve.

Mårten - **12:19**

And I think I've noted a lot of code quality improvements, created a lot of tickets from that. We've actually gotten some outside contributors, some that I would love to find time to do, but we've also been in this sort of constant fire drill with Alpha and all the other stuff, so I haven't gone around to do that. So it would be lovely if we spend some time tending to the quality of that. What do you say? I've also been participating in a crime of writing s***** code because I think the SPDC Alpha code is like from a pure rust perspective, it's a bit better, but I have more respect for the stacks node because the code is still intentional, still a working system. SPDC Alpha is in a pretty bad state because it's been this sort of constant, let's just get it out, no matter how s***** it is.

Mårten - **13:11**

And we've been having that sort of attitude for a very long time and the system is much more complex than it sort of initially set out to be. So that code is also quite s***** also because we didn't sort of plan properly. So I think there's a lot of learnings we can take from developing this new repo and doing SPTC code and still not producing it in a decent fashion. That's a bit like top of mind for me about code quality. There's a lot we can do to improve that.

sayak - **13:47**

Yeah. One sort offhand thought is that I think that in the stacks code base, there is a very poor separation of libraries and binaries. Everything is just sort of like compiling in one big pile. In general, I don't think that's a very good way of doing things, especially in terms of concurrent programming and parallel stuff. You should have library functions that are not dealing with any of the Async stuff and then you have binaries that are sort of like spinning up these tasks and processes that are doing that. So I think that's like a low hanging fruit. I would like just having libraries that we can look at as functionality and not having to deal with all the complexity of running the actual thing.

Kenny Rogers - **14:41**

Yeah.

Will Corcoran - **14:44**

So I want to real quick ask a naive question. When people say test coverage, are you.

Will Corcoran - **14:52**

Saying that we have ten blocks of.

Will Corcoran - **14:57**

Code, like ten small chunks of code, and we have tests in between all ten of those, so we have 100% test coverage? Or is it, hey, we're running this group of tests on this one chunk of code and it's picking up ten out of ten possible issues. So it has 100% test coverage.

Will Corcoran - **15:23**

Sergey.

Will Corcoran - **15:24**

I see you got your hand up. I'm confident you can answer that question too.

Brice Dobry - **15:33**

Yeah, I will try.

Sergey Shandar - **15:34**

So one of the concepts that actually we tried in SBTC, at least for one of the first projects, there was a relay server that I actually created using complete I O isolation. There is no I O in a library at all, but I O it's only on a top level. So this actually helps significantly to test, and actually to test coverage as well. So this actually resonates with what is saying that actually if you can do more describe a state instead of doing async programming or multi threading everything else as soon as you do like a state machine you could push it on top level like all the Async programming and multi threading programming on the top level in the application. So that actually can significantly help in a quality of code like don't program, for example, thread pool in a library. That doesn't make any sense, for example.

Sergey Shandar - **16:45**

And another thing about if you compare like, SBTC repository so one of the things that actually was good, which actually I wish that Stacks blockchain do the same, we actually done properly workspaces in Rust because currently what was the problem with stacks blockchain repo? That it's actually mixing the top level cargo to ML. It's actually mixing workspaces and project settings, and it's usually like an ID, for example. They cannot understand what's going on. So it's actually kind of like, I think we need to fix it. It's not an SBTC repo, and I mean in stock text blockchain, but in SBTC repo, we actually done it properly and it actually works very well.

Brice Dobry - **17:48**

That's my two cent.

Jesse Wiley - **17:52**

That's it. Thanks, Sergio.

Will Corcoran - **17:55**

Martin, I see you got your hand up.

Mårten - **17:58**

Yeah, I just wanted to build on what Sergey says about like I mean, one reoccurring theme to many of these discussions has been sort of separating I O from libraries. You have cycles. I mentioned that here. And we're also talking about test coverage. And I think to me, the important property is testability, right? If you write your code in a way that it's easy to test the code, then you will also get operatability, which is very important because now when we've been working with Stacks Alpha, it's been really painful to operate the stacks node. And that's something we've been sort of seeing throughout here, making stuff easier, more isolated and testable. And to me, I'm going to be a bit controversial. I'm going to say I don't care about 100% test coverage. If something is testable, depending on how you're going to release it and what it is, you don't always have to test it as long as you write it in a testable way, because that's going to make things so much easier to maintain and debug.

Mårten - **19:00**

And you can write tests, you can steal the test driven development. I'm not opposed to that. But testability is the holograph. That's like the word I want to nag about like testable code, testable code. So I just want to nag a bit about that.

Jesse Wiley - **19:14**

Thank you for listening. Thank you.

Jose - **19:19**

I want to add something like a refresher. The weakest measure of coverage is the function coverage percentage. Then were doing in clarity, were doing a line coverage, but because when you cover the function, maybe you are missing some lines, but if you're covering the lines, maybe you are missing some branches. That like logical decisions inside the function. So now we added the branch coverage. So that's the strongest way of covering measuring the coverage is the combination of line percentage coverage and branch percentage coverage.

Jesse Wiley - **20:06**

Thank you.

Will Corcoran - **20:14**

This is not meant to beat up on the blockchain code base at all. A lot of these questions came from blockchain contributors and one of them I.

Will Corcoran - **20:26**

Think would be worth diving into a.

Will Corcoran - **20:29**

Bit more is just very simply like what is the most frustrating aspect of.

Will Corcoran - **20:34**

Interacting with the blockchain at this point in time?

Jesse Wiley - **20:42**

Jacinta.

Jacinta Ferrant - **20:45**

I don't know if this is the absolute most frustrating thing. I've had very limited experience getting the node up and running. I actually found it difficult to do. But the thing that has been a bigger pain point is for example, the next branch I had to interact with and were trying to debug an issue and that branch is so bloated that I couldn't isolate changes to try and find the bug. So what I ended up doing was I went back and I manually pulled out and rebased all of the SPTC changes and created a new branch. And this was a nightmare that took me about two days to do. But if we just made it so that we never merged basically, and we only ever did rebases, that sort of approach would have been easy. Like I could have easily just pulled out the changes I wanted and then iterated through and the history would have made sense.

Jacinta Ferrant - **21:44**

And just having better PR brand chain practices in GitHub would make debugging the stacks node for me so much easier. And that's the only time I've ever had to really interact with it is when I'm trying to debug something that's already exit like already in there. So from my perspective, which has been pretty limited, improving that would be like huge. Yeah, that's it for me.

Mårten - **22:10**

Well, can I add to that? That's an extremely good point. When I first encountered the stacks node code, I do a lot of git blame. I go down in version history, I check out old versions of the code, I look at diffs, I see how it's been developing and I'm going to run to bit here. Like the branching strategy in the stacks blockchain. Again, extreme respect for the actual product and the system, but that almost horrible. It was very hard to make sense of any changes. A lot of changes, small changes were barely explained or linked to any issues or any discussions or context. So it's very hard to understand why things were the way they were. And a lot of the time you would just hit a big merge commit between next and develop and master one way or back. And the merge commits, they are just so hard to read like a linear history.

Mårten - **23:04**

The gold standard is a linear history where every change links to appear and hopefully an issue with some sort of explanation of things. So yeah, team rebase 100%. This is just ranting because I don't know if there's a good way to fix it. We could change the branching structure.

Will Corcoran - **23:26**

Sergio.

Sergey Shandar - **23:29**

I would like to comment on rebase. Honestly, my personal opinion, I hate rebase, but in the same time I completely agree with linear history and what we do in SBTC stacks, SBTC repo, we do allow only squash merge, pull request. So before you actually merge in, you could do whatever you like. I actually keep usually a very complicated history, but as soon as you merge the final history in main branch, it's always like kind of like a pull request only. So that's actually like I don't know what team is thinking. Is it actually helpful for this strategy? Because I don't want to rebase.

Jesse Wiley - **24:19**

When.

Sergey Shandar - **24:20**

I develop because I still would like to keep history. Maybe I can have several branches, but as soon as I actually merge and make a pull request, then we do only and we allow only score merges, we don't allow anything else. And another thing is actually we're working in Main Branch. So the idea is like there's a two strategies usually and this actually is the difference between currently stacks BTC and stacks blockchain that actually in stacks BTC. We do. Work in main branch and we only can like if we need to release, we can tug or make a separate branch for specific version if we need to like an old version. But the main development, it's in main branch and this is actually like in my opinion, this is what currently open source community tends to work on. So if you just go there into the repo, open it and it's actually the main branch, the main page will be the latest code.

Sergey Shandar - **25:26**

So I don't need to switch to development branch or to next branch or to something else. That's one point.

Mårten - **25:36**

Can I add something to the branching strategy? Yeah, I agree. And I feel like when you have a repo and you're working on a single branch, if you can do that's the best thing because it makes things simpler. On PRS, you can do merge commits and still you can squash everything together. It's easy to inform enforce a linear history. Previously, in one of my jobs, I worked in a more complex project with a lot of teams, working in a huge code with like 2 million lines of codes. Quite a messy project, but they still maintained a very good version history. And this was also like one of the setups where we had more needs than we could cover with a single branch. So we had the multi branch strategy, but everything still boiled down into a linear history on every branch. We never merged between big branches.

Mårten - **26:29**

They were still based on each other. So you had like an integration branch where people work together and things were scorched together into releases and they were ultimately making the way to the main branch. And every now and then you would have to make an emergency release and roll it back. But it still made a lot of effort in maintaining this linear relationship between the branches so that you don't have these separate heads of these mythological beasts which are biting each other back and forth. And it's extremely hard to understand it. And I think that's sometimes you need a more complex branching strategy. But being able to maintain a linear history is giving a lot, especially to new developers, but also when you need to debug or when you need to roll back. And that's a value that you're getting as the system grows over time.

Mårten - **27:15**

So I just want to advocate for that.

Will Corcoran - **27:20**

So as we're sergio, I'll come to you in a second, but maybe people could think about this. So as we're ramping up in the size of the team and the complexity of the projects and what we intend.

Will Corcoran - **27:40**

To ship, suppose that complexity needs to.

Will Corcoran - **27:51**

I guess what I'm trying to ask.

Will Corcoran - **27:52**

Is how do you ideally balance new feature time?

Will Corcoran - **28:03**

Like time spent working on new features, time spent working on enhancement to existing features and just refactoring and kind of code health, just trying to improve upon what's already there or simplify it even more. If you're trying to think about like percentages of time, what do you suppose that you allocate now versus what in an idealized situation, you would allocate.

Will Corcoran - **28:37**

Sergio, you want to jump in?

Sergey Shandar - **28:42**

I just like to add to the last comment what is Parting said? Sorry, that a little bit like kind off topic. I will try to finish this. So the thing is actually to work everything, like work in a one branch and just do like a features in separate branch and then merge them always into main branch. I see like, huge companies and huge teams like Facebook, Google and Microsoft sometimes for some projects they use monoreapo with one branch, like with main, and it's actually successful. So it's not like only like that.

Mårten - **29:24**

Depends on a bit on deployment and what the product is. And I agree, if you can do that's the best.

Sergey Shandar - **29:30**

I think it's always possible, in my opinion. I've seen different projects. It's always possible, but you need to organize it properly.

Mårten - **29:36**

But if we take, for example, I'm not sure, it depends on the code organization. If you have higher quality code, then it's probably more possible, especially if you're developing things as libraries. I think everything you're suggesting sort of makes sense. But if we take the current hardball, which is the stacks code, and we're taking the work, which is happening on SPTC, which is a hard fork, you cannot merge like a wire format update in the stacks node, push it on Master and then release it because then you're causing a hard fork on the chain. Right. So you still have to control that. So that would be, unfortunately, a long lived branch. But you would ideally, when you release it, not have a merge commit that just bashes these things together, but actually maintaining this branch as recently as possible and actually rebasing it. But again, I see that my example is also based on a system which doesn't have ideal code quality.

Mårten - **30:34**

And I see how you can ideally, I agree that you should adjust the code base so that it's possible to work in that way. And that has high requirements on the actual code.

Sergey Shandar - **30:45**

We could actually discuss it on separate meeting. I think it's actually like a big and we could discuss it, like the strategy, how we could actually improve this. Okay.

Aaron Blankstein - **30:54**

Yeah.

Will Corcoran - **30:56**

So in terms of trying to improve the situation, suppose that one methodology was documentation.

Will Corcoran - **31:08**

So let's talk about that real quick.

Will Corcoran - **31:12**

I got a couple of questions regarding documentation.

Will Corcoran - **31:16**

If anyone wants to take on any.

Will Corcoran - **31:18**

Of these, like from a high level.

Will Corcoran - **31:20**

Perspective, what do we feel the current.

Will Corcoran - **31:23**

State of documentation is? What parts of the code are well documented versus poorly documented? And who realistically can take on more documentation?

Mårten - **31:35**

So can I just answer, because you had a previous question about how much time we allocated working on code because we're not in sync in questions and discussions. Just to comment on that one, I want to advocate for the Scout Rule, where you're always leaving code that you're touching in a better place than it was when you found it. But also, the way I see it is like personally right now, I think we especially the work I've been doing, I've been focusing much more on features and pushing things out, especially SPTC Alpha. It's not in a good state and we don't dedicate time to code enhancements. And if we even formulate the ticket for a code quality improvement, it's low prior, which is really bad. Ideally, you should spend at least like 20% of a sprint on pure quality enhancements, in my opinion. But on the other side is that when you're actually implementing a feature, if I'm having a feature, because the way you're implementing it, typically, or at least my opinion, the best way to do it is go through the code.

Mårten - **32:37**

And most of the work is like, tweaking the code so that it's prepared for the change and then the actual change that you're implementing is trivial. So in one sense, actually improving implementing features can always improve code quality. And you should always like that's. Again. The scout rule. Like, you should always think about code quality in every feature that you're working. But yeah, that's sort of self evident. So let's get back to documentation, unless anyone else has any comments on the code quality timeline question.

Will Corcoran - **33:07**

Yeah, you'll probably hear me say this a million times in the future, but I was an architect for 15 years until just two years ago. And a saying from the construction industry is measured twice, cut once. Which is really make sure that before you take action, you are very confident of what it is that you're going to do and then spend a minimal amount of time taking action. Another way that it's been said is I think there's an Abraham Lincoln quote, if you gave me 6 hours to cut down a tree, I'd spend the.

Will Corcoran - **33:47**

First 5 hours sharpening my axe.

Mårten - **33:51**

What did you say? Something twice, measure twice, cut once.

Kenny Rogers - **33:56**

Like you can't cut a piece of.

Will Corcoran - **33:57**

Wood twice, so you got to measure it twice.

Mårten - **34:01**

I've heard the Think Twice code once.

Will Corcoran - **34:09**

Kenny, I know that you're specifically interested in documentation. Anything beyond what's on the screen here that you're dying to know or any words of wisdom that you would want to share?

Kenny Rogers - **34:27**

No, I think these are the important questions, just for my part. From what I'm able to work on, I'm very much like sort of a higher level developer documentation person. So I'm able to write documentation on how to build a front end for clarity contracts, how to write clarity smart contracts. But my expertise is definitely not at the protocol level. And so I'm happy to help write documentation, but I will definitely if I do that, I'll definitely need some help with understanding the code base and stuff like that.

Jesse Wiley - **35:04**

Sure. Yeah.

Aaron Blankstein - **35:11**

So I just wanted to chime in on documentation because I think based on Kenny's answer there, I think that really there's at least sort of two broad categories of documentation. I mean, there's like documentation that is supposed to be end user oriented or like developer user oriented. And that is great for people to help the blockchain contributors with people who are contributing to the blockchain, it's great for them to help there. The other category of documentation is like the documentation of the code base itself that we talked about a little bit earlier in this call.

Brice Dobry - **35:59**

And that I think needs to be.

Aaron Blankstein - **36:02**

100% of the responsibility of the people contributing.

Kenny Rogers - **36:07**

Like, if you are writing code, you.

Aaron Blankstein - **36:10**

Are responsible for the documentation of that code. I think this is something that has had broad agreement amongst blockchain contributors since time immemorial. But just because we have agreement on that fact doesn't mean that the code base has documentation because otherwise we would already be there. And so I just want to plus one jacinta's earlier point that automated tools are probably the only way. Out of this, where PRS just have a giant red X next to them if they do not come with some amount of documentation. And it's true that you can defeat that giant red X by just writing bad documentation, but I think that we have to have some level of trust that people won't just do that.

Will Corcoran - **37:14**

I see you have your hand up.

Brice Dobry - **37:16**

Yeah.

Jacinta Ferrant - **37:16**

So this is kind of related to the earlier comments of if the code was broken into clearly segregated libraries, a lot of the documentation would come easier because you're going to end up with a public API from your library. And if you had, for example, you're probably only exposing a certain amount of functions, and those are the ones that need the bare minimum documentation from the user's perspective. And then that just like if you're using Rust properly, your user documentation comes naturally from the code. But then, as Aaron said, if you're the one writing the code, you should be documenting it as well. So I'm very much for a component of the function should be named. I think a lot of people think they don't need documentation because, well, if it's a well defined named function, you don't need the documentation. And that's partly true, but even with private functions, I'd say that isn't the case.

Jacinta Ferrant - **38:16**

Even just a one liner is all you need. But I really think pushing for libraries would help with getting that minimum required documentation that would seem more digestible. Because right now, if you look at the stacks blockchain code, it's so big and the files are so massive that it's hard to know. What would you even define as the bare minimum documentation to be considered acceptable? I think, yeah, they kind of go hand in hand separating into a library, but that's just me. Anyway, that's it.

Will Corcoran - **38:49**

Go ahead, Martin.

Mårten - **38:50**

Yeah, because this discussion often boils down to something about self documented code versus writing comments, which is like I tend to be on the team of being sparse with comments and being very intentional about naming, but of course depends on how you use it. If you have really good code, this is about a developer reading code and understanding what the code does. And if the code is essentially has integrity, it's sort of aligned with the intention of the code and what it does in modules. And like you simply says, if we have better structures, less things needs to be explained. There's always going to be this sort of room in between. Things are not going to be perfect. The worst part is code that people think is self documenting that isn't. So you always have to compensate for that gap with comments or things like that.

Mårten - **39:43**

But if we zoom out a bit, I think now we're talking about code and developer documentation more intended for ourselves, the developers. We do have external documentation. I think that one is pretty decent. I'm not the best person to judge that one. But there's another side of the documentation as well, which I think the code documentation is super great if we compare to how bad we are at documenting design decisions and efforts and issues and what we're about to do whenever we're going to work and plan and do design. Historically, we haven't been very good at doing that. It's very hard, at least for me, to understand the SPTC design. The exception to this is actually the Sips, because I feel like whenever I read Sips, I can actually understand things about the stacks blockchain and things like that. So they are very well curated.

Mårten - **40:40**

That's a slow process, but I think there's a lot of room for us to improve in terms of planning, coordinating and thinking about what we're going to do. Again, think twice, code once. Like, what are the things we need to do, what are the next steps? Coordinate, implement those things and having that sort of intention instead of just coding things together and seeing what happens, actually having a plan, following that plan, iterating on that plan because we're never going to plan correctly first. That's going to make it so much easier for us as developers to also write clean code so this all ties together, right? Because if I know what feature I'm doing, the purpose of that feature is going to be easier for me to structure that code. It's going to be easier to document that code, and then the developer documentation is going to be easier.

Mårten - **41:26**

But it's also going to be easier to trace back context about that code again in the Git history. Seeing why this code is there, how that relates to the plan and how that small piece of code fits into the big picture of what we want to achieve and our long term goals. And I think that set of things is very fundamental to make it easier on the other things. I think we have a long way to go in terms of design and planning and translating that into smaller issues and doing that.

Will Corcoran - **41:57**

Bryce, before I throw it to you, maybe think about this also, but when we have these tasks in front of.

Will Corcoran - **42:06**

Us, like, hey, we could all help.

Will Corcoran - **42:10**

Save ourselves a lot of time and effort and grief and make this project easier to build upon in the future, faster development. If we had a cleanup crew coming up behind us. And basically what I'm trying to ask the documentation and some of the refactoring are these things that you could delegate to someone. Like if were to support people with critical Bounties and say for a period of time, we know that the investment is going to be worth it. And so we just need to identify a couple of people and we can support them. Is that something that can be delegated or not?

Sarala B - **42:58**

Was that directed at you? Go ahead.

Brice Dobry - **43:01**

No, go ahead.

Sarala B - **43:02**

Yeah, I would separate the refactoring of the code from the documentation because delegating and off outsourcing refactoring of code to an external team is just asking for us all to fail collectively very aggressively. And again, documentation. Also, like I said earlier, we need to split that into two categories. Everything that has been talked about right now, the documentation that's related to code, self documentation versus comments, the quality of code readability, et cetera. That needs to be a culture and that needs to be kind of embedded in everything that we do versus outsourcing it. The part that I see that can be delegated outsourced is the external add on documentation that comes as an aid in addition to the code, right. Documentation of how to use the system that's more outside looking in. So I'll be very careful kind of bringing in cleaning crews or refactoring or delegating the responsibility.

Sarala B - **44:10**

It needs to be just everyone taking ownership of that. Easier said than done. But we need guardrails, like we've talked at length about what needs to be done, what are the best practices. There are more points that we're all aligned on. I think it's a matter of how do we get there in terms of, okay, how do we tweak our PRocess so that it's kind of embedded within that. How do we change, how do we change how we commit code, how do we merge code? Just having guardrails around that is the next step. I think just collating all our common pieces right now and taking those action steps is probably worth it for the next sprint, for the sprint that we're talking about. And then I would also include for any PR that goes in checklist of those items and guardrails around that. Does that make sense?

Jesse Wiley - **45:06**

Yeah. Bryce yeah.

Brice Dobry - **45:09**

So I just wanted to add one other kind of piece of documentation that maybe isn't covered by what we've talked about with PRS and making sure everything new is documented. I think there's also the architecture and design documents that are pretty lacking right now that I imagine that living in a wiki in the GitHub repo that can be editable by everybody. And this will be like the place you go when you're jumping into a new section. I can go and see these threads are running and this is how they're communicating. Kind of the higher level stuff that is probably in the comments and in the code. But it's hard to know where to look for it if you don't know where to go. So I think that's a really important piece, especially for getting new programmers in there.

Jesse Wiley - **46:00**

Sign up. Yeah.

sayak - **46:04**

Just to take back what Martin was talking about. So basically, the conversation about self documenting code and writing documentation, I feel a little misguided because when we're talking about self documenting code, we're talking about individual functions. And I think that if you write good code, that's fine. However, we're just working with tons and tons of files, right? So it's like figuring out how the data is flowing is the biggest challenge. So I think that, for instance, when we talk about splitting code up into separate crates, separate modules, it's sort of like limiting the scope of things that could be interconnected to each other. Also like the libraries versus binaries thing, it's sort of like reducing the complexity of things that you need to navigate to understand one piece of code. So to me, documentation isn't necessarily explaining what the function is doing, because you should know that from reading the code, but it's more like how different pieces are sort of fitting together into the bigger puzzle.

sayak - **47:15**

And the bigger puzzle cannot be the whole code base. The bigger puzzle has to be like smaller units. And then once you ungraph the scope of one smaller unit, you move up like one layer and it should be pretty layered.

Jesse Wiley - **47:33**

Excellent.

Jacinta Ferrant - **47:36**

I was just going to partly echo what Soralis said, but in terms of how I would kind of approach this if people are willing to dedicate a sprint to tech debt before even doing. Maybe this is wrong, but before even doing documentation, let's say, like I use the example of, let's say, use clippy. You can just apply it to a single file, and then you tackle that file for a documentation as an example. I would probably not even bother doing that until the architecture design of how it should be refactored. Like, okay, let's use an example of all the Http code being pulled out into a proper library on the side, for example, all of the I can't remember exactly, I was done with the peer to peer network that's set up in the stack. So if that was pulled out, it would be a lot easier after that to then say document.

Jacinta Ferrant - **48:36**

There could be an overarching story ticket that's document the Http code. But when it's built in the way it is now, I think there's a lot of functions combined together that it'd be very hard as someone coming in who isn't the one who first wrote it, to document and to follow the flow. So I would not even bother with the documentation until the refactor of the code layout is decided. Then it's a lot easier to break it down and probably someone who because at this point, it'd be very hard to say. I mean, I think Aaron wrote I don't know how many lines of code he wrote, but will he have the time to document it all? I highly doubt that. So I would definitely make sure that people aren't diving headfirst into documentation before things like the library is being pulled out is done.

Jacinta Ferrant - **49:25**

That's just my viewpoint.

Jesse Wiley - **49:26**

Yeah, that's it.

Will Corcoran - **49:28**

Yeah. No, that's really wise in terms of.

Will Corcoran - **49:31**

Not wanting to just reinforce a bad.

Will Corcoran - **49:35**

Pattern or allocate time towards documentation until that first level of really simplification can be done.

Will Corcoran - **49:49**

So in terms of sorella, I'd love.

Will Corcoran - **49:52**

To hear your opinion on this in.

Will Corcoran - **49:54**

Terms of, like, staffing, something like that.

Will Corcoran - **49:58**

Or considering the time that would go into something like that.

Jesse Wiley - **50:05**

How do you.

Will Corcoran - **50:06**

Consider those decisions and prioritizing that? At what point would that make sense?

Sarala B - **50:15**

It's a question around taking time out specifically for documentation and refactoring.

Will Corcoran - **50:22**

I think prior to that, since I mentioned, like, hey, what if there was a sprint that was just focused on alleviating tech debt before trying to get into documentation and building upon that, really just make sure that we have clarity and clean delineations.

Sarala B - **50:45**

I would even probably step back a little more. I've had my fair share of refactoring throughout my life. For any refactoring to be successful, you need good testing strategy first. Otherwise, how else would you know that your refactored code is actually functioning, working well? That's step zero. So for sprint Zero, as we collating, what needs to be refactored, what needs to be modularized, what needs to be documented, let's work on the guardrails around how do we determine all of this is successful? So starting with how do we improve our testing, then comes refactoring that would be around cleaning up code. Marie condoing it a term that I use internally at Hero quite a bit and also modularizing. Then use that for the next sprint. And then as part of as you're doing both of these, I think documentation needs to go in tandem with that.

Sarala B - **51:44**

It's not one after the other or one is better than the other. It needs to go in tandem, parallel. We can look at the documentation that Bryce and others have highlighted around architecture, design, documentation, external, outside looking in documentation that doesn't have to wait for the refactoring that can continue to happen. That's how I would start.

Will Corcoran - **52:10**

And then, Aaron, I want to come.

Will Corcoran - **52:12**

To you last when you're confronting, hey.

Will Corcoran - **52:16**

We'Ve got this big initiative in front of us with federal blocks that needs to move forward. How do you consider weaving in some of the things that Saral was just mentioning, like this theoretical sprint zero, sprint one, sprint two of testing an architecture and then refactoring and documentation.

Brice Dobry - **52:43**

Yeah.

Aaron Blankstein - **52:44**

So I guess to answer the question about how I think about weaving it.

Kenny Rogers - **52:49**

In for a work stream like the.

Aaron Blankstein - **52:55**

Testing workstream, there are sometimes actual requirements from the testing work stream that things become more modular in the code base. In order to be able to write model tests or property tests, you have a requirement from the code base itself in order to run the tests. And so you need to do some of that refactoring or some of that documentation as part of that work stream in the first place. And I kind of think that better blocks will have some similar shapes. Like, if you think about all the proposed changes to the protocol, the prose changes to the consensus algorithm, that work is going to end up needing to either replace or modify large portions of the code base and that's a good time to think about abstractions. So that's how I would think about weaving it in. But there's also plenty of work that's kind of unrelated to that, but that would also be super beneficial refactoring work.

Aaron Blankstein - **54:13**

And when I think about refactoring sprints or something like that and just in general complaints about code quality, the way that you move from just like a general erring of grievances to actual work completed is like you have to generate really specific tasks. So, like, an example that comes to mind immediately right now is, like, modularizing. The clarity code base sufficiently that the WASM compatible clarity library that gets published on the cargo repositories can actually be built from the main line of the stacks blockchain. Because that seems really annoying, both for clarinet, but also if we wanted to integrate any testing into CI, that's impossible to do. So that's like a specific task with basically a success metric. And so I think actually breaking into things like that would be the idea.

Muneeb - **55:37**

I need to jump on something else. But I think one thing about the code quality discussion that's going on is we should try to check for just at a high level because I think some code might get rewritten for Nakamoto and there might be certain libraries that we start using versus the existing code. So I think that discussion just in terms of sequencing maybe should happen first because it would be a waste of time and effort if we are trying to clean up that depth on code that ends up like getting thrown away.

Mårten - **56:08**

Right.

Muneeb - **56:08**

So I think just a quick comment there and then I heard WASM here and I wanted to propose that earlier when were having that clarity VM discussions that is that going to be vASm? Because I think it should be if it's possible because I think I don't want to open like a can of worms, but I heard the term and I'm very curious. I would be to explore that more. Maybe there's a separate meeting for a.

Brice Dobry - **56:34**

Deeper dive on it.

Jesse Wiley - **56:37**

Great.

Will Corcoran - **56:39**

So you probably saw there's like six more pages of questions to get through.

Will Corcoran - **56:46**

And normally I was going to kick it off to go and turn the.

Will Corcoran - **56:55**

Conversation on Tuesday to green team topics, but I think that we need to keep digging into this seems very top of mind. It's going to help with the testing and hardening work stream. Serology, to your point, I'm going to take these slides and the conversation we just had and start to spill this down into some potential concrete tasks and action items that we can take from this. I think that this is super important and I guess we want to make sure that we're not setting anyone off on a goose chase too quickly and we'll complete the conversation and hopefully pick this up. We might take next Tuesday and next Thursday on this topic, and that will really help us go into the sprint plan for Sprint, too.

Will Corcoran - **57:57**

But I wanted to show you my.

Will Corcoran - **57:59**

Cool Green team slide before I wasted it.

Jesse Wiley - **58:04**

All right.

Will Corcoran - **58:05**

Anything that anyone wants to say before we part ways?

Jesse Wiley - **58:12**

Yes, Jesse?

Will Corcoran - **58:13**

Sorry, I didn't get to yeah, jump in.

Jesse Wiley - **58:15**

Yeah.

Jesse Wiley - **58:15**

So it hasn't been mentioned, but this very much goes to quality of life. The current logging output that we have is horrible. It's been a pet peeve of mine since Specs 2.0 launched a while back. I would love to see better logging, more informative logging, and I think that would help a lot of other areas as well. Today, it's very confusing. If you have a problem and you have to look at the logs, usually what I do is I dig into the code based on the log message. The log message just doesn't really tell me what's happening.

Jesse Wiley - **59:01**

Thanks. Yeah.

Will Corcoran - **59:04**

So I will be taking the firefly notes that gets transcribed from this, and I think a lot of stuff is thrown out there, like, really just trying to find the signal on this and hopefully just keep us moving in the next meeting on Tuesday. Seems like we're surfacing a lot of really important things. It's just a matter of making sure that everyone feels like we're converging on the right protocol in order to move.

Will Corcoran - **59:34**

About improving this situation.

Will Corcoran - **59:40**

But been a lot of fun.

Will Corcoran - **59:43**

I will see you all soon.

Jesse Wiley - **59:49**

Yes. Thank you.