The Single-Page-App Morality Play

Baldur Bjarnason

Progress in web development is meaningless if nobody can sustain it.

I am Vice

Online debates in web and general software development are a sport. People rarely change their minds, standards are minted in a different room with different people, and it would be unusual for it to affect the decisions made by any of the participants or observers.

I've written in the past about how conferences are the business equivalent of morality plays. They don't exist to educate or enlighten but to unify a community around shared values—which is why they are great for networking. That is why they need a liturgical adversary; somebody who can represent the opposing heterodoxy.

Most online debates have this same underlying purpose. You debate with the intent not to change your mind: you participate specifically to clarify your views through opposition. The counter-arguments of your adversary purify your own and arm you against future attempts to change your mind. When we get dragged into a debate with those who fundamentally disagree with us, we are doing them the favour of standing in for Sin and Vice in their morality play. From my 2014 post:

> By providing a clear demonstration of threatening ideas from the outside, we end up giving the orthodoxy's ideological centre a clearer delineation—reinforcing it. We are Vice, Folly, Death, Prodigality, and Temptation in the morality tale. We have to sound plausible, reasonable, and enticing for the drama to work, but are then parodied and mocked by the context. We exist solely to create an uncertainty that can be assuaged by the characters Mercy, Justice, Temperance, Truth, Virtue, and Tenacity, who bring the viewer back into the fold with convictions even stronger than before. Everybody who sets foot on the stage is a stock character serving a stock role that, one way or another, reinforces what the audience considers normal.

They do it. We do it. When we argue in these debates, we aren't performing as ourselves, but as the personification of virtue or vice, as valued by the clashing communities. The clash clarifies and enforces boundaries. Adversity fosters cohesion.

That these discussions and the opinions that drive them are somewhat arbitrary doesn't make them less true. These community boundaries exist to protect the worldviews and philosophical outlooks of those in the community.

One such debate is whether the 'one true way' in web development—a silver bullet if you will—is to use a Single-Page-App or not. The divide here is ordinarily that of progress versus pragmatism.

Sometimes, those who push for Single-Page-Apps advocate for progress, and see them as enabling new ways of making things for the web—this is the future. Progress. Then those who oppose SPAs worry about how good those apps will end up in practice—that most teams won't be able to do it well. Pragmatism. Sometimes it's the other way around. The Anti crowd sees SPAs as standing in the way of progress on mobile and accessibility. The Pro crowd are performing the role of pragmatically accepting what works best in the industry.

That these arguments are arbitrary doesn't make them less true.

You could shorten these debates substantially as most of them boil down to the following:

> **Pro:** SPAs have more capabilities. They will offer substantially better performance for web apps that get used regularly.
>
> **Anti:** In practice, most SPAs are so bad that a user would have to use them for a thousand days, consecutively, with no breaks and no updates before they'd be as good as even the most mediocre multi-page-app. SPAs are evil.
>
> **Pro:** That's just hyperbole. I'm choosing not to believe you at all. Besides, the tools are great, so it can't be true.

I'm caricaturing both sides here. But this, in essence, is the ground that these debates cover.

The answer here is almost always both:

- It depends
- It doesn't matter as much as you'd think, because the root cause of every bad Single-Page-App isn't that it's a Single-Page-App.

The technology isn't at fault. Our ability to use technology is what's at fault and not just this particular one.

I am Folly

Neither Using nor Avoiding a Single-Page-App is a Silver Bullet.

There is no silver bullet, remember?

The pro-Single-Page-App (SPA) crowd keeps underestimating just how bad the median Single-Page-App is. That's because the Pro crowd underestimates just how bad most *management* is.

The Anti crowd keeps underestimating the benefits of a well-behaved app. That's because they don't appreciate just how much can be accomplished with modern SPA tools and a well-managed team. They mistake the attributes of bad management as essential qualities of Single-Page-Apps and the frameworks that enable them.

SPAs are both amazing and horrible. Sometimes in the same project. The web is large; it contains multitudes.

The thing both the Pros and the Antis have in common? They don't consider management to be a factor in a technical decision when it *very much is*. You need to gauge the quality of a team's management first. Before you make any recommendations as to tools, stack, or technology. Same team, same budget, same goals, with the same scope will have entirely different capabilities depending on the management style. That will change all of your recommendations.

You can dramatically change the *effective* scope and a team's real-world productivity just with different management styles.

That said.

I am Temptation

Generally, the delta between a mediocre Single-Page-App and a mediocre Multi-Page-App (i.e. a traditional server-rendered site) is usually massive, especially if you are targeting mobile devices. Given the same end goal and resources, most teams and managers will do a much better job of reaching their goals if they use a Multi-Page-App instead of a Single-Page-App. You can blame the tools, the browser landscape, developer training, or management, but that's just how it tends to pan out in practice.

I keep seeing Single-Page-Apps with huge JS files that only, in terms of concrete User Experience (UX) benefits, deliver client-side validation of forms plus analytics. Apps rarely leverage the potential of a Single-Page-App. It's still just the same 'click, wait for load' navigation cycle. Same as the one you get with Multi-Page-Apps. Except buggier and with a *much* slower initial loading time.

When you look at performance, cross-platform and mobile support, reliability, and accessibility, nearly every Single-Page-App you can find in the wild is a failure on multiple fronts.

Replacing those with even a mediocre Multi-Page-App is generally going to be a substantial win. You usually see improvements on all of the issues mentioned above. You get the same general UX except with more reliable loading, history management, and loading features—provided by the browser.

You don't even have to go all the way. Hybrid systems, such as Hotwire let you gain some of the benefits of simplification (routing and template rendering is managed by the server) while offering the navigation and UX benefits of a Single-Page-App. Other frameworks, such as SvelteKit, can also offer the same benefit through configuration, provided you can resist the temptation to *change* that configuration and go full-on Single-Page-App.

The problem here *isn't* the Single-Page-App or the hybrids. That's the mistake that both the Pro and Anti crowds make in these debates. The Antis look at the above situation and think the problem is either the SPA or that the SPA tooling is bad. The Pros assume that future app 'goodness' is achievable through iteration and progress.

Neither is true. The problem is that these teams are badly managed and are incapable of matching the scope of their project to the resources they can bring to bear on it.

I am Gluttony

The problem is management.

It is a management problem. Truly.

The Multi-Page-App forces the team to narrow the scope to a level they can handle. It puts a hard limit on their technological aspirations. Mandating a traditional Multi-Page-App under the auspices of performance, accessibility, or Search-Engine-Optimisation is a face-saving way to force the hand of management to be more realistic about what their teams can accomplish. When we can accomplish the same by advocating for a specific Single-Page-App toolkit or framework, that's what most of us nominally on the 'Anti' side do. I regularly advocate for Svelte when I think the team can handle its long term implications in terms of complexity. (That might change as Svelte adds more features.)

The problem with Single-Page-App frameworks, even the ones like SvelteKit who could claim to be more Hybrid than just SPA, is that they are very, *very* eager to enable 'scale' of any sort. Features, app size, code complexity, integrations, etc. They are desperate to make sure that you can keep using their framework if you become a mythical 'unicorn' startup and your project grows into the next Facebook. So they put a lot of hard work into making sure that there is no upper limit to the scope of the app you can make with them.

Which, when they present it as 'scale', sounds like a good thing. But it's absolutely a bad thing when you're in an industry that's as mismanaged as ours. We can't handle complexity. Having no upper limit to it is extremely bad.

This is one of the reasons why the Hotwire set of libraries from Basecamp was* so interesting. Being a hybrid system it offers many of the benefits of both approaches But in addition, their approach means that they have hard upper limits to what you can accomplish with them, which most in the Pro camp see as a huge flaw. But, along with Rails integration, it's a boon for the productivity of most teams. The complexity you get, you can manage, and your *future* complexity is more limited than what you'd get with the more 'traditional' API plus SPA approach. Because of this, there are more specialised small-to-mid-sized web services and apps out there running on Rails than many of us appreciate, ranging from intranets to institutional services, to medium-sized Software-as-a-Service businesses.

(I say ‘was’ because the instability of Basecamp’s management and the massive turnover of the team working on Stimulus and Turbo makes me wary of using Hotwire on a project. But given how popular Rails is, that’s probably me being overcautious. The principles they describe in their books, such as Shape Up, are still sound. The problem, as many have pointed out, was that they didn’t follow through in practice on what they wrote.)*

I am Avarice

Single-Page-Apps can be *amazing*.

You can make a great Single-Page-App with a User Experience that a traditional site will never be able to close to matching. But that rarely happens. This industry is *bad* at what it does. Most apps are mismanaged and under-resourced for their given scope. We need tools that accept the reality of bad management.

I tried to highlight in my Software Crisis 2.0 essay just how bad the situation is for most software development teams. If anything web development has it worse. Most managers can't, or aren't allowed, to set the scope of their work to the limits imposed by available resources. They default to specific Single-Page-App frameworks no matter what the actual needs of the business are and no matter what their developer team capabilities are.

This is the norm. We keep talking about web development under the assumption that every manager is of the type that studies best management practices, reads all of the clever blogs, listens to all of the smart podcasts, and then follows through by putting it all into practice. But those managers aren't all that common. Even the great managers that do exist have to answer to people who favour control over good management, which in tech is almost every executive you can find.

Most managers in or adjacent to web development are, to put it plainly, not good at their jobs or aren't *allowed* to do their jobs well.

We've known what good management looks like from anywhere between the 60s to the early 90s, depending on the country, field, and industry. Software isn't all of a sudden going to get good management after decades of preferring control over effectiveness. We've developed entire software development methodologies whose primary purpose is to reduce the harm done by bad management.

It doesn't work that well.

Bad management is what hinders progress on the web, not 'because the tools for creating MPAs historically let non-expert developers do less damage'.

As developers, we need to operate under the assumption that good management is the exception, not the norm. Multi-Page-Apps and hybrid frameworks let under-resourced, mismanaged developer teams deliver reliable and safe code. That should not be dismissed lightly.

A Multi-Page-App forces your team to reduce the project's scope and concentrate on in-page interactions. They have less state management which most have absolutely no resources to handle. And they're easier to test.

I am Vanity

Why should great teams suffer?

Some teams do have good management. Great teams and good managers do exist. If you're on a team like that, stick with it for as long as you can. A great team will choose the right tool for the job. Sometimes that will be something like SvelteKit or Next.js. Sometimes that will be Lit or uhtml. Sometimes it will be something like Hotwire. Sometimes it'll just be a thin layer of standard JS. Or a thick one. There is no one right tool for every job. Under ideal circumstances, the team will be able to choose what works best, based on their skills, capabilities, timeframe, and resources.

But a team is only great if the circumstances aren't, shall we say, *less* than ideal. Most organisations make ideal circumstances impossible.

It doesn't matter if your developers are great or if your manager is excellent if:

- Higher-level management imposes unrealistic or even impossible business requirements.
- The organisation enforces soul-destroying release deadlines.
- Top-level executives *mandate* specific technical approaches because they are trendy or in fashion or because a friend of theirs said so.
- There are no processes, or the processes keep changing.
- Employee churn undermines long-term (3-6 year) organisation cohesion and memory.
- Someone in the hierarchy is a micro-manager who keeps tampering with everything that crosses their path.
- The organisation allows or even tacitly promotes violence and abuse.
- The organisation has no allowance for recovery after failure or disruption.

If there isn't a single item in this list that applies to your current workplace: congratulations, you are the exception.

The rest of us instead experience these situations:

—We have to have first-rate support for mobile phones, but the front end also has to load a 1MB analytics script. No, we can't skip the analytics on mobile. Make it work.

—We need full SAML integration before the end of next month. Preferably in only two sprints. Can you fit it in with what you're doing? No, I don't have any more details than that. I think it's a standard.

—User privacy is a core organisational value, but we also need to find a way to get all users to opt in to let us share their usage data with our partners.

—The CEO thinks GraphQL is the future. We need to figure out a transition plan for our existing products and clients. We might need to maintain two complete implementations of our API in the short term. Only in the short term, I promise. This won't be a repeat of what happened with MongoDB.

—We need top-notch accessibility, but the founder also wants the entire user interface designed around drag-and-drop. No, we're not scheduling additional time to implement keyboard control. That's a separate feature for the backlog.

—We need these five features for feature parity with a competitor, but the CEO wants the interface to look 'clean.' Can't we hide the buttons and widgets using hovers? Oh, and make the hovers work on mobile somehow.

—Make sure the new hire is never alone in a room with the CTO.

—Try not to touch the credit card processing system. The last person who understood it left programming five years ago and became a waiter. We used to contact him semi-regularly for simple updates, but then he died in a freezer fire in a restaurant in Norway.

—The CEO likes this photo of arctic geese. Can't you fit it on the front page somehow? No, someplace above the fold. The CEO doesn't scroll.

—This sprint didn't go too well but that's understandable under these circumstances. We'll make it up in a later sprint by pushing harder for a short time.

This last example is what ruins a lot of the best teams. It is *the* recurring flaw that I see almost everywhere in the industry (other than the abuse and misogyny, but that's a separate essay entirely). That recurring flaw turns even the best dev team into a bad one:

The team is rarely given any time or resources for recovery or cooling down.

I am Sloth

We rarely get any resources dedicated to individual, project, or team recovery after failure or mistakes.

Without recovery, the productivity of most teams suffers and will continue to suffer until you give them time to recover.

This was well illustrated in the following Twitter thread by John Cutler:

> Met a CTO recently who did the right (but so hard) thing. He walked into work and sent out an email: "Today, we are deleting all jira tickets. Stop working. Please use this time for personal education. You have my full support" Why?

Go read the thread. The CTO gave their team months of recovery time and afterwards, they tripled their flow of work.

They tripled the resources they could bring to bear on any given software project by giving themselves time to recover.

Or, put another way: the team was running at a third of its ability and productivity because it didn't have any cool-down periods.

Without recovery, their productivity would have continued to decline until the project failed. No recovery leads to a death cycle.

Basecamp's Shape Up, which is still a decent overview of good single-team practices in software development, describes this as *The Cool-Down* (inventive, no?), which is baked into their process:

> If we were to run six-week cycles back to back, there wouldn't be any time to breathe and think about what's next. The end of a cycle is the worst time to meet and plan because everybody is too busy finishing projects and making last-minute decisions to ship on time.
>
> Therefore, after each six-week cycle, we schedule two weeks for cool-down. This is a period with no scheduled work where we can breathe, meet as needed, and consider what to do next.
>
> During cool-down, programmers and designers on project teams are free to work on whatever they want. After working hard to ship their six-week projects, they enjoy having time that's under

> their control. They use it to fix bugs, explore new ideas, or try out new technical possibilities.

Unfortunately, a standard cool-down period is rare in this industry. You can tell from the software we ship.

Every great team is just one inevitable failed sprint away from a death cycle. Most teams overestimate their long-term resources because of this. They set the scope of their projects well beyond what they can handle long term.

In the long run, all our teams are failed teams. It shouldn't be that way, but that's the reality of our industry.

I am Tenacity

If the morality plays of old had only consisted of an Everyman character, surrounded by the various personifications of sin and vice, doing their best to withstand temptation, they wouldn't have done much to serve their purpose. The goal was to demonstrate the values society deemed positive by contrasting them with their opposite so the Everyman was surrounded and supported by the personifications of the virtues of the day. Often those were the seven virtues that opposed the seven vices. Sometimes they were core values of the churches of the day, such as Mercy or Penance. A slightly less common one was Tenacity who is one of the virtues we need for the problems that plague the web.

Because we've been dealing with these issues for decades. The situation has improved ever so slightly. It's much easier to find blogs, communities, and books that help you learn and discover what good management should look like. Practicing it is *hard* but the information is more accessible now than ever. Making it happen is the tricky bit. Most organisations, especially in tech, are geared towards control and observation over autonomy and productivity. The modern open office is just the latest implementation of Foucault's variation of the Panopticon: constant observation as a form of employee control. This is the reason why, even though study after study on open offices shows a detrimental impact on productivity, creativity, and health, none of it makes even a dent in their adoption. This is also why we're witnessing a substantial backlash against remote work among managers and executives.

Those of us who aren't executives can only create pockets of good management here and there, either in single teams or smaller businesses, and protect them.

Once we have those pockets, then we can start to talk properly about what technology to use best for a specific problem.

Summary

Single-Page-Apps can be fantastic. Most teams will mess them up because most teams operate in dysfunctional organisations. Multi-Page-Apps can also be fantastic, both in highly functional organisations that can apply them when and where they are appropriate and in dysfunctional ones, as they enforce a limit to project scope. It helps to create and popularise tools that limit project complexity, such as Rails and Hotwire or hybrid frameworks. It helps to try and create pockets of well-managed teams. It helps to just understand that the reason why Single-Page-Apps or Hybrid Apps suck isn't that they suck as a concept. Technology implemented by a dysfunctional organisation is almost always going to suck.

I have some doubts about whether hybrid frameworks will end up being useful complexity-limiting tools or not in the long term. The framework drive towards complexity is strong. Many of them go out of their way to enable a never-ending escalation of features and scope. Most teams can't handle an escalation like that over short or long periods.

Until we accept that most of the industry is poorly managed and try to figure out how to solve *that* problem, sites and web apps won't get better, just shinier and with more bounce.

The biggest hindrance to the web's progress isn't non-expert developers, tooling, libraries, Single-Page-Apps, or Multi-Page-Apps. It's bad management.

We've tried fixing management for decades. It hasn't worked so far except in a few lucky corners of the industry. We don't need tools that provide better Developer Experiences (DX). We need tools that mitigate bad management.

Go write a thousand-package-dependency npm-installed CLI that solves *that* problem and I'll happily install it.