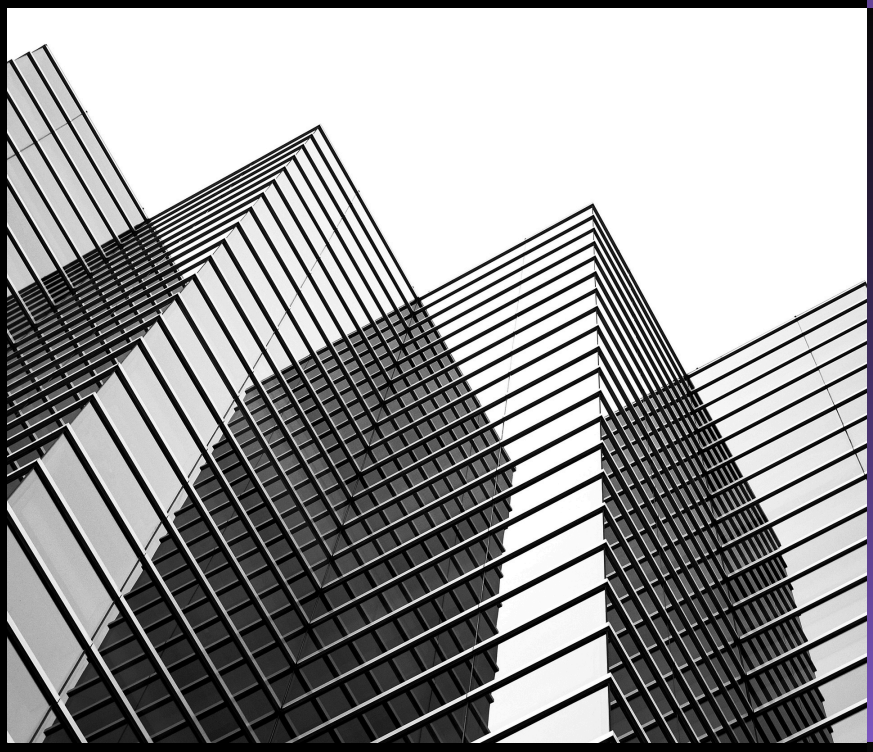


2024

Middleware

State of Open Source DORA Report

middlewarehq.com

Table of Contents

Open Source Landscape

Why DORA Metrics?

DORA metrics are arguably the gold standard for measuring software development and delivery performance.

DORA metrics show how efficiently teams deliver value to users. However, unlike private enterprises, open-source projects rely on global contributors. DORA metrics help measure how well these distributed teams collaborate.
Insights from DORA metrics enable repositories to improve processes, creating a continuous cycle of improvement & process level feedback.

A Snapshot

The report examines 41 repositories spanning a range of projects, including:
- Programming Languages (e.g., Rust, Julia, Carbon Language)
- Developer Tools (e.g., Microsoft VSCode, Jenkins, Kubernetes)
- Web Frameworks (e.g., Django, Svelte, Next.js)

Despite their differences, all repositories face similar challenges in managing bug fixes, feature development, documentation, and technical debt. This report explores how these factors influence core DORA metrics and repository efficiency.

Key Findings at a Glance

- Bug fixes account for over 50% of the workload in many repositories, pointing towards their critical role in maintaining software quality.
- While repositories like Microsoft VSCode excel with cycle times below 3 hours, others, such as NodeJS, experience delays exceeding 70 hours.
- High levels of technical debt correlate with longer cycle and merge times, emphasizing the need for regular refactoring and codebase optimization.
- Repositories with high maintenance activity maintain steady pull request counts despite longer cycle times.

Methodology

At Middleware, we're passionate about helping engineering teams make data-driven decisions, and we believe that starts with accessibility and transparency. That's why we developed Middleware Open Source, a completely free and open-source tool designed to help teams measure and analyze their DORA metrics.

Whether you're part of an enterprise organization or a small open-source community, Middleware Open Source provides the tools you need to track and improve your DevOps performance. If you're curious, you can explore the tool yourself.

Using Middleware Open Source, we gathered detailed data from merged PRs of top Open Source repositories. But we didn't stop there.

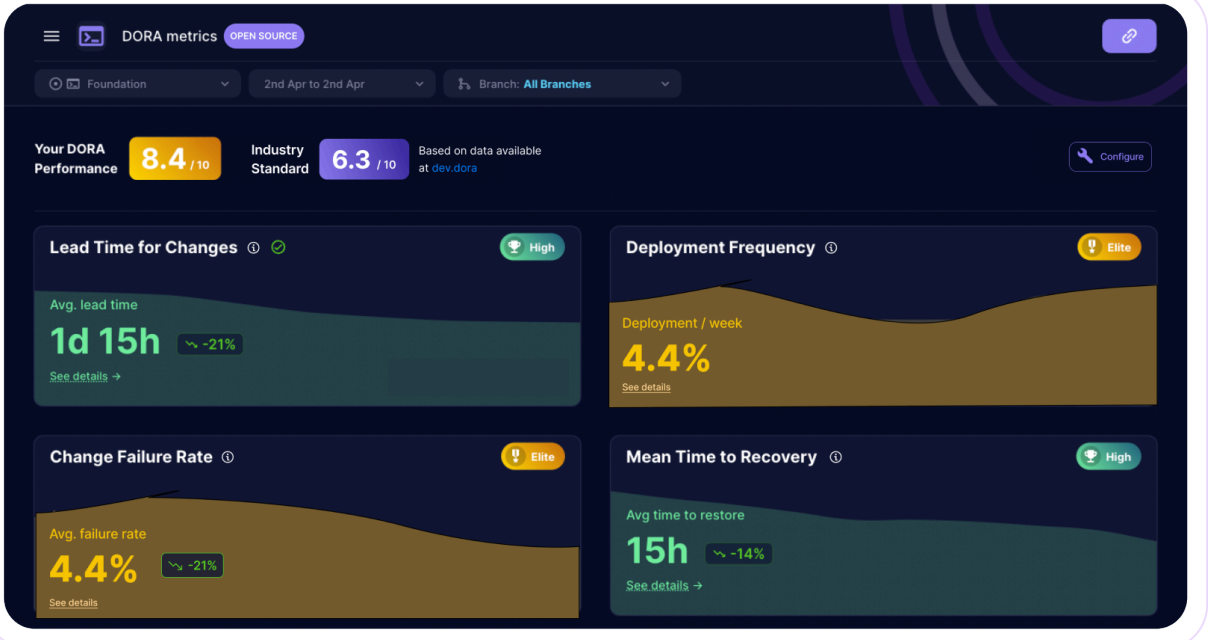
We also took a closer look at the nature of work happening within these repositories. By analyzing pull request data, we leveraged AI to categorize contributions across six key areas: bug fixes, features, documentation, tech debt, maintenance, and dependency changes.

This provided a nuanced understanding of how different types of work influence core DORA metrics and overall productivity.

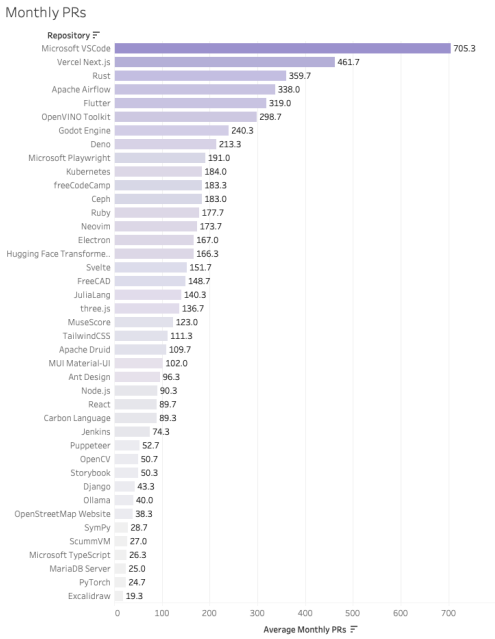
After collecting and organizing this data, we sat down on our desks and built an analysis to discover meaningful patterns and actionable insights.

The goal wasn't just to report numbers. It was to tell a story about how open-source communities are performing, what they're prioritizing, and how their work impacts their speed and reliability.

By leveraging an open-source tool like Middleware, this analysis is something anyone can replicate, customize, and build upon for their own needs. The findings in this report aim to provide actionable insights and spark conversations about what software delivery looks like in the open-source world.



Bird's Eye View



Who is setting the bar?

First Response Time

The fastest responders? SymPy and OpenStreetMap Website. SymPy clocks in at an almost-instantaneous 0.62 hours. VSCode, unsurprisingly, isn't far behind with just under 2 hours.

Merge Time

The MVPs here are Node.js and Ceph, taking less than an hour on average to merge pull requests.

Cycle Time

VSCode stands out again with an unbelievably low cycle time of just 2.35 hours.

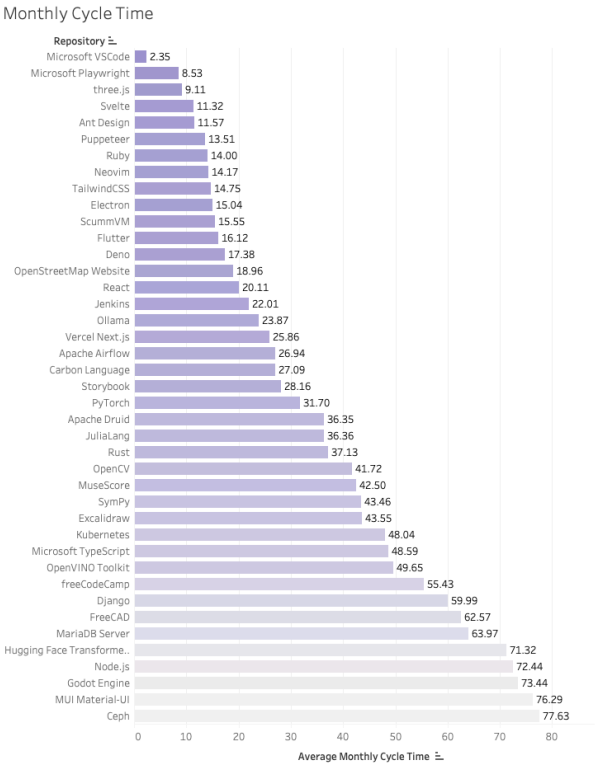
Rework Time

Interestingly, some repos stand at a 0 rework time, possibly indicating a squash and merge.

Where Things Get Sluggish?

First Response Time

On the other end of the spectrum, MariaDB Server struggles, with contributors waiting over 43 hours on average for a response.



Merge Time

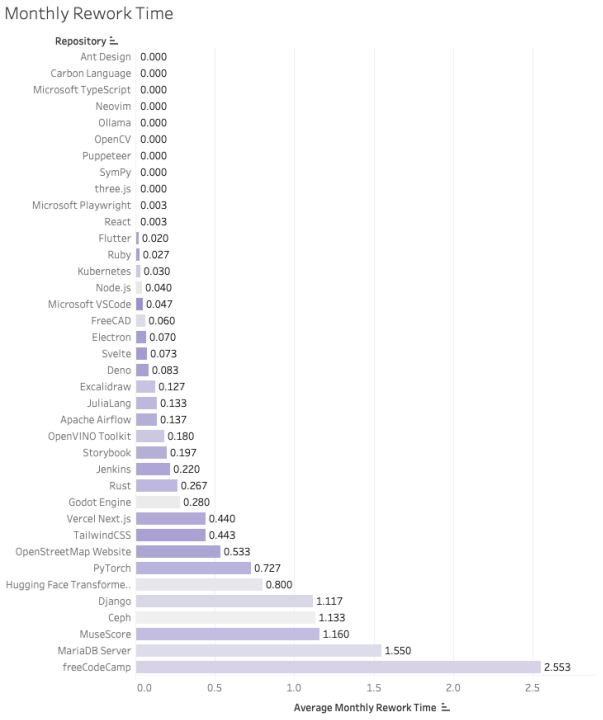
Repositories like React and JuliLang are taking their sweet time with merges - averaging almost 7 hours. While this might reflect careful reviews, it can also signal bottlenecks that slow down progress.

Cycle Time

Ceph and MUI Material-UI have cycle times north of 75 hours. That's not ideal, especially when compared to standout performers like VSCode.

Zero Rework Time?

The near-zero rework times in some repositories might seem like the definition of exceptional code quality and documentation, but a deeper look might reveal a more nuanced story.



Here are a few possible explanations:

Squashing
Achieving zero or close-to-zero rework in open source is rare. The likelihood of consistently high-quality PRs across the board is slim. What's more probable is squash merging, a practice that consolidates changes into a single commit, effectively eliminating traces of rework metrics.

New Contributors
Rework often correlates with the proportion of external contributors in a project. Repositories that encourage participation from newcomers, such as freeCodeCamp, naturally show higher rework times due to the learning curve and adjustments new contributors need to make.

Technical Limitations in Measuring Rework

A key technical limitation in measuring rework time lies in the review process itself. If reviewers leave comments without formally requesting changes via the "request changes" feature, these revisions aren't captured in rework calculations. This means rework might still be happening - it's just not being tracked by the metric.

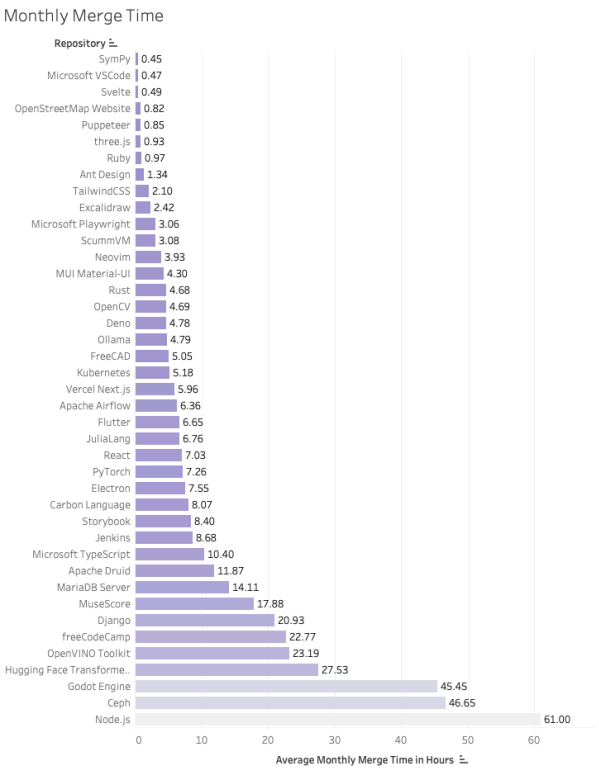
The Story Behind Hardcore Repos

Repositories with little to no rework often belong to technically hardcore domains, where only experienced developers contribute. These projects are less likely to see contributions from casual or new developers, resulting in fewer iterations and changes during the review process.

The Takeaway

While near-zero rework times can signify efficiency, they are also influenced by squash merging practices, contributor demographics, and measurement limitations.

It's an interesting mix of technical processes and community dynamics, showing that what's seen in the data sometimes isn't always the full story.



A Few Things We Didn't Expect

Microsoft VSCode's PR Volume
It's one thing to handle 700 pull requests per month, but to also maintain leading performance across all timing metrics?

MariaDB Server's High First Response Time, Yet Low Merge Time
This is an odd one. While MariaDB Server takes forever to acknowledge contributions, it moves quickly once the PR is active.

Repositories with Low PR Volume but High First Response Times
Some repos, like MariaDB Server, have a lower volume of monthly PRs but have longer first response times. This could imply that even though contributions are sparse, maintainers are not immediately addressing them, perhaps due to resource constraints or prioritization of other work.

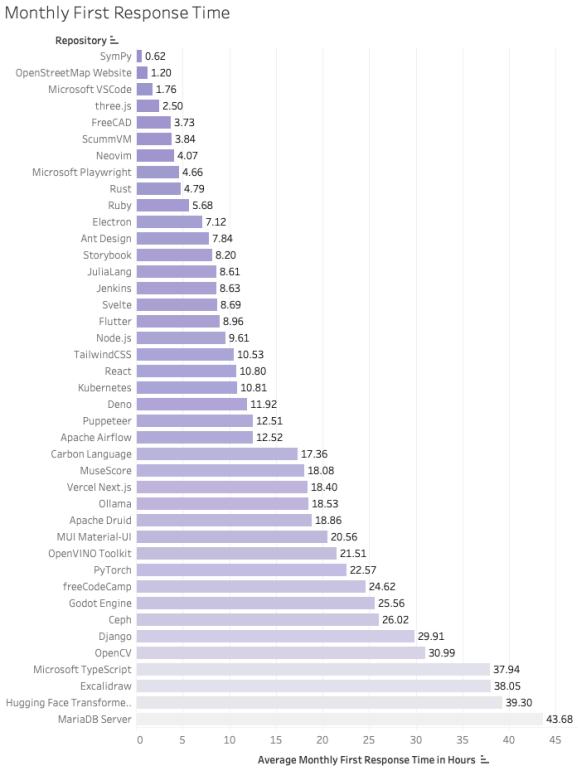
You'd expect repositories with lower PR volumes to handle contributions more quickly. might highlight organizational challenges or misaligned priorities.

The Big Picture

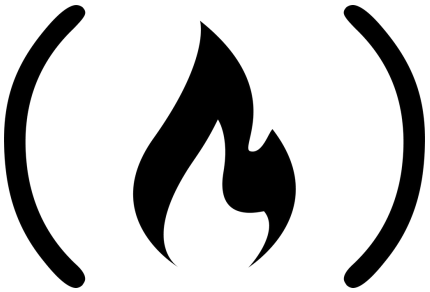
While some repositories demonstrate exceptional performance in specific areas, others face challenges that might surprise you. For instance, SymPy stands out with one of the fastest response and merge times. On the other hand, repositories like MariaDB Server exhibit slower response times despite having fewer PRs, suggesting potential bottlenecks in resource allocation or internal processes.

It's also interesting to see how repositories with high volumes of PRs, such as VSCode and Vercel Next.js, manage to strike a balance between responsiveness and throughput. Their ability to handle hundreds of contributions monthly while maintaining competitive cycle times was quite interesting for us.

As we dive deeper into the report, we'll explore the metrics in greater detail, uncovering insights into how nature of work, community dynamics, and operational practices influence repository performance.



Repository Spotlight



VS Code

VSCode stands out for its incredible efficiency, with the lowest average monthly cycle time of just 2.35 hours, reflecting a highly optimized development workflow.

Its ability to handle a high volume of PRs with minimal delays highlights exceptional project management and technical maturity.

Average Monthly PRs

705.3

Average Monthly Cycle Time

2.35 Hours

freeCodeCamp

FreeCodeCamp is a great community-driven repository, actively encouraging contributions from new developers.

What's impressive is that despite the influx of contributions from varying experience levels, FreeCodeCamp maintains a remarkably low rework time of just 2.55 hours. (even though in all our repos they are at the last position)

Average Monthly PRs

183.3

Average Monthly Rework Time

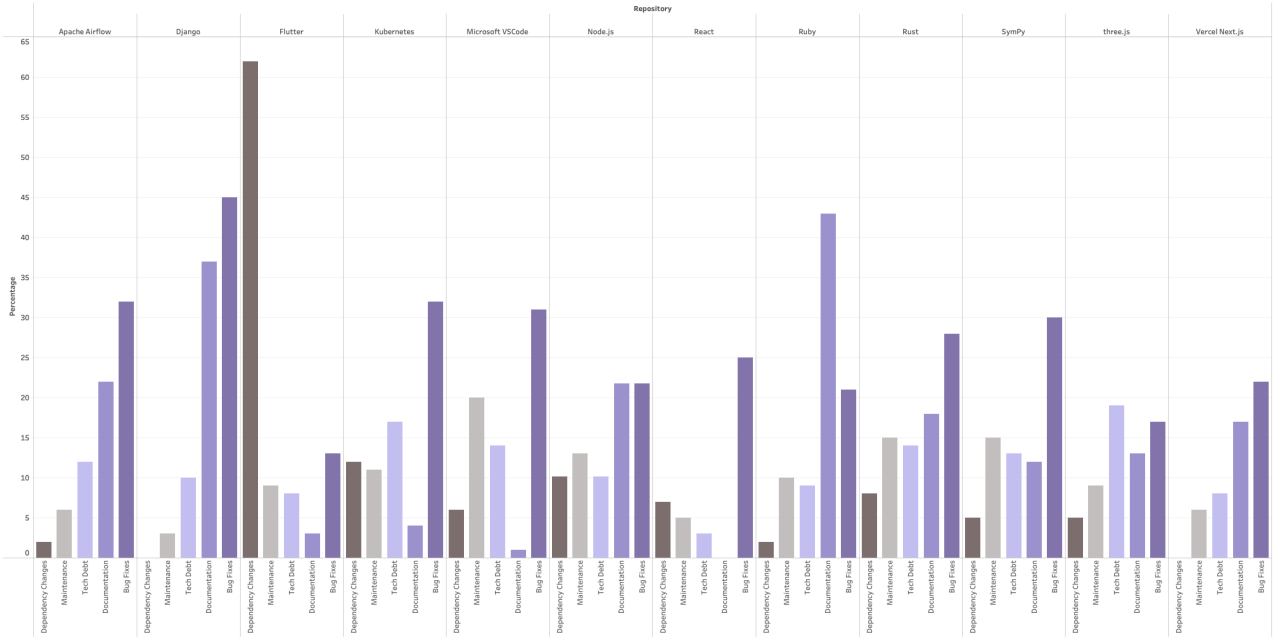
2.55 Hours

Nature of Work

Overview

Every repository tells a story of priorities and purpose, reflected in the type of work being done. From documentation improvements to bug fixes, each PR type contributes to the repository's health, usability, and long-term success. This section provides a snapshot of how open-source repositories distribute their work and what that reveals about their goals.

We leveraged AI to help us categorize the PRs into distinct subsets under nature of work.



Interesting Observations

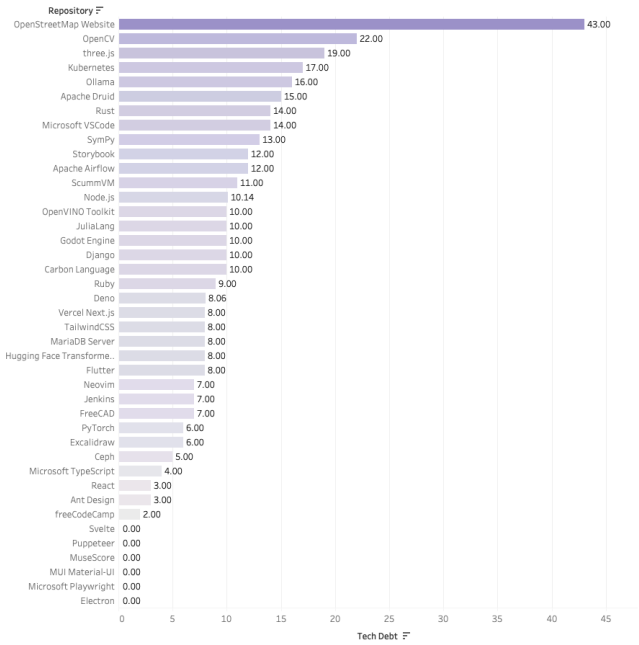
- Flutter leads with 62% of PRs focused on Dependency Changes, possibly indicating a strong focus on staying up-to-date with ecosystem dependencies.
- Ruby stands out with 43% of PRs focused on documentation, demonstrating its commitment to developer-friendliness.
- VSCode and Kubernetes show a balanced distribution across Maintenance and Tech Debt PRs that ensure long term stability.
- Excalidraw prioritizes stability with 58% of PRs dedicated to bug fixes.
- Microsoft Playwright is a leader in maintenance with 69% of PRs focused on long-term operational health.
- Svelte's 71% bug fix PRs highlight its commitment to a seamless developer experience.
- Node.js balances innovation and contributor experience with 23% of PRs for features and 21.74% for documentation.

Tech Debt vs Timing Metrics

Tech debt is an inevitable part of software development. It often represents the trade-offs teams make to meet tight deadlines, tackle pressing challenges, or enable rapid scaling.

In many cases, it's a deliberate choice - a conscious decision to prioritize short-term gains over long-term technical soundness. But tech debt doesn't exist in isolation.

It has far-reaching implications, shaping not only the underlying code but also the speed, efficiency, and sustainability of the development process.



In open source projects, where collaboration spans a diverse and often distributed community, the effects of tech debt can ripple even further. The real question becomes: how does tech debt influence the key workflows that keep open source projects thriving?

By analyzing tech debt alongside timing metrics like cycle time, first response time, merge time, and rework time, we aim to uncover the deeper connections between these factors.

This analysis is driven by a simple but powerful goal: **to bring greater clarity to the hidden dynamics of tech debt and its operational impact.**

Through this lens, we're exploring critical questions such as:

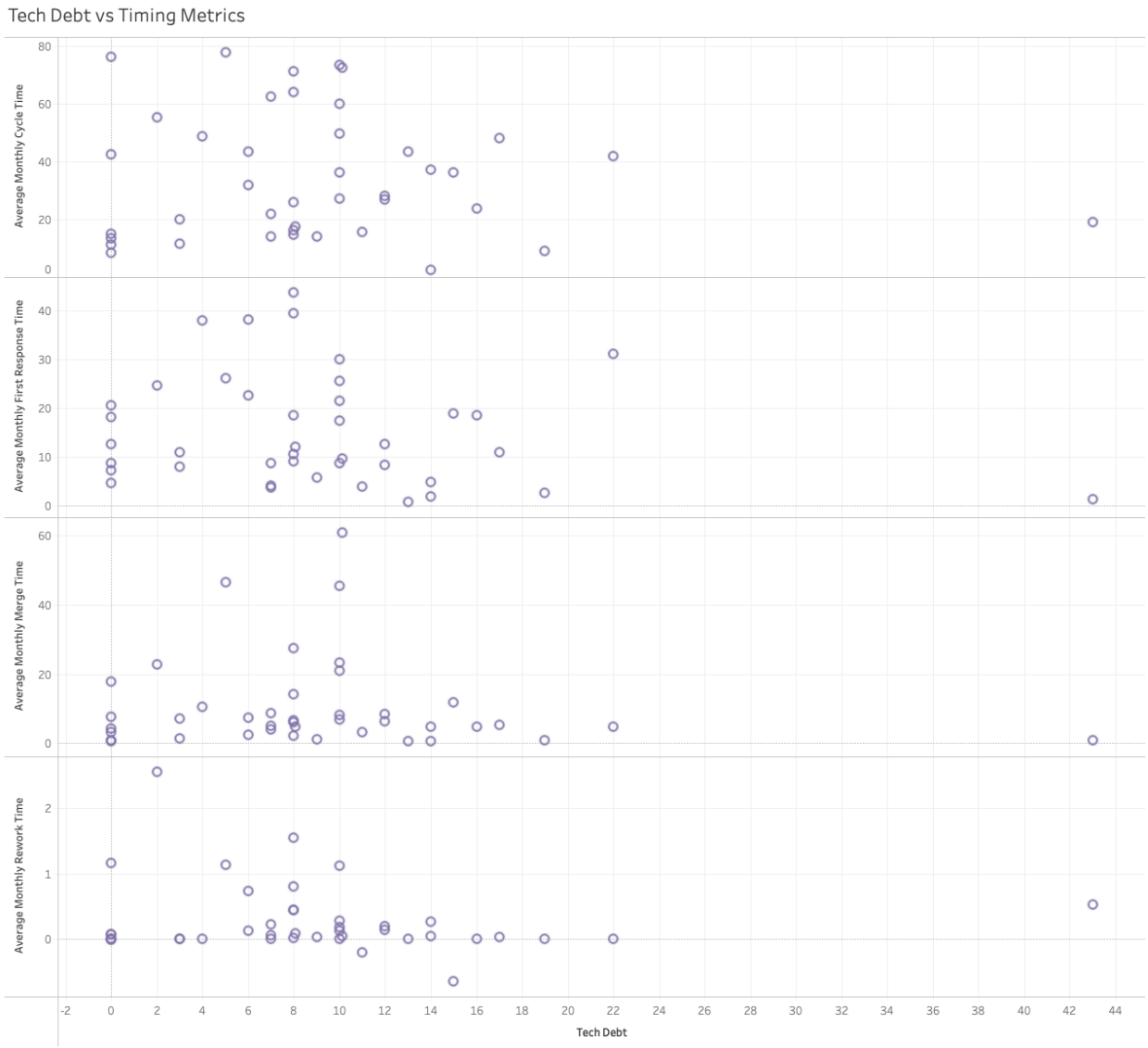
- Is tech debt creating bottlenecks that slow down merges?
- Does it lead to delayed responses or increased rework, particularly for contributors?
- Are repositories with high tech debt trading off speed for sustainability, or vice versa?

Our motivation is not just to answer these questions but to empower open source maintainers and contributors with actionable insights.

By understanding how tech debt interacts with timing metrics, they can prioritize their efforts more effectively.

Ultimately, this analysis seeks to strike a balance. It's not about eliminating tech debt altogether (which is rarely realistic) but about managing it in ways that minimize its negative impact while maintaining high performance.

What we found?



- **Cycle Time**: While tech debt is often assumed to prolong cycle times, our analysis shows no clear correlation between the two. Repositories with high tech debt sometimes manage their cycles efficiently, and vice versa, highlighting the potential influence of other factors like team structure and workflow prioritization.

- **First Response Time**: Repositories with varying levels of tech debt demonstrated no noticeable trends in first response time. This suggests that tech debt may not directly affect responsiveness to new pull requests, with team practices and contributor engagement likely playing a larger role.

- **Merge Time**: Despite expectations, the relationship between tech debt and merge time was also inconsistent. Repositories with both low and high tech debt showed wide variations in how quickly pull requests are merged, indicating that operational factors might outweigh tech debt in this area.

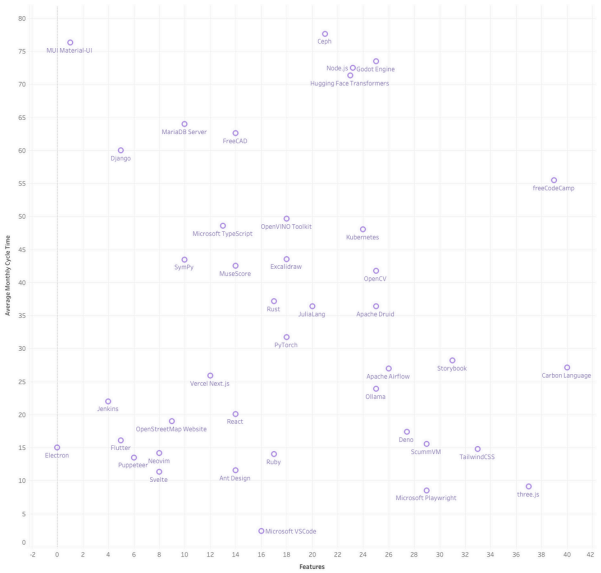
- **Rework Time**: Tech debt and rework time exhibited no strong correlation either. This finding points to the possibility that rework might be influenced by code review quality, contributor experience, and repository-specific workflows than by tech debt levels.

Features vs Cycle Time

Trade-offs of focusing on features?

Features are the lifeblood of any product - they're what make users excited and keep software relevant. But building features isn't always straightforward. It often involves complexities like planning, coding, testing, and feedback loops, all of which can stretch the time it takes to get those shiny new updates shipped.

We looked at the data to see how the focus on feature work correlates with cycle time. The goal? To figure out if working on more features slows open source teams down, and if so, by how much.



What stood out?

Feature Work Often Comes With Longer Timelines

- As the percentage of work dedicated to features increases, so does the average cycle time.
- Makes sense, right? Features typically require more planning and cross-functional collaboration compared to, say, fixing bugs or tweaking documentation.
- Teams that focus heavily on feature work need to be mindful of the trade-offs. More features might mean happier users, but longer cycle times could delay releases or hurt agility.
- A few repositories stood out as outliers - those with moderate feature percentages (~20–30%) but much longer cycle times than expected. This might point to bottlenecks like inefficient workflows or dependencies.
- On the flip side, repos with a smaller focus on features (think ~5–15%) generally had shorter cycle times, likely because they're prioritizing smaller, maintenance-focused tasks.

Why This Matters

Feature work is exciting - it's where the magic happens. But longer cycle times can have ripple effects, from delayed releases to frustrated stakeholders. By understanding this relationship, maintainers and teams can ask the right questions:

- Are longer cycle times okay for the impact these features bring?
- Can workflows be optimized to keep things moving faster?
- Are there bottlenecks slowing down progress that need attention?

Take a closer look at how your team is balancing feature work with delivery speed. If your cycle times feel a little too long, it might be worth digging into workflows or processes to find ways to speed things up without sacrificing quality.

Repository Spotlight

Ruby

Ruby is a great example stability and technical maturity with an impressive 43% of PRs focused on documentation, showcasing a repository designed for long-term clarity and accessibility.
Its average monthly cycle time of just 14 hours means an efficient workflow, even while addressing a mix of bug fixes, features, and other contributions.

Documentation PRs

43%

Average Monthly Cycle Time

14 Hours

Flutter

Flutter takes a distinctive approach with 62% of its PRs dedicated to dependency changes, ensuring the framework stays modern and reliable for developers.

With an average cycle time of 16.12 hours, Flutter balances a high volume of maintenance work while addressing its community's evolving needs.

Dependency PRs

62%

Average Monthly Rework Time

16.12 Hours

Bug Fixes vs Cycle Time

Does resolving issues affect speed?

Bug fixes are the backbone of software maintenance and an essential indicator of a project's health. They reflect the ongoing efforts to address issues, improve reliability, and enhance user experience.

In the context of open-source repositories, bug fixes often dominate contributor focus due to the critical need to maintain high standards across diverse and evolving codebases. However, these fixes might come with trade-offs, particularly when it comes to time.

Key Observations?

Repositories with higher bug fix PRs exhibit a tendency toward increased cycle times, indicating that resolving bugs can be resource-intensive and time-consuming.

However, the relationship is not strictly linear, suggesting other influencing factors.

For example:

- Excalidraw dedicates 58% of PRs to bug fixes but has a relatively moderate cycle time of 43.55 hours.

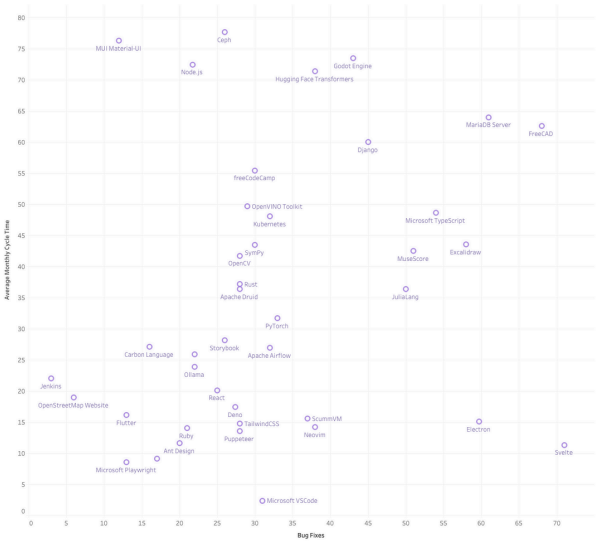
Outliers

- Ceph: Despite having only 5% of PRs for bug fixes, it exhibits a very high cycle time of 77.63 hours, suggesting other bottlenecks in its process.

- VSCode: With only 16% bug fixes, it boasts the shortest cycle time at 2.35 hours, highlighting streamlined processes.

Clustered Insights

- Most repositories with moderate bug fix PR percentages (20–40%) fall within a 15–40 hour cycle time range, indicating a balanced workflow.



Why This Matters

- Efficiency: For teams prioritizing bug fixes, understanding the impact on overall cycle time is critical to balancing resources between addressing existing issues and introducing new features.

- Process Optimization: Outliers like Svelte suggest that even repositories with a high bug fix focus can maintain efficient processes, pointing to opportunities for process refinement.

Invest in process efficiency by streamlining workflows to help manage cycle time effectively, while balancing bug fixes with feature development to ensure that addressing issues doesn't hinder progress on critical tasks essential for long-term growth.

Looking Ahead

Open source is a unique blend of community, technology, and creativity. As repositories grow and evolve, the need to adapt becomes ever more critical.

Here's how maintainers and contributors can leverage the findings from this report:

- Engage contributors with clear guidelines, streamlined onboarding processes, and recognition for their work.
- Leverage tools like Middleware Open Source to continuously track and improve performance.
- Find harmony between bug fixes, features, and other workstreams to ensure a healthy pace of development without burnout.

Challenges

Open-source projects will continue to face hurdles like resource constraints, rising technical debt, and evolving user demands.

Success lies in the ability to pivot and adapt, leveraging both data and community-driven strategies.

Take Action

Whether you're a maintainer, contributor, or open-source enthusiast, the insights here are a starting point.

Dive deeper into your repository's performance, foster collaboration, and keep pushing the boundaries of what open source can achieve.

Get in Touch

Email: contact@middlewarehq.com

Try Middleware