PERFORMANCE ENGINEERING

Fast by Default

— BUILD FAST, STAY FAST —

THE PHILOSOPHY

Fast by Default is a software development model created by Den Odell. It treats performance as a first-class requirement, shaping decisions in architecture, code, testing, deployment, and team habits. The goal is to build systems where speed and responsiveness emerge naturally from everyday work instead of being retrofitted in a rescue mission after users complain.

"Fast by Default is not a tool or framework. It is a disciplined way of building software that stays fast as it grows."

Fast by Default breaks **the Performance Decay Cycle**—the predictable loop of shipping, complaining, panicking, patching, and repeating that plagues most engineering teams. Instead of treating speed as something to fix later, it becomes a natural outcome of how the team builds software every day.

A System Is Fast by Default When:

1. The architecture leaves room for speed by steering clear of designs that will inevitably be slow.
2. Developers can see how their work affects performance while they are still writing the code, not weeks after it ships.
3. Automated budgets and checks catch slowdowns before they reach production.
4. Decisions are guided by data from real users, so the team focuses on the problems that actually affect people.
5. The whole team shares responsibility for speed, working together to keep the software responsive for every user, device, and location.

The Seven Principles

I Design for Speed Before You Write Code

Treat performance as a requirement

II Make Performance Visible During Development

Give developers tools and workflows

architecture, rather than leaving it until later when problems are harder to fix.

speed, so they can spot slow or blocking operations while they are still working on the code.

III Protect the Experience with Clear Budgets

Set clear limits on the things that slow users down, such as response times, payload sizes, and startup costs, then enforce those limits automatically.

IV Optimise Complete User Flows

Users experience complete journeys through your software, not isolated screens, so test and improve the full flow including transitions and the moments between actions.

V Use Real-World Data to Guide Decisions

Let real user experience guide your priorities by tracking what actually happens in production, from response times to errors on the devices your users really have.

VI Treat Performance as a Shared Responsibility

Speed is something the whole team creates together. Everyone has a role to play, whether they work on the backend, frontend, design, testing, or infrastructure.

VII Prevent Decay Through Regular Cleanup and Review

Performance tends to decline as software ages, so set aside time to review dependencies, remove unused code, and check that your most important flows still perform well.

Start Here

If you do nothing else, start with these three moves.

One principle

Think about speed before you start coding, treating performance as something you plan for rather than fix later.

One practice

Make performance visible while you work, so developers can see slow operations and resource costs as they write code.

One check

Set a simple budget for something that matters, like response time or payload size, and have your build process enforce it automatically.

Taken together, these three actions move a team out of the performance decay cycle and into the Fast by Default mindset.

The Fast by Default Cycle

Fast by Default uses a simple recurring loop that keeps systems fast as they grow.

Measure

Start by understanding what users actually experience, then work out where speed matters most and what slowness is costing you.

Build

Design and write systems that perform well under the conditions your users will actually encounter.

Own

Keep checking performance through automated tests, budgets, and tools that make slowdowns visible before they reach users.

Maintain and Refine

Watch how real users experience your software in production, then use what you learn to guide your next round of improvements.

The cycle repeats continuously. Teams that follow it do not wait for problems to appear before acting—speed stays healthy because the loop runs as part of how they normally work.

The Five Practical Steps

HANDS-ON ACTIONS INSIDE THE CYCLE

These are the day-to-day actions that bring the Fast by Default Cycle to life. Together they form a repeating loop that keeps your systems fast even as they grow in size and complexity.

Architect

Begin with designs that leave room for speed by keeping dependencies simple, thinking about caching early, and avoiding structures that will inevitably slow things down.

Implement

Write code with performance feedback close at hand, so developers can see when something is slow or blocking while they are still working on it.

Verify

Check performance automatically as part of your build process, using budgets and tests that catch slowdowns before they reach production.

Observe

Measure what real users experience once your software is live, tracking response

Refine

Take what you learn from real usage and feed it back into your designs and code, fixing the problems that matter and removing unnecessary complexity along the way.

These steps repeat in sequence as part of the larger Fast by Default Cycle, making sure that performance remains a natural part of how your team works every day.

Is Your Team Fast by Default?

Use this quick self-check to see where your team stands today.

Answer yes or no to each question:

1. Do you catch performance problems during development rather than after release?
2. Are performance budgets defined and enforced automatically in CI?
3. Do you use real user data to decide which performance issues to fix first?
4. Do developers see the performance impact of their changes as they work?
5. Do you run a regular cleanup cycle that removes dead code and unnecessary complexity?

If you answered no to more than two questions, your team is probably working inside the

The Fast by Default model offers a way out by making performance part of everyday work instead of emergency work.

Fast by Default Checklist

Here is how to put the model to work in your team.

Architecture

- [ ] Remove unnecessary round trips and sequential dependencies.
- [ ] Keep critical-path work shallow and predictable.
- [ ] Choose rendering and loading strategies that suit your platform.
- [ ] Plan caching, preloading, and data shaping early.
- [ ] Keep services and components small, stable, and easy to optimise.

Development

- [ ] Show long tasks and blocking operations while coding.
- [ ] Show resource costs, response times, and processing overhead as part of normal work.
- [ ] Use profiling tools to spot regressions early.
- [ ] Treat third-party libraries and external services as risky additions.
- [ ] Keep complexity low, especially in frequently used code paths.

Code Review

- [ ] Check for unnecessary work on

CI and Budgets

- [ ] Define budgets for payload size,

- [ ] Look for new blocking operations, expensive computations, or slow queries.
- [ ] Confirm that changes respect existing budgets.
- [ ] Flag hidden dependencies that add latency or load.
- [ ] Require graceful degradation and resilient error handling.

- [ ] Enforce these budgets automatically with predictable checks.
- [ ] Fail builds that slow down key metrics.
- [ ] Track historical performance to spot decay over time.
- [ ] Run both cold-start and warm-path tests.

Testing

- [ ] Test complete flows, not only isolated components or endpoints.
- [ ] Simulate real devices, networks, and geographic regions.
- [ ] Use synthetic tests for consistency and production data for truth.
- [ ] Test interactions, not just load times.
- [ ] Check edge cases like empty states, slow endpoints, and offline scenarios.

Observability

- [ ] Track response times, throughput, error rates, and slow operations in production.
- [ ] Monitor backend latency, memory use, and service health.
- [ ] Instrument the user flows that matter most.
- [ ] Alert on user-visible issues, not internal noise.
- [ ] Use production data to refine budgets and priorities.

Team Habits

- [ ] Include performance goals in planning and design reviews.
- [ ] Keep dependencies healthy and up to date.
- [ ] Celebrate improvements and reinforce good patterns.
- [ ] Run scheduled cleanup cycles to reduce complexity and dead code.
- [ ] Share ownership across frontend, backend, mobile, design, QA, and infrastructure.

Fast by Default is a model you apply every day, not a tool you install. When teams design for speed, verify continuously, and respond to real users, performance becomes the natural outcome of how you work.

More on the Way

Practical guides, starter templates, and tooling for Fast by Default are coming soon. These resources will help you put the model into practice on real projects.

Join the newsletter below to get updates as new content becomes available.

Newsletter

A monthly email from Den Odell with behind-the-scenes thinking on performance engineering, updates on Fast by Default, and more.

your@email.com SUBSCRIBE

One email per month, no spam.

Fast by Default

Projects by Den Odell

DenOdell.com • Fast by Default • Byteshrink AI