INVESTIGACIÓN e INNOVACIÓN en INGENIERÍAS
ISSN2344-8652

Implementación de prácticas DevOps en un Sistema de Mainframe Legado

Implementation of DevOps Practices for a Legacy Mainframe System

Andrés Martínez Villegas **Paola Noreña Cardona**

Elizabeth Suescún Monsalve **Liliana González Palacio**

Universidad EAFIT, Colombia

César Pardo Calvache

Universidad del Cauca, Colombia

Recibido: 14/04/2022
Aceptado: 19/07/2022
Publicado: 14/10/2022

Correspondencia de autores:
ajmartinev@eafit.edu.co

Resumen

Objetivo: Presentar una experiencia de implementación de prácticas DevOps en un Sistema de Mainframe Legado o SML. **Metodología:** Se usó una investigación cuasi experimental, la cual se caracterizó por presentar como grupo de control el proceso de despliegue manual dentro de una organización y como grupo experimental una propuesta de automatización de ciertos puntos del proceso. La metodología para este tipo de investigación permitió observar el comportamiento de ambos procesos, registrar datos y realizar análisis cuantitativo de las variables observadas, entre ellas: el tiempo de despliegue, la frecuencia de despliegue, el tiempo para restaurar, la disponibilidad, la calidad de los entregables y el volumen de cambios. **Resultados:** La implementación de prácticas DevOps en un SML evidenciaron mejoras en las capacidades de TI de la organización, se logró generar mayor velocidad en los despliegues, detección temprana de errores, trazabilidad y control en ambientes de desarrollo y pruebas. **Conclusiones:** El presente articulo mostró que es posible implementar prácticas DevOps en un SML, dichas prácticas emergen como una nueva posibilidad en la mantenibilidad de este tipo de sistemas. DevOps es una promesa que se debe abordar analizando los procesos neurálgicos, midiendo las capacidades y generando cultura en los equipos de desarrollo.

Palabras clave: DevOps, Sistema de Mainframe Legado, Automatización, Medición, Observabilidad.

Abstract

Objective: Presenting an experience about implementation of DevOps practices in a legacy Mainframe system or SML. **Methodology:** We use quasi-experimental research, which is characterized for presenting as a control group manual or actual process of a company and as an experimental group an automation proposal about some points of such a process. The methodology for this research type allowed to observe behavior of both processes, record data, and perform quantitative analysis of the observed variables. Such variables are: delivery lead time, deployment frequency, time to restore service, availability, deliverable quality, and volume of changes. **Results:** Implementation of DevOps practices in a SML evidenced improvements in the IT capabilities of the Company. Also, this was achieved, greater speed in deployments, early detection of errors, traceability and control in development and testing environments. **Conclusions:** This work showed the possibility of implementing DevOps practices in an SML, such practices emerge as a new possibility in the maintainability in this kind of systems. DevOps is a promise, which should be addressed by analyzing the neuralgic processes, measuring capabilities, and generating culture in development teams.

Keywords: Automation, DevOps, Legacy Mainframe System, Measurement, Observability.

Como citar (IEEE): A. Martínez-Villegas., E. Suescún-Monsalve., P. Noreña-Cardona., L. González-Palacio., y C. Pardo-Calvache. "Implementación de prácticas DevOps en un Sistema de Mainframe Legado". Investigación e Innovación en Ingenierías, vol. 10, n°2, 129-146, 2022. DOI: https://doi.org/10.17081/invinno.10.2.6142

Introducción

Un Sistema de *Mainframe* Legado o *Legacy Mainframe System* [1] y en adelante SML es aquel que reúne reglas de negocio y conocimiento que es antiguo en comparación con otras soluciones. Dichos sistemas siguen en uso dentro de algunas organizaciones, y en algunas ocasiones; no cuentan con soporte y mantenimiento y no se pueden reemplazar fácilmente. En muchos casos los SML se convierten en un capital intelectual invaluable para la compañía, dado que este tipo de sistemas de registro proporcionan funciones del negocio relevantes y necesarias.

Actualmente, la transformación digital e innovaciones tecnológicas están dinamizando las organizaciones en términos de cultura organizacional y productividad, asimismo, los marcos de trabajo, ambientes de desarrollo integrados, plataformas, herramientas, arquitecturas, software y los protocolos de comunicación han experimentado grandes cambios en comparación con las generaciones anteriores.

DevOps sigla que integra conceptos de Desarrollo y Operaciones, emerge como una propuesta de mejora continua a nivel tecnológico e incorpora cultura ágil en contextos organizacionales. Sus prácticas se originan en el desarrollo de sistemas basados en la web, por lo que es natural generar preguntas acerca de su aplicación en un SML en un entorno empresarial donde es predominante lo que se conoce como *Big Ball of Mud*, que consiste en miles de sistemas estrechamente acoplados [2,3], también se puede decir que el concepto de integración continua fue el primer paso que inspiró las prácticas ágiles, esto llevó a que los desarrolladores buscarán integrar de mejor manera las modificaciones en repositorios compartidos [4]. También, Forsgren et al. en [5] describe cómo DevOps permite integrar las bondades de prácticas de despliegue automático y continuo en SML basados en principios de enfoques holísticos, [5] también se afirma que las organizaciones pueden adoptar sus prácticas sin importar el tamaño o las características arquitectónicas de las soluciones. Las prácticas en mención son: desarrollo continuo, realización de pruebas continuas, integración continua, entrega continua, puesta en producción y monitoreo continuo. También se puede desarrollar y comprobar sistemas similares a aquellos que se encuentran en producción, implementar procesos fiables y repetibles, validar la calidad operativa, permitir ciclos de retroalimentación y medir las capacidades [6].

En [7] se afirma que el trabajo no finaliza cuando el desarrollo completa la implementación de una función; más bien, este finaliza cuando la solución se ejecuta correctamente en producción, entregando valor al cliente. Según [5], al convertir los procesos manuales de la infraestructura en procesos automáticos se minimizan los errores y se disminuye la inversión del tiempo de despliegue del equipo de desarrollo, también, se afirma que permite habilitar la integración continua entre los ambientes de desarrollo y gestión de la calidad o en adelante QA, siglas de *Quality Assurance*, dichas prácticas se analizaron en el presente trabajo, dado que se tenían múltiples equipos de desarrolladores y estaban enfocados en generar valor estratégico y diferenciador. Otros trabajos que también se consideraron para abordar en este trabajo y que evidencian la adopción de DevOps son [8 y 9].

En este artículo se analizó el proceso de despliegue manual en una organización que cuenta con un SML. El objeto de este análisis era optimizar el tiempo de despliegue entre el ambiente de desarrollo y QA. Para ello, se implementó un proceso de despliegue automatizado mediante el uso del *Deployment Pipeline*, flujo de trabajo propuesto para el despliegue del desarrollo de software basado en prácticas y principios DevOps. De este modo, se generaron datos que permitieron comparar ambos procesos e identificar los mecanismos de adopción de prácticas DevOps para un SML cómo: medir y comparar la velocidad del despliegue, la cantidad de cambios desplegados, la detección de errores y la estabilidad entre los procesos

de despliegue. Las cuáles permitieron agilizar, automatizar y mejorar la productividad en el proceso de despliegue de cada desarrollador.

Teniendo en cuenta lo anterior, este trabajo se estructura de la siguiente manera: Primero, se presenta el marco teórico que describe las bases conceptuales del trabajo; Segundo, se define la metodología a través del diseño cuasi-experimental donde se muestra cómo se abordó la investigación; Tercero, se presenta el análisis de los resultados, y finalmente, se describen las conclusiones y trabajos futuros.

Marco Teórico

La adopción de DevOps tiene como premisa que las acciones relacionadas al desarrollo de los productos de software deben estar enmarcados en una combinación de filosofías culturales, prácticas y herramientas para incrementar la capacidad de la organización, y así; proporcionar aplicaciones y servicios a gran velocidad. La velocidad permite a las organizaciones una mejor experiencia con sus clientes y competir de forma más eficaz en el mercado, velocidad que se podrá ver reflejada en el desarrollo y mejora de los productos tanto para las organizaciones que utilizan procesos tradicionales de desarrollo de software y administración de la infraestructura, como aquellas que están incursionando en el sector de servicios o la industria del desarrollo de software. Y es en este contexto que es posible pensar la adopción de DevOps en SML dado que; en recientes estudios se describe que no existe una correlación significativa entre el tipo de sistema, los procesos de desarrollo y la adopción DevOps en las organizaciones [3, 8, 10 y 11], más bien, el reto se deriva en; identificar las capacidades operativas de una organización, los puntos críticos y en mejorar la velocidad en la entrega de software como un punto diferenciador de las organizaciones [5]. En síntesis, se debe poder validar rápidamente los deseos y necesidades de los clientes y poder hacer despliegues de funciones cuando los clientes las necesitan.

Adicionalmente, las organizaciones que desarrollan y entregan software de forma continua, están mejor capacitadas para experimentar con nuevos enfoques, modelos de capacidad, modelos de satisfacción al cliente, y mantener el cumplimiento y demandas regulatorias [3, 8, 10 y 11]. Según [10], es posible evidenciar que uno de los grandes desafíos de las soluciones SML es incrementar el tiempo de entrega, definir enfoques de transformación que se centren en alinear y sincronizar los esfuerzos de todos en la organización para permitir la entrega de valor. De acuerdo a los resultados obtenidos por [4], se estima que el 90% de las grandes compañías a nivel mundial las apoya un SML que soporta un volumen considerable de transacciones en tiempo real. Estas compañías pueden elegir migrar a nuevas plataformas, o bien, enfocar sus esfuerzos en adoptar prácticas de DevOps para lograr que ellas generen valor, identificando las mejoras del proceso de desarrollo a través de la implementación de prácticas como entrega continua, automatización de pruebas, despliegues que faciliten el mantenimiento y la evolución de dichos sistemas. En este sentido, la adopción de prácticas DevOps para SML tienen como objetivo detectar problemas relacionados a cambios, reducir los errores por procesos manuales, reducir los tiempos relacionados a la integración y entrega y el aumento en la productividad de los equipos.

En este contexto, en [5] se afirma que los modelos de madurez definen un nivel estático de habilidades–tecnológicas y de procesos organizacionales–los cuales no tienen en cuenta la naturaleza siempre cambiante de la tecnología y el panorama empresarial. Además, en [5] se describe que los modelos de capacidad promueven entornos con la facilidad de cambiar dinámicamente y permiten a los equipos y organizaciones centrarse en desarrollar las habilidades y capacidades necesarias para seguir siendo competitivos. Al centrarse en un paradigma de capacidades, las organizaciones pueden impulsar la mejora continua, las capacidades correctas y sus resultados, lo cual permite desarrollar y entregar software con mayor velocidad y estabilidad.

En este sentido, un estudio realizado por Forrester Consulting Compuware (Redacción MCP, 2019, párr. 3) [11], evidencia que el 72% de las organizaciones son completa o altamente dependientes del procesamiento en SML. El mismo informe refleja que el 64% de estas organizaciones ejecutó para el 2019 más de la mitad de sus cargas críticas de trabajo en plataformas de SML, lo que supone un incremento del 57% respecto a 2018. También en [11] (Redacción MCP, 2019, párr.12) se dice que existe una idea errónea en las organizaciones respecto a que el trabajo de los equipos de desarrollo que dan mantenimiento a los SML no es compatible con los enfoques ágiles, los cuales se guían según los valores y principios definidos en el Manifiesto Ágil y DevOps [12 y 13], por tanto, es necesario resaltar que la adopción de enfoques Ágiles y DevOps no es una cuestión de hardware, sino del valor, el proceso y la herramienta. Estos elementos combinados son los que promueven en las organizaciones, el desarrollo y entrega de aplicaciones de forma rápida, otorgando a la vez mayor calidad y eficiencia. En ese sentido, según [7]: es tentador decir que DevOps no funcionará en contextos de SML y que DevOps es solo para organizaciones 'novatas' o 'unicornio'. Aunque los entornos SML tienen diferentes desafíos, DevOps también funciona allí. De hecho, según [7]; la mayoría de las historias de éxito de DevOps provienen de implementaciones en SML, los estudios de casos en organizaciones como Target, CapitalOne y Nordstrom representan algunos de los más populares, y como también afirman los autores, hay una falsa concepción de que DevOps, por una u otra razón, no funcionará en su organización. En este contexto puede que al implementar prácticas DevOps en proyectos con SML algunos de los obstáculos se encuentren relacionados con equipos legados con resistencia al cambio, inercia por continuar trabajando de forma similar o temor por enfrentarse a nuevos desafíos.

Según [14], las prácticas relacionadas con la entrega continua de software es uno de los retos más grandes a los cuales se enfrentan los profesionales del software hoy en día, un comienzo para la adopción de DevOps es proponer un proceso de despliegue que incorpore elementos de automatización en aquellos puntos que son susceptibles de automatizarse, dicho proceso se basa en el *Deployment Pipeline* [6], el cual es denominado proceso de despliegue automatizado para el presente trabajo. Un *Deployment Pipeline* representa un flujo para la implementación automatizada o semiautomatizada, que va desde tomar código de un gestionador de versiones y ponerlo a disposición de los usuarios, en otras palabras; se trata de establecer una fábrica de entrega de valor: un pipeline racionalizado y libre de desperdicios a través del cual el valor pueda ser entregado al negocio con un tiempo de ciclo predecible y rápido [15]. Un *Deployment Pipeline* debe representar los beneficios de la automatización y mostrar al equipo de desarrollo una hoja de ruta sobre la aplicación o solución sobre la cual está trabajando, a la vez, este debe aumentar la retroalimentación y la confianza en el proceso. En ese sentido, es una forma confiable y eficiente de construir, probar e implementar el trabajo, lo que hace referencia al ciclo automático de entrega de software y a la implementación automatizada de los procesos de construcción (Build), despliegue (Deploy), pruebas (Test) y despliegue en ambientes productivos (Release) descritos en [16].

De acuerdo con Humble y Farley en [14], la entrega continua de software está fundamentada en tres prácticas claramente identificadas: Gestión de la Configuración, Integración Continua y Pruebas Continuas. A su vez Forsgren et al. en [5], consideran que las prácticas, tales como: uso de control de versiones, automatización del proceso de despliegue, uso de métodos de desarrollo como "Trunk-Based" o basado en troncales, implementación de integración continua, automatización de pruebas, gestión de datos de prueba, *shift left* en seguridad o traer la seguridad más cerca al desarrollo e implementación y entrega continua mejoran la capacidad de entregar software con mayor velocidad y estabilidad, y a la vez, reducen el estrés y la ansiedad asociados al despliegue de código a producción.

Adicional a eso, a lo largo de los años se han adoptado estrategias de control de versiones de código, el primer paso fue la integración continua, lo cual motivó a los desarrolladores a integrar con mayor frecuencia

las alteraciones de su código y almacenarlo de forma compartida [6], adicional a esto, las estrategias de control de versiones también permitieron a los equipos trabajar en paralelo, acelerar el ritmo de publicación y corregir errores con rapidez. En este contexto, el desarrollo basado en "Trunk-Based", descrito en [14] ha demostrado ser un indicador de alto rendimiento en el desarrollo y entrega de software, mejorando la productividad, los desarrolladores crean ramificaciones de corta duración con algunas confirmaciones pequeñas (p. ej., menos de un día), antes de fusionarse con el maestro o rama principal. A medida que crece la complejidad de la base de código y el tamaño del equipo, el desarrollo basado en troncales (Trunk-Based) ayuda a mantener el flujo continuo de despliegues en ambientes de producción, y como consecuencia, los equipos de desarrollo rara vez tienen períodos de "bloqueo de código", ese estado cuando nadie puede registrar el código o hacer solicitudes de extracción debido a conflictos de fusión, congelamiento de código o fases de estabilización. También hacen parte de las prácticas de entrega continua de software implementar gestión de los datos de prueba e incorporar pruebas de seguridad. Para Kim et al. en [7] las prácticas de entrega continua incluyen también aquellas que son prerrequisitos para implementar el *Deployment Pipeline*, entre las que se cuentan: creación automática y por demanda de ambientes idénticos a producción, reconstrucción de infraestructura en lugar de reparación, implementación de arquitecturas diseñadas para disminuir los riesgos de las liberaciones en producción, entre otras, descritas también en [16].

Metodología

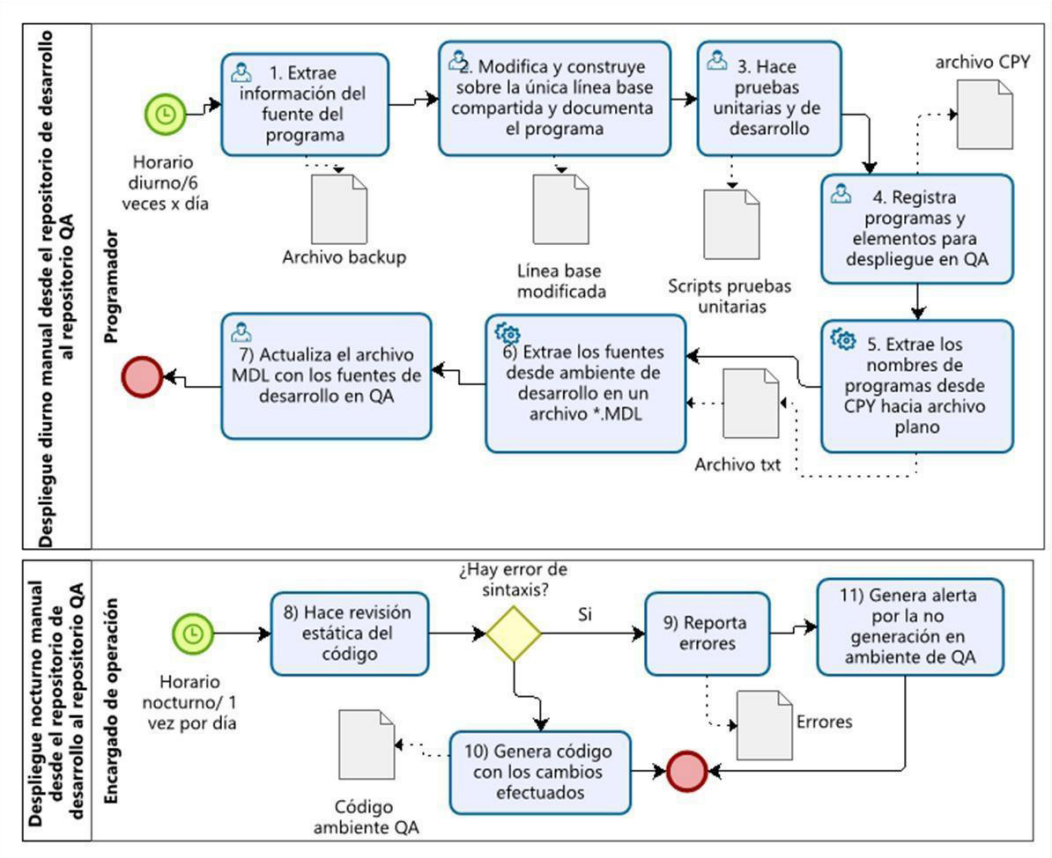
Para abordar la implementación de prácticas DevOps en un SML se hizo un análisis del proceso de despliegue manual a intervenir en la organización objeto de estudio (institucionalizado aproximadamente hace 20 años). Las características generales de este proceso se listan a continuación:

- El grupo de desarrollo estaba conformado por 15 programadores compartiendo un repositorio centralizado con el código fuente de los programas sin un control de versiones. Cada desarrollador contaba con un *framework* de desarrollo que permitía la conexión con el repositorio centralizado para realizar cambios en el código.
- El procedimiento para identificar un código fuente, se realizaba manualmente mediante comentarios escritos en el código por parte de cada desarrollador.
- Una vez el desarrollador finalizaba la construcción, procedía a realizar el ingreso del nombre de la fuente a desplegar en QA en un sistema de registro. La ejecución del despliegue desde el desarrollo a QA se efectuaba cada dos horas durante el día.
- La organización contaba con un ambiente de QA, en el que existía un repositorio con el código fuente de los programas que son desplegados desde el desarrollo. Al finalizar el día, se realizaba una generación total o -build- de la solución, con el fin que el equipo y proveedores de QA realizará tareas de certificación.
- La organización también contaba con un ambiente de producción, que contenía su propio repositorio de código fuente, en el que se encontraban las versiones certificadas en QA.

Es importante resaltar que cuando los despliegues no eran exitosos por errores de sintaxis o por relaciones de dependencia inválidas, la actividad de *rollback*, implicaba que los desarrollos que se esperaban disponibles en QA para realizar pruebas no se ejecutaban en el momento oportuno, generando pérdidas económicas e inconformidades en los equipos de operaciones y desarrollo. En consecuencia, la detección tardía de los errores y la demora en el despliegue representan una debilidad del proceso analizado. El

proceso de despliegue manual desde el repositorio de desarrollo al repositorio de QA se ejecutaba bajo el siguiente flujo de actividades en la Figura 1, los cuales son bloques de actividades ejecutadas por el desarrollador en horario diurno 6 veces por día:

Figura 1. Proceso despliegue manual en el grupo de control



Fuente: Elaboración propia

1) Respaldar manualmente el código fuente del programa en un archivo JMDL y copiar en un directorio personal. 2) Modificar y construir sobre la única línea base compartida con todo el equipo de desarrollo y documentar el programa. 3) Hacer pruebas unitarias y de desarrollo. 4) Registrar en el archivo CPY (copy) el nombre de cada programa y elemento para despliegue en QA. 5) Extraer automáticamente los nombres de programas registrados en CPY hacia el archivo plano (el despliegue se efectuaba cada 2 horas entre las 8am y las 6pm). 6) Extraer los códigos fuentes del txt desde el ambiente de desarrollo en un archivo *.MDL. 7) Actualizar el archivo MDL con los códigos fuentes de desarrollo en QA. Del mismo modo se ejecutaban un par de actividades en horario nocturno: 8) Efectuar revisión estática y si los resultados son exitosos se 9) reportaban los errores y 11) las alertas sino 10) se generaba el código con los cambios efectuados.

En síntesis, las principales actividades encontradas en el proceso de despliegue manual para una mejora entre los ambientes de desarrollo y QA en un SML fueron las siguientes: el repositorio centralizado del código fuente no contaba con un control de versiones, por lo que carecía de trazabilidad en las versiones de los archivos en aspectos como: qué desarrollador, cuándo y dónde se modificó el archivo fuente. El proceso para realizar el despliegue de cambios en el código fuente era netamente manual por parte de los desarrolladores, por lo que, se presentaba dificultad en la detección temprana de errores en la sintaxis del código, retrasos en el ambiente QA y dificultad en la comunicación entre los equipos de desarrollo y QA.

Dando como resultado retrasos en el despliegue. De acuerdo con esto, se hizo necesario un proceso de despliegue automatizado que permitiera mejorar las capacidades en la velocidad de despliegue y control de versiones del código fuente basadas en prácticas DevOps para sistemas de *Mainframe* legado y en esto se basó la creación del grupo experimental. Se seleccionó el método cuasi-experimental para obtener aproximaciones a resultados de una investigación experimental con situaciones en las que no es posible el control y manipulación absoluta de las variables [17-18]. Además se utilizó este método, ya que no era posible asignar de manera aleatoria los individuos al grupo experimental (aquel que se expone al cuasi-experimento) y el grupo de control (aquel que se utiliza para hacer comparaciones entre las mediciones [19]). En el diseño experimental se realizó un cuasi-experimento en una organización en Colombia (cuyos datos específicos son confidenciales como éticas en experimentación [20]) con un sistema de *Mainframe* legado.

A. Grupos

Grupo de control: Grupo establecido por la organización, el cual se compara con el grupo experimental. El grupo de control está conformado por el conjunto de programas sometidos al proceso manual (Figura 1). En el Framework de programación propietario EAE (Enterprise Application Environment, por sus siglas en Inglés), se distinguen varios tipos de programas considerados para el grupo de control (Véase la Tabla 1).

Tabla 1. Programas EAE

Tipo de programa	Descripción
LÓGICA insertable	Fragmento de código reutilizable que se inserta en el programa que lo invoca
LÓGICA perfomable	Función reutilizable para realizar una tarea específica que retorna un resultado
REPORTE	Programas en batch
COMPONENTE	Tablas
PROFILE	Índices tablas
EVENTO	Formularios
DICCIONARIO	Elemento diccionario de datos
GSD	Variables globales

Fuente: Elaboración propia

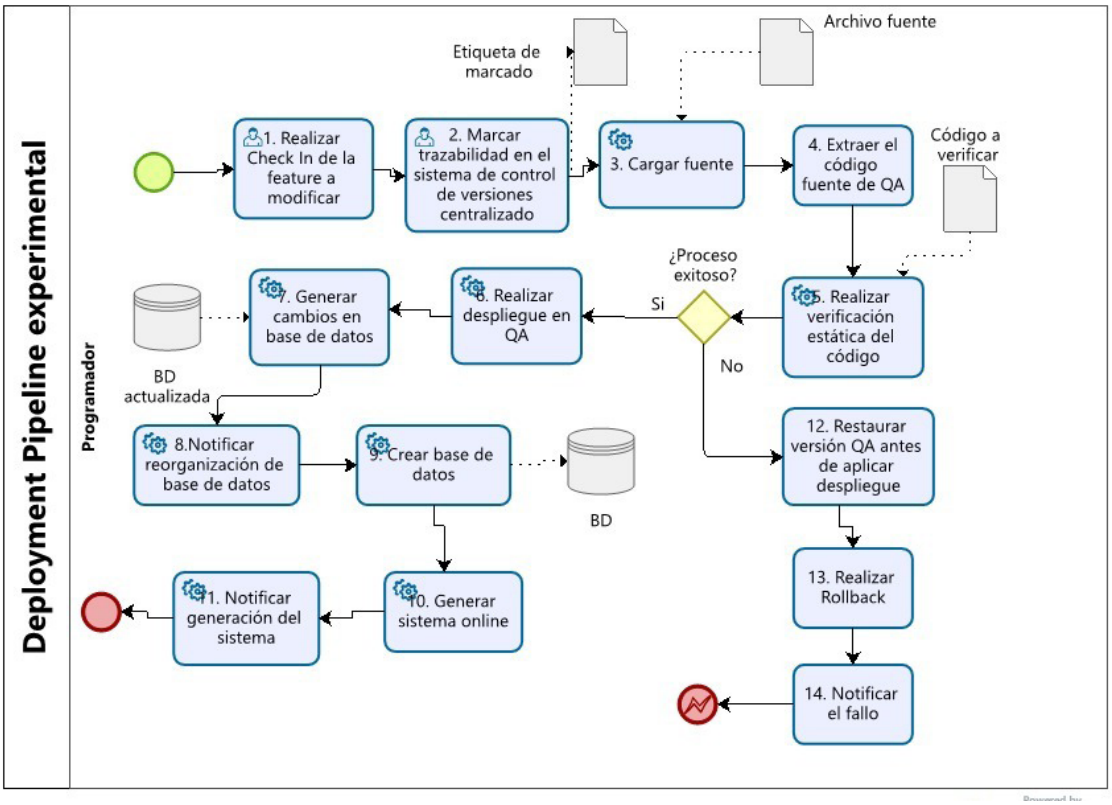
Grupo Experimental: Grupo al cual se le aplicó el estímulo o tratamiento en el proceso de despliegue automatizado mediante el uso del Deployment Pipeline. Se propuso un flujo de trabajo bajo integración continua y control de versiones llamado proceso de despliegue automatizado. Para ello se usó un Deployment Pipeline con automatización de partes del proceso de despliegue manual.

Se definieron las siguientes actividades (Figura 2):

- *Realizar Check* In de la *feature* a modificar: el programador de forma manual integra los cambios de una feature (característica del código) en el repositorio de desarrollo mediante un check In, después de finalizar el ciclo del desarrollo.
- Marcar trazabilidad en el sistema de control de versiones centralizado: el programador marca manualmente con un *label* o etiqueta estándar, la trazabilidad de su actividad en un sistema de control de versiones centralizado (Por ejemplo, TFVC (*Team Foundation Version Control*, por sus siglas en inglés)).

- Cargar fuente: se actualiza automáticamente la fuente del repositorio centralizado de control de versiones EAE y se habilita la integración con el sistema de control de versiones centralizado, mediante el proveedor adecuado (por ejemplo, MSSCCI *(Microsoft Source Code Control Integration* por sus siglas en inglés, interfaz para sistemas de control de versiones)). De esta forma, se actualiza el repositorio de control de versiones en la nube.
- Extraer el código fuente de QA: se realiza el respaldo del programa desde el ambiente de QA para tener un punto de retorno.
- Realizar verificación estática del código: se valida automáticamente el código fuente del programa en QA.
- Realizar despliegue en QA: se realiza el despliegue del código fuente en el ambiente de QA.
- Generar cambios en base de datos: se verifica si hubo cambios en el esquema de la base de datos.
- Notificar reorganización: se notifica al equipo de operaciones que hubo cambios en el esquema de la base de datos.
- Crear base de datos: se ejecutan procesos que permiten la creación del nuevo esquema de base de datos.
- Generar sistema online: se crean de los nuevos objetos ejecutables del Sistema.

Figura 2. Proceso de despliegue automatizado en el grupo experimental



Fuente: Elaboración propia

- Notificar generación del sistema: se envía notificación al programador y al grupo de operación del resultado del despliegue.

A partir de la actividad 6 son actividades automatizadas para el despliegue. En adición, se proponen las siguientes actividades Rollback (véase la Figura 2):

- Restaurar versión QA antes de aplicar el despliegue: se restaura la versión del programa que se encontraba en QA.
- Realizar *Rollback:* se realiza la validación estática del código para la versión que fue restaurada en el paso anterior.
- Notificar fallo: se envía correo electrónico al desarrollador indicando los errores en el código.

Cuando la actividad 3 (cargar fuente) no es exitosa, se ejecuta la actividad 1 del *Rollback*. Sí se detecta que la actividad 5 (realizar verificación estática del código) no se ejecuta correctamente, se realiza el Rollback del programa cargado en QA.

B. Hipótesis

El uso del control de versiones e integración continúa usando el *Deployment Pipeline* como prácticas de DevOps, generan mayor velocidad, detección temprana de errores, trazabilidad y control en el despliegue entre el ambiente de desarrollo y QA.

C. Variables

Las variables independientes son la cantidad de despliegues realizados vs el número de programas desplegados y el tiempo del despliegue, que surgen como parámetros del contexto en datos históricos durante 8 años para observar la capacidad del proceso de despliegue manual. Estos datos se toman como medición inicial para realizar comparaciones con mediciones posteriores a la aplicación del cuasi-experimento. Tales comparaciones son requeridas en los principios de los cuasi-experimentos [19].

En la Tabla 2 se presentan dos fechas para el conjunto de programas EAE del grupo de control, los cuáles se someten al proceso de despliegue automatizado del grupo experimental, con el propósito de establecer métricas que permitan comparar el proceso de despliegue manual y el proceso de despliegue automatizado.

Tabla 2. Selección de Días de Análisis

Programas de despliegue	Fecha 1	Fecha 2	Total
LÓGICA	1	1	2
REPORTE	4	10	14
COMPONENTE	0	4	4
PROFILE	0	8	8
EVENTO	27	26	53
DICCIONARIO	3	0	3
GSD	2	0	2
Total	37	49	86

Fuente: Elaboración propia

En la Tabla 3 se presentan los datos identificados en 8 años de las capacidades del proceso de despliegue manual en la cantidad de despliegues realizados y los programas desplegados. De acuerdo con la tendencia lineal, se analizó que se necesitaban 100 despliegues para transferir alrededor de 203 programas al

ambiente de QA. También, con la variable tiempo de despliegue se identificó que cuando en un despliegue participa sólo un programa, el proceso de despliegue manual tarda 142 segundos, mientras que cuando en un despliegue se involucran más de un programa el tiempo es menor. Para ilustrar, al realizar el despliegue de 32 programas en un solo despliegue el proceso de despliegue manual toma 16 segundos por programa.

Las variables dependientes se seleccionan con base en los criterios de medición del desempeño de los entregables en el desarrollo de software en prácticas DevOps [5]. En la Tabla 4 se presenta la relación entre las variables, las preguntas y métricas para el desarrollo cuasi-experimento.

Tabla 3. Capacidades del proceso de despliegue manual –

Cantidad de despliegues realizados	Programas desplegados
630	1839
150	300
276	92

Fuente: Elaboración propia

Tabla 4. Variables, Preguntas y Métricas

Variable	Pregunta de investigación	Métrica
Tiempo del despliegue	¿En cuántos segundos se realiza el proceso de despliegue en QA, para un programa?	Tiempo de despliegue que toma una implementación aprobada
Frecuencia del despliegue	¿Con qué frecuencia se lanzan nuevas características o capacidades a QA?	Frecuencia diaria, semanal o mensual de los lanzamientos de nuevas características o capacidades a QA
Tiempo para restaurar	¿Cuánto dura en realizar rollback cuando ocurre un incidente en la validación estática de código?	Tiempo empleado en restaurar una falla en el proceso de validación estática de código
Disponibilidad	¿Qué porcentaje del tiempo diario se garantiza el uso del sistema en QA, de forma oportuna y confiable?	Disponibilidad del ambiente de QA
Calidad de los entregables	¿Cuál es el número de programas que cumplen con los atributos de calidad Inmutable, Versionado y Funcional?	Cantidad de programas participantes en el despliegue que cumplen con el nombramiento adecuado de rotulado del programa, versión y pruebas unitarias.
Volumen en los cambios	¿Cuántos cambios diarios a QA atiende el proceso?	Número de cambios de código que el proceso ofrece con cada lanzamiento, o en un período de tiempo determinado

Fuente: Elaboración propia

D. Tratamiento

Todos los programas seleccionados fueron sometidos con el código fuente a la ejecución del proceso de despliegue automatizado mediante un Deployment Pipeline (Véase la Tabla 1). Con el fin de obtener los datos y realizar un comparativo en términos de frecuencia, velocidad, volumen y tiempo para restaurar, entre el proceso de despliegue manual y el proceso de despliegue automatizado.

Resultados

Al ejecutar el experimento, con el proceso de despliegue automatizado mediante el uso del *Deployment Pipeline* (grupo experimental) se logró mejorar las capacidades de control del código fuente, gracias a que en su arquitectura se incorporó un repositorio de control de versiones del código. Este control permite tener trazabilidad y puntos de recuperación en caso de requerir un respaldo. Por lo que, en el cuasi-experimento se pudo identificar quién, cuándo y qué modificó un desarrollador específico. Se logró mejorar la velocidad de despliegue del ambiente de desarrollo a QA debido a que se minimizó la intervención manual del programador en el proceso.

A. Tiempo de Despliegue

En la Tabla 5 se presenta el tiempo en segundos que tomó realizar el proceso de despliegue automatizado en QA de un programa. Desde el análisis de los resultados obtenidos en la Tabla V también se puede mencionar, que el proceso de despliegue realizando el push al repositorio y el despliegue en QA de cualquier tipo de programa, en promedio tuvo una velocidad de despliegue de 57.44 segundos en el grupo experimental. En contraste, la velocidad de despliegue en el grupo de control es de 142 segundos según los datos históricos.

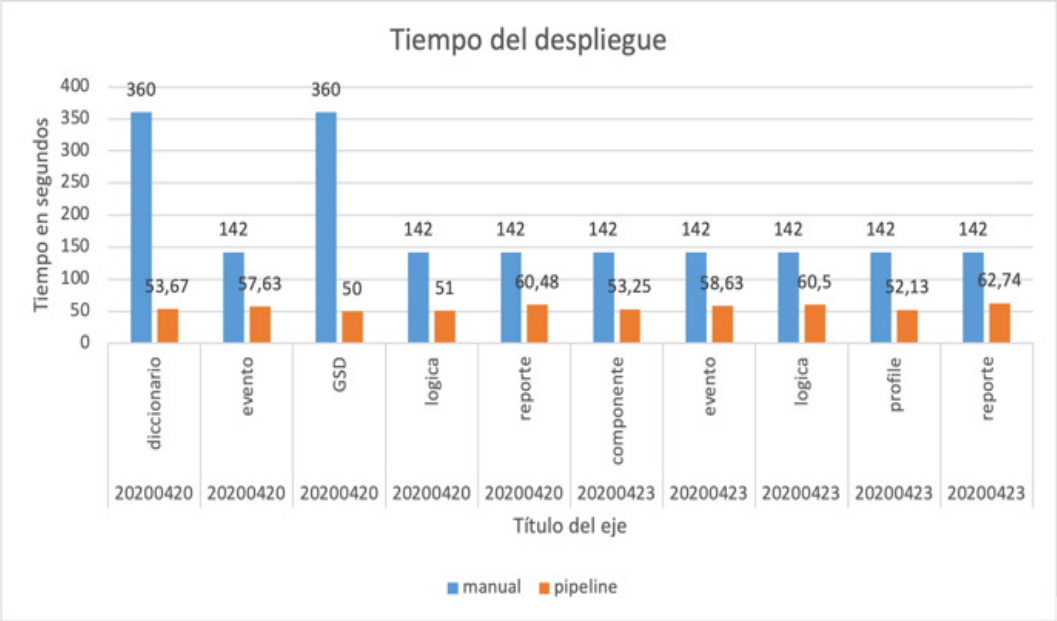
Tabla 5. Tiempo promedio de despliegue del proceso de despliegue automatizado (grupo experimental)

Fechas de despliegue	Promedio velocidad del despliegue en segundos	Número de programas
Fecha 1	57.02 seg.	30
Fecha 2	57.84 seg.	34
Total	57.44 seg.	64

Fuente: Elaboración propia

En la Figura 3, se realiza el comparativo entre el proceso de despliegue manual (grupo de control) y el proceso de despliegue automatizado (grupo experimental). En la misma Figura se puede observar una mejora significativa en la reducción del tiempo de despliegue cuando se realizó el despliegue por medio del proceso de despliegue automatizado. Este análisis permite observar las diferencias entre los tiempos de despliegue del proceso manual y los tiempos después de la implementación del proceso de despliegue automatizado en los diferentes programas EAE.

Figura 3. Comparativo tiempo despliegue entre los grupos de control y experimental



Fuente: Elaboración propia

B. Tiempo para restaurar

En la Tabla 6 se presenta el tiempo para realizar *Rollback* cuando ocurre un incidente en la validación estática de código en el proceso de despliegue automatizado. Así, los resultados muestran que el *Rollback* de cualquier tipo de programa EAE en el grupo experimental es realizado en 60,49 segundos en promedio, a diferencia del *Rollback* en el proceso de despliegue manual que tomó en promedio 6 horas.

Tabla 6. Tiempo en segundos para restaurar en Proceso de Despliegue Automatizado (Grupo experimental)

Fechas de despliegue	Promedio velocidad de un Rollback en segundos	Número de programas
Fecha 1	60.48 seg.	6
Fecha 2	60.5 seg.	5
Total	60.49 seg.	11

Fuente: Elaboración propia

C. Disponibilidad

En la Tabla 7 se presentan los porcentajes del tiempo diario en que se garantizó el acceso oportuno y confiable y el uso del sistema en QA. Según el análisis el ambiente de QA estuvo disponible el 99.999% del tiempo. En todos los casos, se evidencia que en el grupo experimental el ambiente de QA estuvo disponible debido a que el proceso de realizar *Rollback* garantiza que se puedan revertir los cambios en el momento de encontrar un error en el log.

Tabla 7. Disponibilidad, proceso de despliegue Automatizado (grupo experimental)

Tipo de programa	Descripción	Disponibilidad
LÓGICA	Funciones reutilizables	99.999%
REPORTE	Programas batch	99.999%
COMPONENTE	Tablas	99.999%
PROFILE	Índices tablas	99.999%
EVENTO	Formularios	99.999%
DICCIONARIO	Elemento diccionario de datos	99.999%
GSD	Variables globales	99.999%

Fuente: Elaboración propia

D. Calidad de los entregables

En la Tabla 8 se presentan los programas que cumplen con los atributos de calidad de (a) Inmutable, (b) Versionado y (c) Funcional. El grupo experimental permitió realizar la entrega de cada programa asegurando el estándar de nombramiento de programas de acuerdo con las definiciones técnicas exigidas por el cliente. El proceso de despliegue automatizado asegura la inmutabilidad de la versión del código fuente desplegada a QA, puesto que al realizar las actividades 2 a 6 del proceso de despliegue automatizado, la versión del código fuente generado no puede ser modificada por ningún otro miembro del equipo de desarrollo hasta que finalice el despliegue. En el grupo de control esta condición no se cumple, dado que la actividad 2 de marcar trazabilidad libera el programa a la línea base, posibilitando así la modificación por parte de otro miembro del equipo.

Tabla 8. Calidad de los entregables, proceso de despliegue automatizado (grupo experimental)

Tipo de programa	Descripción	Medición
LÓGICA	Funciones reutilizables	a,b,c
REPORTE	GLI -<nombre de la lógica>	a,b,c
COMPONENTE	Programas batch	a,b,c
PROFILE	Tablas	a,b,c
EVENTO	Índices tablas	a,b,c
DICCIONARIO	Formularios	a,b,c
GSD	Elemento diccionario de datos	a,b,c

Fuente: Elaboración propia

E. Volumen en los cambios

La Tabla 9 se presentan los picos históricos de despliegues a QA que puede atender el proceso de despliegue manual (grupo de control), en esta actividad se ejecuta el despliegue cuando el programador realiza la matrícula del programa en el sistema de registro, sin embargo, en este proceso, según datos históricos, se ejecutan 18 despliegues a QA diariamente. En contraste según el presente análisis, el proceso de despliegue automatizado estuvo en la capacidad de realizar 338 despliegues al día, lo que es suficiente capacidad para el equipo de desarrollo actual.

Tabla 9. Selección de Fechas

Fecha de despliegue	Número de despliegues
Fecha 1	173
Fecha 2	127
Fecha 3	108
Fecha 4	107

Fuente: Elaboración propia

En síntesis, todos los programas sometidos al tratamiento quedan correctamente versionados en el repositorio, dando un valor agregado a la organización y al SML utilizado. Adicionalmente, se da trazabilidad del despliegue de cada elemento del proceso despliegue automatizado, lo cual facilita la comunicación entre el equipo de operaciones y desarrollo.

Una amenaza a la validez de los análisis cuasi-experimentales, es que no se puede asegurar el 100% de la equivalencia entre el grupo experimental y el grupo de control, debido a que no existe una asignación aleatoria que empareje el grupo experimental y el de control para que sean equivalentes [13]. Para el tratamiento de esta situación, se solicitaron datos históricos del proceso de despliegue manual para compararlos con el proceso de despliegue automatizado. Se identificaron también dos riesgos que amenazan la validez interna, en aspectos sobre los que no se tuvo control: 1) *El tiempo de despliegue*: en el proceso de despliegue automatizado se realiza la extracción de todas las fuentes registradas por el desarrollador en un único archivo luego se hace un *push* (carga del contenido del archivo) a QA. Esto significó menor tiempo de despliegue en el proceso de despliegue automatizado, que involucró al grupo experimental mientras que el proceso de despliegue manual del grupo de control, se realizó elemento por elemento. 2) El ambiente: se tuvieron en cuenta las mediciones obtenidas del grupo de control en un ambiente productivo, que tiene características más robustas en comparación con el ambiente de laboratorio donde se ejecutó el proceso del grupo experimental. 3) El tiempo de transferencia de archivos: entre el repositorio en la nube y el directorio de despliegue en la instancia local, puede estar condicionado por las características de la infraestructura de los grupos analizados.

Los resultados obtenidos en la investigación se resumen en la Tabla 10. Según Forsgren et al., 2018 en [5] ¿Cómo logran los equipos de alto rendimiento un rendimiento de entrega de software tan sorprendente? “mejorando las capacidades correctas”. En este contexto, con la implementación de control de versiones y automatización del despliegue SML se mejoraron significativamente las capacidades en cuanto a despliegue continuo, mayor velocidad, trazabilidad del despliegue mediante cuantificador de versiones, build automático en QA sin reorganización, trazabilidad de la versión única línea base controlada en TFVC, eliminar el registro manual de despliegues para QA, detección temprana de errores, automatización del despliegue a QA mediante el check-in del programa, sincronización automática de nuevos elementos en el repositorio, rápida detección de errores en el código inyectados al contar con historiales, bloqueo y protección automática al programa, notificación automática al desarrollador del resultado del proceso, facilidad en la identificación de componentes con errores en el despliegue entre ambientes, validación automática de los programas involucrados en el paso a QA, disminución del retrabajo por parte de operaciones asociado a errores en el despliegue y disminución de implementaciones dolorosas del código. La tendencia actual de la industria es mejorar el ciclo de vida del desarrollo de software en las diferentes etapas y con ello entregar productos rápidamente y con calidad. Con ello, las prácticas DevOps no terminan en su adopción y son un continuo a seguir mejorando y colaborando a partir de la implementación de las prácticas asociadas a la entrega continua.

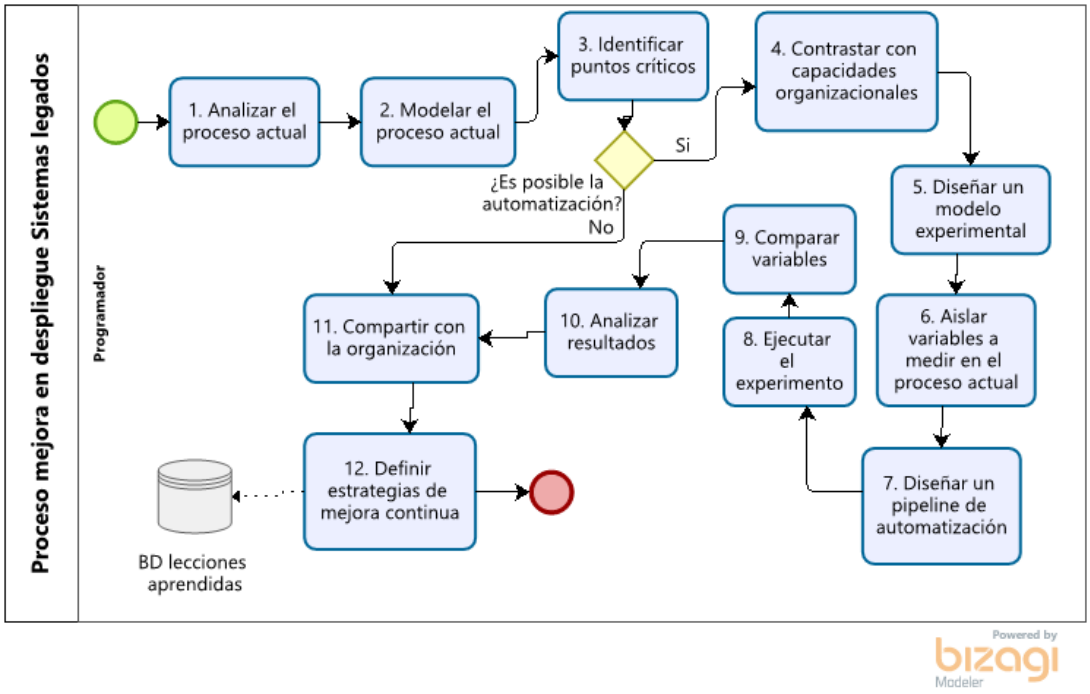
Tabla 10. Calidad de los entregables, proceso de despliegue automatizado (grupo experimental)

	Grupo experimental	Grupo de control
Tiempo del despliegue	57.44 Seg	142 Seg
Frecuencia del despliegue	Medium	Medium
Tiempo para restaurar	High	Medium
	Automática en 60.49 Seg	Tarea manual
Disponibilidad	100.00%	70.00%
Calidad de los entregables	100.00%	Dependencia en el Desarrollador
Volumen en los cambios	338 despliegues en el día	18 despliegues en el día

Fuente: Elaboración propia

Finalmente con los resultados obtenidos, en la Figura 4 se propone un proceso para introducir prácticas de DevOps en SML que puede ser usado en otras organizaciones que estén analizando la adopción de DevOps para mejorar los tiempos de desarrollo, pruebas y despliegue y tengan un SML y así generar valor más rápidamente al negocio, para ello, en este trabajo nos apoyamos en modelos de capacidad clave que contribuyen estadísticamente al mejoramiento en los procesos de entrega de software y a su vez impactar en el desempeño organizacional a nivel cultural, de entrega continua, de la arquitectura, la gestión, monitoreo y principalmente la mejora continua.

Figura 4. Proceso para introducir prácticas DevOps en SML



Fuente: Elaboración propia

Conclusiones

Un SML es una solución que por sus características presenta resistencia a los cambios para su evolución o los mismos exigen procesos complejos, pasos manuales y entregas lentas desde las perspectivas del negocio, sin embargo, por su importancia continúan en uso en las organizaciones, a pesar de la disponibilidad de tecnologías más modernas, los mismos, son mantenidos en servicio debido a los altos costos de reemplazo o rediseño, en muchas, ocasiones un SML requiere la disponibilidad constante por los procesos que soporta en la organización. En ese contexto, la evolución de los SML falla por la complejidad de la modernización, entendimiento de la tecnología o se sugiere que la evolución no es posible. En ese contexto, si se mejoran las prácticas de desarrollo, pruebas y entrega se mejoraría la evolución de este tipo de sistemas, por tanto, en este trabajo fue mostrado a través de una investigación cuasi-experimental cómo es posible adoptar prácticas DevOps en SML a través de un enfoque de capacidades en el cual se mapea el proceso actual y se evalúa los puntos críticos del proceso y susceptibles a automatizar.

También se mostró para la organización evaluada, el primer paso en la mejora de las capacidades de TI, mediante el uso de herramientas, técnicas y prácticas modernas de DevOps en SML y demostrar que al generar mayor velocidad en despliegue, detección temprana de errores, proveer trazabilidad y control del despliegue entre el ambiente de desarrollo y QA se obtiene un proceso repetible y de mayor confianza, también se forma un hito cultural en donde la mejora continua podría ser una constante en la organización. Al incorporar en los procesos buenas prácticas de DevOps que impactan positivamente los resultados, generan valor y eliminan reprocesos susceptibles a errores, que por años se adoptaron como parte del proceso normal, se ahonda por tanto en nuevas oportunidades para equipos que dan soporte a este tipo de sistemas [20, 21, 22].

Bajo un modelo DevOps con un enfoque en la mejora continua, cómo trabajo futuro se puede continuar explorando otras prácticas a nivel de desarrollo, pruebas y entrega de código, algunas de ellas; mejorar la visualización de documentos entregables asociados a cada despliegue, como son: historias de usuario, pruebas unitarias, pruebas de QA, aprobación de cambios y auditorías. Se deben realizar mejoras para habilitar el esquema de despliegue propuesto a QA, de tal forma que permita el despliegue continuo hasta el ambiente de producción, la automatización de pruebas unitarias y trazabilidad y monitoreo de los despliegues mediante el uso de cuadros de mando unificados (dashboards), que habiliten la representación gráfica de los principales indicadores que intervienen en los despliegues.

Este trabajo es mostrado como una lección aprendida para aquellas organizaciones que soportan sus procesos a través de aplicaciones de tipo SML y que no se atreven a adoptar prácticas de DevOps. DevOps permite crear, probar, implementar y monitorear aplicaciones, eliminando pasos manuales, aumentando la agilidad y escalar con mayor facilidad. Lo importante es que no se tenga miedo y que los equipos sean empoderados para innovar y experimentar.

Referencias Bibliográficas

1. B. Wu, D. Lawless, J. Bisbal, J. Grimson, V. Wade, D. O'Sullivan, and R.Richardson, "Legacy System Migration: A legacy data migration engine", C. C. Experts (Ed.), Proceedings of the 17th International Database Conference, October 12–14, pp. 129-138, 1997. https://bit.ly/2VYC6Zk
2. B. Foote, and J. Yoder. Big ball of mud. Pattern languages of program design, vol. 4, pp. 654-692, 1997.
3. González, E. El mainframe resurge para agilizar los entornos de DevOps. Globb Partner, 2016. https://bit.ly/36YtrML
4. V. L. Cruz and A. B. Albuquerque, "A DevOps Introduction Process for Legacy Systems", 2018 XLIV Latin American Computer Conference (CLEI), pp. 139-148, 2018.
5. N. Forsgren, J. Humble, and G. Kim, "Accelerate: state of DevOps report: Strategies for a new economy", DORA (DevOps Research and Assessment) and Google Cloud, 2018. https://services.google.com/fh/files/misc/state-of-devops-2018.pdf
6. D. Farley, and J. Humble, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (1er ed.). Addison-Wesley Professional, 2010.
7. G. Kim, D. Patrick, J. Willis, J. and Humble, The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press, 2016.
8. M. Vizard, IBM avanza DevOps en el mainframe, 2019. Devops.com. https://bit.ly/3oEo7UP
9. Micro Focus, Ejemplos prácticos de DevOps para empresas de mainframe (Informe técnico oficial), 2017, https://bit.ly/33SjHkX.
10. Compuware, "Parámetros que valen la pena medir. Representación de los KPI DevOps en su mainframe", Compuware a BMC Company; Compuware a BMC Company, 2018. https://bit.ly/3mWaGiu
11. Compuware, "Las integraciones DevOps para mainframe de Compuware aumentan la automatización de pruebas de software de forma temprana en el ciclo de desarrollo" *Compuware a BMC Company*, 2019. https://bit.ly/2JDW2hC
12. B. Kent, B. Mike,B. Arie, C. Alistair, C. Ward, F. Martin, G. James , H. Jim, H. Andrew, J. Ron, K. Jon, M. Brian , M. Robert, M. Steve,S. Ken, S. Jeff, and T. Dave. Agile Manifesto, 2001. https://agilemanifesto.org/iso/es/manifesto.html
13. C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano. DevOps. IEEE Software, 33(3), 94-100, 2016.
14. J. Humble, and D. Farley, Continuous delivery Reliable Software Releases Through Build, Test, and Deployment Automation. MA: Pearson Education Inc, 2011.
15. Schwartz, M. The Art of Business Value. Revolution Press, 2016.
16. P. A. Castañeda García, Prácticas DevOps de entrega continua de software para la transformación digital de los negocios (Doctoral dissertation, Universidad EAFIT), 2019.
17. G. Agudelo, M. Aigneren, and J. Ruiz Restrepo, EXPERIMENTAL Y NO-EXPERIMENTAL. La Sociología En Sus Escenarios, (18), 2010. https://revistas.udea.edu.co/index.php/ceo/article/view/6545
18. B. A. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-based software engineering", 26th international conference on software engineering, pp. 273-281, 2004.
19. H. White, and S. Sabarwal, Diseño y métodos cuasi-experimentales, No 8; Síntesis Metodológicas Sinopsis de la Evaluación de Impacto, 2016. https://bit.ly/37RmJHB

20. E. J. De la Hoz Domínguez, T. J. Fontalvo Herrera, y A. A. Mendoza Mendoza, "Aprendizaje automático y PYMES: Oportunidades para el mejoramiento del proceso de toma de decisiones", *Investigación e Innovación en Ingenierías*, vol. 8, n.º 1, pp. 21-36, 2020. DOI: https://doi.org/10.17081/invinno.8.1.3506

21. C. Pardo, E. Suescún, H. Jojoa, R. Zambrano, y W. Ortega, "Modelo de referencia para la adopción e implementación de Scrum en la industria de software", *Investigación e Innovación en Ingenierías*, vol. 8, n.º 3, pp. 14-28, 2020. https://doi.org/10.17081/invinno.8.3.4700

22. C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen, Experimentation in Software Engineering. New York: Springer, 2012.