2026-04-22 Governance weekly

- Date: 2026-04-22
- Participants: ???

Meeting notes / Discussion

Agenda

- Codeownership of f2/testlib (removal of more broad owners)
- Perfectionism vs. incremental changes
- Flakes eval/language restrictions
- Discuss moving the testlib bits outside of the monorepo to address Python dependencies issues

Code ownership of F2/testlib

- rootile: Entire test folder is owned by everyone. It makes sense, everyfew can make tests, delete them, etc. This does includes the testlib directory. IMO, while it belongs to tests, it is its own codebase, it should not be changed or owned by everyone.
- raito: proposal, put OWNERS in testlib with disabled inheritance and put the ownership to F2's codeowners.
- rootile: I'd also like to remove helle from code ownership of F2 because they have not been responsive in the past, if they become active again, I'd add them back. The only codeowner of F2 is me then.
- raito: currently, there's "root code owners" / "default code owners", so you are not the only CO on F2 in practice
- kate: I'd suggest code owners for F2 pieces a couple of core team members e.g. Osiria, just there, so that there's review distribution besides rootile.
- jade: a general comment on things of the shape of testlib, this is a thing that is well trodden, the google stuff is in a testing subdirectory of a source directory, we do not have a good place for python sources, it might be reasonable put testing sources in a src directory as well. Folks mentioned they would like to use F2 for other code outside of Lix, perhaps, we could consider F2 as its own software outside of Lix.
- kate: let's consider using other repositories even if Raito is a monorepo maximalist for things that are not just for Lix.
- qyriad: there might be python packaging concerns
- jade: i am very against moving anything outside of the lix repo for now because you have to deal with lockfiles, nix-eval-jobs was moving inside because it was breaking it all the time; we could use copybara which is how i release snowydeer, to create an export of that part of the lix tree if that's something useful
- moved into its own topic: moving Python outside the monorepo

Action items

- `set noparent; set rootile` and add folks who wants to help
- disboard helle from the F2 code ownership

Perfectionism vs incremental changes

- raito: [insert later the contextualization] (editor's not: lolsob)
- jade: this was written in the context of trying to get the work of unpacking changes in and it did not feel like we were start incrementally merging under a flag
- kate: i feel like there's a bunch of topics that was merged into this loaded topic; what is our obligation and standards of what we were doing, what are expectations for experimental, code that is going to grow vs. code we know that is less likely to have bugs — i don't find the culture of code perfectionism does much to address the number of bugs there, i do think it's a complete separate topic from one of the thing we have issues with: communication, plans on how to get things merged, what is acceptable, general communication about it — there's a definitive issue i have seen a lot, it's that people have a lot of trauma, their own expectations of themselves, things they expect to burn them in a codebase ; someone make an issue, "i don't want to deal with this trauma issue again" and it creates a situation where this is not even about a technical argument anymore because it is going to cause something negative to me happening — in a summary: we have a problem of discussing how to communicate to each other, what to do about the fact that a lot of us are afraid to drive changes in the codebase
- piegames: i'd like to split up the perfectionism from the culture of wanting to do things right, i do think it's a very good thing, that i think lix prides itself with, this can be a thinner line at times; one of the values i wrote back for the first fork and i think lix adopted was that we are willing to go to the extra mile and to do things right and prefer solutions that are long term and sustainable than quick and easy hacks to get things working if that's possible
- jade: i totally agree with kate that the underlying problem is the fear more so than it is anything else, i'd really like to decrease the fear, we can improve our testing story, we can start having fuzzing, we can start having other ways to hunt down bugs before merging, there's verification methodology to decrease the sense of fear, it's a pretty large problem; other methodologies could include having ways to persist warnings, we could surface later
- raito: [oh no no one took note :D]
- kate: the thing i said i realize could be taken the way raito took it :D — there's a difference between doing perfectionism and having a culture of doing things well; perfectionism encapsules to me the standard of "i have a way i think this should be done and i won't tolerate deviating from the way i think this should be done", i think we have accomplished a lot compared to CppNix in how we are taking time to plan and execute them properly, i think this is a culture of doing things right and i think this is different from a culture of perfectionism; can we introduce bugs as part of experimental features as long as they are caught in user testing? these kind of things are implicit in the idea of perfectionism
- horrors: phrasing this entire thing in testing or the fear, it does touch on important points, but not all of them, regarding the macOS stuff that created this topic, there were bugs surfaced and due to communication disasters, it ended up being a minor incident; if we

add features that people really obviously want and there's dangerous, they should come with grave warnings they are dangerous, we only have experimental features with various degrees of communicating dangers from none to "hey this is stable!"; experimental features alone is just not a good framing for the kind of rare-to-common bugs that such features may bring and we do need get better to communicate to our users too

- piegames: about the testing, in spirit of doing things that are long term sustainable, i'm really fan of formal verification, strong type system, things that are resilient/correct/secure by design, fault tolerant by design; as much as I appreciate the efforts like doing the fuzzing stuff, doing these things have an high opportunity cost on NOT writing new code in Rust, NOT writing new systems that can improve upon the old ones that doesn't have failure modes in the first place. About the experimental features, everything is tangled with everything and all failures are very tricky and catastrophic. The focus to me is rather on the modularization and kinda refactoring in Rust and being fearless over there, if something goes wrong, it won't be that bad.
- kate: first, I'd like to address what piegames brought with formal verification, for 10 years, I was the head of being computer division, making a very very high assurance product with a full budget, trying to make it happen and it's — in scale — formal verification is not used because it's in general intractable for something that is a side effect manager. It cannot be the sole method of architecturing code, throwing so much money did not get fractions of the Linux kernel running correctly. I don't think fuzzing is taking time away from formal methods, because formal methods are inherently very hard to apply.
- piegames: Having type systems or encoding grammars are also acts of formal verification in my eyes.
- kate: language constructs as a FV is also insufficient — making a sound language that is capable of having things like that typing depends on data, you start more and more into mathematics and formalism. I don't think getting to Rust is going to change dramatically this. Things that can find bugs in other methods can inform on the overall software engineering. This is kind-of like defense in depth for testing.
- kate: we should talk about ways on reducing conflicts we had, for example, with the archival CL, something that could have solved the whole thing is that if there was a communication: I'm going to do this and I will architect and can we figure out the whole way on how to solve the problem? That didn't seem to happen in a public space.
- jade: I did!
- kate: We did find a better solution by doing investigations on this topic (wrt filesystem slowness on macOS) and we were able to come up with a better plan for virtual filesystem management that both solves the NAR archival by engaging a group of people.
- jade: I completely agree with many of this, it would be incorrect to say this was not discussed before it got written ; the implementation we came up was suggested by horrors. Anyway, we should lean on how other projects improve their quality: we can do somethingASAN, we can fuzz more bugs to appear or to be found. We can decrease the scariness of things. This is what I want to work on in Lix in the next few months.
- kate: As one of the two Darwin maintainers, the filesystem issue was not ever brought up to me or Osiria in a way that this is the plan we are going to do and discussing it. We should agree on a policy to develop expectations on developing of any piece of development.
- raito: re: Kate's thing about formal verification

    - Worked on translating Rust to Lean
    - There's a project called "Signal Shot" (?)
    - Formal verification is still worth investigating
    - Though overall agrees with Kate
    - Thinks the true problem is in the communication
- Raito: Re: Darwin maintainers were not made aware of the unpacking thing design
    - Doesn't think that Kate/Qyriad were not yet considered the true Darwin maintainers
    - Thinks what horrors tried to do was give incremental feedback as they found issues
    - Thinks it's not about fear of the codebase and more about discovering things as we go
    - When you open the Lix devshell it asks you to reach out
        - We don't do that that much these days

Action items

- Raito: thinks that further chat about this can be asynchronous
    - Can create Zulip topics about e.g. fuzzing, reducing conflicts, developing expectations (reinforcing the policy we have on the devshell contribmsg)

Flakes eval/language restrictions

- piegames: Flakes uses a lang subset, which is full of defects and is an enormous layer of violations; this has been shakey grounds because of the layer violations, my current plans with bytecode conflicts with that piece of code. I'd like to make some breaking changes: removing all of the language restrictions in flake.nix. I don't know enough about Flakes to fully grasp the consequences but I do know that it will break CppNix unless they also do something, this will require coordination.
- Qyriad: this is an incompatibility we would be introducing with CppNix users (rather than a Breaking Change™ for Lix users).
- jade: exactly the same thing. Actually, it's less bad than we would see from touching libfetchers, we have some existing incompatibility in libfetchers, this will probably break some Lix users. We have been wanting to do this for a quite while. also, there's holes in the current model which makes it possible to evaluate complicated thunks in CppNix by abusing dynattrs!

(continued in next meeting)

Revision #1
Created 2026-05-13 20:22:30 UTC by piegames
Updated 2026-05-13 20:29:50 UTC by piegames