IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

DevOps Enablement in Legacy Insurance Infrastructure for Agile Policy and Claims Deployment

Balaji Adusupalli[1], Lahari Pandiri[2], Sneha Singireddy[3]

DevOps architect - Small commercial Insurance, ORCID: 0009-0000-9127-9040[1]

SR Systems Test Engineer, ORCID ID: 0009-0001-6339-4997[2]

Masters, ORCID ID: 0009-0009-8450-5404[3]

Abstract: DevOps is an important concept in the world of software development and deployment that aims at marrying development (Dev) with operations (Ops). Its application throughout organizations has yielded great results. Not only has it bypassed the problems of rigid software processes, but it has increased product velocity, thereby increasing customer satisfaction. Operating with DevOps principles leads to high performing technology organizations. Over the years, technology has advanced. Businesses now desire to act at DevOps speed. Inside the realm of enterprise technology, however, several long-standing problems still exist. Dealing with those problems is often a painful uphill battle.
In this paper, we explore DevOps enablement in a legacy insurance enterprise with a multi-decade on-premises primary claim processing infrastructure and address subjects like culture, legacy transition, and individual forms of legacy enablement that allowed a large enterprise to embrace larger transformations while modernizing their infrastructure in a secure, risk-mitigated way. All of this while giving business units, stakeholders to the transformation, the ability to deploy policy or claim changes frequently, thereby providing continual business value. As noted earlier, applying DevOps at the enterprise level has been considered difficult and not something that has been widely practiced. Lessons learned, best practices established, and caveats across several industry verticals of secure DevOps enablement across legacy infrastructure are detailed in this paper. Embracing these practices will provide a seamless but secure DevOps capability for many enterprises running legacy infrastructure inside their firewalls.

Keywords : DevOps, enablement, legacy systems, insurance, infrastructure, Agile, policy deployment, claims deployment, modernization, continuous integration, continuous delivery, CI/CD, automation, digital transformation, system integration, scalability, microservices, containers, cloud adoption, orchestration, API integration, legacy modernization, deployment pipelines, version control, configuration management, monitoring, testing automation, security, compliance, DevSecOps, collaboration, agile workflows.

I. INTRODUCTION

Insurance enterprises familiar with agile principles desired rapid DevOps enablement to enhance productivity and address the complexity of rapidly evolving defect and change requests for legacy policy and claims processing infrastructure. However, a substantial release cycle and inadequate institutional knowledge resulted in difficulties deploying change across disparate insurance products and platforms. A legacy system mitigative approach to application migration limits could expedite non-intrusive DevOps enablement without external dependency or risk of production outages. Creation of a right-sized, federated DevOps model along with legacy PMO-RM-Release Train inspired rituals support the removal of silos among business, IT, and other shared services. IT architecture supports major technology roadmaps in synchrony with business capabilities, along with quick response to business needs arising from new technologies or acquisition activity.

Today's enterprises must leverage people and technology for rapid delivery of new capabilities to the market. Agile methodologies increase the frequency and speed of change delivery compared to traditional methods. The speed of change and technology proliferation require enterprise architectures to be dynamic in nature, avoiding monolithic approach constraints to centralized decision-making, but rather mirroring the decentralization of business model and the distribution of financial transactions and resources. DevOps is an emerging paradigm for coordinating tasks in the software development and IT operations phases, leveraging the ability to analyze and automate every stage of the delivery and deploy pipeline while ensuring high quality and maintainability of IT capabilities. The delivery pipeline is only as strong as its weakest link, typically represented by data connectivity to software ecosystem participants that have been

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

in traditional silos for decades. Insuring data model enablement together with enterprise security and privacy policies and risk governance models must occur before the enterprise can fully leverage cloud, machine learning, and other emerging technology enablers.

A multi-year journey of DevOps enablement transforms compliance and policy libraries, risk-aware development and service design, automated and predictive infrastructure build and deploy related functions, and informed management of shared service and third party dependency risk.

II. UNDERSTANDING LEGACY INSURANCE INFRASTRUCTURE

The insurance industry is one of the oldest and most conservative service sectors. The insurance mission is to provide the risk services to individuals and businesses. Insurance is and continues to be for the most part a “paper-based” world, where the economic exchanges between individuals and insurance companies are backed by contracts, which are essentially pieces of paper. The insurance companies are the custodians of the contract backing the economic exchange. The business model for the insurance industry is a very simple one involving taking premium money now and paying it back, with growth, often decades into the future. To generate the required growth, insurance companies take the premium money and invest it in an array of capital markets.

Fig 1 : The Many Benefits of Insurance Legacy System Transformation

The information services of the insurance industry, the underwriting, policy service and claims service, are also grounded in paper documents. Businesses rely on file-based systems and still have, in many instances, as much investment in storage cabinets and filing systems as they do in IT. These file-based processes exposed insurance firms to the same set of problems that have plagued a huge number of businesses over the last 15 years, namely the problems of speed and scale affecting all businesses, the capacity to service continually increasing numbers of transactions, the speed at which these transactions can be processed, and the ease with which these transactions can be processed. The paper-based systems were fundamentally brittle and costly to service. Business and personal customers expect service levels akin to those offered by contemporary banks, hiring travel agents, and online marketplaces, with phone, website, and 24/7 service.

2.1. Definition and Characteristics

Information systems are defined as the collection and integration of components for collecting, storing, processing, and delivering relevant information to the stakeholders of an organization for supporting decision-making, coordination, control, and analysis within that organization. Interestingly, nowadays, all operational organizations are information-based; they use information technology for supporting their mission and business goals. Moreover, whether intentional or not, an organization "outsources" part of its memory through its information system, and, therefore, organizational learning is at its core based on the experience accumulated in the information system containing organizational data as well as its specific contents and the management and development procedures.

In this context, legacy information systems represent the knowledge embedded in the software technology: its structure, design, functionalities, procedures, and, moreover, it specifies the company processes supporting the business model adopted by the organization over time. The legacy software and technology, indeed, might present both negative and positive implications for the organization. From the negative side, the legacy information systems tend to be the result of

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

years of incremental development and they may become enormous, unwieldy, full of bugs, errors, and failures, and incapable of adapting to support changing business requirements, and, therefore, they hinder and hamper the agility of the organization. From the positive side, a legacy system, as a store of the corporate knowledge, tacit and formal, represents a strategic resource, irreplaceable, confidential, and unique to the organization and, therefore, it operates as a barrier to copying, thus, creates a competitive advantage.

2.2. Challenges in Legacy Systems

Legacy systems are not necessarily monolithic or obsolete. They can continue to work very well for their original purposes while still comprising numerous unsustainable problems. Being made to last for several decades often forces these systems to decompose into fragile components for which other, newer systems store crucial dependencies. While being difficult to replace due to their core position in the business, the consequences of change become greater as time progresses. Business-oriented and technology-oriented flaws manifest increasingly severe pain points. Our observations suggest that, although true for all legacy systems, these challenges are tenfold for those in the insurance sector.

The volume of business queries and interactions are currently reaching new highs mostly due to explosive growth in service quality demand, the increasingly active role of clients and for some lines of business the ever-declining cycles in their life. Early insurance products were designed to survive decades without either modification or interaction with customers. These products require no longer be routed through branches revisiting the same employees every year but are due for their modern overhaul with portals replacing physical presences. Product renewals need no longer be handled by the same company. New product lines have to be enabled much faster than just years today while being governed by ever-fiercer cost and compliance pressure. Policyholder death has become a major business driver, which in combination with decimated administrative staffing levels has turned into dropping service quality.

Modern clients and competitors want instant results, which forces IT organizations to provide rapid reactions. Most legacy infrastructures allow for little flexibility towards either rapid changes in policy/product structure or connecting to new external partners with alternative data workflows. The painful business consequences of delays or failures in administration tasks can be pushed through IT because it is invisibly executing the services, but clients will decide on what to do even more quickly. Service outages or hiccups in what should be a workflow-based enterprise today are a business risk.

III. THE IMPORTANCE OF DEVOPS IN INSURANCE

Insurance infrastructure has some unique challenges in terms of agility and speed. Insurance operations have not changed as much over the years and have had the same systems for underwriting, payment of claims, actuarial valuations, and portfolio management for ages. However, these large systems need updates and upgrades on a constant basis to add regulatory features, new products, make it easier for customers to access services, use RPA to reduce human effort, and make data available to customers and partners securely. With Covid-19, the customer obviously wants help in getting services from the insurance company digitally and has impatience with traditional processes that involve paper and voice. The traditional role of an insurance company, which has been to compensate the customer at the time of need, and give back their investment as a valued, has also been forced by corporates to be digitally managed and supported.

Companies have been investing heavily on their IT capabilities but project deployment times are still not at an acceptable level. DevOps as a practice has shifted focus from Application Development only to making the entire System Development Life Cycle more efficient, integrated, and collaborative. DevOps provides for faster product delivery, predictable timeframes, reduced bottlenecks through enhanced collaboration through more frequent interactions and shared code repositories, and higher quality products through greater number of unit tests. These benefits are still restricted to a few companies for the following reasons - in the case of large programmes of work, the individual projects overlap only for a small duration, and hence the significance of operational collaborative tools is diminished; tools are new and are not widely available, the integration of tools with readiness, security testing, production monitoring, and operational support systems require a lot of effort; large programmes still require a significant amount of manual effort for both development and operations; and policy and compliance aspects require closer dependencies of the application teams with the regulatory teams, especially in the insurance domain.

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

Fig 2: What is DevOps Integration in Java Development

3.1. Agility and Speed in Deployment

The insurance industry has long been looking to become as advanced as those in the tech sphere. Functions normally associated with technology companies, such as customer onboarding, customer experience and customer journey, are being expanded into the insurance business as well. But insurance is not about products that are manufactured and sold. Most products just sit there, waiting to be consumed. Every so often, a consumer actually uses the product. This is when Cloud Computing needs to deliver. We must put the infrastructure, processes, checks and balances in place to enable a people, enabled by technology, process and system interface, that can deliver at the speed expected of every consumer. However, deployment of change in a 24/7 economy, into a complicated patchwork of technology pieces that have been evolved since the 1970s, is not a simple task. It involves not just the technology, but the foundation of how insurance companies run.

Drawing in thought leaders from outside the demand function, but heavy in trade knowledge, is essential to deliver and bring to fruition agile deployments of development system changes. It's strange that insurers still see developer operations, DevOps, as an "add-on" to milestone-based insurance software deployment. Instead, speedy agile delivery of changes to development systems should be at the heart of the change. The idea is to have a consortium of representatives from all interested parties — claims and other operational areas who would be the initial end-users, marketing for customer journey experience management and call center experience, and actuaries for data integrity. They would have a seat at the table to guide the development changes, can make easy agile decisions, quickly review changes as they develop, and ensure the agile development team builds alongside. This company consortium makes DevOps in the insurance space not just a technology tool, but an essential component to be taken advantage of through the unique relationship aspect of the business.

3.2. Collaboration Between Teams

The organization of teams inside insurance is one of the key aspects to make any initiative succeed and especially be effective. Three key roles in DevOps are overlapping, aligning, and cooperating. An end-to-end oversight organization looking across product delivery, business operations, and enterprise architecture should help various Agile delivery teams—those dedicated to formal product releases, those responsible for continuous delivery of critical services, and those that enable continuous delivery of shared components in the technology stack.

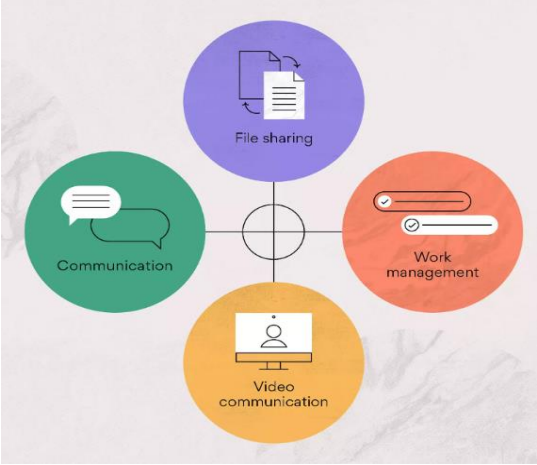

Fig3: Team Collaboration

DOI 10.17148/IJIREEICE.2019.71209

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

In the case of core systems, insurance normally does not run on Agile processes. Enhancing a cross-functional joint collaboration between the teams and supporting people to feel comfortable about sharing each other's responsibilities would be very important, as their resourcing commonly expresses little flexibility. Therefore, it may be interesting to use a model of symbiotic team patterns to plan and operate the teams involved in the initiative. This means structural configuration of the organization that enhances interaction and collaboration is necessary to balance the speed of delivery with the risk of failure for both core and company related delivery teams.

In financial, risk and compliance domains, common endeavors are basically limited to incident management and some emergency actions. However, when transformation efforts are in place, a stronger interaction is sustainable and can be sustained. Over the last five years, we promoted an essential change in the resiliency governance and assurance, expanding the resiliency theory and practice applied at the core systems towards the whole enterprise asset – governance, appropriate modeling, consider resilience as a business aspect, not merely an IT concern, and promote shared assets in deliveries, like measurement and incident management platforms. We found that achieving the corporate resiliency is a strategic action that every company should privilege.

IV. KEY PRINCIPLES OF DEVOPS

Software development practices have changed significantly over the years, but the same problems seem to arise again and again. Historically, software development and technology operations were two distinctly separate functions of an organization. Development teams wrote the software, while operations teams deployed and maintained the software. The two functions had very different goals: development teams focused on building new features and better products, while operations teams focused on ensuring reliability and stability of the existing systems. These different priorities led to friction between teams and a lack of alignment around business goals.

DevOps frees developers from the constraints of traditional silos and red-tape one is used to with legacy deployments. When your operations team is in a distinct silo, it will always be tempting to add process around everything, even the low-risk deployments. When checking in, validation and deployment are all seamless and automated (and all team members collaborate), high-frequency deployments become low-cost and low-risk. Small failures that do occur can be easily detected and monitored. The focus can instead lie on the short- and long-term impact of the changes instead of just making sure nothing bad happens as quickly as possible. This takes away a lot of the perceived value in centralized ops, as they do not add to the business push in their own right: they only promote business continuity. The DevOps process allows a business to change its ways without losing focus, getting stuck waiting for the perfect moment or losing stability. Diminishing deployments are about moving power to the frontline. The final goal is to make fail-stop faults show up to your users as sporadic failures, and have simple core business logic minimize downtime. A lot of other principles help towards that goal, especially around CI/CD, IaC and monitoring.

4.1. Continuous Integration and Continuous Deployment (CI/CD)

CI/CD, or Continuous Integration and Continuous Deployment, is a set of practices that enable specialized teams to deliver code changes more frequently and reliably by automating the process of integrating and deploying code. Continuous Integration refers to the practice of merging all developer working copies into a shared mainline several times a day, making a build and running automated tests against it. CI's goal is to avoid integration problems, and point out errors quickly. CI/CD is an assembly line production process for the software development lifecycle.

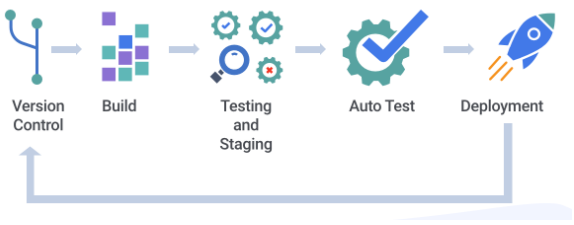

Fig4: Complete Overview of Continuous Integration and Delivery (CI/CD)

With CI/CD, developers' frequent code submissions trigger automated builds and testing of the applications. And, the more frequently that the software is built and tested, the easier and less risky it is to deploy the final product to a production environment. By deploying more often, organizations are able to get smaller, more targeted releases into the market. Deploying frequently also allows organizations to get feedback from users sooner than they would if they were following a more traditional software release schedule. And then they can use that feedback to add features and make

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

improvements. The second aspect of CI/CD is Continuous Deployment. In Continuous Deployment, every change that successfully passes the automated tests is deployed to staging or production automatically. Neither the developers nor the IT staff have to push the button, making it simpler than Continuous Delivery. Organizations vary in their approaches to deployment automation. In some, development teams take responsibility for deploying their code into production environments using shared operations infrastructure. In others, a centralized IT operations group conducts deployments.

4.2. Infrastructure as Code (IaC)

While continuous deployment enables teams to quickly deliver value to end-users, infrastructure as code allows them to adopt the same agile development practices that they benefit from. Infrastructure changes are often out of sync with code changes—when the underlying environment is not changed to support the incoming release changes, the feature under implementation may either fail or behave differently when deployed to Production. Adopting CI/CD principles for Infrastructure as Code means infrastructure is defined in code like application software, which both Dev and Ops teams can collaborate on and version control. When moving between environments, the IaC tool that processes the defined infrastructure code handles translating it into the appropriate target changes. Anything that needs to be deployed—application binaries, data, and infrastructure—should be defined in its respective repository as code. This drives systems behavior across every layer—from the runtime layer through the associated physical or virtual cloud stack—all the way up to the Kubernetes configurations for microservices running in containers.

Tools for IaC have massively evolved over the last decades, with options out there that are specifically made for specific needs—ensuring that the aforementioned goal of a single source of truth for a deployed system is achieved. A huge motivator for presenting infrastructure changes as code is the ability to use diff tools for detecting drift—the output of diffing presents the user with the differences between the code-determined state and the actual state of infrastructure on the target systems. Tools also have the ability to automatically rectify drift; however, users are often still encouraged to use diff tools.

4.3. Monitoring and Feedback Loops

The reciprocal effect and relationship between development and operations must be defined with monitoring and continuous feedback mechanisms as a common element. Monitoring goes beyond logging mechanisms. It identifies pre-formulated service level indicators, which underpin the definition of service level objectives. These are “good enough” limits for service level agreements and understanding how service level objectives relate to business processes. Instrumentation layer components must be provided at the design stage. The company must also have a data science team to analyze the information feeding the feedback loops and predictive algorithms, which is not a trivial task, given the great risk involved. This helps to understand the state of applications and infrastructure, the areas of user experience with problems, and their latency, through application performance management and real user monitoring. This should also be aligned with business indicators, such as the company’s net promoter score.

The feedback loops must exist in different directions and strata. From the operations to development, both qualitative and quantitative data must transfer to avoid technical debt in improving the application. The tool to do this could be a simple ticket system. Other typical feedbacks to development are event reporting, through an error tracking webhook and the previous telemetry of real user experience with application performance. This feedback informs maximum performance access time for the critical path of business transactions and latency from the alerting integrated tool to business metrics monitoring tools. Other sources could be the availability of companion applications or deployment uses that are important for the business, shown on business metric monitoring dashboards. From development to operations, release tracking tools must indicate deployment schedules, and dashboards alerting development to business-related errors must be created from monitoring tools.

V. ASSESSING CURRENT INFRASTRUCTURE

Throughout the legacy system modernization process, there are many assessments that will be made. Some assessments are high-level and should include stakeholders from across the organization, while others are low-level and involve only the technical staff. These assessments may be repeated multiple times as the process of defining the new system continues. The assessments provide important information about how the development teams will deploy, monitor, and scale the new systems—especially in an existing infrastructure with limitations brought on by technical debt.

The assessment on the current infrastructure in support of the new CI/CD process must be very broad and include every relevant feature of the current system. This is an important process that must incorporate the entire team. Each development team needs to provide input on what changes should be made to the infrastructure, what changes can be made to help enable the new CI/CD process, what is not working today, and what obstacles have kept them from finishing development and QA testing on their backlog items. This will involve as many high-level discussions as is needed to

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

help quantify, if possible, the current limitations of the existing customer and internal systems. The discussions must also encompass a plethora of low-level conversations to determine the how, what, and where of the existing internal and customer infrastructures.

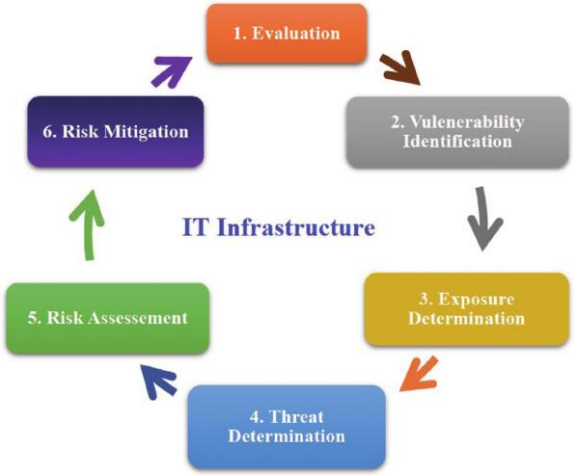

Fig 5: Risk assessment life cycle in IT infrastructure

The inventories produced will need to include the software applications and supporting services that are deployed in the existing development and QA environments. Each inventory must contain the general functions of that software or service, what systems the software or service interacts with, how the software is tested, how the software is patched, and who is responsible for those activities. In addition, if known, an inventory of how many customers are using the existing software or service must be included, along with the customer pain points. Without an accurate identification of the current situation, any efforts to modernize will be ingesting a large amount of risk. It is most important to first outline the areas of greatest risk.

5.1. Inventory of Existing Systems

DevOps automation tooling works with terminals: an SDK that interacts with deployed services or applications and a collected configuration management and monitoring tool. The SDK, in turn, typically works with Git or internal SDK code storage and cloud storage, or internal cloud repositories. DevOps tooling for an Agile Insurance service or application will run from a CI/CD - shipping coding commits into service regions - in particular Dev, Test, QA, and Production, where code validation is performed, and the regulatory compliance, including security checks, is followed for every code version deployed or revised in Production environments. The internal SDK and DevOps tooling used can set up every stage pipeline - usually from Pipeline as Code methods, perform storefront tests, domain-level checks, and external tests, from UI Testing and Pen Testing methods; activate the service for company-affiliated docs and support business, risk models, handling, Business Process Management, work process, and inbox handling customized business rules.

The installed solutions may include different versions of external and internal software services from all the supported Product areas - from Front-end and Mobile, ID&TRI and Claims, Risk Models, and Analytics for Data Warehousing and BI to Back and Microservices, including Inventory and Account, Payment, Policy Management and Billing, Infrastructure, and UI Integration services. The configuration of Data Storage and Data Factory, used for Configuration and PubSync services or File and DB Links integration tasks, and Warehouse, where needed, may vary and include different versions of required Micro- and Back External, Internal, UI Collection and Services. It includes the old and new versions of different products by domain name and service support area due to state and government regulations and data validity period, supporting interchange rules.

5.2. Evaluating Technical Debt

Measuring technical debt consists of monitoring key indicators of efficiency and stability over time and comparing their empirically observed values against an ideal. This may cover very distinct practices, such as analyzing source control logs to calculate code churn from which we can infer quality decay, running testing and validation tools, using continuous integration, and monitoring failure rates, response predictive models trained on several life cycle and incident variables to assess the respect of some best practices on the organization's project life cycle constitutes one of the most common forms of measuring technical debt.

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

A second important type of practices used to evaluate technical debt are the less formal KPIs crafted inside the organization, which usually monitor team internal performance, through user-reported bug tracking, expense tracking, user adoption preventing time delays or proper alignment between operations teams and the main support teams, such or the tracking of long maintenance periods for some functionalities, which usually indicate some inadequate work. Although some predetermined calculations and benchmarks can describe the performance, the custom nature of most of the KPIs tailored by team inside the organization allows a fine-tuning of the results that allows to identify problems that are occurring during a given time in the organization. While detecting problems or difficult situations, neither of these types of technical debt assessment practices may give insight into how to handle them, and, more importantly, how to handle the associated business risk.

VI. DEVELOPING A DEVOPS STRATEGY

Developing a viable and successful strategy for implementing DevOps practices to a legacy, core systems, IT environment is a challenge. It requires knowledge and experience with legacy IT, modern agile delivery practices, and modern cloud infrastructure to be successful. A one-size fits all approach cannot be found. The project must be crafted to the current state of the organization, the people involved, and the corporate culture. Quite often, multiple approaches must be developed and deployed in parallel to cover the various needs of the core IT environment.

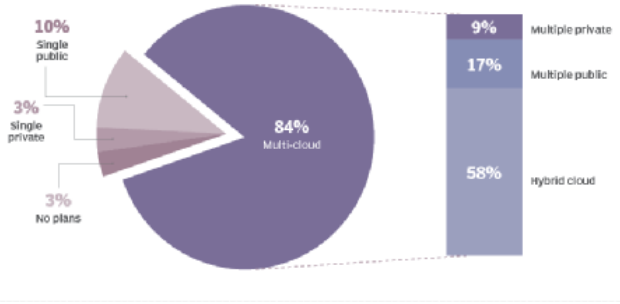

Fig: tips for building a multi-cloud DevOps strategy

Before beginning to build the plan, it is prudent to have a sense of the desired goals of the DevOps strategy. Goals may include the ability to run more experiments in a manner similar to A/B testing in marketing; support for constantly changing insurance customer expectations; adjust insurance models and claims processing based on internal and external stimuli; ability to service customers' post sale, developing to core disciplines of risk underwriting and claim fulfillment; reduce costs by streamlining infrastructure; and facilitate partnerships with others in the risk ecosystem. While no specific set of goals will serve all companies, this list attempts to represent a cross section of companies' possible objectives.

An equally critical task in building a DevOps strategy is the task of identifying the key stakeholders who will shape and affect the roadmap. The makeup of this group is very much organization specific, but a diverse group composed of a cloud solution architect, agile specialist, and representatives from the core disciplines is a recommended start. The group must balance the technical capabilities with pragmatic business knowledge of claims, risk management, and solution design. Its support is crucial because the roadmap cannot solely be a sequence of tasks, but must also be a change management and culture altering plan, without which technology may not bring about the desired results. Business representatives may need coaching to express their needs for change and process improvement.

6.1. Setting Clear Objectives

DevOps enables organizations to shorten time-to-market through continuous delivery of features, updates, and repairs for existing services and products. Many goals for a DevOps initiative are therefore directly aligned with how the larger organization is evaluated for success or failure. Operating in large streams, considering DevOps a path for group comfort and streamlining, rather than a tool for the performance of operations tools and typical project waterfall stasis, DevOps makes the job easier for everyone by sharing responsibility for the mission, and the successes/failures therein.
But objectives are more than simply accelerated time to market. Applications and services deliver value over time, and it's that value delivery that has been unsustainable for many organizations, leading to long product development life cycles. That software delivery is a factory, and DevOps makes the tools available to make it an efficient factory is the mission. If customers are not making and spending more money, this goal ends up loudly defeating better time to market, eliminating the speed advantage altogether.

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

DevOps makes development and delivery easy. Using release automation tools, testing tools, and continuous integration/delivery tools, iteration cycles shrink and features are easily made and integrated. Performance monitoring and testing tools ensure problems become knowable and easily solvable. Communications tools promote pervasive awareness of all activity. All of this reduces the time between ideas and value. Everything we do in terms of adopting these tools, no matter how much effort, expense, or grumpiness it generates on its own, is for the single, short objective of value delivery acceleration. Anything else is wasting the time and resources to devote to the tools and principles of DevOps.

6.2. Identifying Key Stakeholders

Identifying key stakeholders for executing a DevOps onboarding plan requires balancing IT and business needs to realize value fast without operational overhead inefficiencies and unnecessary distraction for people already stretched thin. A participant explains that she felt lucky in that it was clear that the focus needed to be on IT operations and development and that the executive levels were willing to push for change. However, it was an unusual case because the CIO wanted to spread the word about Agile and DevOps practices so that others could start to implement change. A support participant felt that without a champion at the C-level as well as the manager level, change would be slower but possible. In contrast, a participant noted that all teams outside of development, QA, and operations are to be included in the internal stakeholder discussions, but the footprint of internal stakeholders needs to be small enough to see progress before the spotlight turns on customer delivery.

Eqn1: Stakeholder Priority Score (SPS)

Where:

- I = **Influence** of the stakeholder (e.g. 1–5 scale)
- P = **Power** to affect the project (e.g. 1–5 scale)
- S = **Support** or attitude (positive or negative)
- W_i, W_p, W_s = **Weights** assigned to each factor based on project context

$$\text{SPS} = (I \times W_i) + (P \times W_p) + (S \times W_s)$$

Selecting and communicating with people currently involved in supporting customer solutions, either by promoting or getting the full picture of stakeholder needs and deliverables, can help prioritize change. Because business area stakeholders and senior management are often remote from day-to-day change implementation, their involvement should be minimal, while still encompassing the extent of what will be affected by DevOps changes. A developer suggested that ambassadors between internal and external stakeholders would help manage expectations. However, maintaining contact with remote locations, especially older legacy infrastructures, is necessary for effective delivery of repeated updates and fixes.

6.3. Creating a Roadmap

The third step is creating a roadmap. A roadmap is different from a project plan in that it isn't a schedule showing when specific tasks are to be accomplished. Instead, it outlines areas of particular strategic interest, painting a rough picture of the tasks, resources, and timelines required to move toward each goal. DevOps improvement is a long, complex process with many twists and turns. Your roadmap helps you see where you are going. Without a roadmap, you might get lost in the trenches, tending to everyday fires instead of making progression down the path of DevOps maturity.

There are several factors to consider when making a roadmap. Your high-level DevOps objectives help guide the priority of work. Areas of particular strategic interest help ensure that the most important business objectives are being addressed. There are usually many more possible improvement ideas than you can address in the next six to twelve months. You could graft identified improvement items onto a timeline, but that wouldn't account for dependencies among the items or for the priorities and durations of the improvement items. A roadmap helps capture this risk in a visual format, emphasizing that you have roughly mapped out a set of priorities.
These are not meant to be concrete promises but are intended to help you quickly explain to your stakeholders how you plan to do what you do. Your roadmap will serve several important purposes.

VII. TOOLS AND TECHNOLOGIES FOR DEVOPS

DevOps is the result of better collaboration between Development, Operations, and Quality Assurance. Doing DevOps right means having the right toolset that works well together. This chapter introduces some of the most popular tools, technologies, and solutions used to implement DevOps. Because of space limitations, we only cover those tools that are most widely used in the enterprise, not every tool available. By no means do we mean that the tools covered in this

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

chapter and the next are the only tools available. Any tool that is built on the principles of DevOps can be used in the place of the ones covered here. Version Control Systems. Version Control Systems are sometimes referred to as Source Code Control Systems. The use of VC systems is not limited to Development.

Eqn2: Conceptual DevOps Success Equation

$$\text{DevOps Success} = (T + A + C) \times CI$$

Where:

- T = **Tools** (automation tools for CI/CD, monitoring, infrastructure as code)
- A = **Automation** (build, test, deploy, monitor)
- C = **Collaboration** (between Dev, Ops, QA, and other teams)
- CI = **Continuous Improvement** (feedback loops, iterative learning, and adaptation)

It is used by Dev, Ops and sometimes even by Test to version everything related to software development and infrastructure. Everything used to create the enterprise's applications is stored in VC Systems, source code, scripts to deploy applications and configure servers, and text files that hold configuration for testing and production. Disparate teams may develop and maintain source code, testing scripts and production deployment scripts, but using the same VC system allows easy collaboration between teams. Using the same system offers the capability to provide fine grained access control, hold each team accountable for their contributions and have a single source of truth. Automation Tools. Automation is the key element that makes DevOps enable faster and continuous deployment of applications. Without automation, processes will remain slow, manual and time-consuming. Automation tools to support DevOps practices can be placed into three categories, Configuration Management, Test Automation and Application Release Automation. Any automation tool is only one piece of the puzzle, the complete chain is only as strong as the weakest link. So, deficiencies in any piece of the Automation Toolchain will lead to poor automation support of Development and Operations.

7.1. Version Control Systems

Version Control Systems (VCS) allow DevOps practitioners to track and control changes in source code over time. Out of the many VCS tools available, GIT is arguably the most popular, owing to its distributed architecture and increased collaboration capabilities. GIT also allows better branching and merging, which are essential to Agile Development in Enterprise Platforms, where multiple teams must develop on the same code base while isolating their changes to avoid cross-pollination of errors. It is free to use for open-source projects and offers a simple and competitive pricing model for software vendors that require a commercial license. A popular Development Operations platform is often used for GIT. While many organizations use this platform or continue to support commercial license for enterprise projects, other organizations prefer an enterprise solution that supports private repositories.

VCS tools offer several advanced capabilities such as intelligent diff and merging capabilities that save on build and deploy times during policy or claims deployments in production. By making snapshots of code and document changes over time, developers and DevOps practitioners can easily rollback to a version prior to a change that caused errors. Users can also create tags to capture the most stable versions of the code, which should be deployed to production. Support for webhooks in vendor products enable automated build and deploy solutions by triggering CI/CD pipelines. Labels on Work Items that are associated with versioned code allow the development and QA teams to find out which deployment is related to a given fix or feature request. This increases audibility and compliance of the solution.

7.2. Automation Tools

As development and operations are kept separate, the administration of servers and environments has become a specialized skill for system administrators. These administrators invest a significant amount of time and effort into the mundane task of configuring systems and deploying applications, all of which is prone to human error. Enterprises typically evolve their server provisioning and configuration infrastructure over time and, as it tends to grow organically, custom frameworks are created with varying degrees of functionality and maturity. These internal solutions may work well during their infancy, but they are not built to scale or handle the complexities of enterprise applications and development processes. Thus, such infrastructures need to be completely rebuilt with specific configuration management tools and provisioning automation.

By providing lightweight abstraction layers, these automation tools allow Dev and Ops to deploy, configure, and maintain applications on any size environment regardless of whether those servers are in the cloud or reside on-premises and provide DevOps with a common framework to define the properties of the application and the steps required to birth and nurture it into production. While there are a number of tools available, the choice of the infrastructure automation should depend on existing processes, desired integrations, and the end objective. Switching between tools adds complexity and increases the support burden. Technologies work in different ways and have specific strengths that may make them more

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

or less suitable for particular use cases. Regardless of implementation choices, infrastructure-as-Code tools allow teams to treat infrastructure as they do the code and track and version changes, share and collaborate on new components and automated procedures with peers, and exercise version control processes as they evolve.

7.3. Monitoring Solutions

In the current competitive landscape, it is not enough for IT Services to merely provide Level 1 and 2 support to customers. Significant value can be added to the client IT Service by ensuring that it is monitoring the multitude of systems and tools available to identify early issues and fix them prior to the client being impacted. With the constant increase in size and complexity of the interactions between systems, the historic Passive Approach to Support Services is no longer effective. Monitoring solutions need to be proactive.

The previously described tools offer some monitoring capability, but primarily rely on alerts being sent from the various tools when they identify an issue. A Monitoring Framework is used in conjunction with multiple other market tools to provide a comprehensive monitoring capability to insurance clients. This can even include predicting issues that are going to occur in a system based on historic performance information.

A Monitoring Framework has been developed. The solution was developed with the goal of unifying the monitoring for multiple disparate systems. The framework offers an out of the box solution that accelerates the monitoring implementation process. The service uses a multi-stage approach examining each level of the application or architecture stack. The use of a single tool means that issues can be detected via a single interface for systems and application logs, performance and health metrics, as well as UI monitoring.

All issues discovered through the Monitoring Framework are categorized by severity level, and can be assigned to one of the Support or Monitoring Teams to troubleshoot the issue or perform corrective actions if the issue is severe enough. The team will need to check any alerts that may have been generated. In conditions where the issue cannot be resolved via the suggested task, the team will escalate it to Level 3 Support.

VIII. CULTURAL SHIFT TOWARDS DEVOPS

When a DevOps implementation is starting, one of the more challenging aspects is the cultural shift that has to take place. At large companies, IT organizations frequently have multiple departments for Operations and Development where both groups are sometimes at odds with each other. Therefore, it is essential to establish a collaborative environment that recognizes the mindset change required to enact the relationship shift you are facilitating. Trust is very important and can be earned through transparency, enabling the capabilities for smooth planning, improving incident resolution, and executing successful deployments to production.

Eqn 3: Cultural Shift Towards DevOps Equation (Conceptual)

$$\text{DevOps Culture} = (M + S + T) \times L$$

Where:

- M = **Mindset Change** (from siloed to collaborative, shared ownership)
- S = **Shared Responsibility** (Dev + Ops + QA working together)
- T = **Trust and Transparency** (open communication, visible pipelines, feedback)
- L = **Leadership Support** (organizational buy-in, support for experimentation)

We have already expressed the need for both Operations and Development roles to work with an Agile methodology which can start a joint planning and automation conversation of the tasks and responsibilities of both teams; this Agile approach can help reduce work siloing. A good first step is sharing sprint planning, refinement, and review meetings. Additionally, improving Production Quality begins a relaxation of the Dev+Ops tensions by minimizing critical incidents in the Production environment. By reducing PR responses and creating engagement in shared Sprint activities, the requirement for joint effort increases Reliability and thereby Team Morale. Ideally, the effectiveness of these efforts can be tracked and presented to stakeholders. This critical path then requires visibility and communication of any incidents and quick remediation response.

Hiring and training for cross function skills helps blur the boundaries of the two roles further diminishing team silos. A small Operations team that recommends and creates Developer training to support the ongoing incident response needs while also creating automated monitoring for Dev+Ops would further enhance Team Trust. Formalizing Events with a Lessons Learned review can help capture knowledge and foster a sense of sharing joint responsibility.

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

8.1. Fostering a Collaborative Environment

A desire to improve relationships within the organization is paramount for an effective DevOps implementation. This means that companies should consider accelerating their transformation journey towards an Agile model, with the goal of developing shared objectives and motivational factors to alleviate siloed and conflicting stances. Joint measurement practices and placing DevOps champions and change agents with the right influence and communication skills within each team to incrementally facilitate their adaptation to a DevOps-friendly culture are helpful in accelerating transformation.

Effective collaboration across the value stream enables success through having more direct links regarding customer experience, faster identification of issues and result feedback, and increased technical leverage through the sharing of service ownership. In this transparency-driven environment, risks are openly shared. Technical ties and emotional ties create a bonding effect that further strengthens collaboration, while friendly competition keeps teams motivated. Having dedicated product teams and reducing dependencies for customer experiences accelerates value delivery and customer trust upon receipts of quality products early and often. To improve effectiveness and reduce dependencies, a third-party API gateway is successfully used by the Internal Cloud Team to unify how all teams manage their customer inflows after services go into production. In the process of implementing API gateways, modeled design and essential resource sharing are conducted by small collaborative groups while specializing the API on microservices and dedicating the microservice backends to the various individual teams.

8.2. Training and Skill Development

Continuous training and skill development may be limited in scope and potentially expensive but they are critical to actively reinforce the culture changes associated with DevOps. With previously discrete infrastructure-centric teams blended with increasingly Agile policy and claims development teams, new skills are required across functional boundaries. For infrastructure teams, this includes in-depth knowledge of agile tools, continuous integration and deployment processes, stakeholder communications and accountability for code quality. For Agile policy and claims development teams, this includes investment in building understandable and manageable infrastructure code, capacity planning, domain expertise and working knowledge of cloud infrastructure management, continuous integration and other DevOps tooling to complement their professional focus on code quality. Active engagement and knowledge sharing across functional boundaries, above and beyond traditional Agile Demo cycles, are crucial for continued skill development in both groups – perhaps weekly brown bag sessions. A deeper understanding of how each team works, the constraints on the others' systems, and the significant benefits of coordination via shared tooling are prerequisites for a mature training and skill development program.

Equipping the insurance enterprise with the knowledge and skills to gradually design, implement, and manage a specialized toolchain of DevOps tooling is one of the primary considerations to successfully building and continuously improving a custom DevOps-enabled strategy. This training and knowledge transfer can take place via a carefully sequenced presentation of workshops or focused, role-based enablement sessions led by outside expertise or via extensive running of assessment-driven DevOps Assessment sessions culminating in a phased-in DevOps accelerator toolchain enablement. Different toolchains may require different enablement paths – PaaS, serverless, and application container-based microservices each require different complementary toolchains, with shared sessions at the higher principles and Agile patterns level plus separate base enablement.

IX. IMPLEMENTING DEVOPS PRACTICES

9. Implementing DevOps Practices Maintainability and flexibility are some requirements of legacy systems for the insurance domain. Agile delivery, and in particular the need for a shorter and predictable delivery frequency are increasing issues in this domain. With the continuously increasing business demand, in the era of competition start-ups and digital transformation, there is a need for organizations to adopt organization-wide DevOps practice to enhance the business value of software development. We first discuss the findings of a pilot project within a large insurance firm to experiment with DevOps principles. Then, we discuss the challenges faced and the practices that were adopted to enable the implementation of effective CI/CD in their legacy infrastructure and the recommendations for the organizations willing to adopt DevOps for agile policy change and claims management.

However, for the implementation, the organization faced many issues. While pilots are needed to prove the proof of concept, they were not enough to implement the concrete DevOps practices. These pilots focus on frontend project elements but during scaled implementation, there were challenges on business logic code and the legacy infrastructure. Based on the experimental projects, many organizations adopt specific practices and technologies. However, while enterprise DevOps focused on using commercial, off-the-shelf tools, during the scaled implementation, it was found that

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

the key was to first implement effective CI/CD processes, along defined stages for legacy components and services interacting with cloud-native frontend elements. Once the CI/CD process is in place within the defined stages, then suitable new technologies and tools could be adopted for automation.

9.1. Pilot Projects

With the understanding that innovative deployment practices would have to compete for attention with regulatory and internal deadlines, we prioritized policy and claim updates that were relatively big and complex. We needed an internal champion who was seeking an innovative approach to have a good chance of success. In most cases, we relied on developers writing the update request from scratch because the legacy assembler code was too complex to decipher and normally faced myriad dependencies that further complicated the change. Most were changes that impacted a broad base of variables and conditions. Most requests also timed out on automated builds, a typical symptom of higher complexity changes. Because these requests typically took several months to go live and had a small number of source rules performing most of the work, we scaled back the amount of new logic we needed to build on top of these changes. In one way or another, they would all require rewriting code that had been converted multiple times over the decades. Even though we only used two sample requests, we found small sample sizes offered better estimates of what contractors might achieve. Consequently, we avoided risking time-consuming overestimates that could occur with larger sample sizes.

A key part of defining the estimates was to involve all the right participants in the estimation process. To ensure we got the participation of the internal stakeholders, we had them step in as editors. By involving the assessment participants in developing the estimates as editors or scribes, they took greater ownership of the results. We also wanted to ensure the right internal participants were helping develop the estimates and budgets so we invited product managers to decide how to allocate business and operating budgets. Over time, we made sure the industry participants were also available for the retrospective assessments.

9.2. Scaling Successful Practices

Through the pilot projects, we proved that it is possible to increase the velocity with which software updates are produced and tested. However, the pilot projects were able to do so only for a narrow use case, since they were not critical for the daily operation of the business. Therefore, it is insufficient to prove that increasing velocity is possible. It is also necessary to scale this increased velocity to majority use cases. In particular, insurers rely on their policy and claims processing systems for their daily operations. Any software update to these systems can impact all active deals, which can number in the tens of thousands or more. This risk creates a sizable bar for deploying updates to these systems frequently. Nevertheless, in order to achieve the benefits of moving towards cloud-native technologies and architectures, it is necessary to build and deploy applications written in these technologies and using these architectures frequently. Hence, it is important to scale improvements to the critical policy and claims processing systems as quickly as possible.

There are many dimensions along which scaling can occur. One dimension is the coverage of the policy and claims processing systems. The pilot projects modified only small and isolated areas related to simple policy and simple claims processing. Backward-compatible code changes were needed in large parts of the existing systems to ensure that when a policy or claim was updated, the underwriting artifacts or interactions produced by the prospective front end were compatible with existing rules and intelligence, written in 20-year-old business language and sysadmin scripts, and now executed from modern backend APIs written in Java. Eventually, conditions had to be made compatible, either by having backward-compatible configurations overridden, or by simply re-evaluating the conditions stored in the database for updating the artifacts. However, the pilot projects were not able to deploy their new capabilities to the frontend for simple policies and claims, and to based on updated artifacts, because both the frontend and the had not been decoupled from existing configurations.

X. MEASURING SUCCESS

At the end of the day, were our efforts successful? It would be easy to say yes. After all, we implemented digital methods, open-source tools, cloud solutions, and automation into an area of Aviva's business that had not seen a significant upgrade for nearly a century. However, it is important not to fool ourselves. Thus, understanding KPIs and feedback mechanisms, good programs help their teams, partners, and clients define clear goals for their work and measure their success.

The deliverables or outcomes of our efforts are the first step in answering this question. Certainly, new technology would allow for shorter cycle times, happier partners and teams, more efficient developers, and more satisfied clients. Fulfilling these would be major steps toward passing the success test if our work enabled Aviva's I.T. partners to help their business teams deliver against their plan goals. However, would success be measured during just one anniversary period? What about scaling the existing tablet and mobile DevOps pipelines into other types of policy and claims data or other business teams? That may be too unnecessary.

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

What if our tunnels broke down and we were only able to provide skiing rates for governmental users seasonally at an off-peak time? Or if other accounts without weather driven models still had missions suffering from their toxicity of their dependent workloads? Or if team members were not reaching out to each other for help?

Thus, while we did not believe that one year periods were sufficient measures for most standard KPIs, we pressed our partners to create their own feedback mechanisms and survey partner, internal, and external users of these DevOps pipelines as they ran over time through the various seasons and account levels.

10.1. Key Performance Indicators (KPIs)

Infrastructure and thus development in the enterprise-scale insurance industry tends to be monolithic and legacy-based. Large-volume processing systems, like claims and policy systems in the insurance industry, evolved specialized maintenance, application, storage, and process methodologies to support large-volume data transactions. These systems tend to be poorly instrumented as compared to transactional environments, which deploy a plethora of state-of-the-art metrics to improve performance, maintain user experience, and fine-tune processing requests 24/7. Low system dynamic throughput for development and low business value throughput for DevOps for Claims and Policy Management processes is attributed to poorly defined and poorly maintained metrics, feedback, and postmortem processes. Defining appropriate key performance indicators and metrics is difficult however, as the types of changes process through legacy large-volume management systems are not detectable through system-generated reports.

Expanded DevOps for large-volume transactions also requires policy and claims microservices deployed as APIs, thus introducing DevSecQA for API Management. This appendix tries to delineate new KPIs for enterprise-level DevOps enablement for policy and claims processing to allow business units to transit to Agile DevOps for Claims and Policy Manage Serverless Applications. The KPIs are also used for provide and measure corporate compliments and security requirements, and provide enterprise audit trails for partner large-volume transactions. Minimally acceptable effectiveness and enterprise security audit trail response times are measured by E50, E95, E99, and worst userax.

10.2. Feedback Mechanisms

The case described in this paper is from a feedback-based study where new practices and processes were identified and used from the feedback gathered at the end of each cycle. The details describing these practices and processes come from the feedback gathered and were implemented in the following 4-5 feedback cycles. The outcomes of these feedback cycles are coded using the same codes from the first feedback and thus referenced recursively. This recursive referencing enables the reader to visualize the evolution of the project. This section describes drawing from research best practice, results of how to enable maturity in DevOps Enablement amongst non-technical people combining project-wide training, identifying local champions, mentoring, documenting, gamification and a 2-tier approach.

In practice, we provided feedback input to the project at the beginning, and at the end of each one or two-weeks cycles. After the first input, and based on the responses to our requests gathered from teams, we implemented feedback further formalizing input per team. These inputs were revisited and if required further detailed in inputs up to final delivery times at the end. Some of the initial deliveries and last deliveries were iterations of proof of concepts only for reviewing and accepting requested changes before the outside iterations also span the last feedbacks inputs discussing overall project's last-time learning points input and project's final output review. These last two cycles learned points and milestones review were based on input on general questions that we sent describing and detailing the points we wanted to cover.

XI. CASE STUDIES

The insurance legacy environment is often perceived as unsuitable for agile and devOps enablement due to the constraints imposed by a long tally of industry-specific regulations, lack of late entrance competitive pressure, larger scale of operation and a skew in incentives for driving down costs at the expense of investing in improving operational efficiency. The banks of the large insurance company were housed at different venues and were on different journeys in the utilization of contemporary risk-relevant technologies such as AI, blockchain and cloud computing. Each project was unique and had the predictability and unpredictability of a regular IT project and a GTD project respectively. However, policy administration and claims settlement processes mandated collaboration among and across banks launching different products and projects, and consistency around customer experience throughout the entire policy lifecycle and across lines of business – Life, Property & Casualty, or Disability – is key to fostering a lasting customer relationship and achieving competitive advantage. The absence of such collaboration can give rise to higher operational and expansion costs and poor customer experience.

The devOps enablement of shared code bases and the infraDevOps automation of Insurance Cloud enablement are especially critical to driving down the walling across banks and across lines of business and enhancing returns on investment in cross-insurance product and policy administration and claims servicing DevOps initiatives. These case

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

studies democratize knowledge capture and sharing of what accelerated or de-accelerated the embrace of devOps and Agile by insurance companies no matter their size and experience and operating and build characteristics. The approach can accelerate the maturation in the depth and breadth of the use of DevOps practices and Lean thinking across the entire organization, not limited to IT, and enable the transition from an imperfect, phased agile implementation to the adoption of agile methods as lean practices for building the software and release trains as a regular and continuous for the deployment of software utilization for collaboration across IT, business and everyone at the organization who have a vested interest in the deployment.

11.1. Successful Implementations

In recent years, the Federal Insurance Contribution Act (FICA) mitigated the cost of legacy infrastructure replacement by allowing some insurers to reduce their cost basis associated with legacy systems. Many carriers have either moved to cloud-enabled infrastructures or have introduced hybrid policies that still support the legacy system while allowing for innovation on more modern technologies. Some carriers opted for generated analytics and chatbot-based tools, which solved many client-side issues in an expedited fashion. In addition, they realized the importance of exposing those capabilities through both internal and external services. APIs became the standards-based method of getting data into and out of the various policy and claim-driven processes. Other vendors have invested in policy and claims cloud-based solutions that allow for users to rapidly prototype and design insurance-related processes. These cloud-based solutions provided pre-existing flows that reduced a carrier's cost for those lines of business from millions down to hundreds of thousands.

While some of these vendors have demonstrated initial company launches, the entire insurance company ecosystem must also be cloud-enabled and reliable for large-scale business adoption. Test-driven frameworks based on mocking and virtualization must be developed within the vendor's organizations and their ecosystem to allow continuous integration and deployment. Only through massively-parallelized testing can the initial design, development, and deployment cycles approach industry-proven best-practice timelines of weeks to months. The new carriers modeled on venture capital investments must also allow for a variable traditional cost structure until they achieve scale. Partnerships between the traditional players and the new carriers must be struck to allow for policyholder protection while the new companies scale.

11.2. Lessons Learned from Failures

When discussing our findings about failed or stalled DevOps work, we include events that were realized as potential ingredients for a successful outcome in the projects that were stalled. These include lack of necessary data in both pipeline and its crossing points, poor choice of tech stacks for enablers, time inconsistency as linked to round tip time for execution, refusal to follow some common previously agreed milestones, insufficient push for some interconnections whose maintenance is not anybody’s responsibility, unrealistic expectation of some complexity being unit tested without having action prior to DevOps, refactoring on top of new feature development to enable more concurrent unit testing execution at some production stages. These ingredients became evident to us with time when facing stalled projects, many subjects in these events kept on iterating over non-separated software dualities, monolith versus micro, together. These items are not connected to management decision faster execution reaction, but more related to organization of code pieces that help on its neutrality for some defined behavior for separated actions moving data from A to B, including all interconnections to facilitate semi-automation or at least data qualification before quartered production processes.

We use at a high level to differentiate a DevOps project status and a simple unit testing automation DevOps initiative a window and a door. In a window, data comes in and out with some definition consistency for the action while in a door, a complex piece of software is tested without pre-defined output state consistency so that the unit test needs to constantly verify it, increasing its execution time. The action for the door state at special organizational pieces, helping others reaching the window state and keep it, are with a significant share from quality assurance analysis and a significant behavioral share from engineering systems analyses, focusing on architecture in its business and product use aspects. Without defining who is going to help on the design focus of time consumed on both states differentiation, the tendentious effects keeps on software projects recurring to the divide from monolithic high technology for micro technology and modularization, plus operational behavior budget. Without this behavioral definition, customers and users may expect the high technology produced to run as a micro behavior module, with low speed, low complexity, and low cost on operational activities.

XII. FUTURE TRENDS IN DEVOPS AND INSURANCE

Twelve years ago, the authors chronicled the tentative entrance of the DevOps movement into the worlds of technology and finance. Quickly after, industry standards commandeered the enthusiasm of change in the financial markets by engendering a giant monster called the Audit Control. Insurance, with a hand guiding its own pace of change known as

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

the Model Audit Rule, has its own, slightly more subdued, history of compliance dragging its feet letter-of-the-law adoption of major innovations. Today, the major players in both financial technology and insurance are ready for change and for the application of the full weight of supply chain automation technologies in the cryptocurrency and fault-tolerant transaction-enabling industries.

Today, however, the weight of emerging technology cannot be ignored. Cloud services sit above the layer of infrastructure that they long ago commoditized and that forms a basis for automation that serves to short circuit the many layers of approval that such automation avoided in the past. Layering on top these snares of innovation - of software and process enabling service and user Experience Management - creates an Adjacency Effect that inhibits growth of organizations that need to transform. Not all industry players can afford to follow cloud-first strategies; monetization can often be unlocked while utilizing resources and assets that contain immense latent value.

12.1. Emerging Technologies

When looking at insurance technology as a whole, it is undeniable that an entire landscape of technological capabilities are continuously emerging and being pushed into the broader insurance industry. There are outbound technologies allowing for better integrations with third parties; inbound technologies allowing third parties to capitalize on insurance as a service; optimizing technologies which facilitate efficiencies within a given operation; and eye-catching capabilities built on breaking technology that have the power to fundamentally change a large sector of the industry. While much of the value of these emerging technologies can indeed be extracted with or without utilizing a suitable framework, when combined, the ripped value approach capitalizes themselves.

Firstly, and without pomp or gravitas, are the last couple of decades of enabling tech infrastructure and capabilities. Cloud and microservices have allowed for better integration and innovative connections with third parties. Open-source languages coupled with agile and devops have allowed organizations to really optimize their own internal operations. With insurance as a service being on the horizon, it's still cloud and microservices that play the enabling role for traditional providers. The influx of companies entering the transitional optimization market, looking to get behind a screen and help drive down costs for carriers, have made the enabling power of innovation tech's infrastructure capabilities palatable.

However, these technologies would just be enablers at a more optimized state of the technology landscape were it not for the eye-catching capabilities that have emerged, making use of some of tech's newer capabilities.

From AI and ML to automation and data traceability, it's now possible for a tech provider to pitch a vendor agnostic solution that succinctly sells itself to a given department of a given company. And if the sales and marketing pitch is good enough, it convinces an entire organization to trust in one of these products to ground its operation as a whole, in fear of losing out on an allocation of investment funding for not committing to become more efficient, more innovative.

12.2. Evolving Regulatory Landscape

In both policy sales and claims processing, there is often significant risk for insurers in terms of unrecoverable funds. Defined risk-based enterprise-wide limits on self-service tools and capable internal controls to prevent financial impacts and increased regulatory scrutiny should be a focus for all market participants and regulators. Likewise, institutional knowledge at companies that are 100+ years old is extremely valuable and should be utilized in validation models for both claims processing and policy issuance for automation/increased speed. This may not necessitate as rigorous central control and additional time for all low value items, reducing ultimate economic loss and improving customer satisfaction. Doing this while remaining compliant is an increasingly more difficult task due to multiple governments adding layers of scrutiny and regulatory hurdles throughout the processes. In addition to state regulators, procedures are required to prevent the claims being declared as taxable or to prevent policy loans from being used to defraud the government, as well as internal processes designed to prevent insureds from violating reporting acts.

As customers become used to even more real-time services in other areas of their lives, technology will have to stretch even further. Whereas data was the main bottleneck in the past, today attention is moving over towards transparency and accountability. Customers are requesting more real-time reports on explanatory signal risk in order for companies to remain compliant and profitable while at the same time signing service-level agreements voiding the company's right to accept/decline business; not only from a timing perspective, but how they internally handle tax-insured communication. There is no longer the ability to decline requests for assisted claims and policy functions when they go outside of normal working hours. This is obvious from the increase in unplanned overtime related to claims and creative policy structuring outside of the typical death benefit.

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 7, Issue 12, December 2019

XIII. CONCLUSION

Applying Agile Release Train and Value Stream organizational structures to an operations business area provided a necessary framework for implementing DevOps enablement across disparate teams situated within a traditional hierarchical organizational structure. Simplifying decision-making through facilitated workshops, with membership solely from impacted client-facing teams, cleared the way for improving operational efficiency while implementing compliance safeguards. Creating new compliance functions in accordance with DevSecOps principles eliminated the identified bottlenecks to continuous flow and lowered incident post-mortem customer distractions by more than two-thirds while safeguarding compliance needs with security and fraud mitigation. Rate-limiting tool use in the development domain allowed for introduction of a strategic technology initiative to help alleviate pressure on operations teams.

The effectiveness of coding tools must be recognized at a vendor-neutral level. Vendors are not compensated for creating tools that may severely limit the upper bound of what an organization may accomplish via code, be it the utilizing of constrained resources for strategic advantage or the introducing of operational efficiencies and their concomitant financial drops to the bottom line. They build these unique business assets, their code, from preexisting libraries of other code, with minimal added marginal cost of incremental customer and external partner-facing new functionality enablement. Organizations should collectively reward coding professionalism by embedding coding tool decision-making feedback into their consumption contracts.

Cascaded at an enterprise-level, reduced tooling latitude will enable improved decision-making through local domain groups focused on incremental releases for their specific areas. With releases moving from quarterly to monthly to weekly, business knowledge, too, must coalesce. Consideration must move towards a strategic initiative not for code base ownership, but for shared ownership of the enterprise business capabilities delivered by the multiple touch points enabled by the multiple codebases while balancing risk and reward at all levels.

REFERENCES

[1]. Serón, M., Martín, Á., & Vélez, G. (2019). Life cycle management of automotive data functions in MEC infrastructures. *arXiv*. https://arxiv.org/abs/2303.05960

[2]. Mandala, V. (2018). From Reactive to Proactive: Employing AI and ML in Automotive Brakes and Parking Systems to Enhance Road Safety. International Journal of Science and Research (IJSR), 7(11), 1992-1996.

[3]. Berezovsky, A., El-khoury, J., Kacimi, O., & Loiret, F. (2018). Improving lifecycle query in integrated toolchains using linked data and MQTT-based data warehousing. *arXiv*. https://arxiv.org/abs/1803.03525

[4]. Mandala, V. (2019). Integrating AWS IoT and Kafka for Real-Time Engine Failure Prediction in Commercial Vehicles Using Machine Learning Techniques. International Journal of Science and Research (IJSR), 8(12), 2046-2050.

[5]. Shiklo, B. (2017). IoT in the Automotive Industry: Concept, Benefits, and Use Cases. *ScienceSoft*. https://www.scnsoft.com/blog/iot-in-automotive-industry