Shifting Left: Improving the Quality and Speed of Software Development

Felix Qvarfordt

April 2023*

*I/We certify that generative AI, incl. ChatGPT, has not been used to write this essay. Using generative AI without permission is considered academic misconduct.

1 Introduction

The field of software development is quickly evolving and new paradigms, processes, and methods appear every few years. A term and concept that has gained a lot of traction in the last decade is "DevOps," a term coined in 2009 made up of the words "development" and "operations" [11]. This idea focuses on bridging the gap between the developers and operations teams in an organization, which historically has led to many problems. It achieves this goal mainly by merging the two teams together to form cross-functional teams and promoting collaboration and communication, this is usually accompanied by several new automating technologies and other practices like continuous integration and delivery with the goal of delivering software faster and more reliably. As the concept matures, core ideas, practices, and patterns are becoming more established. One such idea is called shifting left, and while invented before and grew alongside the agile and DevOps movements, it captures the essence of DevOps by creating a smoother development lifecycle with and maximize the benefits derived from DevOps as a whole.

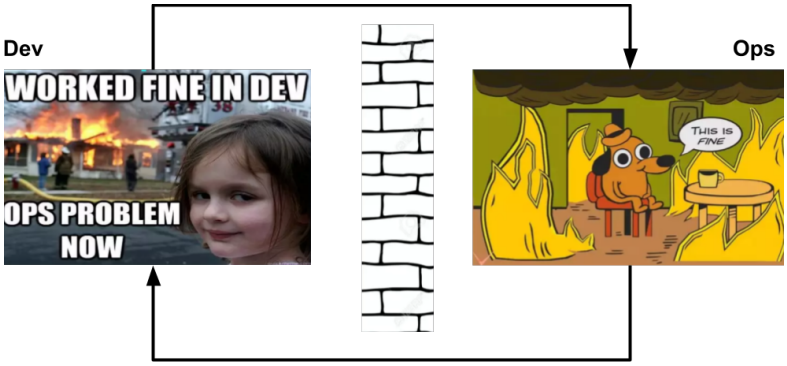

Figure 1: Comical representation of the gap between the development and operations teams before the introduction of DevOps.

Shifting left involves integrating quality assurance (QA) and testing into the software development lifecycle from an early stage whereas it previously mostly has been done in the later stages. The idea is quite intuitive, the earlier issues are detected, the less effort and money will be needed to solve the issues. Using different tools and techniques enables more reliable delivery to production and also leads to better software and reduce costs.

In this essay we will explore the benefits, strategies for implementation, and potential challenges and problems associated with shifting left. It also aims to provide a deeper understanding of the concept of shifting left, how it impacts the workflow in an organization, and potential pitfalls to look out for when applying the approach in practice.

1.1 State-of-the-art

Since shifting-left is quite a broad concept, many of the new and upcomming DevOps terms can be more or less caught under the shifting-left umbrella. As long as they have some way of enforcing or promoting quality earlier in the development lifecycle they would be considered the state-of-the-art of shifting left. A few examples would be DevSecOps, NoOps, MLOps, cloud-native technologies, etc. Therefore, it is hard to capture the state-of-the-art of shifting left in a short description without missing out on important and relevant examples.

2 Impact of Shift-Left on Software Development Processes

Since the main point of shifting left is to make sure quality assurance and testing are preformed as early as possible and consistently throughout the software development lifecycle, this forces us to use new tools and techniques impacting our workflow.

In this section we will go over a few of the key concepts that have can have a large impact on an organizations workflows when transitioning into the shift-left mindset.

2.1 Continuous Integration (CI)

Continuous integration is one important strategy. By continuously and regularly merging branches and running automated tests, we can avoid a common problem that arises with long-lived branches, sometimes called "merge hell."[1] This phenomenon refers to problems that occur when a branch has significantly diverged from the main branch, causing merging to become a time-consuming and labor-intensive process, if not entirely unfeasible.

There have been instances where organizations have abandoned feature branches that have been under development for months, simply because the effort and cost of merging them back into the mainline outweighed the benefits. While the scale of the problem can vary, developers working on large projects will often lose some work due to this. The more frequently branches are merged, the less work is at risk of being lost, and in the most extreme settings one would try to keep the total work on a separate branch to a number of minutes rather than hours simply to avoid any large work losses due to this issue. [8]

2.2 Emphasis on Unit Testing

Another crucial part in the shift-left approach is the importance of unit testing. [10] By testing all written code with unit tests, the most common bugs like off-by-ones, variables outside of scope, etc. can be caught already at this early stage. Identifying such problems early on in the CI/CD pipeline allows developers to

quickly fix the bug before the change makes it deeper into the pipeline where the issue would take more time and effort to resolve. This idea often goes hand in hand with the idea of test driven development[6], or TDD for short, where you write your unit tests before starting to write your code. Using TDD is not mandatory but a good practice to ensure the use of comprehensive tests for every developer on your team. Using comprehensive unit tests we can also rely on fewer integration tests, which are not only harder to debug but are also significantly slower than unit tests. Therefore, migrating from a pipeline with many integration tests that really could be expressed as unit tests can often also lead to a large speed increase in the pipeline.

2.3 Integration Tests

While the shift-left approach highly values unit testing, integration tests are still necessary to make sure that the interaction between different components of the software work as intended.[3] However, the goal is to minimize the number of integration tests by relying on comprehensive unit tests. Catching errors with integration tests is still better than finding them further down the pipeline; for example, using canaries might prevent rolling out a bug to the entire user base, but feedback might take days or weeks to find, and by then, it can be more expensive to fix.

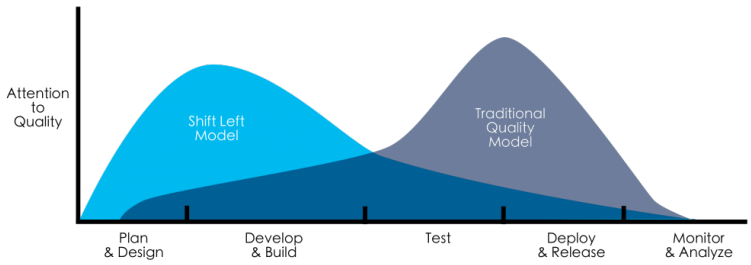

Figure 2: Testing/Quality shift from a traditional model

2.4 Monitoring and Observability

Monitoring and observability are also key concepts that should be implemented when shifting left. By monitoring software in production, we can detect issues as they occur. While commonly seen as a shift right strategy, it also relates to shift-left since we can identify problems earlier in the production phase and solve them before they become even worse and impact the end-users.[7]

Observability is a more broad concept and refers to being able to examine the internal state of our system by looking at its external outputs. By using a unified platform for observability that takes in data from the whole organization one can improve the workflow in most parts of the software development lifecycle, avoid problems and issues before they arise but also help solve them when they do. [9]

3 Risks and Challenges of Shift-Left in DevOps

Now that we have gone over many of the core concepts of what shifting left means and how it differs from more traditional software development practices, let us look at some risks and challenges that we should be aware of when implementing the approach.

3.1 Increased Complexity of Developer Roles

Now that DevOps have blurred the lines between the developers and operations teams and shifting left moves more practices and responsibilities over to the developers desk, developers are expected to be involved in several processes and aspects of software development and use new tools to help with their new workflow.[2]

Shifting left gives more concrete guidelines to what the developer is supposed to be doing, and also importantly, places accountability for the written code more directly on the developer who wrote it by detecting the errors and quickly giving feedback. Monitoring, observing, maintenance, and deployment now also have to be in the back of the mind of every developer and the idea that all of that can be fixed by someone else is thrown out the window.

This poses the risk of overwhelming the developers which ultimately might lead to burnout or a loss of focus on the core task which is to develop good software. To mitigate this risk, organizations should provide proper training and resources to their developers in their new roles. One also needs to make sure that the new cross-functional teams that are created get appropriate tasks and that the responsibilities are evenly distributed so that each team can perform at its best.[5]

3.2 Integration of Tools and Technologies

Naturally, when shifting left and creating a totally new workflow for your organization, a new set of tools and technologies are needed to carry out the work in a streamlined and automated way. Tools such as version control systems, build and deployment tools, testing framework and monitoring/observability solutions are some examples of this, but more might be needed in any specific case. Integrating these tools poses a great challenge and moving over to a completely new structure might take years. Some tools might not be compatible with each other and others might need to be researched and set up delicately to work effectively within the new and quickly changing infrastructure that works.

Of course, you can try and carefully plan and evaluate your new tools before implementing them to try and make sure that they will work in the new system and that they can be integrated without any large issues. However, planning can only take you so far, and by sticking to some basic concepts such as agility and modularity, one will have an easier time changing the tools as needed when problems arise. One perhaps easier approach is to go for a locked-in solution where many tools are already tested and can be seamlessly integrated together, but if some limitation of the tools are discovered later on, switching out the tools might be very cumbersome and this can be avoided by sticking to open standards and API:s as much as possible.[4]

3.3 Organizational and Cultural Changes

Accompanied by shifting left is often a significant cultural and organizational change. As mentioned, development and operational teams have traditionally been more independent with less interaction, collaboration, or communication between the teams. By now it should be clear that this structure would not work well in a shift-left environment since the knowledge of the developers is needed further down the CI/CD pipeline and the knowledge of the operations team is needed further up. The change implies creating cross-functional teams with a high emphasis on collaboration and knowledge sharing but this can be easier said than done in an organization where many employees and are set in their ways.

It is no small task to change from a traditional approach to the shift left mentality in a company and there are several things that might end up creating a disastrous situation. But organizations can try to address these issues proactively and continuously evaluate how the integration is doing to make sure that everyone feels onboard and avoid creating a toxic environment in the workplace. A key piece here is active leadership that understands these issues and listens to feedback from the different teams.[5]

4 Reflection

Some DevOps practices and ideas are only fit some companies, take for example microservices. A small company just starting out does not gain anything by implementing microservices from the beginning and it could even be their downfall since the many added layers of complexity slow down development compared to a monolithic development process during the most fast-paced phase of the company's life. These kinds of practices seem to be better used when implemented when the company grows and is in need of the scalability of such ideas and where the initial investment and potential drawbacks start to get outweighed by the gain of the new tools and practices.

When instead looking at the shift-left approach, we realize that this can be used to different degrees, and the core concepts of the approach, like writing good and comprehensive unit tests and creating cross-functional teams, can be done on a smaller scale without any larger drawbacks. The problems connected to implementing the approach mainly seem to stem from the change the company has to go through.

For larger companies operating with traditional practices, several challenges will show up during the process of shifting left. This will not only be felt by the company itself but also by the individual developers whose roles suddenly start to become more fluid and who might need to educate themselves to keep up with the changing environment. There are likely a lot of developers currently working who do not have any deeper knowledge of DevOps and/or specialization in other areas of software engineering. For them, shifting left could then pose a threat of losing their job if they do not learn and adapt fast enough, and this is most likely one of the hardest problems to solve when trying out the approach.

Looking around, one can find many opinions and tips about how to make the transition go smoothly, but they often seem to boil down to the fact that shifting left mostly is about shifting the mentality and culture in the workplace. Tools and new techniques can often be implemented if you just take the time to learn them correctly and in some cases get outside help during the transition. But changing the culture in the workplace is a different problem entirely and comes down to people skills, good management and clear communication. It's no wonder you often hear from developers that your technical skills are not what will make your career, but rather your people skills.

5 Conclusion

Because of the significant benefits shifting left offers, it has become a prominent tool in the DevOps movement to lead the way on how to streamline your workflow.

However, while shifting left offers many benefits, adopting the mindset still comes with considerable challenges and risks that have to be dealt with for a successful result. The increased complexity of the roles of developers, the integration of new tools and technologies and most importantly, organizational and cultural changes. By addressing these challenges and using different strategies to avoid the overhanging risks organizations can hopefully make the road to change less bumpy and also reap the benefits of the shift-left ideas.

References

[1] Alberto de Murga. 2022. Escaping a git merge hell. https://threkk.medium.com/escaping-a-git-merge-hell-e08f37511f37

[2] Hannah Foxwell and Andy Burgin. 2021. Blame DevOps: Shifting Left the Wrong Way. https://www.youtube.com/watch?v=muJqAhfBW9s

[3] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Pearson Education, Limited, Hoboken.

[4] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Pearson Education, Limited, Hoboken.

[5] George Lawton. 2019. Take a shift-left approach, but avoid developer burnout. https://www.techtarget.com/searchsoftwarequality/tip/Take-a-shift-left-approach-sans-developer-burnout

[6] Silverio Martínez-Fernández, Anna Maria Vollmer, Andreas Jedlitschka, Xavier Franch, Lidia López, Prabhat Ram, Pilar Rodríguez, Sanja Aaramaa, Alessandra Bagnato, Michał Choraś, and Jari Partanen. 2019. Continuously Assessing and Improving Software Quality With Software Analytics Tools: A Case Study. *IEEE Access* 7 (2019), 68219–68239. https://doi.org/10.1109/ACCESS.2019.2917403

[7] Mallory Mooney. 2021. Best practices for shift-left testing. https://www.datadoghq.com/blog/shift-left-testing-best-practices/

[8] Asif Nahian. 2018. Git — How often should you commit? https://medium.com/@asifnahian/git-how-often-should-you-commit-c133e5473d76

[9] Ilan Peleg. 2022. Shifting Left Observability in Practice — An Overview. https://thenewstack.io/shifting-left-observability-in-practice-an-overview/

[10] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2017. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access* 5 (2017), 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629

[11] Dinesh Venugopal. 2020. DevOps: Driving innovation with old habits. https://devops.com/devops-driving-innovation-with-old-habits/