Reduce cycle time to deliver value

How teams can meet business demand to innovate and remain competitive

GitLab

What's inside?

Introduction

Why does cycle time matter?

» Adding tools is not the solution

» How to reduce cycle time

The impact of faster cycle time

Conclusion

About GitLab

Introduction

In today's modern landscape of rapidly changing priorities, rising expectations from customers, and new market opportunities, IT leaders must discover new ways to streamline delivery of digital applications and systems. The pace of change in the market has reached a point where the need for velocity and rapid response is the new imperative. To remain competitive, organizations must reduce cycle time.

In 2014, Marc Andreesen wrote, "Cycle time compression may be the most underestimated force in determining winners and losers in tech." He observes that organizations face a future where faster delivery and shorter cycle times will be a driving force in the global market. The challenge facing IT leaders is how to adopt cultural and technical changes to achieve the **faster delivery and shorter cycle times** our business partners expect.

Marc Andreessen @pmarca

Cycle time compression may be the most underestimated force in determining winners and losers in tech. The first clear instance of cycle time compression: Cloud/SAAS vs on-premise enterprise software. Cloud/SAAS development cycles can be far faster than on-premise software; single instance deployed instantly to all customers. Further, customers can try and adopt cloud/SAAS far faster than they can try and adopt on-premise software.

Implication: Cloud/SAAS is probably impossible to compete with for on-premise software across multiple product cycles.

The second clear instance of cycle time compression: Product improvement and customer upgrade cycles for phones vs TVs and cars. Consumers can upgrade their phones every 1-2 years, vs TVs at 5-8 years? Cars at 10-12 years? With phones improving by leaps & bounds.

Implication: At given point in time, your TV can be 4-6 years behind your phone; your car can be 9-10 years behind your phone.

Implication: TVs and cars will become accessories for phones, not the other way around. And it's already happening: Airplay, Chromecast.

Interesting note: Web cycle times still much faster than mobile app cycle times due to restrictive mobile app store policies.[1]

[1] https://pmarcasays.golaun.ch/2014/06/07/cycle-time-compression-the-most-underestimated-force-in-determining-winners-and-losers-in-tech/

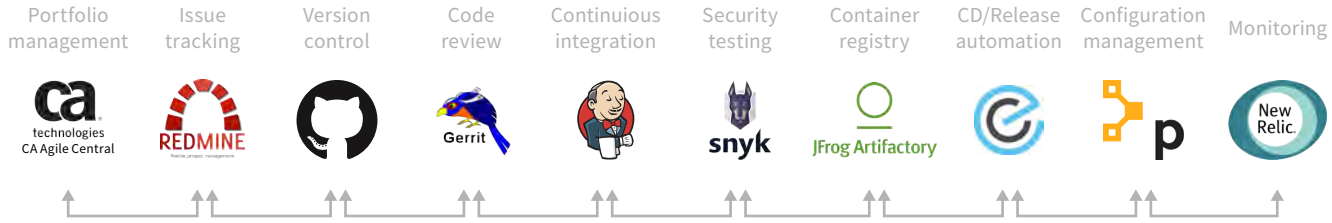
Why does cycle time matter?

In an ever-changing technological landscape, the pace at which organizations release features can impact success. Reducing the time between thinking of an idea and having the code in production is vital to providing value to customers. If an organization is too slow to innovate, its competitors can rapidly absorb the market by becoming the first organization to meet the needs of customers.

Adding tools is not the solution

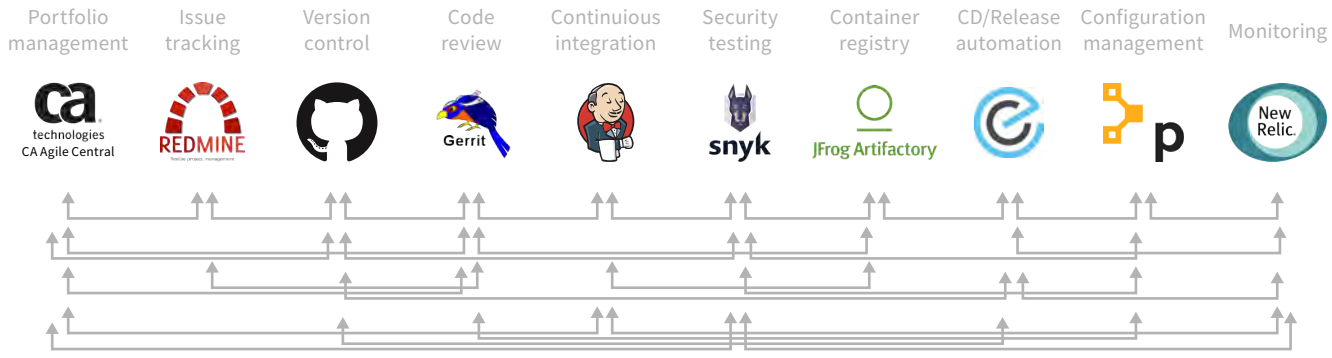
Agile planning and DevOps delivery have demonstrated reliable and scalable solutions to streamline and accelerate application delivery. Recently, deployment teams have taken advantage of new tools to assemble toolchains designed to enable automated integration, test code, manage digital assets, scan for vulnerabilities, and deploy and configure applications.

A toolchain



While these integrated toolchains help accelerate application delivery, they also introduce a new challenge in the form of complexity, islands of data, inconsistent security settings, reporting, and compliance issues. Each tool adds a new integration and a factor of complexity in a team's workflow. What development teams now have is a complex, fragile, and expensive mashed-up toolchain, on which they are forced to **waste cycle time tinkering on tools** and not delivering business value.

A toolchain integration headache



The idea behind using a variety of tools is understandable: Teams can take advantage of the best software in the industry. But, having such a large toolset hinders progress rather than enables it, causing longer cycle time, a lack of visibility, and the creation of silos. These silos take overhead to integrate and manage, slowing down teams and preventing collaboration and communication. While it's tempting to add more tools, a complicated toolchain is not the answer to reducing cycle time.

How to reduce cycle time

What development teams need is a simple, comprehensive toolchain that can easily build, test, and deliver applications without the waste and overhead of managing dozens of disparate tools and integrations.

To resolve bottlenecks and wasted effort, development teams need to simplify their toolchains and establish solutions that incorporate multiple capabilities with fewer fragile integrations. Specifically, to reduce cycle time, teams should look for ways to consolidate interrelated development activities, such as:

1. Issues and planning

The planning stage of the software development lifecycle has historically been a source of confusion and difficulty, with 25% of product managers struggling to create a consistent project management methodology or scoping document, according to The State of Project Management Survey. Teams must be able to capture, discuss, prioritize, and define requirements and use cases. A simple solution to understanding use cases and requirements from end users about the specific capabilities they need is to use issues. Using issues, product managers, delivery teams, and customers define use cases, elaborate on needs, and prioritize new features. Issues initiate future development and assist in the creation of comprehensive planning, since all scoping occurs before development begins.

2. Distributed source code management

Designing and developing applications is an intensive activity that requires managing branches in the source code, tracking frequent changes of multiple files, and merging and integrating changes to the main trunk. Distributed source control allows developers to work independently from a centralized repository and keep their work in synch with the larger team. A distributed source code management enables coordination, sharing, and collaboration across the entire software development team.

3. Code reviews and approvals

Peer reviews and rigorous approvals are essential to ensuring that new code changes address user needs and do not introduce logic errors, defects, or security vulnerabilities. Typically, approvals for

code changes must be clearly documented and tracked to demonstrate compliance. This critical oversight and review process should be a core capability to ensure both quality, accountability, and compliance.

4. Continuous integration for every commit

Continuous integration is the cornerstone of modern software development, and teams that want to embrace rapid delivery require a CI pipeline to ensure the right sequence of automated tests, scans, and compliance checks. Continuous integration ensures:

a. Software quality

The CI pipeline manages an automated unit, API, and functional and nonfunctional tests to verify new code changes do not introduce new defects.

b. Security (code scans, container scans, license management)

Application security scans should be incorporated into the CI pipeline to provide rapid feedback about any software changes that introduce new vulnerabilities. It is critical to have security feedback as early as possible in the development lifecycle in order to minimize the risk of security issues delaying software delivery.

5. Repository to manage binary assets

The output of the continuous integration pipeline is the binary code and libraries which comprise an application. These assets must be managed and tracked through the testing, validation, and deployment of an application.

6. Continuous delivery

The continuous delivery pipeline automates testing and deployment capabilities so that software can be developed and deployed rapidly, reliably, and repeatedly with minimal human intervention. The CD pipeline can be a natural extension of the CI pipeline, continuing the assembly line of tasks from development to deployment into pre-production and finally production.

7. Dynamic test environments / infrastructure

In order to streamline development work, a development lifecycle should support dynamic test environments that can be deployed on demand to support the testing needs of individual developers and teams. Traditionally, new code changes queue up to wait for a limited set of testing environments and resources. A development lifecycle should take advantage of containerization and cloud technology to reduce or eliminate delays.

8. Incremental deployment

Rapid delivery requires reducing every change proposal to its absolutely minimally viable form to allow teams to ship almost anything within a single release, obtain immediate feedback, and avoid deep investments in ideas that might not work. Incremental deployments reduce risk and enable quick iteration, helping teams consider the full scale of development complexity but focus on moving a project forward with the simplest solution. Techniques such as canary deployments or feature flags give software development teams the flexibility to ship code quickly while actively managing and mitigating risks.

9. Application monitoring

Quickly delivering what customers want requires a modernized software development lifecycle that saves time, effort, and cost. Feedback from an application in production is an essential part of reducing cycle time. Rapid and actionable insight from application monitoring empowers product developers to detect issues, take action, and continuously improve an application.

The impact of faster cycle time

After reducing cycle time, organizations experience benefits throughout the business, including saving costs, developing innovative features, satisfying customers, and increasing competitiveness. When businesses get to market faster, they can continuously improve to deliver the features that customers want.

Conclusion

Reducing cycle time is vital to remain competitive and meet customer needs. Rapid delivery is no longer a luxury but a necessity to compete in an evolving industry, and organizations must find ways to encourage collaboration, sharing, and teamwork. To address the challenges of rapidly building and delivering business applications, organizations can simplify the toolchain and empower teams to focus on innovation and delivering value to customers faster.

About GitLab

GitLab is the first single application for the entire DevOps lifecycle. Only GitLab enables Concurrent DevOps, unlocking organizations from the constraints of today's toolchain. GitLab provides unmatched visibility, radical new levels of efficiency and comprehensive governance to significantly compress the time between planning a change and monitoring its effect. This makes the software lifecycle 200% faster, radically improving the speed of business.

GitLab and Concurrent DevOps collapses cycle times by driving higher efficiency across all stages of the software development lifecycle. For the first time, Product, Development, QA, Security, and Operations teams can work concurrently in a single application. There's no need to integrate and synchronize tools, or waste time waiting for handoffs. Everyone contributes to a single conversation, instead of managing multiple threads across disparate tools. And only GitLab gives teams complete visibility across the lifecycle with a single, trusted source of data to simplify troubleshooting and drive accountability. All activity is governed by consistent controls, making security and compliance first-class citizens instead of an afterthought.

Built on Open Source, GitLab leverages the community contributions of thousands of developers and millions of users to continuously deliver new DevOps innovations. More than 100,000 organizations, including Ticketmaster, ING, NASDAQ, Alibaba, Sony, and Intel trust GitLab to deliver great software at new speeds.

Start your free trial


GitLab