#learning

How large pull requests slow down development

November 21, 2023

Share this article:

One of the fundamental challenges in software engineering is managing and minimizing complexity. This challenge is not just a theoretical concern; it has real and tangible impacts on the pace of development. Overly complex codebases slow teams down, making even simple changes cumbersome and time-consuming. This phenomenon is known as "change amplification:" the more complex a codebase becomes, the more modifications across different files and functions are required to implement a simple change.

Change amplification: a symptom of complexity

John Ousterhout explains change amplification in "Philosophy of Software Design" as "a symptom of complexity which is that a seemingly simple change requires code modifications in many different places". This problem was especially rampant in early web development, where a single design element change (i.e.

Written by

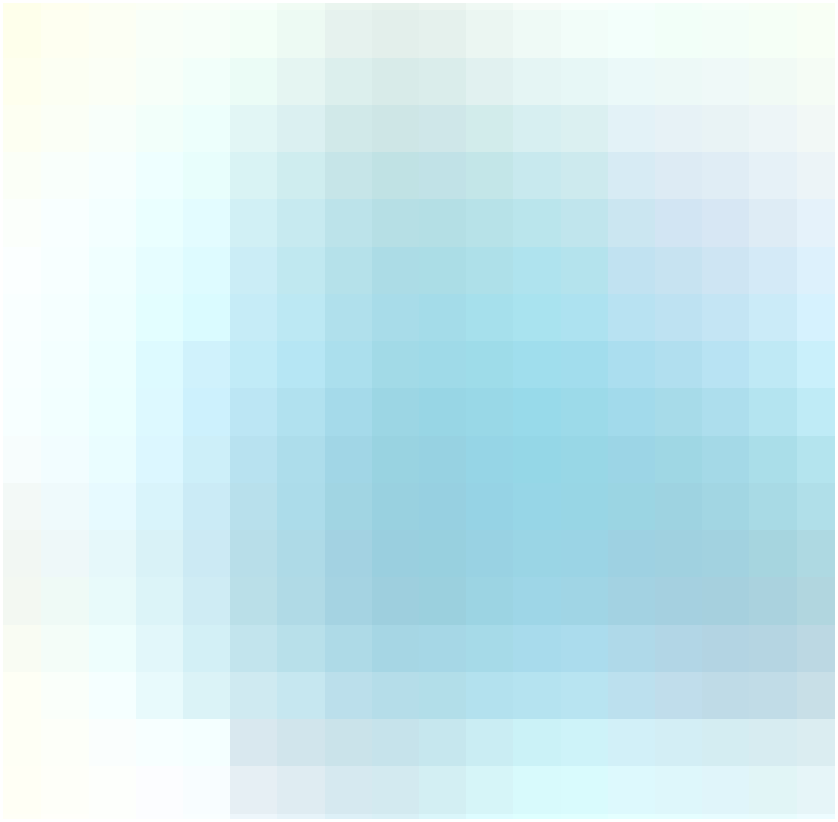
Greg Foster

In this post

a banner color) required updates on every page. Modern web development practices have evolved to centralize such elements, significantly reducing the need for widespread code changes to implement a visual update. The goal is clear: good software design should limit the amount of code affected by each design decision. In a well-architected software system, changes with high amplification - those that impact many areas - signal a problematic level of coupling.

Here at Graphite we wanted to better understand and quantify the costs of change amplification at scale, so we looked to our dataset of millions of PRs created by top engineering teams for answers.

Measuring the**impact of PR scope**

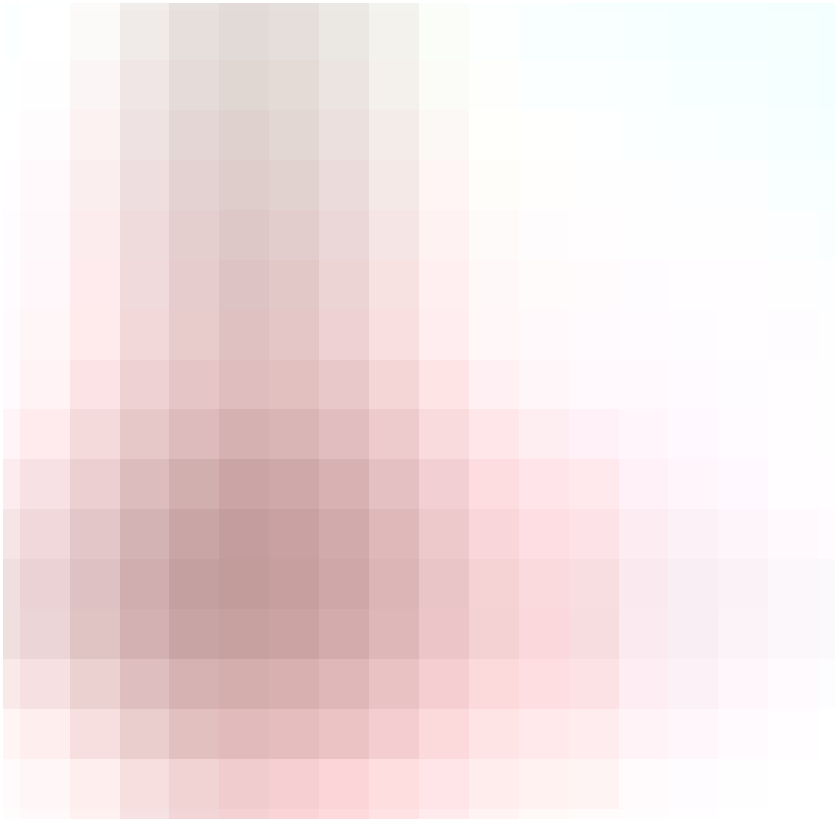


In order to investigate how change amplification affects engineering efficiency, we looked at 1.5 million pull requests and compared the number of files that were changed (as a proxy for complexity) to the time these PRs took to merge.

Our data brings a few interesting patterns to light:

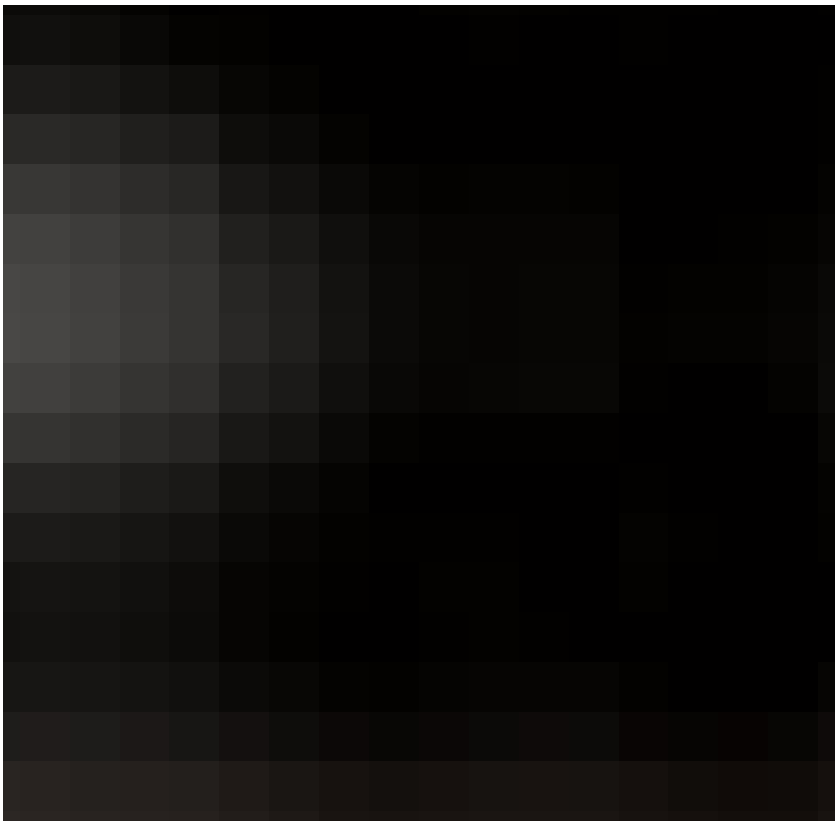
- Unsurprisingly, the fastest PRs are those that change the fewest number of files. This suggests that high-velocity engineering organizations should architect their systems to minimize file touch points in each PR.
- Notably, even a slight increase in the number of files changed can more than double the time-to-merge. This isn't merely a matter of file quantity; it indicates higher risks, coupled code, and a greater likelihood of failing continuous integration (CI) processes.
- Review complexity also increases with the number of files, requiring more time and cognitive effort from reviewers. Moreover, with Git operating at a per-file level, more files mean a higher chance of rebase conflicts, which can further slow down development.

A closer look at review efficiency for varied PR sizes



The graph depicting the average time to review per file against the number of files in a PR reveals a counterintuitive trend: as PRs grow beyond a moderate level of complexity, the time spent reviewing each file notably decreases. Initially, one might expect review time to increase monotonically with PR complexity, yet the data

indicates a pivot point where reviewers spend less time per file as file count grows.



This could be explained by reviewers shifting their strategy from a meticulous line-by-line assessment to a broader, risk-oriented evaluation of larger PRs. Large PRs may also include more auto-generated and/or repetitive changes, which are quicker to verify once the reviewer identifies the recurring pattern. Additionally, cognitive limits mean that reviewers may simply lose focus across a large number of files, inadvertently speeding up the review process.

The overall takeaway from the data is that small PRs lead to faster merges, and large PRs containing more files get less concentrated focus from reviewers. This is critical for engineering teams to understand - if you want to create high-quality code while maintaining an efficient review process, you have to keep PRs sufficiently small.

Best practices for minimizing complexity

The implication of the data is clear: if you want to create high-quality code while maintaining an efficient review process, you should aim to **limit PRs to three or fewer files changed**. Beyond this threshold, there's a significant

increase in time to merge. If maintaining small PRs is challenging, consider the following strategies:

1. **Try out "stacking":** stacking is a source control workflow that allows for smaller, more manageable PRs. By parallelizing review and development, stacking keeps you unblocked while waiting on code review and lets you push changes up for review as you write them. Tools like Graphite make stacking easy for teams that host their code on GitHub.
2. **Simplify software design**: As Ousterhout notes, complexity often stems from an accumulation of dependencies and obscurities. By simplifying design, you reduce change amplification, cognitive load, and the potential for 'unknown unknowns'. This makes it easier and safer to modify existing code bases.

Conclusion

Change amplification is not just a theoretical problem - it clearly manifests in the time it takes to merge large & complex PRs. By keeping changes small and manageable, we can simultaneously increase development velocity while reducing the risk of regressions - and modern source control tooling such as Graphite can help you and your team achieve this.

Related posts

#learning

Understanding Git: The history and internals

November 9, 2023

#learning

How long should yo

November 2, 2023

Kenneth DuMez

 Greg Foster

Graphite © 2023

Product

Features

Pricing

Docs

Customers

Company

Blog

Careers

Contact us

Resources

Community

Privacy & security

Terms of service

Stacking workflow

Developers

Status

GitHub