Reflections on the React community

2025 – lee robinson

I've been using React professionally for almost 10 years across three different companies. **I love React**. It's responsible for much of my own personal growth as a product engineer and has helped me build wonderful things.

I started using Next.js in 2018. From 2018 to 2020, I independently created content and taught the React community. And then for the past 5 years, I helped manage and grow the Next.js community at Vercel.

This is a post I've wanted to write, but felt I could only write it after stepping away from my official Next.js role.

The overwhelming positive side

React is "boring tech" at this point.

Every year, React continues to grow and gain adoption. The "React way" clicks with many developers, and that has led to it becoming one of the largest ecosystems of developers, packages, and educational content for building on the web.

This comes down to two main things for me: composition and stability. The composition model of React was the right architecture choice in the beginning, and continues to have benefits. For example, AI models are able to generate encapsulated components you can drop into your app without affecting global state. Local reasoning is awesome.

React is also stable. The core APIs rarely change and there's great backwards compatibility, and most of the innovation comes from other tools or frameworks trying new ideas on top of the core React foundation. You could argue that ecosystem thrash reflects on React itself, but I'm not so sure. It's popular enough that some people are bound to use it incorrectly.

With that being said, you might have felt that the React of today is not the same as the React of

2015-2020. Why is that?

Communities are hard

Let me start with a personal note. Five years of managing the Next.js community was hard. There are so many great things to reflect on, but also a lot of unneeded stress and frustration. I'm proud of the work I did and the community that we all built together, but at the same time, it does feel good to step away and write this.

It takes dedicated people to persevere in growing a large, open-source project over 10 years like React. This is why some of the most successful OSS projects have BDFLs (Benevolent Dictator For Life). DHH has some great rants on this. He's building Rails as a gift back to the community, not as a commercial OSS project. He doesn't mind telling people no when they make their demands. "PRs are welcome" as they say.

The reason I'm starting by talking about community management, is that I can empathize with the React team (this includes current and former members, those at Meta and outside). They built something incredible and poured their heart into it. Showing up and having a voice in the community takes a lot of time and energy. I get why some have tuned out and stopped engaging.

The problem to me is that React isn't a self-governing community. It requires work.

Listening to feedback

As the size of the React community has grown, so has the amount of feedback. Some of that feedback is good, some bad, but in both cases community members are taking time to share how they feel about a piece of technology. That's a gift, even if there is a signal/noise filter that needs to be applied to some percentage.

Now, if we take the most pessimistic view, you could argue: Lee, none of that community management stuff matters. Clearly React continues to grow and get more popular, even without the React team being active in community discussions. They built a great tool and that's all that matters. Why waste time on all of those discussions? They know the right thing to build and all of that feedback is just noise.

There's some truth to that. Will React continue to grow even if the team never responds to another tweet, GitHub discussion, or Reddit post? Yep, definitely. There's also the human side of that argument. It is draining to process and filter negative feedback into something constructive that can

be used to push an OSS project forward.

But when you separate yourself from the community as a tool author, it means other voices will fill the void. Those voices have their own incentives and don't have the best knowledge about how React works. This can lead to FUD (fear, uncertainty, doubt) and topics running on for much longer than they need to.

A few recent examples that come to mind are the CRA deprecation drama and the "Vite isn't as prominent in the docs" drama. Did both the community and React core team have some good points? Yeah. Did it need to escalate that far? No. Were there good intentioned people on both sides? Yes.

If for nothing else, one of the reasons to have DevRel for OSS projects, is so that it is someone's full time job to do this work. If it's not their job, then other stuff rightly comes up and gets in the way, like building the actual library. Core developers don't want to sink a bunch of emotional and mental energy into wording the right response, knowing they will invariably get negative feedback.

Commercial vs non-commercial OSS

React is not a commercial project. Meta/Facebook made React and give it away for free.

You can try to argue the other side that React is beneficial for them: React brings talented developers to Meta, it helps improve their core product experience, they get bug fixes and improvements from outside the Meta team, and more.

And yes, some of that is true. I'd argue there's not that many React bugs being fixed outside the core team. My larger point is that those benefits are ***tiny*** in comparison to the cost of employing people who work on React. Think about how many R&D dollars were spent on the React Compiler, just for them to give it to everyone 100% free.

Meta isn't making money off React. They are not financially incentivized to grow the project. They built something amazing and gave it back to the community. We should be thankful for that.

Where things start to get tricky is when there are ***commercial*** OSS projects built on top of React. Their incentives are different. They ***do*** want to grow their project, because it is usually relevant to the growth of the company. For example:

- Gatsby, which was a React framework, had an infrastructure business called Gatsby Cloud. They were incentivized to grow the usage of Gatsby in the hope that some percentage would choose to use Gatsby Cloud. Their business ultimately did not work and they were acquired by Netlify.
- Remix, which was a React framework, had a business model where you would pay for access to the

framework. This was probably one of the clearest incentive structures possible. It did work, and they were acquired by Shopify.

- Astro, which is a general web framework, took funding and eventually launched a managed database from the backing company. The DB was spun down. I haven't caught up on the latest there.
- Redwood, which is a React framework, had an initial business model to help support startups. It was funded by the founder of GitHub. They have since pivoted to be a framework specifically for Cloudflare. There is prior art with Flareact, and then Hono, which is built by a Cloudflare employee.

And then there's Next.js. The business model of Vercel is to primarily provide managed infrastructure, or a "platform as a service", which aligns with the incentive to grow Next.js.

Both of these things are true: Vercel doesn't need to monetize Next.js to have a successful infrastructure business, as they support many different frontend and API frameworks. But also, as Next.js becomes more popular, some percentage of that usage does end up on Vercel.

As if getting eyeballs and usage of your OSS project wasn't enough, you must also build a sustainable business on top. And if you take funding from venture capital, the goal isn't a business that just does well, but a business that does exceptional in the category on both adoption and revenue metrics.

Companies have tried the Vercel & Next.js business model, but almost always fail. Vercel has been an ***exception*** at $200M in revenue and growing, not the rule to follow going forward. Expo is another successful example.

So why did I give all of that context? Because commercial and non-commercial OSS have different relationships with the community. React doesn't have a financial obligation to respond to GitHub issues. It's more like "read-only OSS". Next.js ***does***. If Next.js doesn't do right by the community, the project will not succeed.

This is why you continue to see Next.js growing and evolving based on community feedback. I spent a lot of energy helping people self-host Next.js successfully, and Next.js will soon have deployment adapters to make it easy to deploy to any managed cloud provider.

React Server Components (RSC)

The React team had the ambition to extend React's composition model across the network boundary, bringing the React you love on the frontend to the backend as well.

This was a massive project. One that would require financial backing to build out. As mentioned

above, Meta was not incentivized to build this. So what can the React team do to make this vision a reality?

This is how Vercel got more involved in helping contribute to React core going back to 2021. Some members of the React team joined Vercel, and React evolved from a single-company to multi-company project. There are other independent, non-Meta folks on the React team as well, but Vercel was now the biggest outside corporate backer.

So, Vercel invests millions of R&D dollars into building new React features. This includes React Server Components, Actions, any other features eventually included in React 19 years later.

Many of these new React APIs weren't yet usable by the broader community until they had been validated in the real world. This meant being integrated into frameworks and getting deployed in customer-facing applications. Like I mentioned, this wasn't happening at Meta.

This is why the pairing with Next.js made a lot of sense. The React APIs were being co-created and validated in the Next.js App Router, such that they could then be released more generally on React stable channels for the broader community. My favorite example of this are React Actions, which work without a server or framework!

Now, we can litigate exactly how these improvements were communicated and released to the world. There's plenty of missteps here, including those made by yours truly. The App Router was likely marked stable too soon, with the intention of declaring the API surface area as stable (not that it was generally bug-free). Obviously that was a mistake, as people felt it was pushing a sub-par RSC experience too soon.

Further, the incentive structure here meant that it was unlikely and/or difficult for other frameworks to invest the same amount of time and energy into integrating these APIs as the Next.js team did. I do think the React team could have done a better job of bringing the community along for this ride in the open, and at the same time, I understand their hesitation.

Ideas are fragile. I'm sure the React team felt like everyone would have strong opinions about RSC and the API design, and that led them to be more reserved in what they shared. Plus, they probably felt burned from sharing Suspense and Concurrent Features work early in the past. Sure, there were some React Labs posts every now and then, but RSC was developed quite differently than say Svelte adding async support. I see both sides.

I wish I had been more vocal in the community about how RSC was being built. I did talk about this at conferences, which had calm, intentional, and thoughtful discussions with members of the React community.

Posting online, I'd get feedback like "you're a shill stop pushing RSC to sell more Vercel". This made me more reserved sharing openly, even though I did believe we were building something amazing for the ***entire*** React community.

RSC in 2025

Now in 2025, we have RSC stable and related features like Actions and Transitions shipped. But putting technology out into the world doesn't mean the community is immediately going to run with it and build an ecosystem on top.

Again, this is likely a place I could have done better with Next.js. Most of the "RSC innovation" happening in the ecosystem was from those building on Next.js. It was still very difficult to build non-Next.js RSC things, largely because RSC is a ***bundler*** feature.

Some savvy developers were able to build their own bundler hacks, or build off the webpack plugin, but there weren't docs and any guidance for how to navigate that journey[1]. Moment of vulnerability here, this was also not my strong suit. I'm not a bundler developer and in fact, part of the reason I liked Next.js was because I never wanted to think about bundlers again. Shoutout to Dan for his great writing on RSC.

The result of this was that it took many years before there was a community-led effort to get RSC in other places besides Next.js, in a ***production-like*** way. There were plenty of experiments and side projects, like Waku and others, but Parcel was the first bundler to give us an abstraction that you could ship to prod with.

Since then, many of the Vite plugin authors building their own RSC implementations collaborated on building an official Vite plugin for RSC. And thanks to hi-ogawa and others, we now have `@vitejs/plugin-rsc`. We also have RSC for React Router. Nature is healing?

Community burnout

One of the most frustrating parts of my previous job working on Next.js was trying to explain the above context to the React community. It was a far easier story for people to say "Vercel is taking over React so they can sell more compute in the cloud". Nuance didn't scale there. Maybe a skill issue on my part.

And I can understand the hesitation, especially when you re-read the list above of other React

frameworks trying to do something similar. As I mentioned, the Next.js and Vercel business model is ***not*** one that normally works.

So eventually, I stopped trying to fight this fight, and then ultimately stopped engaging. You might say: but Lee, there are millions of Next.js developers who are happy building things and never contributing to the discourse. And sure, that is true! I did appreciate those people.

My bigger issue was that other prominent people in the React community never took the time to fully grok what I was saying above. It often felt like I was talking to a brick wall trying to convince others what we were doing at Vercel and Next.js was good for the community and worth their time.

In the fullness of time, I expect this to change. I have confidence RSC will be incredibly impactful and be largely regarded as the third generation of React (if you consider hooks the second). I expect a lot of people who dogged on RSC will eventually come around. But hey, now I get to watch how the story plays out from the sideline.

My call to action for anyone in the React community today, and hoping to grow with it over time, is the following: remember there are humans working on the software, and they want to be treated with kindness and respect just like you. Remember that OSS is a gift and you are not entitled to make demands without putting any energy forward yourself. And above all else, be the optimistic change you want to see in the world.

[1]: While the code was open source, this doesn't mean the average person could figure out how to build RSC things. Hydrogen used RSC + Vite back in 2021, before moving away to Remix.