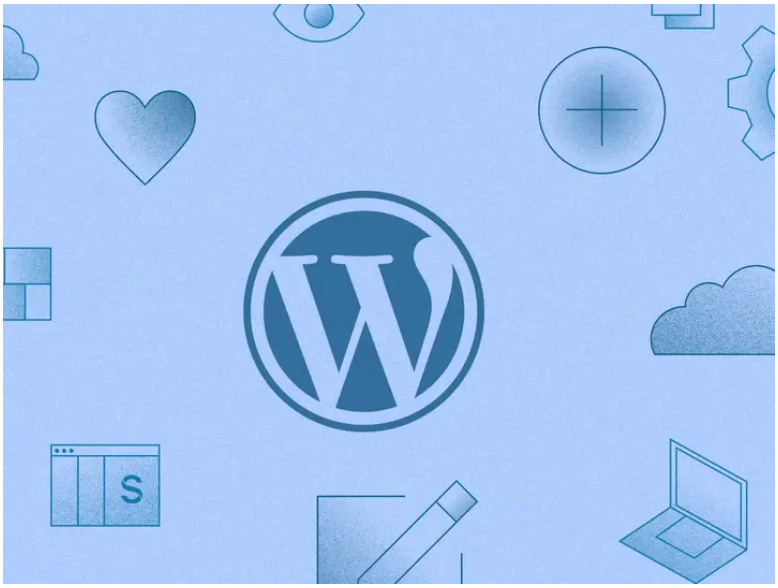
PUBLISHED MARCH 16TH 2023

5 Disadvantages Of Wordpress That Are Holding You Back

Your content is too important to leave in a system designed to manage blogs over twenty years ago. Here are 5 major disadvantages of WordPress as a content solution in the composable era.

Knut Melvær
Principal Developer Marketing Manager



You should think twice before adopting WordPress as the main platform for your content. I'm sure it isn't surprising to get this advice from me—after all, I work for Sanity, a competing content platform.

But my reasons for recommending against WordPress are actually based on my personal experience. The reason I got enthusiastic about Sanity in the first place was that after 15 years of working around all the limitations of WordPress to solve my clients' digital challenges, I finally found something that made much more sense and allowed me to do so much more.

Many of us started our careers with WordPress; it deserves its honorable place in web publishing history. But history is also where it should stay — at least if you are managing applications beyond a single website with only a few pages. Your content is far too important to leave it in a system designed for managing blogs over two decades ago.

In this post, I connect the dots between the technical foundations of WordPress and their strategic implications for how you can (or can't) make the most of your content. I hope you'll find inspiration, whether you're currently looking to move on from WordPress or not.

1. WordPress is not built for the modern, composable era

Let's start with the obvious: Today's customer and user journeys are digital. They span different devices and domains, outside and inside of authenticated surfaces. They can be personalized, localized, dynamic...you name it. We aren't just publishing static marketing brochures on the web as we did in the 90s and early 00s. We aren't just building websites anymore; we're building *digital experiences*.

That's why so many organizations are adopting a composable approach to technology. It's near impossible for one solution to adequately meet all of your complex requirements while being flexible enough to be tailored to your unique needs.

Instead of a one-size-fits-none solution, we're building our tech stacks with more specialized tools to help us advertise, market, sell, provide support, and deliver the product and services themselves. Modern digital tools need to have APIs and integrations. These systems can be combined in different ways to solve individual problems better.

Expectations for tools have changed

Over the past several years, there's been a paradigm shift in how we build modern web experiences—and WordPress hasn't caught up. Today, more and more developers want to work with serverless environments and composable frameworks.

They expect to be able to ship faster and more reliably thanks to tools that enable continuous delivery and integration (CD/CI). They want to leverage specialized solutions to solve complex problems like global scalability.

These services and APIs provide abstractions that allow your developers to focus on the technical challenges specific to your goals and business and drive value. Furthermore, developers today seek a great developer experience. They want to enjoy and feel good about the technology they're using (doesn't everyone?).

Layering APIs and plugins don't fix underlying problems

One big problem with WordPress is that the self-hosted version relies on legacy technology. It's built to be run on a traditional web server on top of a relational database.

That means your developers must spend their time working around limitations and maintaining a vast amount of plugins to get the necessary functionality. Even WordPress developers will tell you that although it's a small task to update plugins, the risks are large: the site could go down, and you'd need to spend time on laborious recovery.

Yes, you can put an API on top of WordPress. Still, these implementations deliver content in a way that makes it hard to integrate—especially compared to other, more modern content platforms. Ultimately, this means that your team spends precious time and energy making WordPress work for *them* instead of the other way around.

Here's an example: GraphQL has become a popular way for developers to integrate content into digital experiences when they build with modern frameworks. So, while the GraphQL plugin for WordPress is an impressive feat of engineering, it's still severely limited in how you can query your content. In contrast, Sanity was designed to enable you to query your content however you'd like.

2. WordPress is open-source but not free

WordPress markets itself heavily on being open source. But does that mean it's free (both in terms of cost and time you must invest)?

Not exactly. You can indeed access and alter WordPress source code to use it for (almost) anything you want. However, to follow upgrades, you'll also need to keep your own "fork" compatible and ensure your customizations don't break, adding maintenance.

And it's also true that you can install WordPress on any compatible web server without paying licensing fees to its parent company, Automattic. But then *you* are responsible for putting in the work of installing WordPress on the web server and dealing with all the implications.

Think about it: in most cases, when hosting WordPress, you take on responsibility for maintaining not only the core software but also its database, server environment, and caching layer to make sure your site is fast and performant. Someone has to invest the time and money to configure, scale, and maintain this infrastructure.

So, what about specialized WordPress hosting environments? These services are still locked into a restrictive model based around a specific website instead of giving you the freedom to innovate and experiment across multiple domains.

Take multi-tenant/sites. With Sanity, you pay for API usage, but you can consume the APIs from as many sites and services as you like. Modern hosting platforms like Vercel, Netlify, and Amplify also have usage-based pricing, which means that you can get started with a multi-site setup for free, and the cost will be driven by the different sites' scale— not the number of them.

Functionality is commodified

Many people will bring up WordPress' extendability through plugins. If you want to go beyond a simple blog, you'll need to install plugins like Jetpack, Yoast SEO, Imagify, WP Rocket, and Advanced Custom Fields. Or you can install page-builder applications like Elementor and Divi that replace the WordPress content creation tooling. These plugins are commercialized, so you must pay different vendors for functionality.

So, how is this different from composable architectures? With WordPress, you pay for plugins locked to a single CMS and a specific number of sites. With a composable stack, you pay for APIs that you can use across many applications and platforms, lending you the flexibility you need in the composable era.

3. WordPress treats your content like it's 2003

In the composable era, content is multi-channel, multi-context, and multi-experience. In most cases, content is not just flowing to a single page on a single website. Think about how an e-commerce product description needs to be used not only on a product page but across all marketing channels, personalized carts and order pages, support documentation, and so on. You need to be able to easily reuse content without copy-pasting it into different systems. And you need good content workflows and integrations where your content can seamlessly flow between different services and applications without risking unintentionally overwriting people's work due to race conditions and creating errors.

Content is locked in

But WordPress locks your content in, making it hard to reuse, repurpose, or move to a different platform. This stems from technological decisions made over 20 years ago:

- Your content is stored in a relational database, which is akin to storing all your content in a spreadsheet with only rows and columns, limiting your content modeling opportunities
- The data model hasn't been changed, so you will find that *all* your content—regardless of how custom it is—is stored in the table for "posts"
- There's no standardization for how plugin content is supposed to be stored and structured, leading to all sorts of idiosyncracies that make it painful to redesign and migrate
- The content that you put into "posts" and "pages" is stored in the database as HTML, which makes it hard to process programmatically and integrate into anything beyond a web page

No flexibility for different presentations

WordPress originally caught on among content creators because it gave you the power to publish blog posts without writing (that much) HTML. You could type your blog post in a basic text editor, hit Publish, and see it live on the web. Later, you could also preview the content to see how it would look before sharing it with the world.

With the launch of WordPress' What-You-See-Is-What-You-Get (WYSIWYG) block editor, Gutenberg, the core product took strides into becoming the *page builder* people had started using it as. This means that the UI gives you more control over how content is visually presented on a website. (Interestingly, the introduction in 2018 didn't go well: official numbers say only 6% of WordPress sites use the block editor today.)

This model works great for articles that only appear in a single place. But it starts to break down when the content needs to be reused across different contexts, channels, and presentations. Updating the design of your website using only WordPress theming capabilities gets harder if you use Gutenberg or other visual plugin additions like Elementor because your content is littered with design-specific markup. And someone needs to spend time cleaning it up if you want to present it in a different place or design.

4. WordPress doesn't enable collaboration...

Developing, building, maintaining, and improving digital experiences requires constant collaboration between different functions and teammates. To enable this, tools should fit seamlessly into modern product development practices and decrease friction between teams through integrations.

WordPress was never designed to do any of this. As I mentioned, it was built on top of a simple database model that requires document locking to integrate with other services. The UI is also designed for single authors. It's not impossible to do WordPress work at scale, but it comes with cost and friction because you're working against its design.

...between content creators

Have you ever used modern collaboration tools like Google Docs, Notion, Figma, or Miro? If so, you're likely used to working on content in the same space, at the same time, with multiple other people.

In WordPress, collaborative editing isn't possible. If someone else is editing content, you aren't able to edit that post or page without taking over access. This is called "document locking," a user experience hack to prevent unintentionally overwriting someone else's work.

Furthermore, WordPress' revision system is very rudimentary: You can't roll back individual changes, you don't get nuanced attribution to who changed what, and it forces content creators to read and compare HTML code to see what's changed. This becomes unreasonably hard when the HTML is littered with specific code for presentation and layout.

...between content creators and developers

When content creators feel frustrated by needing to go through developer teams to get stuff shipped, it often points to a larger problem: the developer experience of the tool is poor, and so developers are reluctant to say yes to requests because the solution would involve hacks or workarounds.

Conversely, when technology prioritizes great developer experience, teams are likelier to say "yes, of course!" to requests and ship them quickly. Sadly, WordPress' developer experience hasn't kept up with the developments we have seen in the last 10 years.

Many content creators like WordPress because it allows them to add HTML snippets and embeds to their content. It's understandable that feels empowering because it allows you to quickly do things that seem simple on the surface, like adding a contact form, independently.

This power comes with great responsibility, though. There are countless cases where this capability has led to design breaking, the introduction of 3rd-party JavaScript that impacts SEO and performance negatively, and other user experience issues. It's not reasonable to expect that content creators will be thinking about all of these implications, and—unlike when developers ship code—these changes are not peer reviewed and do not use version control.

...between designers, content creators, and developers

Similar concerns arise regarding how content and code work with the design implementation. Many modern organizations with needs beyond a simple marketing site have invested in design systems and component libraries to make building new presentations and surfaces easier while providing consistent user experiences and staying on brand.

This design approach maps well with developer frameworks that are built around the idea of composing components and integrating the content from APIs. React, Vue, Next.js, Angular, and others are popular web technologies built to enable developers to reuse visual components in and between web apps.

These frameworks have something in common: they work best with content structured as data. When the underlying structure is HTML—the format that WordPress saves its content in—it's much more likely that arbitrary code will be introduced, making it harder to integrate with these other systems.

More design challenges come from the Gutenberg block editor. If someone makes design changes—for example, padding in pixel values, colors, or how a module looks—it can introduce errors and inconsistencies in other places the content appears. Component systems, on the other hand, allow content to be accessed from different devices, with different needs for accessibility, brand and style guides, and the many other things that make design a discipline.

...between content creators and integrations

Real-time collaboration isn't just important between teams and colleagues. It's essential if you want any kind of automation and content enhancement.

In the past few years, we've seen services and integrations that let you automate content operations. The onset of AI-powered content services will only make this more relevant. To support this, you need content platforms that allow you to update content programmatically without unintentionally writing over someone's work or locking them out of documents.

5. WordPress is familiar, but that doesn't mean it's user-friendly

People want to work with tools that make them comfortable and productive. And change always feels uncomfortable at first.

But we're quick to embrace tools that let us respond better (and more pleasurably) to new circumstances. Not long ago, many of us preferred to have most meetings in a room at the office; now, we recognize the upside of having more flexibility with Zoom meetings. It's the same in the composable era: we're becoming accustomed to specialized tools that integrate with other systems.

WordPress makes many things possible, but that doesn't mean it's easy in the short term or optimal in the long term. Developers are usually reluctant to implement workarounds, making them more likely to say "no" to stakeholders' requests.

This post is mainly focused on technology choices, but it's important to point out that how you introduce and onboard teams to the technology you choose will be key to everyone's success. You have to understand that the change will hurt a little

while making sure that they feel empowered as early as possible. Fortunately, (good) modern tools are often designed to provide an early sense of accomplishment.

To modernize, you may also have to change how you've worked. And you need your team to be open to the fact that they may be using inefficient workflows simply because they're familiar with them.

Moving on from WordPress

It might not always be an easy sell to suggest that your company move from something familiar into something unknown that—on the face of it—seems complex.

But complexity is unavoidable, so it's really about solving for the complexity that brings you more value sooner and sets you up for the future instead of continuing to deal with increasing complexity that will only hold you back from a tool that has outlived its purpose.

If you want to build a digital experience beyond publishing a single website with a blog and some simple web pages, you should move on from WordPress. If you want to attract web development talent, take advantage of the many technological innovations of the last decade, and be better prepared for the next ones, you should look into doing it with a composable stack.

I hope I articulated some of the shortcomings you might have felt with WordPress or made you think about it differently. I also expect some to disagree and take issue with the points I've made above. Any reaction you might have, I would love to hear from you!