UNIVERSIDAD AUTÓNOMA GABRIEL RENÉ MORENO

FACULTAD DE INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN Y TELECOMUNICACIONES

UAGRM SCHOOL OF ENGINEERING

Diplomado en Automatización en Devops Essentials V1 E2.

Estrategias para Fomentar una Cultura DevOps y Mejorar la Implementación de CI/CD en Equipos de Desarrollo.

Monografía para optar al Certificado de Culminación de Estudios y al Título de Licenciatura en Ingeniería Informática.

Autor: Yuliana Marcela Montaño Perez

Santa Cruz de la Sierra - Bolivia

Mayo, 2024

Índice general

Índice de figuras

CAPÍTULO 1

ASPECTOS GENERALES.

El capítulo 'Aspectos Generales' contextualiza la evolución de DevOps desde sus antecedentes históricos hasta su relevancia actual en el desarrollo de software. Explora cómo DevOps aborda los desafíos de colaboración entre equipos y la necesidad de eficiencia en el ciclo de desarrollo, estableciendo así una base para comprender los problemas, justificaciones y objetivos que se abordarán en la monografía.

1.1. Antecedentes y Contextualización.

En el contexto histórico del desarrollo de software, es esencial comprender la dinámica entre los diversos equipos involucrados en el proceso. Aunque idealmente los product owners, el desarrollo, QA y IT operations trabajarían en estrecha colaboración, en muchas organizaciones existe una percepción de que los departamentos de desarrollo y operaciones de TI están en conflicto.

Un paralelo valioso puede trazarse con la revolución manufacturera de la década de 1980, donde la adopción de los principios Lean transformó radicalmente la productividad de las organizaciones manufactureras.Aquellas que implementaron prácticas Lean lograron mejorar la productividad, los tiempos de entrega al cliente y la calidad del producto, ganando una ventaja competitiva significativa en el mercado.

De manera similar, en el ámbito del desarrollo de tecnología, las demandas y expectativas de los clientes han evolucionado rápidamente. Lo que fue aceptable en décadas anteriores ya no lo es en la actualidad. Hemos sido testigos de una reducción significativa en el tiempo y el costo requeridos para desarrollar y desplegar nuevas capacidades empresariales estratégicas, gracias a enfoques como la integración continua (CI) y la entrega continua (CD). Estos principios, fundamentales en la metodología DevOps, han revolucionado la forma en que las organizaciones desarrollan, prueban y despliegan software.

Según lo señalado en el libro “DevOps Handbook”, DevOps es el resultado de aplicar los principios más confiables del dominio de la fabricación física y el liderazgo al flujo de valor de TI. Se basa en cuerpos de conocimiento de Lean, Teoría de las restricciones, el Sistema de Producción de Toyota, ingeniería de resiliencia, organizaciones de aprendizaje, cultura de seguridad, factores humanos y muchos otros. Otros contextos valiosos de los que se nutre DevOps incluyen culturas de gestión de alto nivel de confianza, liderazgo servicial y gestión del cambio organizacional. El resultado es calidad, confiabilidad, estabilidad y seguridad de clase mundial a un costo y esfuerzo cada vez menores; y un flujo y confiabilidad acelerados

en todo el flujo de valor tecnológico, incluyendo la Gestión de Productos, Desarrollo, QA, Operaciones de TI y Seguridad de la Información. Muchos también ven a DevOps como la continuación lógica del viaje del software ágil que comenzó en 2001.

Las organizaciones que han adoptado DevOps, con un enfoque claro en la implementación de CI/CD, han logrado reducir drásticamente el tiempo de desarrollo y despliegue de nuevas funcionalidades, permitiendo incluso la realización de experimentos para probar ideas de negocio y descubrir rápidamente las que generan más valor para los clientes y la organización en general. Si bien la colaboración entre product owners, desarrollo, QA y IT operations es esencial para el éxito en la entrega continua y la mejora continua del desarrollo de software, esta colaboración puede enfrentar obstáculos debido a percepciones pasadas de rivalidad entre los equipos.

1.2. Formulación del Problema.

En el contexto del desarrollo de software moderno, la implementación efectiva de prácticas de Integración Continua (CI) y Entrega Continua (CD) en un entorno de Cultura DevOps plantea desafíos significativos para los equipos de desarrollo.

Estos desafíos pueden manifestarse en formas diversas, incluida la resistencia al cambio organizacional, la falta de colaboración entre equipos, procesos obsoletos y una comprensión limitada de los principios DevOps.

El problema central que esta monografía se propone abordar radica en la dificultad que enfrentan los equipos de desarrollo al implementar prácticas de CI/CD en un entorno de Cultura DevOps. Esta dificultad puede conducir a retrasos en la entrega, errores de software y una calidad inferior del producto final, lo que afecta negativamente la eficiencia y la competitividad de las organizaciones en el mercado actual.

1.3. Justificación del Problema.

La implementación efectiva de prácticas de Integración Continua (CI) y Entrega Continua (CD) en un entorno de Cultura DevOps es crucial para mejorar la eficiencia, la calidad y la velocidad en el desarrollo de software. La adopción exitosa de CI/CD puede proporcionar una serie de beneficios tangibles, como una mayor satisfacción del cliente, una mayor competitividad en el mercado y una reducción de costos operativos.

En el contexto actual de la industria del desarrollo de software, donde la velocidad de entrega y la calidad del software son factores críticos para el éxito empresarial, la implementación efectiva de CI/CD se ha convertido en un imperativo para muchas organizaciones. Sin embargo, la dificultad en la implementación de estas prácticas en un entorno de Cultura DevOps plantea desafíos únicos que requieren una comprensión profunda y estrategias efectivas para superar.
Por lo tanto, esta monografía busca explorar los puntos clave en la implementación de CI/CD en equipos de desarrollo, centrándose específicamente en cómo construir una cultura DevOps efectiva que promueva la adopción exitosa de estas prácticas. Al abordar este problema, se espera contribuir al avance y la mejora continua en el campo del desarrollo de software y la adopción de prácticas DevOps.

1.4. Formulación de Objetivos.

En esta etapa, nos adentramos en la definición clara y precisa de los objetivos que perseguimos con nuestra investigación.

1.4.1. Objetivo General.

Analizar las estrategias clave para fomentar una cultura DevOps efectiva en equipos de desarrollo de software web, a fin de implementar exitosamente prácticas de Integración Continua (CI) y Entrega Continua (CD), mediante una revisión de la literatura, con el objetivo de mejorar la calidad del software, reducir el tiempo de desarrollo y aumentar la satisfacción del cliente.

1.4.2. Objetivos Específicos.

- Analizar los conceptos fundamentales de la cultura DevOps y su importancia en el desarrollo de software moderno.
- Identificar los principales desafíos y obstáculos en la implementación de CI/CD en equipos de desarrollo dentro de un entorno de Cultura DevOps.
- Proporcionar recomendaciones y mejores prácticas para fomentar una cultura DevOps efectiva en equipos de desarrollo, específicamente en relación con la implementación de CI/CD.

CAPÍTULO 2

MARCO TEÓRICO.

El capítulo de Marco Teórico proporciona una visión integral de los fundamentos y principios esenciales de DevOps, así como de los componentes clave de una cultura DevOps efectiva. Se explora la definición y concepto de DevOps, así como los principios básicos que subyacen a esta metodología, incluidas las Tres Vías fundamentales.

2.1. Pruebas de Software.

Las pruebas de software implican la detección y corrección de errores antes del lanzamiento, asegurando que el producto cumpla con los requisitos establecidos y funcione de manera confiable para los usuarios finales, siendo fundamentales en el desarrollo de productos de calidad.

2.1.1. Conceptos Fundamentales de las Pruebas de Software.

Las pruebas de software, una práctica esencial en el desarrollo contemporáneo de software, representan un proceso sistemático para evaluar y verificar la funcionalidad de una aplicación o sistema informático (**ammann2016introduction**). Estas pruebas tienen como objetivo principal identificar defectos, errores o fallos en el software antes de su implementación, con el propósito de mejorar su calidad y confiabilidad. En casos extremos, un error o defecto puede degradar los sistemas interconectados o causar fallas graves.

Por ejemplo, Nissan tuvo que retirar más de 1 millón de automóviles debido a un defecto en los detectores del sensor de la bolsa de aire. Otro caso preocupante fue el fracaso del lanzamiento de un satélite militar de 1,200 millones de USD debido a un error de software. Estos incidentes ejemplifican el impacto significativo que pueden tener los fallos de software. Según datos, en los EE. UU., las fallas de software costaron a la economía USD 1.1 billones en activos en 2016 y afectaron a 4.400 millones de clientes. Estas cifras hablan por sí solas sobre la importancia de asegurar la calidad del software y la necesidad de realizar pruebas exhaustivas para evitar tales consecuencias.

Aunque las prueba cuestan dinero, las empresas pueden ahorrar millones por año en desarrollo y soporte si cuentan con una buena técnica de prueba y procesos de control de calidad. Las primeras pruebas de software descubren problemas antes de que un producto salga al mercado. Cuanto antes reciban los equipos de desarrollo los comentarios de las pruebas, antes podrán abordar problemas como:

- Defectos arquitectónicos.

- Malas decisiones de diseño.
- Funcionalidad no válida o incorrecta.
- Vulnerabilidades de seguridad.
- Problemas de escalabilidad.

Cuando el desarrollo deja un amplio espacio para las pruebas, mejora la confiabilidad del software y las aplicaciones de alta calidad se entregan con pocos errores. Un sistema que cumple o incluso supera las expectativas del cliente genera potencialmente más ventas y una mayor cuota de mercado (**ibmsoftwaretesting**).

2.1.2. Ciclo de Vida de Pruebas de Software (STLC).

El ciclo de vida de las pruebas de software (Software Testing Life Cycle o STLC, por sus siglas en inglés) es un proceso sistemático que guía el desarrollo y la ejecución de las actividades de prueba a lo largo del ciclo de desarrollo de software. Este ciclo ocurre en paralelo al ciclo de vida de desarrollo de software y es crucial para garantizar la calidad y confiabilidad del producto final. A continuación, se describen brevemente las principales fases del STLC:

- **Análisis de Requisitos.** Durante la fase de análisis de requisitos, todos los requisitos de características recopilados durante el proceso SDLC se evalúan para determinar su idoneidad para la prueba y se identifican los aspectos comprobables. Los equipos de prueba pueden consultar con las partes interesadas relevantes para aclarar los requisitos. Durante esta fase, se identifican los entornos de prueba, donde se supone que se llevarán a cabo las pruebas. El análisis de viabilidad de la automatización también se lleva a cabo durante la fase de requisitos para identificar la posible necesidad de pruebas automatizadas y realizar cálculos económicos de los costos laborales basados en la estimación del proyecto. Los criterios de entrada y salida durante esta fase son los siguientes:

- **Criterios de entrada:** Requisitos del documento, criterios de aceptación y arquitectura del producto previsto.

- **Criterios de salida:** Matriz de trazabilidad de requisitos (RTM) aprobada e informe de viabilidad de automatización (si es necesario).

- **Planificación de Pruebas.** La planificación de pruebas durante STLC es la fase en la que la estrategia de prueba se describe en un documento de plan de prueba. Este plan generalmente lo determina y aprueba un Gerente Senior de Garantía de Calidad. Incluye detalles como las herramientas necesarias, el entorno de prueba, el cronograma de pruebas, los pasos de prueba y las funciones y responsabilidades. El plan se determina en base a un Análisis de Riesgos y Costos (RCA) y un cronograma aproximado para las pruebas.

  - **Criterios de entrada:** Análisis de requerimientos, RTM e informe de factibilidad de automatización.

- **Diseño o Desarrollo de Casos de Prueba.** Durante la fase de desarrollo del caso de prueba, se crean casos de prueba y scripts de prueba, y cada caso define todas las entradas de prueba, procedimientos, condiciones de ejecución y resultados anticipados. Los casos de prueba involucrados deben ser transparentes, eficientes y adaptables, con una cobertura del 100 por ciento de todos los resultados esperados. Los casos de prueba se desarrollan en base a los datos de prueba identificados, que han sido identificados y trabajados en base a las condiciones previas. Según el informe de viabilidad de la automatización, los scripts de automatización también deben crearse durante esta fase.

  - **Criterios de entrada:** Plan de prueba aprobado, incluidos los cronogramas y RCA.
  - **Criterios de salida:** Casos de prueba y scripts de automatización aprobados.

- **Configuración del Entorno de Prueba.** En la fase de configuración del entorno de prueba, se configuran e implementan los entornos de prueba, es decir, las condiciones de software y hardware para probar el producto. Esta fase suele incluir muchas herramientas de prueba como TestComplete, Selenium, Appium o Katalon Studio. La configuración del entorno de prueba se puede realizar simultáneamente con la fase de desarrollo del caso de prueba y, a veces, todo esto incluye la configuración de servidores de prueba aislados. Una vez que se ha configurado el entorno, se realizan pruebas de humo o comprobaciones de preparación para garantizar que los entornos funcionen con todas las funcionalidades previstas.

    - **Criterios de entrada:** Definición del diseño del sistema y de la arquitectura del proyecto.

    - **Criterios de salida:** un entorno de prueba completamente funcional, resultados de pruebas de humo y casos de prueba aprobados.

- **Ejecución de Pruebas.** Durante la fase de ejecución de pruebaa, las funciones del software se prueban en el entorno de prueba implementado utilizando los casos de prueba establecidos en función de los datos de la prueba. Durante esta fase se realizan la ejecución del script de prueba, el mantenimiento y el informe de errores, y los resultados de la prueba esperados se comparan con los resultados reales. Los resultados de la fase de ejecución de la prueba se recopilan y se informan a los equipos de desarrollo.

    - **Criterios de entrada:** Todos los criterios de salida de pasos anteriores.

    - **Criterios de salida:** Se realizan todas las pruebas y se documentan los resultados.

- **Cierre del Ciclo de Pruebas** La fase final de STLC es el cierre del ciclo de prueba STLC, durante la cual un informe de resultados de la prueba resume todo el proceso

de prueba con comparaciones entre los resultados esperados y reales preparados. Todos los defectos encontrados se registran como casos fallidos y se asignan a sus respectivos casos de prueba en el RTM. Cada gran solución realizada también se vuelve a probar para su cierre.

- **Criterios de entrada:** Resultados de las pruebas y registro de todas las fases anteriores.

- **Criterios de salida:** Informe de cierre de pruebas entregado y aprobado(**theknowledgeacader**
.

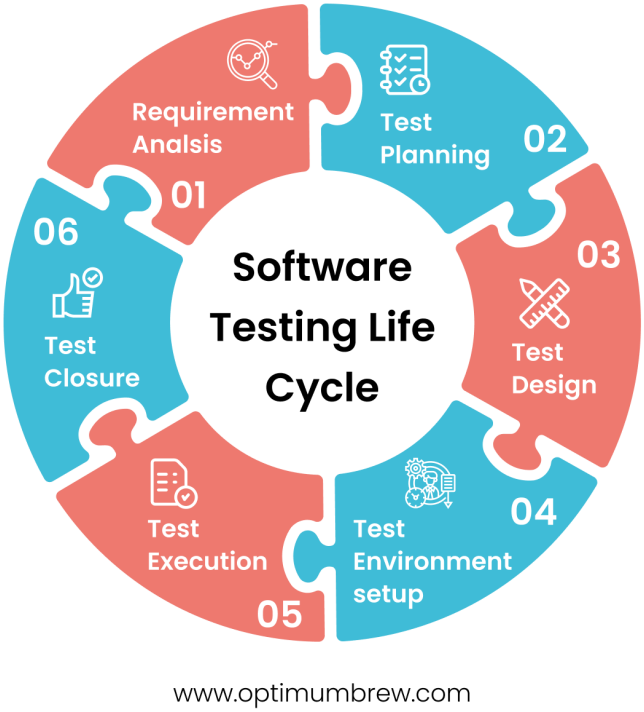

Figura 2.1: Ciclo de Vida de Pruebas de Software

2.1.3. Fundamentos de un Caso de Prueba.

En el contexto del Ciclo de Vida de Pruebas de Software (STLC), la fase de Diseño de Pruebas desempeña un papel fundamental. En esta etapa, se elaboran los casos de prueba que servirán como base para evaluar el comportamiento y la funcionalidad del software. Un

caso de prueba (test case) se define como un conjunto específico de condiciones y acciones diseñadas para verificar el comportamiento de un componente o sistema de software. (**pressman2014software**).

Los casos de prueba deben ser precisos y claros, abordando una única funcionalidad del software de manera independiente y con un propósito específico definido. Es fundamental utilizar un lenguaje claro y comprensible, evitando la inclusión de pasos innecesarios para mantener la legibilidad y la eficiencia. Además, deben ser relativamente pequeños y contener pasos atómicos para evitar la complejidad excesiva. La repetibilidad es clave, permitiendo su ejecución en diferentes momentos del ciclo de vida del software, y se debe mantener una terminología consistente para una comunicación efectiva. Estas prácticas aseguran la calidad y la eficacia de los casos de prueba en el proceso de desarrollo de software.

La estructura típica de un caso de prueba incluye varios elementos importantes (**pressman2014softwar**

- **Descripción del caso de prueba.** Una breve descripción que identifica el propósito y el objetivo del caso de prueba.
- **Precondiciones.** Las condiciones o situaciones que deben ser verdaderas antes de ejecutar el caso de prueba.
- **Pasos a seguir.** La secuencia de acciones específicas que deben llevarse a cabo para ejecutar el caso de prueba.
- **Datos de entrada.** Los valores o parámetros necesarios para ejecutar el caso de prueba.
- **Resultados esperados.** La salida o comportamiento esperado del software después de ejecutar el caso de prueba.
- **Condiciones de aprobación.** Los criterios que deben cumplirse para considerar el caso de prueba como exitoso.

Un ejemplo de un caso de prueba puede ser el siguiente:

Ejemplo de Caso de Prueba

Descripción: Verifique la respuesta cuando se ingrese un correo electrónico y una contraseña válidos.

Precondiciones: El usuario debe tener una cuenta registrada en el sistema.

Pasos:

1. Abrir el navegador web e ir a la página de inicio de sesión.
2. Ingresar el nombre de usuario y la contraseña registrados.
3. Hacer clic en el botón de ”Iniciar Sesión”.

Datos de entrada:

- **Nombre de usuario:** usuario_prueba
- **Contraseña:** contraseña123

Resultados esperados:

- El sistema redirige al usuario a la página de inicio después de iniciar sesión correctamente.
- Se muestra un mensaje de bienvenida al usuario.
- Se habilitan las funcionalidades específicas del usuario (por ejemplo, acceso a su perfil).

Condiciones de aprobación:

- Los pasos se completan sin errores.
- Los resultados esperados se cumplen.

BIBLIOGRAFÍA

Continuous Delivery. (s.f.). *Atlassian*.

Coupland, M. (2022). Five Cultural Changes Needed for DevOps Success. *Transparity*. https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/digital-blog/five-cultural-changes-you-need-for-devops-to-work

de Autores del Manifiesto Ágil, E. (2001). Manifiesto para el Desarrollo Ágil de Software.

Defense.AI, C. (2023). How to Implement an Effective CI/CD Pipeline.

Deshpande, A. (2016). DevOps: an Extension of Agile Methodology – How It will Impact QA?

G., N. (2023). *DevOps Tools for 2024*. https://nithinguruswamy.medium.com/devops-tools-for-2024-40112e1e657c

Gardini Miguel, P. (2023). 6 Essential DevOps Team Roles for Growing SaaS Companies. *The CTO Club*. https://thectoclub.com/news/devops-team/

Humble, J. & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*.

Kim, G., Behr, K. & Spafford, G. (2013). The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win.

Kim, G., Humble, J., Debois, P. & Willis, J. (2016). *DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*.

Pittet, S. (s.f.). How to setup continuous integration.

Šulák, V. (2018). The Three Ways of DevOps.

The Three Ways: The Principles Underpinning DevOps [Accedido el 12 de mayo de 2024]. (2024). http://itrevolution.com/the-three-ways-principles-underpinning-devops/

Van Merode, H. (2023). *Continuous Integration (CI) and Continuous Delivery (CD): A Practical Guide to Designing and Developing Pipelines*. Apress.

VersionOne. (2015). VersionOne Named a Leader in Gartner Magic Quadrant for Application Development Lifecycle Management.

ANEXOS.

ANEXO A. Hoja de Vida

Yuliana Marcela

MONTAÑO PEREZ

EGRESADA DE INGENIERÍA INFORMÁTICA

CONTACTO

78559065

yulianamarcela200@gmail.com

www.linkedin.com/in/yuliana-montaño

https://github.com/marceyuli

APTITUDES

- Capacidad de identificar y resolver problemas
- Trabajo en equipo
- Disposición a probar cosas nuevas
- Capacidad de investigación
- Creatividad

IDIOMAS

- Español - Nativo
- Inglés - Fluido

CAPACITACIONES

Bootcamp Frontend Mujeres 360
Desarrollo frontend con React, Bootstrap, CSS.

Programa Mujeres 360
Capacitación en habilidades blandas y tecnologicas.

CONOCIMIENTOS Y HABILIDADES

Desarrollo de Software

- Java
- libGDX
- HTML
- CSS
- Bootstrap
- Javascript
- React
- Flutter
- Nest

Control de versiones

- Git

Metodologías de Gestión de proyectos

- SCRUM
- PUDS

FORMACIÓN ACADÉMICA

Diplomado DEVOPS Essentilas
School of Engineering, Unidad de Postgrado
UAGRM - FICCT
Septiembre 2023 - Presente

Ingeniería Informática
Universidad Autónoma Gabriel René Moreno
Egresada

Idioma inglés
Centro Boliviano Americano
Programa de inglés, Avanzado
CertificaciónToefl ITP , Nivel C1

EXPERIENCIA LABORAL

Linkser| Analista de Desarrollo y Proyectos
Abril 2024 - Presente

Callnovo| Representante de atención al cliente
Diciembre 2023 - Marzo 2024

- Atención al cliente en ingles

Síntesis | Pasante en el área de desarrollo
Marzo 2023 - Septiembre 2023

- Documentación
- Pruebas de aplicacion
- Desarrollo frontend (Flutter)

TQ Group| Asistente Administrativa
2019 - 2023

- Control de inventario
- Control de cuentas de clientes