ISSN: 2583-9888
DOI: doi.org/10.51219/JAIMLD/amit-gupta/167

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

Vol: 1 & Iss: 1 *Case Report*

A Case Study: DevOps Transformation in Mobile Application Products: Journey, Challenges, and Solutions

Amit Gupta*

Amit Gupta, San Jose, CA, USA

Citation: Gupta A. A Case Study: DevOps Transformation in Mobile Application Products: Journey, Challenges, and Solutions. *J Artif Intell Mach Learn & Data Sci 2022*, 1(1), 655-660. DOI: doi.org/10.51219/JAIMLD/amit-gupta/167

Received: 03 September, 2022; **Accepted:** 28 September, 2022; **Published:** 30 September, 2022

***Corresponding author:** Amit Gupta, San Jose, CA, USA, E-mail: gupta25@gmail.com

ABSTRACT

In the quest for enhanced agility, speed, and quality in software development, many organizations are embarking on DevOps transformations. A leading Enterprise Mobile Management suite undertook this journey two years ago, driven by goals of faster releases, better quality, increased team flexibility, and quicker feedback. This paper outlines the initial conditions, the challenges encountered, and the solutions implemented during this transformation. Key challenges included resistance to change, pressure for rapid feature delivery, technical debt, team structure, skill gaps, ineffective automation, environmental constraints, and lack of cross-team collaboration. By addressing these issues through strategic interventions such as securing management support, continuous education, adjusting team velocity, integrating quality engineering into development teams, upskilling, and refining automation strategies, the suite achieved significant improvements. The results included faster release times, reduced unplanned patches, improved productivity, and enhanced team collaboration. This paper provides detailed insights into the transformation process and offers recommendations for other organizations seeking to adopt DevOps practices, emphasizing the importance of careful planning, management alignment, and a commitment to continuous learning and improvement.

Keywords: DevOps, Transformation, Agile, Scrum, DevOps Journey, Mobile Applications, Management, Process

1. Introduction

In the quest for faster releases, better quality, increased flexibility, agility, and quicker feedback, many companies are undergoing a DevOps transformation. This shift is crucial for meeting the ever-evolving demands of the market and staying competitive. A leading Enterprise Mobile Management suite is no exception to this trend. Recognizing the need to adapt and innovate, our journey began two years ago, driven by these specific goals. This transformation was not merely about adopting new tools and processes but also about fostering a cultural change within the organization. This paper details the obstacles we faced along the way, ranging from resistance to change to technical challenges. Additionally, it outlines the solutions and strategies that kept us moving forward, ensuring that our DevOps transformation was both effective and sustainable. Through this detailed account, we aim to provide insights and guidance for other organizations embarking on similar journeys.

2. Initial State

When we embarked on our DevOps journey, we had some initial ingredients in place that were crucial for the transition. These included an agile process, although its effectiveness was debatable at the time. We also had a dedicated operations team, various automation tools, and modern technologies at our disposal. Despite having these essential elements, they were not functioning together as a cohesive unit, which significantly hindered our progress. Each component operated in isolation, leading to inefficiencies and a lack of synchronization across teams. The agile process was not being fully utilized to its potential, the operations team worked separately from the developers, and the automation tools were not integrated effectively into our workflows. Recognizing these shortcomings, we understood that fundamental changes were necessary. We needed to align these components to work seamlessly together, fostering collaboration and enhancing overall productivity. This realization marked the beginning of our journey towards restructuring and optimizing our processes to truly embrace the DevOps methodology.

3. Challenges and Solutions

3.1. Cultural shift

Challenge: Convincing the team to adopt significant changes and drive towards a cultural shift was undoubtedly one of the most challenging tasks we faced. Fear, uncertainty, and doubt naturally created an environment of rejection and resistance among the team members. Many were comfortable with the existing processes and hesitant to embrace new methodologies.

Solution: To tackle this, we initiated a program of constant education and awareness about the benefits and necessity of DevOps. This involved regular training sessions, workshops, and open discussions to address concerns and highlight the advantages of the new approach. We implemented gradual process changes to ensure a smooth transition. Teams were encouraged to adopt changes slowly and incrementally, which allowed them to see the practical benefits of DevOps adoption without feeling overwhelmed. This step-by-step approach helped in reducing resistance and building acceptance. Additionally, the unwavering support of our management team played a crucial role. Their alignment with the DevOps vision and their active participation in the transformation process made a significant difference. They provided the necessary resources, motivation, and encouragement to the teams, reinforcing the importance of this transformation. Without their support, the DevOps transformation would have been considerably more challenging, if not impossible.

3.2. Feature delivery pressure

Challenge: Teams were under constant pressure to deliver new features quickly, which placed a significant strain on the development process. Developers were primarily focused on delivering working code as rapidly as possible to meet tight deadlines. However, quality assurance was the responsibility of a separate Quality Engineering (QE) team. This separation of duties often resulted in a disconnect between development and quality, leading to numerous defects and issues that were discovered late in the development cycle. Despite the developers' best efforts to produce functional code, the lack of integrated quality checks throughout the development process meant that many bugs and defects went unnoticed until they reached the QE team. This not only slowed down the overall development process but also led to frustration and rework, further exacerbating the pressure on the teams.

Solution: To address this issue, we implemented a strategic shift in our approach. First, we decided to reduce the team's development velocity to 50% of their regular capacity. This reduction was crucial in ensuring that developers had adequate time to focus on quality rather than just speed. By lowering the velocity, we aimed to create a balance where the quality of the code was given as much importance as the speed of delivery. Additionally, we redefined the concept of "done." The new definition included not only the completion of coding but also thorough quality checks, testing, and validation. This holistic approach ensured that a feature was only considered "done" when it met the highest quality standards. We also emphasized root cause analysis to identify and address the underlying issues causing defects, rather than just fixing the symptoms. This proactive approach helped in preventing the recurrence of similar issues. Furthermore, we enhanced our automation efforts to support continuous integration and continuous testing. Automation played a pivotal role in providing early feedback and identifying defects early in the development process, thereby reducing the likelihood of issues slipping through the cracks. By integrating these solutions, we were able to improve the overall quality of our products and ensure a smoother, more efficient development process.

3.3. Technical debt

Challenge: Like many teams working with legacy code, we faced a significant amount of technical debt that needed addressing. Technical debt accumulated over time as new features were added and quick fixes were implemented to meet immediate needs. This debt manifested in various forms, such as outdated code, lack of documentation, and obsolete technologies. The presence of technical debt not only slowed down the development process but also made the codebase more difficult to maintain and enhance. It created obstacles that hindered our ability to innovate and adapt quickly to new requirements. Addressing technical debt was crucial to improving the overall health and sustainability of our codebase, but it was challenging to find the time and resources to tackle it amidst the ongoing pressure to deliver new features.

Solution: To effectively address our technical debt, we made the strategic decision to slow down our development velocity. By reducing the team's workload to 50% of their regular capacity, we created the necessary breathing room to focus on technical debt without compromising ongoing feature development. This approach allowed us to tackle smaller, more manageable chunks of technical debt within the team's capacity, rather than trying to address it all at once, which would have been overwhelming and impractical. We collaborated closely with product management to prioritize technical debt and engineering improvements alongside new feature development. This prioritization ensured that technical debt reduction became an integral part of our planning process, rather than an afterthought. By integrating technical debt management into our regular workflow, we were able to make consistent progress in improving the codebase. This approach also involved identifying the most critical areas of technical debt that posed the greatest risk to our projects and addressing those first. Additionally, we focused on enhancing our engineering practices to prevent the accumulation of new technical debt. This included improving code reviews, enforcing coding standards, and investing in better documentation. By balancing the reduction of technical debt with the development of new features, we ensured the long-term success and sustainability of our products, leading to a healthier, more maintainable codebase that could support future growth and innovation.

3.4. Team structure

Challenge: Initially, Quality Engineering (QE) teams operated independently of developers, which did not fit well within the DevOps structure. This separation created a siloed environment where developers focused solely on writing code and QE teams were responsible for testing and ensuring quality. This division led to several issues, including delays in identifying and fixing bugs, miscommunication between teams, and a lack of shared responsibility for the end product's quality. The developers and QE teams often had different priorities and goals, which made it difficult to achieve a cohesive, high-quality product. This independent operation of QE teams was fundamentally at odds with the DevOps philosophy, which emphasizes collaboration, shared responsibility, and integration of all functions involved in software development and delivery.

Solution: To align with the DevOps principles and address these challenges, we decided to merge the QE teams with the development teams, creating integrated, cross-functional DevOps-style teams. This restructuring was aimed at fostering a culture where quality was everyone's responsibility, from the initial stages of design and development through to the final stages of deployment and maintenance. By integrating QE with development, quality engineers began to work closely with developers, participating in the entire development lifecycle rather than just the end phases. This close collaboration meant that quality considerations were embedded from the very start of the design process, leading to more robust and well-thought-out solutions. Quality engineers and developers now reported to the same managers, ensuring better alignment of goals and priorities. This change also facilitated better communication and faster feedback loops, as issues could be identified and addressed in real-time. Pair programming and joint problem-solving sessions between QE and developers became common practice, which not only improved the quality of the code but also enhanced the skill sets of both groups. By engaging in quality activities from the outset, the teams were able to detect and fix defects early, reducing the cost and effort of addressing issues later in the development process. This integrated approach helped us to build a more cohesive, efficient, and high-performing team, ultimately leading to the delivery of higher quality products.

3.5. Skills and knowledge gaps

Challenge: Changing the team structure to make everyone responsible for product quality revealed significant skills and knowledge gaps. While the intention was to create a more integrated and collaborative environment, it quickly became apparent that both the Quality Engineering (QE) teams and the developers lacked certain essential skills needed to fulfill their new roles effectively. For instance, many quality engineers were not familiar with the intricacies of code development, making it difficult for them to engage in meaningful code reviews or contribute to automated testing frameworks. On the other hand, developers often lacked a deep understanding of quality assurance principles, test planning, and comprehensive testing strategies. This disparity in skills and knowledge posed a risk to the quality of the product and the overall success of the DevOps transformation.

Solution: To address these skills and knowledge gaps, we initiated comprehensive upskilling and cross-skilling programs. These programs were designed to equip both QE teams and developers with the necessary skills to effectively contribute to all aspects of product development and quality assurance. For the QE teams, we provided training in code development, focusing on programming languages, coding best practices, and the use of development tools and environments. This training enabled quality engineers to better understand the code they were testing, participate in code reviews, and even contribute to writing automated test scripts. For the developers, we organized training sessions on quality assurance principles, test planning, test strategy, and overall quality mindset. This included workshops on writing effective test cases, understanding different types of testing (such as unit, integration, and system testing), and the importance of thorough and systematic testing processes.

The upskilling and cross-skilling initiatives were supplemented by practical, hands-on experiences. We encouraged pair programming and collaboration sessions where developers and quality engineers worked together on real projects. This not only reinforced the training but also fostered a deeper understanding and respect for each other's roles and contributions. Knowledge sharing became a regular practice, with team members presenting their learnings and insights during team meetings. This continuous learning environment helped to bridge the skills gap and cultivate a culture where quality was a shared responsibility. By fostering a quality mindset across the entire team, we ensured that quality assurance was integrated into every stage of the development process, leading to higher quality products and more efficient workflows.

3.6. Automation

Challenge: Initially, automation was not used effectively, and test results were often ignored due to a lack of trust in the automation scripts. This mistrust stemmed from several issues, including flaky tests that produced inconsistent results, poorly written scripts that were difficult to maintain, and a lack of alignment between the automation tools used by the QE team and the development tools preferred by developers. As a result, the potential benefits of automation were not being realized. Instead of providing reliable and timely feedback, the automation framework was seen as an unreliable and cumbersome addition to the development process. This undermined confidence in the automated tests and led to a heavy reliance on manual testing, which was time-consuming and error-prone.

Solution: To address these issues, we undertook a comprehensive review of our automation test strategy. The first step was to select tools and technologies that were familiar and comfortable for developers to use. By aligning our automation tools with the development environment, we made it easier for developers to write and maintain test scripts. This familiarity encouraged greater participation from developers in the automation process. We chose technologies that integrated seamlessly with the existing development tools, such as integrated development environments (IDEs) and version control systems, ensuring a smooth workflow.

Additionally, we focused on improving the design and architecture of our test automation framework. This involved creating more robust and maintainable test scripts, establishing clear guidelines and best practices for writing tests, and ensuring that tests were reliable and produced consistent results. We also implemented continuous integration and continuous testing practices, where automated tests were run as part of the regular build process. This provided immediate feedback on the quality of the code, allowing developers to identify and fix issues early in the development cycle.

Training and support were crucial components of our strategy. We provided developers and QE team members with training on how to write effective and reliable test scripts. Workshops and hands-on sessions were conducted to help them understand the nuances of automated testing and the specific tools we had chosen. We also encouraged collaboration between developers and quality engineers, fostering a shared responsibility for the quality of the automated tests.

By making these changes, we improved the stability and reliability of our test automation. Developers were more engaged in the process, contributing to and maintaining test scripts. As a result, the automated tests became a trusted part of our development workflow, providing valuable and consistent feedback that enhanced the overall quality of our products. This strategic overhaul of our automation approach not only boosted confidence in automated testing but also significantly reduced the reliance on manual testing, leading to more efficient and effective development cycles.

3.7. Environment availability

Challenge: The availability of on-demand test environments was a significant bottleneck due to the complexity of the product. Our enterprise mobile management suite is a multifaceted and highly integrated system with numerous dependencies and configurations. Setting up test environments manually was time-consuming, error-prone, and required substantial resources. This complexity meant that teams often had to wait for shared environments to become available, leading to delays in testing and development processes. The lack of immediate access to suitable test environments hindered our ability to perform continuous integration and continuous testing effectively. This bottleneck not only slowed down our release cycles but also impacted the overall quality and reliability of our product.

Solution: To address this critical issue, our infrastructure team embarked on a comprehensive project to develop a system for providing on-demand test environments. This system was designed to offer test environments through a self-service portal and REST APIs, making it easy for teams to create and manage their own environments as needed. The self-service portal allowed team members to quickly and efficiently request, configure, and deploy test environments tailored to their specific requirements without relying on manual processes or waiting for shared resources. This automation dramatically reduced the time and effort required to set up test environments, enabling teams to focus more on development and testing activities.

The use of REST APIs further streamlined the process by allowing seamless integration with our existing automation frameworks and tools. Teams could incorporate environment provisioning directly into their automated workflows, ensuring that test environments were always available when needed. This integration facilitated continuous integration and continuous testing practices by providing consistent and reliable environments for every build and test cycle. It also ensured that environments were configured correctly, reducing the likelihood of environment-related issues that could skew test results or cause failures.

Additionally, our infrastructure team implemented robust monitoring and management capabilities to ensure the stability and performance of these on-demand environments. This included automated scaling, resource allocation, and health checks to maintain optimal operation and availability. Regular audits and maintenance ensured that the environments remained up-to-date with the latest configurations and dependencies, further enhancing their reliability.

The introduction of on-demand test environments via the self-service portal and REST APIs had a transformative impact on our development and testing processes. It eliminated the bottleneck caused by the complexity of manual environment setup, enabling faster and more efficient testing cycles. Teams were empowered to provision and manage their environments independently, leading to increased productivity and reduced downtime. This solution not only improved the speed and quality of our releases but also fostered a culture of self-sufficiency and innovation within our teams.

3.8. Cross-team collaboration

Challenge: Teams often worked in silos, focusing primarily on their specific deliverables without sharing best practices, insights, or reusable code. This siloed approach led to a lack of collaboration and communication across teams, resulting in duplicated efforts, inconsistent standards, and missed opportunities for innovation and efficiency. Each team operated in its own bubble, with little to no visibility into what others were doing, which made it difficult to align on common goals or leverage each other's work. This fragmentation not only slowed down the development process but also increased the likelihood of errors and inconsistencies in the product. The absence of shared knowledge and reusable code meant that valuable resources were being wasted, and the overall potential of the organization was not being fully realized.

Solution: To overcome these challenges, we took a strategic approach by forming a dedicated coordination team. This team's primary responsibility was to facilitate communication and collaboration across all development teams, ensuring that efforts were aligned and resources were used efficiently. The coordination team worked to align release cycles, making sure that different teams' timelines and milestones were synchronized. This alignment helped in avoiding bottlenecks and ensured that dependencies between teams were managed effectively.

Moreover, we emphasized the importance of reusing code and sharing best practices. The coordination team played a crucial role in identifying and promoting reusable components and modules that could be leveraged across different projects. This not only reduced duplication of effort but also ensured that high-quality, well-tested code was being used consistently. To support this initiative, we invested in improving our documentation practices. Comprehensive and up-to-date documentation was created and maintained, making it easier for teams to understand and adopt reusable code and best practices. This documentation served as a valuable resource for onboarding new team members and for ongoing reference.

Additionally, we created a cross-platform team tasked with decoupling and writing reusable code that could be used across various product lines. This team consisted of experienced developers and architects who specialized in creating modular, scalable, and reusable code. They worked closely with other teams to identify common functionalities and design reusable components that met the needs of multiple projects. By fostering a culture of code reuse and sharing, we were able to significantly improve the efficiency and quality of our development process.

The formation of the coordination and cross-platform teams transformed the way we worked. It broke down the silos that had previously hindered collaboration and innovation. Teams began to communicate more openly and frequently, sharing their successes and challenges, and learning from each other. The use of reusable code became standard practice, leading to faster development cycles and more robust products. This collaborative approach not only enhanced the overall productivity and effectiveness of our teams but also created a more cohesive and supportive work environment. As a result, we were able to deliver higher-quality products to our customers more efficiently and effectively.

Findings

- Our DevOps transformation led to several improvements:
- Increased collaboration across teams.
- Enhanced quality ownership by all team members.
- Greater focus on root cause analysis and preventive measures.
- Better alignment between development and quality assurance processes.

We also observed significant improvements:

- Release speed increased by 50% on iOS and 25% on Android.
- Unplanned patches and minor releases decreased by 58% on iOS and 29% on Android.
- Escalation counts were reduced.
- Overall productivity improved.

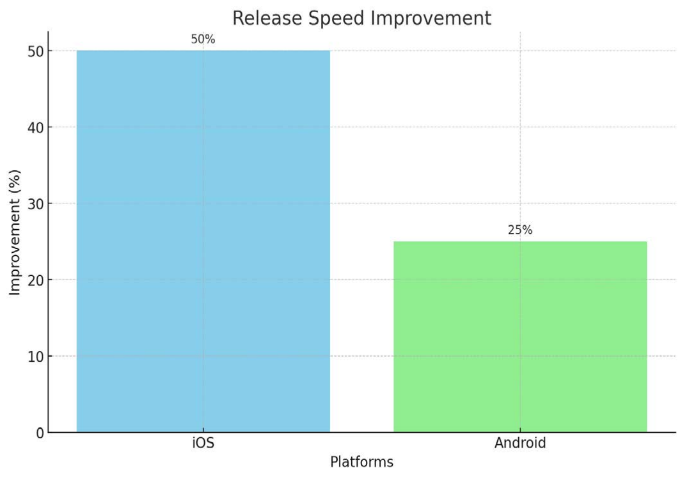

Figure 1: Release speed improvement after one year in transformation.

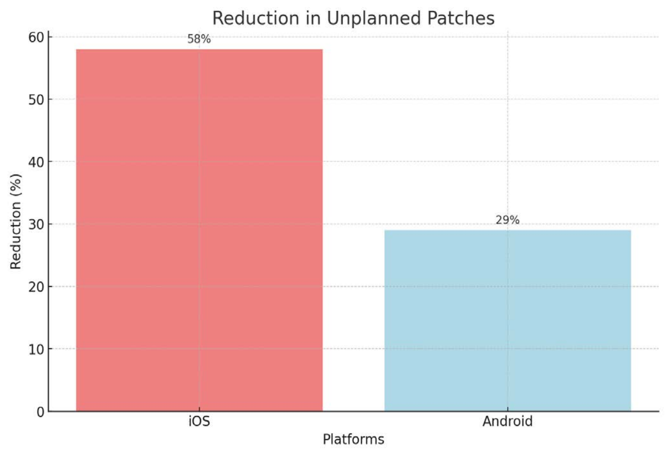

Figure 2: Unplanned patch release reductions after 1 year in transformation.

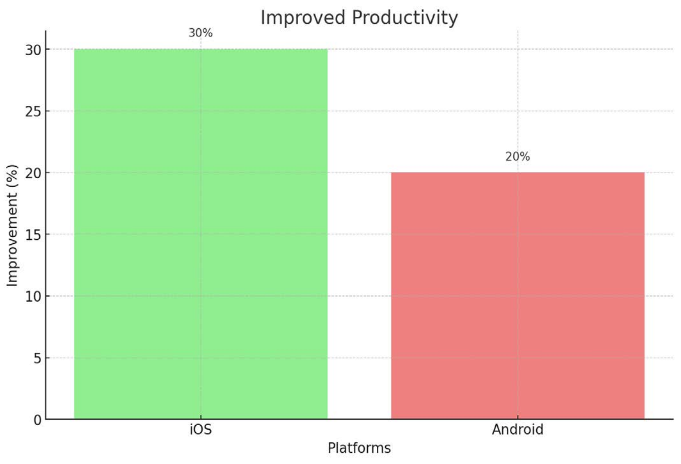

Figure 3: Developer productivity improvements after 1 year in transformation.

Recommendations for Adopting DevOps Transformation

Based on the findings and experiences from our DevOps transformation journey, here are some recommendations for organizations looking to implement similar changes:

Secure Management Support

Action: Ensure that the management team is fully on board and supportive of the DevOps transformation. Their alignment is crucial for driving the necessary cultural and structural changes across the organization.

Emphasize Continuous Education and Gradual Changes

Action: Start with constant education and process changes, implementing them gradually. This helps teams understand and adopt DevOps principles without overwhelming them. Use workshops, training sessions, and regular updates to keep everyone informed and engaged.

Adjust Team Velocity and Redefine "Done"

Action: Consider reducing the team's development velocity to allow for a greater focus on quality. Redefine the concept of "done" to include quality checks, root cause analysis, and preventive measures. This approach ensures that quality is not compromised for speed.

Tackle Technical Debt Incrementally

Action: Address technical debt in smaller, manageable chunks as part of the regular planning process. Collaborate with product management to prioritize technical debt alongside new feature development to ensure long-term sustainability and success.

Integrate Quality Engineering into Development Teams

Action: Merge QE teams with development teams to create DevOps-style teams where quality is a shared responsibility. This integration fosters collaboration and ensures that quality considerations are included from the beginning of the development process.

Upskill and Cross-Skill Team Members

Action: Invest in upskilling and cross-skilling programs to bridge gaps in skills and knowledge. Train QE teams on code development and developers on quality and testing practices. This cross-functional expertise is vital for building robust, high-quality products.

Focus on Effective Automation

Action: Review and refine the automation test strategy. Choose tools and technologies that developers are familiar with to encourage their participation in writing test scripts. Ensure that automation is stable and reliable to provide early and consistent feedback.

Provide On-Demand Test Environments

Action: Develop infrastructure solutions to provide on-demand test environments for teams. Use self-service portals and REST APIs to simplify the process of creating and managing these environments, ensuring they are readily available for testing needs.

Foster Cross-Team Collaboration

Action: Break down silos by forming coordination teams to align release cycles, share best practices, and create reusable code. Encourage collaboration and knowledge sharing across teams to enhance efficiency and innovation.

Measure and Evaluate Progress

Action: Regularly measure and evaluate the progress of the DevOps transformation using both qualitative and quantitative metrics. Use graphs and data to track improvements in release speed, quality, productivity, and other key performance indicators. And if required, make adjustments wherever necessary and suit the need of team and enterprise.

4. Conclusion

A successful DevOps transformation requires careful planning, management support, and a commitment to continuous improvement. This transformation is not a one-time event but an ongoing journey that evolves with the organization's needs and the technological landscape. By addressing challenges with strategic solutions, organizations can achieve significant improvements in release speed, quality, and team collaboration. Effective planning involves not only setting clear goals and objectives but also ensuring that all stakeholders understand and are aligned with the DevOps vision. Management support is crucial for providing the necessary resources, fostering a culture of collaboration, and removing barriers that teams might face. A commitment to continuous improvement means being open to feedback, constantly evaluating processes, and being willing to make adjustments as needed.

Start your DevOps transformation with thorough and thoughtful planning. Define what success looks like for your organization and outline the steps needed to achieve it. Involve all relevant teams and stakeholders from the beginning to ensure buy-in and alignment. Make sure that management is fully on board, as their support will be critical in driving the cultural and operational changes required for a successful transformation. Understand that DevOps is a journey, not a destination. It's a continuous process of learning, adapting, and improving.

Be mindful of potential impediments, such as resistance to change, skill gaps, and technical challenges. Develop strategies to address these issues proactively. Encourage a culture of experimentation and innovation, where teams feel empowered to try new approaches and learn from their experiences. Enjoy the process of transformation, celebrating small wins along the way to maintain momentum and morale.

Cntinue to learn and evolve, staying abreast of the latest DevOps trends, tools, and best practices. Foster an environment where continuous learning is valued and supported. By embracing this mindset, organizations can create a dynamic and resilient DevOps culture that drives long-term success and adaptability in an ever-changing technological landscape.

5. References

1. Bou Ghantous G, Asif Gill. DevOps: Concepts, practices, tools, benefits and challenges. *PACIS2017* 2017.
2. Lwakatare, Lucy Ellen. DevOps adoption and implementation in software development practice: concept, practices, benefits and challenges. 2017.
3. Fernandes Marcelo, et al. Challenges and recommendations in devops education: A systematic literature review. *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*. 2020.
4. Yiran, Zhou, and Liu Yilei. The challenges and mitigation strategies of using DevOps during software development. 2017.
5. Hamunen, Joonas. *Challenges in adopting a Devops approach to software development and operations*. MS thesis. 2016.
6. Wang, Cheng, and Changling Liu. Adopting DevOps in Agile: Challenges and Solutions. 2018.
7. Mandepudi, Snehitha. Communication Challenges in DevOps & Mitigation Strategies. 2019.
8. Riungu-Kalliosaari, Leah, et al. DevOps adoption benefits and challenges in practice: A case study. *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17*. Springer International Publishing, 2016
9. Senapathi Mali, Jim Buchan, Hady Osman. DevOps capabilities, practices, and challenges: Insights from a case study. *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*. 2018.
10. Ibrahim Mahmoud Mohammad Ahmad, Sharifah Mashita Syed-Mohamad, Mohd Heikal Husin. Managing quality assurance challenges of DevOps through analytics. *Proceedings of the 2019 8th International Conference on Software and Computer Applications*. 2019.
11. Maroukian Krikor, Stephen Gulliver. Defining leadership and its challenges while transitioning to DevOps. 2020.