Shifting Left into the Cloud

A whitepaper by Simon Pearson, Coforge Business Consulting

May, 2023

Coforge

Table of Contents

1. Introduction

Shift-left is a testing approach that emphasises testing early and often in the Software Development Life Cycle to catch defects and vulnerabilities as early as possible. Whilst this concept is not new, there are some emerging choices that engineering leaders can analyse and decide upon.

This is especially true for the cloud migration journey. Cloud migration is moving a company's digital assets, services, databases, IT resources, and applications partially or wholly into the cloud.

This whitepaper provides a straightforward approach to answer the critical questions on what activities form Shift-left testing, which to apply, and how to decide what resources may be required when adopting automation of testing.

Its primary position is to advocate for Shift-left, written in the context of cloud migration, as this is where we see an opportunity to effect change.

In this whitepaper, we will look at

- The types of Cloud migration routes
- What factors to consider when assessing an application for shift-left testing
- Which level of shift-left to apply
- The resources you may need to affect success.

1.1 What do we mean by shifting left?

Whether you are an engineer at the coal face, a product owner, or you are in a delivery role, you will be acutely aware of the need for testing software against requirements. In addition, you may already have a testing strategy for functional and non-functional tests operating at all levels of the testing pyramid (and in the right proportions).

The crucial part of this is **when** you do your testing.

Shift-left is a testing approach that emphasises testing early and often in the Software Development Life Cycle to catch defects and vulnerabilities as early as possible. Doing so improves the pace of delivery whilst keeping software stable by achieving good quality in engineering.

This approach contrasts with traditional methods, treating testing as a separate phase (often after completing the code) which can lead to delays and quality issues as defects in both code and requirements are discovered much later in the lifecycle.

By shifting the testing stage "left" in the Software Development Life Cycle, testing starts in the planning stage, where teams can identify and fix issues before they become more costly and time-consuming.

What shifting left means in real terms is:

- **Developers own the outcome:** Software engineers take accountability for their code, which starts with ensuring good test case coverage.
- **Quality is planned for all stages of the Testing Pyramid:** Automated testing strategies consider covering Unit, Component, Integration, and End to End tests.
- **Implementing Test-Driven Development (TDD)**: Write failing automated tests before you write any code – a fundamental building block of TDD, and written about extensively in "Test Driven Developement: By Example" by Kent Beck.
- **Adopting Analysis Tools**: Secondary tools beyond the IDE provide coding quality and coverage feedback.
- **Cohesion and Reusability through Frameworks**: Prioritising the development of reusable standard components with high-quality levels built in.
- **Leveraging continuous integration and continuous delivery (CI/CD) pipelines:** By automating integration and deployment, teams can catch issues earlier in the development process and reduce the time between identifying and fixing a problem.

1.2 Cloud: Knowing your Cirrus from your Stratus

Cloud computing involves hosting and deploying applications and services over the Internet rather than on physical servers or devices, offering numerous benefits, including:

- **Scalability**: Application resources can scale up or down as needed, based on demand
- **Flexibility**: Developers can quickly provision and deploy resources without manual hardware provisioning
- **Cost-efficiency**: Businesses can save costs, as organisations only pay for the resources they use

Typically cloud computing is divided into three categories:

- **Infrastructure as a Service (IaaS):** This involves providing virtualised computing resources over the Internet, including servers, storage, and networking. You must still manage the Operating System, runtime environment and any middleware
- **Platform as a Service (PaaS):** This involves providing a platform for building, testing, and deploying applications, including tools and services for development and deployment. You must still manage the application and data
- **Software as a Service (SaaS):** This involves providing access to software applications over the Internet without needing local installation or maintenance. You're not developing any code; you are just using the software

2. Shifting Left into the Cloud

Migrating applications to the cloud will likely require changes to your application's core code, be it simply updating libraries and recompiling or making holistic changes to take advantage of serverless services and capabilities.

Opening your IDE and inspecting that code resembles a mechanic lifting the hood of a car to work on its engine and make improvements. Whilst the hood is open, you might as well check the oil, fill the washer reservoir and inspect the timing belt to see all is well. Maybe you want to upgrade your turbocharger so you can go faster, and why not? The hood is open!

Like our mechanic working on the car, migrating to the cloud offers an opportunity to introduce better ways of working and improving the application – because whilst the hood is up, you are already in the mindset for change.

Shifting left during a migration provides a huge opportunity to add long-term value.

1. **It allows you to predict risk.** Being predictable means that you already know you can innovate faster when making future decisions on your applications
2. **You reduce wastage.** Since you are automating testing, you won't need repetitive regression testing that takes days to complete
3. **Elimination of reengineering bottlenecks.** Our Quality Engineering team has shown fixing defects found in production costs 20x more than those found at the requirements stage

Mindset: "Let's *just crack on and get our application into the cloud – we'll just use virtual servers."*

Reality: ...said every engineering manager, ever. Probably.

Have you ever heard the proverb "more haste, less speed?" My modern take on that is **"go fast, safely."**

Cloud migrations are not just about saving money; with modern deployment techniques such as automation of Infrastructure As Code (IAC), you can become more agile in your delivery, which means less time from concept to production. And faster delivery allows Engineers more time for innovation *(because who doesn't love a bit of Greenfield development?!)*

But you cannot just go fast – you must go safely.

I've seen Engineering teams shifting left and Cloud teams migrating apps as separate initiatives, with little consideration for their interdependence. But these are highly complementary – running automated tests as part of your continuous integration and deployment to cloud resources is at the heart of the practice.

If you are migrating legacy applications to the cloud, you must ensure you can do so safely. Regardless of whether you lift and shift or completely reengineer your applications, you must ensure that developers take accountability for quality, compliance and security through good levels of testing.

Mindset: *"You want me to change our app so it is Serverless? Oh, yeah, we'll add that to the backlog."*

Reality: We all know that new features take priority over technical debt.

If rehosting takes time, refactoring takes even longer. **Simply shifting left isn't going to speed up that migration either, making it hard to justify from a Business perspective: and this is the crux of the matter.**

We must understand when and how far to shift left and under what circumstances this would be appropriate. We can introduce levers to help you decide how fast you go to the cloud and how far left to shift.

We must also acknowledge that shifting too far to the left could slow a Cloud migration down. But, conversely, not moving the needle far enough left prevents adopting an agile stance and loses out on the benefits associated with shifting left. It also accumulates an ugly amount of technical debt that may eventually consign your applications to their ultimate demise.

2.1 Introducing our Levers

We will use three levers to help us consider our shift-left strategy.

Our first lever is where we consider our **Route To The Cloud**, which may depend on factors such as the complexity of the application, budget, business requirements and appetite for destroying existing code bases and legacy infrastructure. Here are the common routes:

- **Rehosting:** using the chosen cloud provider's infrastructure (IaaS) involves moving the application to the cloud without changing its architecture. Also known as a lift and shift
- **Replatforming:** involves making minimal configuration changes to the application architecture to enable it for the cloud platform (PaaS) environment without changing the core functionality
- **Reengineering:** involves making substantive code changes to the application to optimise it for the cloud platform environment necessitating redevelopment of the core functionality
- **Refactoring:** or re-architecting as it is sometimes known, involves redesigning the application architecture from the ground up to leverage cloud-native features, such as serverless scalability and elasticity

Two other considerations that may also be relevant to your organisation:

- **Repurchasing**: involves replacing the existing application with a cloud-native alternative, such as switching from an on-premise CRM system to a cloud-based one (Saas)
- **Retiring:** involves decommissioning applications that are no longer required

Our second lever relates to the application we are migrating, for which we will provide an **Application Assessment Score** derived from multiple inputs. Key to the scoring is defining the bounds to attribute to a low, medium, high and critical scale.

- **Application Criticality:** not all applications are created equal; some may be more critical to an organisation's operations than others, namely how they affect your bottom line. Organisations can mitigate risk and ensure that their most critical applications are secure and reliable by prioritising migrating "critical" applications to the cloud and incorporating shifting-left practices into their development process. The higher the criticality, the higher the score
- **User Base:** the user base is another important input. By prioritising applications with a larger user base, organisations can ensure they provide the most value to the most people. Additionally, applications with a more extensive user base often significantly impact an organisation's operations, making them more critical to migrate to the cloud and incorporate shifting-left practices. The more users affected by the change, the higher the score
- **Transactions:** organisations can ensure they provide the most value to their customers by prioritising applications with higher transactions. Additionally, applications with increased transactions often significantly impact an organisation's revenue and profitability, making it more critical to migrate to the cloud and incorporate shifting-left practices. The more transactions, the higher the score
- **Rate of Change:** the final element to score in our second lever is how often the application is changed. If you keep something updated with new features, visuals or patches, you will want to ensure it has the most efficient path to production. The more frequently you release, the higher the score

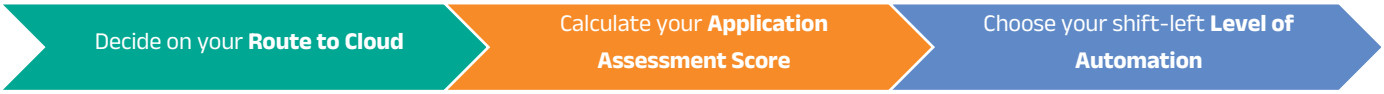
Shifting left is not a single activity; we can apply varying levels of shift left related to testing automation. So our third and final lever will be the **Level of Automation**. It is worth noting these levels are not prescriptive; you can change them to suit your needs. What is important to note is that the levels scale, meaning the implementation difficulty increases at each step, and so does the implementation time.

- **Level 1**: manual testing performed through peer code reviews and unit and exploratory testing.
- **Level 2:** automated functional testing within the codebase provides reliable and repeatable unit and integration testing. Automation will include dynamic analysis with tools supporting visual regression
- **Level 3**: simplistic testing replaced by advanced framework-based capabilities that focus on creating environments that support all functional and non-functional testing across multiple system components
- **Level 4:** testing is integrated into the CI/CD pipeline, enabling automated build and testing processes to follow your git branching strategy. Static analysis tools are added to your toolchain to provide information on coding errors, security vulnerabilities and generic coverage

Knowledge and skills gaps must be in consideration here. Therefore, one would recommend a good gap analysis to help you understand the areas where you are strong or weak and help define your desired future state. In turn, that analysis will drive any training areas required to bring your teams to the level they need to be.

2.2 Setting your Levers to enter the Matrix

You will be taking a three-step sequential approach:



a. Decide on your **Route to Cloud**

   Not all Cloud Migrations are suitable for all types of applications or shift-left levels. Therefore, your decisions in choosing which route your applications take to the cloud are fundamental.

b. Calculate your **Application Assessment Score**

   The key here is understanding if your application has a Low, Medium, High or Critical score. Again, your scoring mechanism will depend on your organisation.

c. Choose your shift-left **Level of Automation**

   The cloud migration route and the application scoring will output a *recommendation* on which shift-left type you wish to apply.

Let's translate that into our Decision Matrix:

	(c) Level of Automation	(b) Application Assessment Rating			
		Critical	**High**	**Medium**	**Low**
(a) Route to Cloud	**Rehosting**	Level 3	Level 2	Level 2	Level 1
	Replatforming	Level 3	Level 3	Level 2	Level 1
	Reengineering	Level 4	Level 3	Level 2	Level 2
	Refactoring	Level 4	Level 4	Level 3	Level 2

Level 1: Manual testing; **Level 2**: Automated Testing; **Level 3**: Frameworks for Functional and Non-functional testing; **Level 4**: CI.CD Pipeline Integration.

2.3 Resourcing for Success

When we discussed the gap analysis required to ensure you reach your desired outcomes, you may already have started thinking about what type of books to read[1] and training courses to enrol your team on. But there's a vital distinction between upskilling and adding skills to your teams.

Training alone may not allow you to achieve the desired velocity: High-flying teams perform because of the hybrid nature of the availability of expertise – the right people can play pivotal roles and make a difference in your capacity.

In this instance, recommendations on who would make a difference to your Release Train are crucial to the construction of the team. Deploying these key roles are vital for success.

Therefore we can extend our model to include recommended roles for the varying levels of shift-left.

	L1	L2	L3	L4
Quality Assurance: responsible for reviewing requirements, ensuring acceptance criteria is testable and measurable, reviewing testing strategies and plans, and performing non-automated testing such as exploratory and regression.	✓	✓	✓	✓
SDET (Software Development Engineer in Test): responsible for designing, developing, and maintaining automated tests as a part of the development process.	-	✓	✓	✓
Non-Functional SME (Subject Matter Expert): responsible for testing non-functional aspects of the application, such as performance, scalability, and security.	-	-	✓	✓
Test Architect: responsible for designing and implementing testing frameworks and strategies.	-	-	✓	✓
DevOps Engineer: responsible for managing the CI/CD pipeline, configuring build tools, managing version control systems, and automating testing and deployment processes.	-	-	-	✓

Of course, these roles are part of the broader Engineering team, which may be considering investing in a multi-team, multi-application migration. At scale, an enterprise organisation with many teams and applications may benefit from establishing two key functions:

- a **Center of Excellence (CoE)** to manage the shift-left and cloud migration program. A CoE can provide centralised guidance and oversight while allowing individual teams to maintain autonomy and flexibility
- a **Cloud Factory** for standardising processes, automating cloud infrastructure provisioning, implementing tools for management costs, and centralising monitoring and management of Configuration Management Databases

Together, these teams can facilitate a command and control approach to manage the program with solid governance, decision-making processes, and reporting structures.

[1] Kent Becks's **Test Driven Development: By Example** is an absolute banger from start to finish.

3. A quick recap

I know there's a lot to take in here, so I'd like to summarise my approach:

- **Set your mindset.** Define the objectives and goals of the initiative so that you have clear boundaries on what you want to achieve
- **Conduct a gap analysis.** Identify the current state of the organisation's shift-left and cloud capabilities and the desired future state. Assess the availability of internal and external resources and determine any gaps that need addressing.
- **Use the levers.** These will impact the pace and scale of the initiative.
    - Decide on your **Route to the Cloud**
    - Calculate your **Application Assessment Score**
    - Choose your shift-left **Level of Automation**
- **Develop a roadmap.** Focus on the types of resources for the initiative and how to govern the initiative.
    - Determine the types and quantities of resources required to execute the roadmap, considering the levers and the desired future state.
    - Develop a resource plan that outlines the types and quantities of resources required, the timeline for resource acquisition, and any costs associated with these.
    - Implement your Center of Excellence and Cloud Factories.

Coforge has extensive experience helping companies shift left into the cloud: **Let's engage** and see how we can help you and your business.

About the author

Simon Pearson is the Practice Head for Europe in Coforge Business Consulting. With a background in Software Engineering, Simon has spent 25 years leading Digital Transformations, pioneering customer experience technologies across industries and developing a passion for all things relating to customer journeys. Simon regularly advises clients on programme delivery and change, advocating for quality engineering through early intervention strategies.

About Coforge

Coforge is a global digital services and solutions provider, that enables its clients to transform at the intersect of domain expertise and emerging technologies to achieve real-world business impact. A focus on very select industries, a detailed understanding of the underlying processes of those industries and partnerships with leading platforms provides us a distinct perspective. Coforge leads with its product engineering approach and leverages Cloud, Data, Integration and Automation technologies to transform client businesses into intelligent, high growth enterprises. Coforge's proprietary platforms power critical business processes across its core verticals. The firm has a presence in 21 countries with 25 delivery centers across nine countries.

Learn more: www.coforge.com

For more information, contact information@coforge.com

Coforge