PRÁCTICAS DEVOPS DE ENTREGA CONTINUA DE SOFTWARE PARA LA TRANSFORMACIÓN DIGITAL DE LOS NEGOCIOS

PABLO ANDRES CASTAÑEDA GARCIA

Trabajo de Grado para Optar al Título de Magister en Ingeniería

Asesor

Rafael David Rincón B.

Profesor

Departamento de Informática y Sistemas

Universidad Eafit

UNIVERSIDAD EAFIT

MEDELLÍN

ESCUELA DE INGENIERÍAS

MAESTRÍA EN INGENIERÍA

2019

CONTENIDO

TABLAS

ILUSTRACIONES

RESUMEN

Las organizaciones hoy enfrentan una nueva dinámica económica: un mundo híper conectado, tecnologías accesibles a la mayor parte de la población y empresas haciendo grandes disrupciones y creando experiencias digitales que establecen un nuevo estándar para la interacción de los clientes con las marcas. En respuesta a esto todas las compañías, sin excepción, deben iniciar un proceso de transformación digital, donde el software es uno de los elementos claves. Así pues es relevante contar con capacidades para crear y evolucionar software con criterios de velocidad, calidad y eficiencia para preservar la existencia de las compañías. El movimiento DevOps, particularmente en las capacidades de Entrega Continua de Software define las prácticas que debe incorporar el proceso de ingeniería de software de una empresa para lograr producir y mantener software bajo las exigencias mencionadas. Esta investigación centra su atención en los procesos y prácticas del ciclo Construcción (Build), Pruebas (Test), despliegue (Deployment) y liberación en producción (Release). Aborda conceptos, definiciones y prácticas, y analiza la importancia de los procesos de gestión de la configuración, integración continua, automatización de pruebas y automatización del despliegue y la liberación en producción. Asimismo revisa las herramientas necesarias para implementar, simplificar, automatizar y administrar las prácticas de cada proceso. Finalmente se hace una propuesta para guiar la adopción y mejoramiento de las prácticas de entrega continua de software y sugiere un conjunto de métricas para evaluar el desempeño del equipo responsable del ciclo.

Palabras clave: Transformación Digital, DevOps, Entrega Continua de Software, Gestión de la Configuración, Integración Continua, Pruebas Continuas, Despliegue y Liberación Continuo, Ciclo Automático de Entrega de Software, Métricas de Desempeño de Entrega de Software.

ABSTRACT

Today organizations face a new economic dynamic: a hyper-connected world, technologies accessible to the majority of the population and companies making major disruptions and creating digital experiences that establish a new standard for the interaction between customers and brands. In response to this, all companies, without exception, must start a process of digital transformation, where software is one of the key elements. Therefore, it is important to have capabilities to create and evolve software with criteria of speed, quality and efficiency to preserve companies' existence. The DevOps movement, particularly in the Continuous Delivery of Software Capabilities, defines the practices that a company's software engineering process must incorporate in order to produce and maintain software under the aforementioned requirements. This research focuses its attention on the processes and practices in the Deployment Pipeline Cycle what comprises: Build, Tests, Deployment and Release. It approaches concepts, definitions and practices, and analyzes the importance of configuration management, continuous integration, test automation and deployment and release automation. It also reviews the tools required to implement, simplify, automate and manage the practices of each process. Finally, a proposal is made to guide the adoption and improvement of software delivery practices and suggests a set of metrics to evaluate the performance of the team responsible for the cycle.

Keywords: Digital Transformation, DevOps, Continuous Delivery, Configuration Management, Continuous Integration, Continuous Testing, Continuous Deployment and Release, Automatic Software Delivery Cycle (Deployment Pipeline), Software Delivery Performance Metrics.

INTRODUCCIÓN

Cuando se analizan los departamentos/áreas/direcciones/vicepresidencias de tecnología/sistemas/informática, con el propósito de determinar su razón de ser, la mayoría de los elementos nos llevan a concluir que su misión es la de crear y entregar soluciones capaces de proporcionar valor al negocio, es decir, Servicios de TI con la habilidad, de:

- Resolver problemáticas específicas de la organización
- Impactar positivamente los indicadores de tendencia y resultados del negocio
- Generar ventaja competitiva para la empresa
- Proporcionar capacidades para enfrentar los retos y desafíos propios de un entorno de negocios globalizado y altamente competitivo

Así pues, hoy en día la función TI juega un papel preponderante en el logro de los objetivos estratégicos de las corporaciones, dado que las tecnologías de información se están constituyendo como el vehículo por excelencia para la materialización de las estrategias y tácticas empresariales.

El reto para la función TI cada día es mayor, pues podemos evidenciar claramente que vivimos en un mundo donde las compañías se están transformando en entidades (negocios) digitales, y con ellas, sus interacciones con los clientes, aliados, accionistas, empleados, entes de regulación y control, entre otros. Es suficiente con tomar un teléfono móvil inteligente (Smartphone) de una persona del común para validar lo anteriormente mencionado, un simple ejercicio de exploración

a un dispositivo como estos nos daría cuenta que a través de unos cuantos clicks / touches es posible, entre otras: Pagar facturas, solicitar asistencia de una aseguradora en el lugar de un accidente, solicitar un servicio de taxi, reservar un hotel o restaurante, realizar compras on line. Luego, los momentos de verdad que experimentan los clientes con las compañías con las cuales interactúan, tienden cada vez más a ser interacciones digitales, “conversaciones” a través de una aplicación instalada en un móvil, tableta o una accesible a través de un navegador de internet de un computador portátil o de escritorio. Entonces, ahora la calidad de las experiencias de los clientes con las compañías ya no se medirá en términos del trato amable, respetuoso y cordial brindado por un representante de la empresa, mucho menos por una rápida y acertada respuesta proporcionada por este representante; por el contrario, la satisfacción del cliente estará determinada por los tiempos de respuesta a las transacciones en una aplicación, el nivel y extensión de funcionalidades a las que se puede acceder, las características de usabilidad, La confianza y seguridad de sus transacciones, entre otras.

Estamos, tal vez sin darnos cuenta, inmersos en lo que se ha denominado por muchos la “Economía de las Aplicaciones”. Forrester Research, de acuerdo con los datos suministrados por SNL Financial en un estudio reciente (Forrester, 2014), indica que los bancos en los Estados Unidos han cerrado alrededor de 2.700 sucursales en 2012 y proyectan cerrar un número cercano a las 13.000 para final de la década; lo anterior, como resultado de las estrategias de crear “Sucursales Virtuales en el Bolsillo” a través de aplicaciones móviles (Forrester, 2014) . Luego, no debe ser sorpresa que un banco como JPMorgan Chase tenga más desarrolladores que Google, dada la alta dependencia de TI para el desarrollo de su actividad económica.

Con estos antecedentes en mente, es posible inferir que uno de los mayores retos que enfrentan las organizaciones, es el de construir y modificar continuamente su software de forma más rápida, de mejor manera y a un menor costo. Sin duda, esto requiere de grandes transformaciones en términos de procesos, herramientas y

personas, para poder reducir los ciclos de entrega de las aplicaciones. Para esto es necesario contar con procesos de creación y evolución de software rápidos, repetibles, confiables, y lo más importante, altamente automatizados, de tal suerte que el tiempo que transcurra entre tener una idea y materializarla en una aplicación sea cuestión de horas o días, y no de meses o años.

Como respuesta a estos desafíos han surgido varios movimientos que vienen transformando los procesos de ingeniería del software para responder a la necesidad de crear aplicaciones a la velocidad, con la calidad y niveles de eficiencia que exige el entorno de competencia y transformación digital. Uno de ellos es el movimiento DevOps, este promueve la adopción de prácticas agiles y lean, poniendo un especial énfasis en la automatización, las personas y la cultura para lograr la colaboración entre los equipos de desarrollo de software y operaciones (infraestructura, plataforma) y así poder reducir el tiempo que transcurre entre el momento que se realiza un cambio en código y su liberación en producción funcionando correctamente.

Otro es el movimiento de entrega continua de software (Continuous Delivery), este plantea un conjunto de prácticas para liberar en producción, de forma rápida, consistente y con bajo riesgo, los cambios para modificar el software. El movimiento tiene como eje central un ciclo denominado Ciclo Automático de Entrega Continua de Software (Deployment Pipeline en idioma ingles). Con las prácticas del ciclo se pretende asegurar que el software puede ser liberado de manera rápida y confiable en cualquier momento.

De acuerdo con (Forsgren, Humble, & Kim, 2018) las prácticas de Entrega Continua de Software son una porción del movimiento DevOps. El movimiento DevOps completo está compuesto por las anteriores y las capacidades y prácticas en las áreas de Arquitectura, Gestión de Producto y Procesos, Gestión Lean y Medición, y Cultura. El propósito de este trabajo es identificar cuáles son las practicas DevOps que permiten lograr entregar software de forma continua, con los niveles de calidad y eficiencia requerida por los negocios en su proceso de transformación digital. Esta

investigación documenta las técnicas que permiten reducir el tiempo que transcurre entre la escritura del código y la liberación de software funcionando en producción. También considera los aspectos claves de las tecnologías requeridas para la implementación de prácticas base del ciclo automático de entrega de software como lo son: Gestión de la Configuración, Integración Continua, Automatización de Pruebas, Automatización del Despliegue y la Liberación en Producción. Finalmente se documentan en 11 fases unas consideraciones sugeridas por el autor para la implementación de las prácticas DevOps para la entrega continua de software, esto busca, construir una primera versión de un cuerpo de conocimiento capaz de orientar a los interesados en los aspectos claves para abordar un proceso de implementación

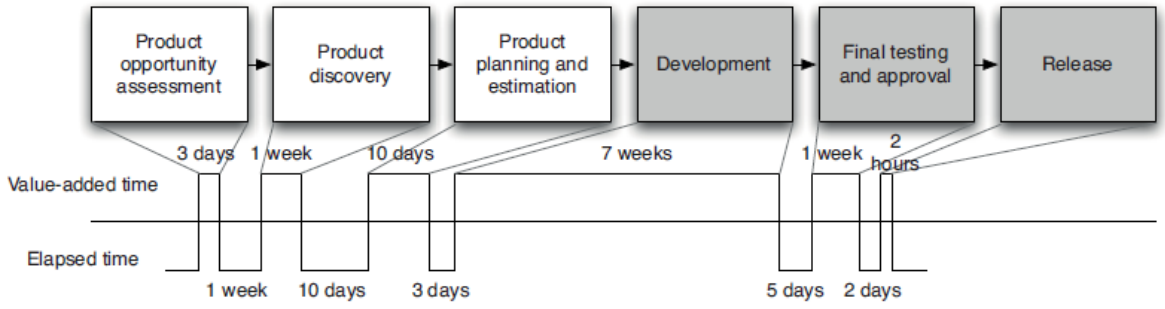
PLANTEAMIENTO DEL PROBLEMA

DECLARACION DEL PROBLEMA

Las empresas requieren construir aplicaciones y/o modificar las existentes con mayor velocidad, calidad y eficiencia para ser competitivas y mantenerse relevantes en la nueva realidad de la Economía Digital. Hoy día las compañías no cuentan con prácticas (procesos, técnicas y herramientas) que permitan lograr este propósito, y si bien, muchas han adoptado prácticas agiles como SCRUM, XP, SAFe, entre otras, éstas solo aceleran una parte del ciclo de vida del desarrollo de las aplicaciones y no logran reducir los tiempos totales de entrega de software a los clientes y/o procesos de negocio. Esto ocurre porque no se interviene el proceso completo.

De acuerdo con (Forsgren, Humble, & Kim, 2018), el ciclo de vida de las aplicaciones, el cual inicia con una idea o solicitud del mercado y finaliza una vez el software está liberado en producción, funcionando correctamente y a disposición de sus usuarios se divide en dos partes: una etapa de diseño y desarrollo, y una etapa de entrega del producto.

Ilustración 1: Flujo de Valor del Software



(Humble & Farley, CONTINUOUS DELIVERY Reliable Software Releases Through Build, Test, and Deployment Automation, 2011)

La etapa de diseño está representada en la Ilustración 1: Flujo de Valor del Software por los procesos representados en los cajones en blanco y los procesos nombrados como: evaluación de la oportunidad (Product Opportunity Assessment), descubrimiento del producto (Product Discovery), planificación y estimación del producto (Product Planning and Estimation). El trabajo acometido en esta etapa es altamente variable pues las actividades usan técnicas de ideación y creatividad para las cuales no existen parámetros de estimación que puedan estandarizarse (Kim, Patrick, Willis, & Humble, 2016). En consecuencia, hay una dificultad inherente por la naturaleza de las tareas, y si bien técnicas como Visual Story Map y la reducción de documentación exahustiva, entre otras, utilizadas por los equipos en las practicas agiles han ayudado a reducir los tiempos de conceptualización, los tiempos totales para entregar el software siguen siendo inaceptables. Por otro lado, está la etapa de entrega del producto representados en la gráfica (Ilustración 1: Flujo de Valor del Software) por los cajones sombreados y los procesos nombrados como: desarrollo (Development), pruebas y aprobación final (final testing and approval), y liberación (reléase). Durante esta fase, el trabajo a ejecutar es bien conocido y puede estimarse con exactitud, adicionalmente las tareas son de baja variabilidad en su esencia.

Hoy, una gran mayoría de empresas desconocen cuáles son las otras prácticas adicionales a las de desarrollo ágil (Etapa de Diseño), que contribuyen a disminuir los tiempos del ciclo de entrega, tener software con menos defectos y reducir los costos. Un estudio reciente de Forrester indica que el 31% de la industria no está utilizando prácticas y principios que son ampliamente considerados como necesarios para acelerar la transformación digital (Stroud, Oehrlich, LeClair, Kinch, & Kinck, 2017). La realidad es que durante la etapa de entrega del producto, prolifera la ejecución manual de actividades de construcción (Build), pruebas unitarias y de aceptación, despliegue (Deploy) en los diferentes ambientes y liberación en producción (Release). Asimismo no existe colaboración entre los equipos de operaciones y desarrollo, la comunicación se limita a generación de

solicitudes (tiquetes) formales con acuerdos de servicio que se extienden por días o semanas para la entrega de recursos o capacidades de infraestructura para los ambientes, motores de bases de datos, configuraciones en equipos de red y seguridad, instalaciones en ambientes de producción. Esto deja múltiples posibilidades para mejorar los tiempos de ejecución no solo de la etapa sino también del ciclo de entrega de software completo. Pues no responder con la celeridad requerida a la necesidad de modificar una característica, adicionar una funcionalidad o corregir un error tiene impactos económicos para las compañías, bien sea por:

- Pérdida de clientes potenciales por incapacidad de ejecución.
- Pérdida de ingresos derivados de una o más transacciones.
- Pérdida de clientes por insatisfacción.
- Multas o penalizaciones impuestas por organismos de regulación.

Pero no solo basta con aligerar el proceso de entrega de software, también es necesario asegurar que el software es capaz de entregar resultados confiables y predecibles, so pena de impactar el negocio, como se acaba de mencionar.

Se precisa entonces que los procesos de Ingeniería del Software incorporen prácticas para crear o modificar aplicaciones de software con los criterios que se han venido enunciando. En una economía movida por negocios digitales, donde el software es un actor fundamental para las interacciones entre los clientes y los proveedores de bienes y servicios es imperativo que el despliegue en producción de una nueva versión de una aplicación sea un tema que se pueda resolver en minutos, un par de horas a lo sumo, en lugar de meses o años.

Para que exista esta adopción en las empresas es necesario dar a conocer las prácticas, reconocer los beneficios de las mismas y contar con una guía para ayudar a su implementación.

JUSTIFICACIÓN

Para poder sobrevivir en el entorno de competencia de la era digital las empresas necesitan crear y ajustar sus tecnologías, incluyendo sus servicios y aplicaciones de software con mayor agilidad, pero sin ir en detrimento de la calidad de las mismas y sin elevar las inversiones, los costos y los gastos requeridos para ello.

Las compañías que carezcan de la capacidad para hacer y/o modificar mejor software, de manera más oportuna y a niveles de costo más óptimo serán sobrepasadas por aquellas que hayan implementado prácticas para lograr este cometido. Cobra importancia entonces, sintetizar un cuerpo de conocimiento que pueda ser tomado como referencia por parte de los líderes de los procesos de Ingeniería de Software y gestión de infraestructura y plataforma, para iniciar la transformación de los procesos actuales de entrega de software.

Esta investigación es pertinente toda vez que sus resultados permiten poner a disposición de los interesados en la materia una descripción de las prácticas DevOps para la entrega continua de software, junto con recomendaciones de adopción para facilitar su implementación. Asimismo, da cuenta de las herramientas de software que son indispensables para lograr que las prácticas logren los niveles de eficiencia y eficacia requeridas para crear valor.

OBJETIVOS

GENERAL

Identificar y documentar aspectos claves sobre las prácticas DevOps que permiten a las organizaciones lograr la entrega de software de forma continua con la velocidad, los niveles de calidad y eficiencia requerida por los negocios en su proceso de transformación digital.

ESPECÍFICOS

1) Describir los movimientos DevOps y Entrega Continua de Software, así como su importancia para la transformación digital.

2) Describir las prácticas de Ingeniería del Software alineadas con DevOps para la entrega continua de software.

3) Identificar y documentar los aspectos claves de las tecnologías requeridas para la implementación de prácticas DevOps orientadas a entrega continua de software.

MARCO TEÓRICO CONCEPTUAL

TRANSFORMACIÓN DIGITAL Y SOFTWARE

Mantenerse vigente y rentable en un entorno de negocios cada vez más globalizado y de competencia creciente es uno de los principales retos que enfrentan las compañías. Luego, todas ellas, independientemente de su tamaño o sector de la economía a la que pertenecen, requieren de diversas metamorfosis para lograr dar respuesta a estos desafíos. Para muchas organizaciones estas transformaciones tienen como objetivo principal lograr la evolución de la empresa en su modo presente y tradicional, cualquiera que éste sea, a un estado conocido como Negocio Digital, en el cual sea posible crear valor para los clientes a través de sus nuevas experiencias con los productos o servicios, y nuevos modelos de negocio basados en fuentes y esquemas de generación de ingresos diferentes a los actuales.

Se considera un negocio digital aquel que ha implementado nuevos diseños de negocio eliminando las barreras y las diferencias entre el mundo real (físico) y el digital (GARTNER, 2018). Si bien el término digital podría cambiarse por virtual o ciberespacio, parece ser más cómodo para las personas (Raskino & Waller, 2015). Estos nuevos diseños de negocio tienen sus fundamentos en la convergencia sin precedentes entre las personas, las empresas, y los objetos inteligentes y conectados (productos y/o servicios). Se espera que estos diseños de negocio superen, incluso a aquellos que emergieron durante el boom de Internet y los e-Business, debido a la integración de los tres elementos mencionados (Lopez, 2014).

Así pues, un Negocio Digital utiliza tecnologías de información (TI) para la incorporación de nuevas e innovadoras prácticas para crear, no sólo nuevas formas de generar riqueza sino también para implementar mecanismos orientados a simplificar y automatizar los procesos creados para resolver las necesidades y exceder las expectativas crecientes y cambiantes de los clientes. Esto significa que

estas compañías ahora dependen completamente de las TI para materializar la estrategia de negocios y desarrollar su actividad económica.

Considerando esto, se requiere una nueva función de TI dentro de las empresas, preparada para responder de forma oportuna a las dinámicas de la empresa y los requerimientos de mercado, regulación y clientes, entregando soluciones de TI capaces de crear el valor esperado por los clientes y las mismas organizaciones. En concreto, una nueva función de TI capaz de poner en funcionamiento, cada vez que se requiera, software con niveles adecuados de calidad en periodos de tiempo que se pueden medir, en horas, máximo en días, y no en semanas.

En respuesta a esto, metodologías ágiles para el desarrollo de software como Scrum y Extreme Programming (XP) han emergido y han sido implementadas con éxito y su adopción viene en auge. Una encuesta realizada por versionOne entre Julio y Noviembre de 2015 en los 5 continentes, entregó como resultado que el 95% de las compañías han institucionalizado prácticas ágiles para el desarrollo de software, donde el 62% de ellas lo han hecho buscando acelerar la entrega de aplicaciones al negocio (VersionONE Inc, 2016). Sin embargo, la misma encuesta muestra que las prácticas consideradas como claves para reducir el tiempo que transcurre entre una idea y software funcionando que materializa la misma, se encuentran en niveles de implementación inferiores al 50%, con un promedio del 30%. De ahí se infiere que las organizaciones, en su mayoría, se han concentrado en diseñar e implementar procesos y prácticas como Scrum y Extreme Programming (VersionONE Inc, 2016), las cuales se centran en escribir software y no en las actividades requeridas para implementar en producción, de forma rápida y confiable, el código escrito por los desarrolladores.

CICLOS DE VIDA DE DESARROLLO DE SOFTWARE

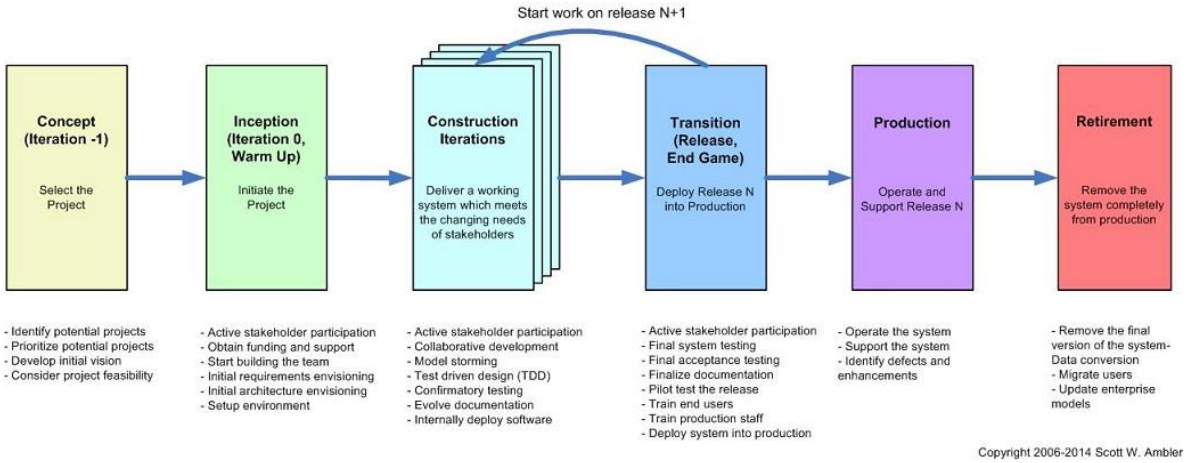
El software o aplicaciones son entidades o productos complejos de construir, mantener, mejorar y evolucionar. Por esta razón es necesario contar con métodos estructurados para su creación. A estos métodos, se les conocen como Modelos de Ciclo de Vida de Desarrollo de Software, en inglés y por sus siglas The Software Development Life Cycle (SDLC). Existe un número importante de modelos, entre los más conocidos y utilizados están: Cascada, Modelo-V, Prototipado, Espiral, Iterativo, Incremental y Ágil (CLEVERISM, 2018). En términos generales todos los modelos contemplan, a su modo las etapas de planificación, Análisis, Diseño, Desarrollo, Despliegue y Mantenimiento. Como se ha dicho, la transformación digital requiere prácticas y modelos de Ingeniería del Software que permitan poner a disposición de las necesidades cambiantes de los negocios, las aplicaciones de software que estos requieren en periodos de tiempo muy cortos; por tal razón, esta investigación propone como modelo de ciclo de vida de desarrollo de software el ágil, que comprende las siguientes fases o etapas, de acuerdo con (Ambler & Lines, 2012):

- Concepción
- Incepción
- Construcción / Iteraciones
- Transición
- Producción
- Retiro

En ese orden de ideas, las prácticas DevOps para la entrega continua de software que se abordan en este trabajo son algunas de aquellas requeridas durante las fases:

- Construcción / Iteraciones
- Transición

Ilustración 2: Ciclo de Vida de Desarrollo de Software Ágil



Ambler, S., & Lines, M. (2012). *DISCIPLINED AGILE DELIVERY*

Si fuese necesario contextualizar las prácticas abordadas en este trabajo de acuerdo al modelo de ciclo de vida de desarrollo de software cascada, el cual, de acuerdo con (Winston, 1970) comprende las siguientes etapas:

- Requerimientos de sistema y software: capturados en un documento de requerimientos de producto.
- Análisis: Los resultados son modelos, esquemas y reglas de negocio.
- Diseño: Los resultados son la arquitectura de software.
- Codificación. Desarrollo, pruebas e integración de los componentes de software.
- Pruebas: Descubrimiento y corrección sistemática de defectos.
- Operaciones: Instalación, migración, soporte y mantenimiento del sistema complete.

Las prácticas se enmarcarían en las etapas de:

- Codificación.
- Pruebas.
- Operaciones.

DEVOPS

Conceptos y Definiciones

Las prácticas y capacidades necesarias para implementar en producción software de calidad con la velocidad y los niveles de eficiencia requeridos provienen del movimiento que hoy conocemos como DevOps. A continuación se proporcionan algunas definiciones plausibles:

De acuerdo con (Bass, Weber, & Liming, 2015) “DevOps son un conjunto de prácticas orientadas a reducir el tiempo que transcurre entre el momento de la adición de un cambio en las líneas de código al código fuente y el instante en el cual dichas líneas de código han sido implementadas en producción, asegurando alta calidad” (p, 4).

Para (DEVOPS AGILE SKILLS ASSOCIATION DASA, 2016) “DevOps es un modelo cultural y operacional que fomenta la colaboración para permitir áreas de TI de alto desempeño para lograr las metas del negocio” (p, 9).

En términos de (GARTNER, 2018) “DevOps representa un cambio en la cultura de TI, foco en la rápida entrega de servicios de TI a través de la adopción de prácticas Agiles y Lean. Enfatiza en las personas (y cultura), busca mejorar la colaboración entre los equipos de operaciones y desarrollo”.

En palabras de (Willis, 2012) DevOps representa la convergencia de muchos movimientos filosóficos y de gerencia documentados en marcos de referencia y cuerpos de conocimiento como Lean, Teoría de Restricciones, El Sistema de Producción de Toyota y el Movimiento Toyota Kata. Tal convergencia ha dado como resultado un conjunto de prácticas técnicas, de arquitectura y de cultura para resolver los desafíos planteados de velocidad, calidad y eficiencia en la entrega de servicios y aplicaciones cuya base fundamental es el software.

Importancia de DevOps

Investigaciones realizadas durante más de 4 años y documentadas por (Forsgren, Humble, & Kim, 2018) dan cuenta que organizaciones de todos los tamaños, industrias y tipo de tecnologías de software que han implementado prácticas DevOps obtienen mejores resultados, comparativamente contra aquellas compañías donde las prácticas son incipientes o nulas, así:

- Despliegue de código 46 veces más frecuente.

- Tiempo Commit to Deploy (tiempo que transcurre entre el momento de la adición de un cambio en las líneas de código al código fuente y el instante en el cual dichas líneas de código han sido implementadas en producción) 440 veces más corto.

- Tiempo promedio de recuperación (MTTR) de un error desplegado en producción 170 veces más corto.

- Tasa de Falla de los cambios desplegados en producción 5 veces menor.

Adicionalmente, otro estudio de (Puppet Labs, 2014) asegura tener evidencia cuantitativa que el desempeño de TI y las prácticas DevOps contribuyen al desempeño organizacional. Este estudio encontró que las compañías con áreas de TI de alto desempeño tienen una probabilidad dos veces mayor de superar sus metas de rentabilidad, participación de mercado y productividad.

Historia de DevOps

De acuerdo con (Willis, 2012), es el resultado de varios movimientos, a saber:

El Movimiento Lean, el cual surgió en los inicios de los años 1980 como un intento de codificar el sistema de producción de Toyota y el cual hizo popular las técnicas de Mapa de Flujo de Valor (Value Stream Map), los tableros Kanban y el mantenimiento productivo total.

El Movimiento Agil, iniciado en el 2001 con la promulgación del manifiesto ágil por parte de diez y siete pensadores del desarrollo de software con el propósito de alivianar los métodos y procesos de desarrollo de software y teniendo como principio básico “Entregar software funcionando de manera frecuente, desde un par de semanas hasta un par de meses, con preferencia de tiempos cortos”.

El Movimiento de la Conferencia Velocity, durante la cual, en el año 2007 se presentó el trabajo “10 Deploys per day: Dev and Ops Cooperation at Flickr”.

El Movimiento de la Infraestructura Ágil, durante la conferencia de Toronto en el año 2008, Patrick Debois y Andrew Schafer realizaron una presentación sobre la aplicación de principios ágiles a la infraestructura. Allí se gestó el primer DevOpsDay que tuvo lugar en Ghent, Belgica en 2009 y fue acuñado el término DevOps.

El Movimiento Entrega Continua, esta idea fue presentada inicialmente en la conferencia Agile 2006 por Tim Fitz, en un blog titulado “Continuous Deployment” y luego fue enriquecido por Jez Humble y David Farley con su patrón “The Deployment Pipeline”.

Practicas DevOps

Investigaciones de (Forsgren, Humble, & Kim, 2018) han identificado 24 capacidades claves que contribuyen estadísticamente al mejoramiento en los

procesos de entrega de software y a su vez impactan el desempeño organizacional. Ellos las han clasificado en 5 categorías, a saber:

Categoría 1 – Capacidades de Entrega Continua de Software

1) Utilizar control de versiones para todos los artefactos de producción.

2) Automatizar el proceso de despliegue.

3) Implementar integración continua.

4) Utilizar métodos de desarrollo “Trunk-Based”.

5) Implementar pruebas automáticas.

6) Implementar gestión de los datos de prueba.

7) Incorporar los aspectos de seguridad en las fases de diseño y pruebas.

8) Implementar entrega continua de software.

Categoría 2 – Capacidades de Arquitectura

9) Utilizar arquitecturas de bajo acoplamiento.

10) Permitir a los equipos definir sus propias arquitecturas.

Categoría 3 – Capacidades de Producto y Procesos

11) Recolectar e implementar las recomendaciones de los clientes.

12) Hacer visible el flujo de trabajo durante todo el ciclo

13) Trabajar con lotes o cantidades pequeñas.

14) Fomentar y habilitar al equipo para realizar experimentos.

Categoría 4 Capacidades de Gestión Lean y Monitorización

15) Tener un proceso liviano para la aprobación de cambios.

16) Monitorizar las aplicaciones y la infraestructura para tomar decisiones de negocio.

17) Monitorizar la salud de los servicios y aplicaciones proactivamente

18) Administrar y establecer límites para el trabajo en progreso (Work-In-Progress)

19) Visualizar el trabajo para monitorizar la calidad y comunicarlo al equipo

Categoría 5 - Capacidades Culturales

20) Implementar una cultura tipo generativa.

21) Animar y apoyar al equipo a aprender.

22) Apoyar y facilitar la colaboración entre los equipos.

23) Proporcionar recursos y herramientas para que el trabajo sea significativo para los individuos.

24) Apoyar el liderazgo transformacional.

Las prácticas descritas en la categoría 1 y denominadas como capacidades de entrega continua de software son las prácticas en las cuales este trabajo hace énfasis. Si bien el resto de las practicas también tienen un impacto estadísticamente significativo, de acuerdo con (Forsgren, Humble, & Kim, 2018), no han sido consideradas en esta investigación.

ENTREGA CONTINUA DE SOFTWARE

Conceptos y Definiciones

Para lograr la puesta en funcionamiento del software con los criterios de velocidad, calidad y eficiencia necesarios, se requiere además de escribir código que satisfaga los requisitos de negocio y del cliente, ejecutar otro conjunto de actividades en los procesos orientados a la construcción, prueba, despliegue, y liberación del software en producción. Esos procesos diseñados y ejecutados de forma orquestada, y en la medida de lo posible completamente automatizada, se conocen con el nombre de Entrega Continua (En inglés, Continuous Delivery). Este es un enfoque de la Ingeniería del Software en la cual los equipos están todo el tiempo produciendo software de valor para el negocio en ciclos cortos y asegurando que el software puede ser liberado de manera confiable en cualquier momento (Chen, 2015). Según (Fowler, 2013), una organización ha implementado Entrega Continua cuando se satisfacen las siguientes condiciones: “

- El software se encuentra siempre listo para desplegar en cualquiera de sus etapas del ciclo de vida.

- El equipo da prioridad a tener el software siempre listo para desplegar, en lugar de trabajar sobre nuevas funcionalidades.

- Cualquier persona puede obtener, en cualquier momento, retroalimentación de manera rápida y automatizada acerca de si los sistemas están preparados para el despliegue de software, luego que alguien ha ejecutado un cambio sobre ellos.

- Es posible ejecutar un despliegue de cualquier versión del software a cualquier ambiente con sólo dar clic en un botón.

Se trata entonces de un conjunto de principios y prácticas cuyo propósito es reducir el costo, los tiempos y los riesgos asociados con la entrega de software con nuevas funcionalidades o corrección de errores a los usuarios. Vale la pena recordar que con la creciente adopción de prácticas ágiles, el software se desarrolla de manera iterativa e incremental, por lo cual, sólo es después del primer despliegue y liberación en producción que ocurren los mayores esfuerzos, no sólo para mantenerlo funcional, sino también para su evolución, por lo cual estas prácticas resultan relevantes.

Hoy en día, el despliegue y liberación de software en cualquiera de los ambientes de su ciclo de vida, puede y debería ser un proceso: frecuente, predecible, rápido, de bajo riesgo y costo (Humble, ThouhgtWorks, 2014); a la fecha, existen casos de referencia, buenas prácticas documentadas y herramientas de software para la orquestación y automatización de las actividades requeridas para la Entrega Continua. Los principios en los cuales se fundamenta la entrega continua de software, según (Humble & Farley, 2011), se resumen a continuación:

Crear un Proceso Repetible y Confiable para Liberar Software. Al ser un proceso que se repite con frecuencia, más temprano que tarde terminará siendo un proceso que entrega resultados plausibles y predecibles. Sin embargo, para lograrlo se requiere de otros dos principios; automatización y control de versiones.

Automatizar Todo lo que sea Posible. Producir resultados adecuados y pronosticables, así como lograr el despliegue y liberación del software con sólo dar clic en un botón, sólo es posible encargando a las máquinas de ejecutar tareas repetitivas, en las cuales la intervención humana, en lugar de ayudar, sólo genera propensión a la comisión de errores.

Mantener todo Bajo el Control de Versiones. Cualquier artefacto que se requiera para construir, desplegar, probar y liberar una aplicación, debe mantenerse bajo control de versiones, incluyendo: Scripts de prueba, casos automatizados de prueba, scripts para configuraciones de red, scripts para configuración de parámetros en servidores web, servidores de bases de datos y servidores de aplicación, entre otros.

Incorporar la Calidad en Todas las Actividades. La mayoría de prácticas y procesos de la Entrega Continua están pensados para lograr la identificación de defectos tan pronto como sea posible durante el ciclo de despliegue y liberación. Específicamente, se propende por hallarlos antes que cualquier artefacto sea ingresado al control de versiones. Se es conforme con la premisa que establece que entre más temprano se encuentren los errores, menos costoso será repararlos. Lo anterior establece tácitamente que, las pruebas no son una fase, y la responsabilidad de realizar pruebas es de todos los miembros del equipo y no una responsabilidad exclusiva de los ejecutores de pruebas.

Completado significa Liberado. Una nueva funcionalidad o corrección de errores está completada sólo hasta que ha sido desplegada o liberada, al menos en un ambiente similar al de producción, pues realmente una característica ha sido finalizada si y solo si ha llegado a manos de los usuarios del software. Es comprensible que hay situaciones en las cuales no resulta conveniente desplegar en el ambiente de producción, por lo cual es aceptable el despliegue en un ambiente de características equivalentes.

Todas las Personas son Responsables en el Proceso de Entrega. La colaboración entre todos los que participan en la entrega de software es fundamental. Al final de cuentas, si la entrega no se cumple, el equipo falla como equipo y no como individuos, ya que entregar software no depende de una sola persona o equipo funcional, por lo cual el trabajo en equipo desde el inicio es crucial.

Mejora Continua. La primera vez que se despliega y se libera en producción una aplicación es solo el inicio; con seguridad, la aplicación va a requerir de cambios para su evolución y continua adaptación a las necesidades del negocio y la corrección de errores será necesaria para su correcto funcionamiento. Luego, más despliegues y liberaciones serán necesarios, por lo cual el proceso también debe evolucionar para ser más eficiente y eficaz todo el proceso.

Importancia de la Entrega Continua de Software

De acuerdo con (Humble & Farley, 2011), el principal beneficio de la entrega continua de software es la creación de un proceso de liberación de software en producción que es repetible, confiable y predecible, el cual a su vez genera reducción en los ciclos de entrega, lo que permite que nuevas funcionalidades o corrección de errores lleguen a manos de los usuarios más rápido y con los niveles de calidad exigidos.

Prácticas de Entrega Continua de Software

Las prácticas de entrega continua de software buscan resolver el problema que según (Humble & Farley, 2011) es la dificultad más grande que enfrentan los profesionales del software, ésta es: “Si alguien tiene una buena idea sobre un software, ¿Cómo la ponemos a disposición de los usuarios tan pronto como sea posible?”

(Humble & Farley, 2011) definieron un patrón nombrado en inglés como Deployment Pipeline y para el cual no es posible hallar una traducción precisa al español, pero que llamaremos Ciclo Automático de Entrega de Software. Este es la implementación automatizada de los procesos de construcción (Build), despliegue

(Build), Pruebas (Test) y Liberación en producción (Release) de una aplicación o software. Este patrón tiene tres propósitos:

- Hacer visible para todos los involucrados cada parte del proceso de construcción (Build), despliegue (Build), Pruebas (Test) y Liberación en producción (Release).

- Mejorar la retroalimentación para que los problemas se identifiquen y resuelvan tan pronto como sea posible durante el proceso.

- Permitir a los equipos desplegar (Deploy) y liberar en producción (Release) una versión del software a cualquier ambiente a través de un proceso completamente automatizado.

De acuerdo con (Humble & Farley, 2011), la entrega continua de software tiene su sustento en tres prácticas claramente identificadas: Gestión de la Configuración, Integración Continua, y Pruebas Continuas. A su vez (Forsgren, Humble, & Kim, 2018), consideran que las prácticas, tales como: Automatizar el proceso de despliegue, Utilizar métodos de desarrollo “Trunk-Based”, Implementar gestión de los datos de prueba e Incorporar pruebas de seguridad, también hacen parte de las prácticas de entrega continua de software. Para (Kim, Patrick, Willis, & Humble, 2016), 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.

En lo sucesivo, de este capítulo se describirán sólo algunas de las prácticas DevOps de naturaleza técnica para lograr la entrega continua de software.

GESTION DE LA CONFIGURACIÓN

Conceptos y Definiciones

De acuerdo con (Humble & Farley, 2011), es común que la gestión de la configuración sea tratada como un sinónimo de control de versiones y la definen como: un proceso por el cual se almacenan, recuperan, identifican de manera única y se modifican todos los artefactos relevantes al software y las relaciones entre ellos.

Para (CMMI Product Team, 2010) el proceso incluye actividades, como:

- Identificar la línea base de configuración de cada artefacto.
- Controlar los cambios a los artefactos o ítems de configuración.
- Construir o proporcionar las especificaciones para construir artefactos a partir del sistema de gestión de la configuración.
- Mantener la integridad de las líneas base
- Proporcionar información exacta del estado y datos de configuración actual a los desarrolladores, usuarios y clientes.

Algunos ejemplos de artefactos, son:

- Código de software y sus dependencias, incluyendo librerías, contenidos estáticos, entre otros.
- Documentación de historias de usuario, requerimientos y procedimientos.
- Documentación de arquitectura.

Importancia de Gestión de la Configuración.

Para (Humble, 2008), la gestión de la configuración persigue, principalmente dos propósitos; el primero, capacidad de reproducir, esto es, tener la posibilidad de proporcionar cualquier ambiente de forma completamente automatizada y tener la certeza que cualquier ambiente creado de esta manera tiene la misma configuración. El segundo, trazabilidad, es decir, para cualquier ambiente es posible determinar sus versiones de forma rápida y precisa, y es posible comparar versiones previas para encontrar diferencias.

Según (Humble, 2008), a partir de estos dos propósitos será entonces posible obtener beneficios en las siguientes perspectivas:

Recuperación de Desastres.

Poder construir rápidamente el ambiente caso de ocurrencia de evento catastróficos o fallas que obliguen a estas medidas para recuperar los servicios.

Auditoria.

Para demostrar la integridad del proceso de entrega de software es necesario poder suministrar información sobre los componentes, sus versiones y su historial.

Calidad.

La reducción de los tiempos de aprovisionamiento de ambientes permite la posibilidad de mayores tiempos para construir calidad en cada componente y realizar pruebas con mayor nivel de cobertura.

Velocidad en Corrección de Defectos.

Ante el descubrimiento de errores o vulnerabilidades de seguridad, es posible tener en corto periodo de tiempo una nueva liberación en producción del software, dada la capacidad de reproducir ambientes de forma automática.

Lo anterior es compatible y complementado por (Humble & Farley, 2011), quienes plantean que la gestión de la configuración permite: reproducir ambientes completos a partir de sus componentes y configuración, incluyendo: sistemas operativos y sus niveles actualización (parches), configuración de red, configuración de los componentes de capa media de software y las aplicaciones o software desplegados sobre el mismo. Asimismo, establecen que la gestión de la configuración da la posibilidad de identificar cada cambio en los ambientes, dando cuenta sobre: qué fue el cambio, quién lo hizo y cuándo fue realizado. También afirman que la gestión de la configuración permite demostrar y asegurar la conformidad con los requisitos de seguridad y cumplimiento establecidos por la propia organización o los organismos externos encargados de su vigilancia.

Por otro lado, las investigaciones de (Forsgren, Humble, & Kim, 2018) revelan que se puede predecir un alto desempeño de la organización de IT cuando se mantienen bajo un sistema de control de versiones tanto el código como la configuración del sistema, la configuración de las aplicaciones y los scripts que automatizan la construcción (Builds) y configuración.

Por último y no menos importante, de acuerdo con (Humble & Farley, 2011), la manera como se desarrolla el proceso de gestión de la configuración determina tres aspectos importantes en la entrega de software, ellos son: la administración de cambios en el proyecto, el registro de evolución de los sistemas y del software y la forma como los equipos pueden desarrollar un modelo de trabajo colaborativo.

Prácticas de Gestión de Configuración.

Control de Versiones.

Un sistema de control de versiones es un mecanismo típicamente automatizado a través de una herramienta de software que permite guardar múltiples versiones de

los archivos o elementos constitutivos de un software o aplicación. Dicho mecanismo permite el acceso a las diferentes personas del equipo responsable por la entrega del software y mantiene registro detallado de los cambios que se estos realizan a los archivos.

Practica 1 – Todo Bajo el Sistema de Control de Versiones

La experiencia práctica y las investigaciones tanto de (Humble & Farley, 2011) como de (Kim, Patrick, Willis, & Humble, 2016) sugieren que los siguientes ítems se incluyan bajo el control de versiones:

- Código fuente
- Scripts, Archivos, procedimientos o Documentos con definiciones de pruebas
- Scripts para bases de datos
- Scripts para construcción (Build) y despliegue (Deployment)
- Documentación de cualquier índole relevante al proyecto, como: documentación de requerimientos, procedimientos de instalación, notas de liberación, entre otras.
- Librerías
- Compiladores
- Sistemas Operativos

- Software de capa media
- Archivos de configuración DNS
- Archivos de configuración de Firewall
- Cualquier archivo para crear contenedores (Docker, Mesosphere, Kubernetes)
- Archivos de configuración de servicios de nube pública, como: Plantillas de AWS Cloudformations, Archivos de Microsoft Azure Stack DSC, Archivos OpenStack HEAT, entre otros.
- Scripts y archivos de configuración requeridos para crear infraestructura compartida, como: Enterprise Bus Services, Sistemas de gestión de Bases de Datos, Dispositivos de red y conectividad.

Como particular, (Humble & Farley, 2011) sugieren no mantener bajo el control de versiones los binarios que resultan de la compilación de la aplicación, primero por el tamaño de los archivos, los cuales con frecuencia son muy grandes; segundo, porque un proceso automático de construcción (Build) debe ser capaz de recrearlos y de ese modo poder identificar sin equivocación una sola versión de la aplicación y no entrar en confusión cuando hayan dos envíos (Commits) para la misma versión, una para el código fuente y otra para los binarios.

Practica 2 – Adicionar con Frecuencia al Tronco (Trunk)

Los estudios de (Forsgren, Humble, & Kim, 2018) encontraron que el desarrollo basado en el tronco (Trunk-Based Developement) en lugar de usar ramas (Branches) para las nuevas funcionalidades, tienen un impacto significativo estadísticamente hablando, en el desempeño del equipo responsable de entregar software. Según los autores, los equipos que mejores resultados obtuvieron fueron aquellos que tenían menos de tres ramas activas (Active Branches) en cualquier momento; para lograr esto, el tiempo de vida de las ramas (Branch Lifetime) es de menos de un día, esto significa que el código de cada rama se integra al tronco una o más veces al día.

En resumen, la recomendación es desarrollar nuevas funcionalidades o refactorizar funcionalidades ya existentes de manera incremental y confirmar su código sobre el tronco tan pronto como sea posible, de modo que se pueda asegurar la integración y funcionamiento del software.

Administrar las Dependencias.

El software o las aplicaciones grandes y modernas que conocemos no son entidades monolíticas puestas en vacío, por el contrario, están construidas de diversos elementos, cada uno con propósitos específicos, que interactúan entre sí para dar lugar a su funcionamiento. Se requieren prácticas conocidas como división por componentes (Componetization), para lograr:

- Lograr mantener la aplicación en estado Listo para Liberación en entornos altamente cambiantes.

- Permitir el trabajo colaborativo de forma eficiente y eficaz cuando los equipos están compuestos por un alto número de personas.

- Lograr arquitecturas de bajo acoplamiento.

- Evitar el uso de ramas (Branching) para adicionar nuevas funcionalidades o refactorizar código.

Esta división por componentes requiere, igualmente, de prácticas para lograr los objetivos que se pretenden.

Una dependencia se define según (Humble & Farley, 2011), como la situación en la cual una pieza de software requiere de otra pieza para poderse, bien sea, construir o ser ejecutada. De acuerdo con los mismos autores, una taxonomía de alto nivel nos habla de dos tipos de piezas de software, ellas son, Librerías y Componentes.

Una librería es un paquete de software, comúnmente desplegado en forma de binarios, que el equipo de desarrollo no controla y que se actualiza con poca frecuencia, mientras que los componentes son piezas de software que son desarrolladas por el mismo equipo de desarrollo u otro equipo de desarrollo dentro de la misma organización.

Practica 1 – Administrar Librerías

De acuerdo con la visión de (Humble & Farley, 2011), se deben mantener copias locales de las librerías en algún repositorio, asegurando que el sistema de construcción (Build) especifique la versión exacta de las librerías en uso. Ellos también sugieren que las librerías se guarden teniendo en cuenta su momento de uso, por lo cual el repositorio debería contener 3 niveles, marcados como: Build-Time, Test-Time, Run-Time. Todo lo anterior supone también la utilización de una convención de nombres para su identificación.

Se propone también por parte de (Humble & Farley, 2011) la utilización de herramientas como Mave e Ivy para el manejo automático de dependencias, permitiendo declarar exactamente cuáles versiones de las librerías hacen parte del proyecto de software.

Practica 2 – Administrar Componentes

Es una buena práctica de arquitectura dividir las grandes aplicaciones en componentes para obtener beneficios como la reutilización de la misma pieza de software en diferentes aplicaciones. En tal caso debe realizarse una separación de los procesos automáticos de construcción (Build) del componente, es decir, uno para cada aplicación, de modo que pueda asegurarse que los artefactos que se están creando son los correctos. Si los componentes no son compartidos entre aplicaciones es posible tener un solo proceso automático de construcción (Build).

Administrar los Ambientes.

Será tratado en el capítulo Gestión de Infraestructura y Plataforma para DevOps.

Herramientas para la Gestión de la Configuración

Descripción General de las Herramientas para la Gestión de la Configuración

Las herramientas de gestión de la configuración (SCCM – Software Change and Configuration Management) son habilitadoras clave de los equipos de desarrollo de software. Esta categoría de herramientas abarca sistemas de control de versiones, anteriormente conocidos como sistemas de administración de código fuente (Source Code Management - SCM), CVCS y DVCS (Centralized and Distributed Version Control System), que pueden versionar y administrar clases amplias de objetos tipo fuente, binarios, metadatos y cambios a los objetos. Es importante destacar que estas herramientas protegen el código fuente y otros activos de desarrollo de modificaciones accidentales o malintencionadas. También permiten a los equipos compartir el acceso al código sin producir conflictos que resulten en daños, al tiempo que establecen la trazabilidad, la responsabilidad y la separación de tareas. (GARTNER, 2015).

Necesidad de Herramientas para la Gestión de la Configuración

En general, las organizaciones invierten en herramientas de Gestión de la configuración del software para aumentar sus capacidades de velocidad y agilidad, tanto de los procesos de desarrollo de nuevas funcionalidades como la modificación de las existentes (GARTNER, 2015) , pero a su vez para tratar con retos y desafíos, como:

- La necesidad de dar respuesta a interrogantes sobre el cómo, dónde y quién realiza los cambios en el proceso de desarrollo de software, lo cual ha provocado cambios sustanciales en la adopción de herramientas para este equipo.

- Adopción de prácticas ágiles, como refactorización, rama por historia e integraciones continuas.

- Incremento en la cantidad y variedad de información que debe ser versionada.

- Requisitos para soportar múltiples clientes y plataformas.

- La necesidad de coordinar el versionamiento de software empaquetado y de sistemas mecánicos.

- La demanda de soporte global y distribuido como las ofrecidas por Git y Mercurial ha aumentado, por lo que, las herramientas open-source y de pago han respondido agregando soporte para DVCS y mejorando la seguridad, la capacidad de auditoría y la integración con herramientas de terceros. Git, dominante en el mercado, y Mercurial son utilizadas por aproximadamente el 30% de los desarrolladores. Mientras que otro 30% usa Subversion o las ofertas CVCS de código abierto anteriores. Productos como los de Borland, IBM, Perforce y Serena Software, representan ahora alrededor del 40% del total.

- Muchas organizaciones desconocen los problemas de seguridad, auditoría y propiedad intelectual (PI) de sus desarrollos de software. Es probable que las grandes empresas sigan implementando herramientas locales o privadas en la nube.

- Las empresas más pequeñas y algunos grupos de trabajo están usando instancias de SCCM basadas en la nube como GitHub y Visual Studio Online de Microsoft. Algunos de ellos sólo están aprovisionando el SCCM a nivel departamental, mientras que otros están usando la nube para ampliar su acceso al código fuente, casos de prueba y otros artefactos del proceso de

desarrollo. Sin embargo, la adopción de SCCM basada en la nube por parte de las empresas representa menos del 5% del total

Funcionalidades y Características Principales de las Herramientas

Las herramientas se ofrecen en diferentes versiones, desde aquellas que entregan funcionalidades básicas para permitir el desarrollo del núcleo del software, y otras más avanzadas para coordinar el acceso concurrente y el control de versiones en múltiples flujos y múltiples etapas del ciclo de vida de desarrollo por parte diferentes equipos y personas (GARTNER, 2015). Los productos más básicos proporcionan control de acceso y control de versiones de archivos y objetos individuales. Los productos más elaborados agregan metadatos, conjuntos de cambios, funciones para brindar administración del flujo de trabajo, acceso concurrente y acceso a archivos para otras partes del ciclo de vida (GARTNER, 2015).

Las soluciones open-source o de fuente abierta cumplen con los requisitos funcionales de muchos equipos de desarrollo, sin embargo, las ofertas comerciales ofrecen capacidades de escalabilidad mejorada, rendimiento, seguridad e integración (GARTNER, 2015) . En Tabla 1: Funcionalidades Herramientas de Gestión de Cambios y Configuración de Software, se destacan las funciones típicas que se encuentran en las herramientas de SCCM e identifica a sus principales usuarios.

Tabla 1: Funcionalidades Herramientas de Gestión de Cambios y Configuración de Software

Usuario Principal	Funcionalidad Básica	Funcionalidad Intermedia	Funcionalidad Avanzada
Desarrollador	Control de acceso Versiones de artefactos	Branching Agrupación de módulos	Desarrollo paralelo Build support Conjunto de cambios Refactorización

Equipo de Desarrollo	Asignación y seguimiento de items de trabajo	Staging Aprobación	Coordinación a través de múltiples variaciones Informes
Equipo de despliegues	Flujo de trabajo Programación	Logging Recuperación	Coordinación de procesos paralelos

GARTNER. (2015). Market Guide for Software Change and Configuration Management Software

A continuación, se describen de manera general las funcionalidades principales de una herramienta de Gestión de la configuración – SCCM.

- Gestión de versiones y compilaciones.
- Control de acceso y Colaboración global.
- Asignación y seguimiento de ítems de trabajo.
- Análisis de impacto integral.
- Flujos de trabajo.
- Branching.

Gestión de versiones y compilaciones.

Proporciona visibilidad coherente a todos los elementos de versiones del ciclo de vida del software. Los participantes pueden desplazarse fácilmente a cualquier instancia de tiempo para obtener y manipular el estado de cualquier configuración o para determinar las actualizaciones asociadas con las nuevas versiones (GitLab, 2018).

Control de acceso y Colaboración global.

Control de acceso y colaboración global para equipos de desarrollo, ya sean desde una única ubicación o distribuidos geográficamente. Todos los integrantes se benefician de la visibilidad y el control centralizados que proporciona un único repositorio, lo que finalmente reduce los costos y el mantenimiento asociado a tener las réplicas (GitLab, 2018).

Asignación y seguimiento de ítems de trabajo.

Los equipos de desarrollo pueden rastrear y gestionar los cambios en una amplia gama de activos de desarrollo digital, como archivos de origen, peticiones de cambio, defectos, tareas, requisitos, historias de usuario y discusiones (Perforce, 2018).

Análisis de Impacto integral.

Proporciona la capacidad de crear compilaciones automatizadas y generar informes de versiones. Ofrece un almacén de datos (DataMart) para entregar análisis completos de datos y asistencia en la toma de decisiones para ayudar a mejorar la visibilidad de los proyectos de software (GitLab, 2018).

Flujos de trabajo.

Todos los requisitos, las tareas y los archivos de origen pertinentes se llevan directamente a los desarrolladores y los gestiona su IDE. El diseñador de flujos de trabajo permite a los usuarios diseñar el mejor conjunto de procesos y normas para la entrega de software (GitLab, 2018).

Branching.

Permite la creación de nuevas ramas (Branches) de código. Para los proyectos agiles proporcionan un espacio para almacenar los cambios que están siendo usados por algunos equipos de desarrollo pero que todavía no están completos para ser fusionados con el tronco.

INTEGRACIÓN CONTINUA

Conceptos y Definiciones

Integración Continua.

Integración continua es un proceso donde los miembros de un equipo, valga la redundancia, integran el código resultante de su trabajo de desarrollo. Por lo general, la frecuencia típica es que cada persona realice al menos una integración diariamente, lo que se traduce en múltiples integraciones por día. Cada integración se verifica a través de una construcción automatica (Build) que incluyen pruebas para detectar los errores de integración tan pronto como sea posible (Fowler, 2006). En términos prácticos, esto significa que cada desarrollador mantiene su trabajo en progreso continuamente integrado con el de otros desarrolladores. El objetivo es asegurar que el software siempre está en estado operacional estable, dado que se

asegura que el código escrito por cada miembro del equipo de desarrollo es compatible con el de sus pares.

Construcción Automática (Build).

La definición anterior de integración continua hace necesario abordar el concepto construcción automática (Build) para una correcta y completa comprensión del concepto. Para (Duvall, Matyas, & Andrew, 2007), una construcción automática, es: “Mucho más que una compilación, una construcción automática (build) puede estar compuesta de la compilación, prueba, inspección, despliegue entre otras actividades. Una construcción automática (Build) es el proceso de juntar el código unir y verificar que el software funciona como una unidad cohesionada”.

En adelante, en este trabajo el término construcción o construcción automática se emplearán como sinónimo.

Pruebas.

Como se dijo en la definición de construcción automática, esta puede incluir pruebas. En esta etapa las pruebas pueden ser de diferentes tipos; unitarias, de componentes, de sistema, de desempeño, de seguridad, entre otras (Duvall, Matyas, & Andrew, 2007).

Ciclo de Integración Continua

A continuación, para ayudar al entendimiento del proceso de integración continua se utiliza el ejemplo propuesto por (Fowler, 2006), con la intención de explicar el concepto, tomando en muchos casos el texto exacto:

1) El ciclo inicia cuando se requiere modificar el código que actualmente funciona, bien sea porque se requiere adicionar una nueva funcionalidad, corregir un error identificado en producción, refactorizar el código para reducir la deuda técnica, entre otros.

2) Para esto, el desarrollador a cargo hace una copia del código (esta copia se denomina código integrado), en la maquina local de desarrollo.

3) Luego, el desarrollador realiza lo que sea necesario para completar el propósito del desarrollo y/o adicionar cambios a las pruebas automáticas existentes.

4) Después, el desarrollador realiza una construcción automática en su propia máquina de desarrollo, si no hay errores después de realizar la compilación, crear los ejecutables y ejecutar las pruebas automáticas, se considera que la construcción es exitosa

5) Una vez se tiene una construcción automática satisfactoria en la máquina local, el desarrollador toma nuevamente una copia del código del repositorio de versiones (dado que es posible que hayan ocurrido cambios desde la copia en el paso 1 y este momento). Se adicionan los cambios que se realizaron en el paso 3 y se realiza de nuevo una construcción automática en su máquina local, tal como se hizo en el paso 4. Si esta construcción es satisfactoria, el desarrollador ahora puede proponer (Commit / Check-In) sus

cambios. Por el contrario, si los cambios que se hicieron en el paso 3 dan como resultado errores en la construcción automática, el desarrollador debe realizar las modificaciones hasta lograr que los cambios que él ha hecho funcionen correctamente con la copia más actualizada que se toma del repositorio del control de versiones para que pueda proponer (Commit / Check In) de sus cambios.

6) Una vez se proponen los cambios (Commit / Check In) por parte del desarrollador, es momento para realizar una nueva construcción automática (Build), pero ahora no en la máquina local del desarrollador sino en una máquina conocida como Máquina de Integración. Para esto se toma el código fuente [este contiene los cambios propuestos (Commit / Check In)] del repositorio. Si los resultados de la construcción automática (Build) es exitosa, se dice que los cambios se han completado. Si hay errores en la construcción, nuevamente el desarrollador debe trabajar en reparar el código hasta que se obtenga el resultado esperado en la construcción.

Automatización del Proceso de Integración Continua

El proceso que se describió en la sección anterior es un proceso altamente automatizado para permitir el trabajo de múltiples personas del equipo de desarrollo de forma concurrente, evitar conflictos entre desarrolladores y permitir encontrar errores, evitando construcciones automáticas cuando se detecten cambios que son incompatibles entre sí.

Ilustración 3: Componentes del Sistema Automático de Integración Continua

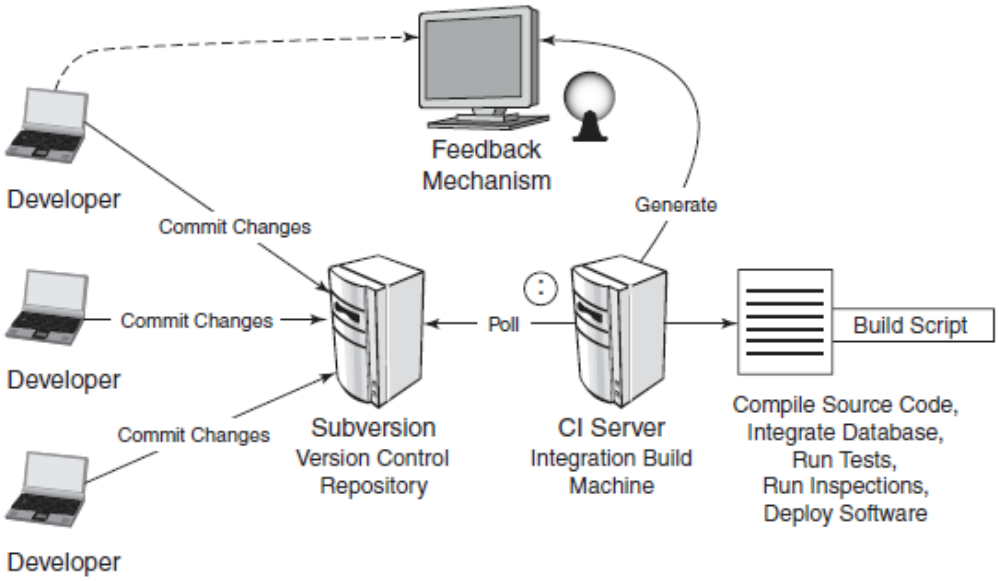


FIGURE 1-1 The components of a CI system

Duvall, P. M., Matyas, S., & Andrew, G. (2007). Continuous Integration: Improving Software Quality and Reducing Risk

En el proceso automatizado los desarrolladores envían cambios (Commit/Check-In) al repositorio de control de versiones el cual está siendo constantemente monitoreado por el servidor de integración (CI Server) para establecer si ha habido cambios en el código. Una vez el servidor de integración detecta modificaciones, hace una copia del código cambiado y ejecuta el script de construcción automática realizando la integración del software y notificando a los desarrolladores de los resultados. El proceso se repite de forma sucesiva cada vez que el servidor de integración descubre que ha habido cambio en el código que reposa en el repositorio de control de versiones.

Los componentes del sistema que automatiza el proceso de integración continua, son:

- Repositorio de control de versiones

- Un servidor de integración

- Un script o conjunto de scripts para compilar, probar, inspeccionar y desplegar el software

Importancia de la Integración Continua

Para (Duvall, Matyas, & Andrew, 2007), encarna las tácticas que le permiten a los desarrolladores hacer cambios en el código, sabiendo que si el software deja de funcionar recibiremos retroalimentación inmediata sobre esta situación y en consecuencia, será posible realizar los ajustes necesarios para regresarlo al estado funcional. Así pues, estas prácticas contribuyen no solo a obtener software de calidad, sino también a la velocidad y eficiencia de su entrega, pues los defectos que hasta esta etapa se hayan podido introducir se encontrarán más rápido y con menos esfuerzo por parte de los desarrolladores.

Los autores (Humble & Farley, 2011) afirman que la integración continua es el proceso que permite mantener siempre una versión de software que funciona, pues permite identificar y reparar con prontitud cualquier cambio que derive en funcionamiento incorrecto del mismo.

Por otro lado, las prácticas de integración continua habilitan el desarrollo en paralelo, permitiendo que múltiples equipos puedan avanzar en la creación de código con la confianza que proporciona saber los conflictos e incompatibilidades que pueden hacer fallar el software, serán encontrados rápidamente y podrán ser reparados de la misma manera.

Adicionalmente, las investigaciones de (Forsgren, Humble, & Kim, 2018) encontraron que los equipos que integran con frecuencia, tienen un mejor desempeño en términos de entrega de software que aquellos que mantienen el código separado en ramas o derivaciones (Branches).

Prácticas de Integración Continua

Enviar (Commit/Check-In) Cambios Frecuentemente

Para (Humble & Farley, 2011), la práctica más importante de integración continua es enviar cambios con frecuencia al repositorio de control de versiones para iniciar el ciclo de construcción automática y validar que no existen errores. Para lograr que esta actividad se haga con alta periodicidad (Duvall, Matyas, & Andrew, 2007), sugieren que los cambios se hagan de modo que solo se modifique un componente o elemento a la vez, idealmente una vez se finalice una tarea que agrega, elimina o actualiza el código.

No enviar (Commit/Check-In) Cambios cuando hay errores

En caso que el proceso de construcción anterior haya fallado, no se debe hacer copia del último código del repositorio de control de versiones. Se requiere entonces disponer de mecanismos para que los desarrolladores puedan identificar el estado de salud del código en el repositorio (Duvall, Matyas, & Andrew, 2007).

Ejecutar construcciones privadas (Private Builds)

Los desarrolladores, una vez finalizan sus modificaciones y ejecutan las pruebas unitarias, deben tomar el código más actualizado del repositorio de control de versiones y ejecutar la construcción automática para adicionar los cambios realizados (Duvall, Matyas, & Andrew, 2007). El proceso anteriormente descrito es comúnmente automatizado por parte de los servidores de integración (CI Server) modernos que ofrecen la funcionalidad conocida con diferentes nombres, como: envío previamente probado (Pretested Commit), construcción personal (Personal Build), entre otros. Con esta característica, será el servidor de integración el que tome los cambios que el desarrollador hizo localmente en su máquina y realice la

construcción automática, tomando el código más actualizado del repositorio de control de versiones (Humble & Farley, 2011).

Avanzar luego de haber completado las pruebas e inspecciones

Hasta que las pruebas no demuestren que el código satisface completamente los requisitos y los estándares, los desarrolladores no deben iniciar una próxima modificación o adición de código, pues en caso de detectar errores, la prioridad para el desarrollador debe ser la corrección de los defectos.

Herramientas para la Integración Continua

Descripción General de las Herramientas para la Integración Continua

Las herramientas para la Integración Continua posibilitan la colaboración y la productividad del equipo de desarrollo de software, al mantener la integridad del código fuente mediante su centralización y la automatización de las compilaciones, pruebas e inspecciones. Las más modernas se integran con otras herramientas de DevOps para crear un canal automatizado de entrega de código, comúnmente denominado CI / CD (integración continua / entrega continua) (Forrester, 2017).

Necesidad de Herramientas para la Integración Continua

Los desarrolladores a menudo están desarrollando componentes de software que se comparte entre varias personas del equipo. En este típico escenario sin el uso

de herramientas de esta naturaleza el proceso puede resultar en errores y retrasos en el despliegue de nuevas versiones de las aplicaciones.

La adopción de micro-servicios por parte de los desarrolladores, combinada con la entrega continua, les impone una mayor demanda para escribir y mantener pruebas unitarias automatizadas para garantizar el cumplimiento funcional. Los desarrolladores necesitan herramientas que les posibilite ejecutar pruebas de manera eficiente.

Los desarrolladores se enfrentan de manera frecuente a errores o demoras en el proceso de compilación y con herramientas que entregan información reducida o limitada sobre el uso de los sistemas o estadísticas sobre los ambientes de prueba, lo que incide en altos tiempos y esfuerzos para identificar los errores de código.

Los equipos de desarrollo y entrega de las aplicaciones que ya han adoptado soluciones ágiles y DevOps para optimizar sus procesos de desarrollo y despliegue, han identificado que de manera continua deben optimizar sus procesos y adquirir nuevas tecnologías para obtener mayor escalabilidad, simplicidad, desempeño, analítica, entre otros. (Forrester, 2017)

Entre mayor sea la fase en la que se detecte un defecto de código, mayor es el riesgo y el costo. Típicamente, los defectos se detectan en las fases tardías de producción.

Funcionalidades y Características Principales de las Herramientas

A continuación se describen de manera general las funcionalidades principales de una herramienta de Integración Continua:

- Flexibilidad en el modelo de despliegue.
- Agilidad y escalabilidad en el desarrollo.
- Soporte a Kubernetes y Docker.

- Capacidad de autoservicio.
- Control de acceso.
- Gestión centralizada del equipo.
- Integración y compilación automática.
- Versionamiento automático de pruebas.
- Detección ágil de código defectuoso.
- Seguridad del desarrollo.
- Alertas y notificaciones.

Flexibilidad en el modelo de despliegue

Las herramientas de CI/CD ofrecen modelos flexibles de despliegue, ya sea en las premisas o en la nube. Permite implementar las herramientas en las premisas con su propio motor Kubernetes administrado o en cualquiera de los servicios de Kubernetes populares basados en la nube. No existe una dependencia del proveedor, ya que una arquitectura en las premisas se puede migrar, en cualquier momento, a los servicios en la nube. Además, soporta la integración con la mayoría de los proveedores de nube pública (CloudBees, 2018).

Algunas herramientas ofrecen modelos de despliegue en la nube como servicio, que ofrecen la plataforma de integración continua que entrega la infraestructura y el ambiente de prueba necesario, en conjunto con los objetivos de despliegue, una gran variedad de dependencias ya preinstaladas y listas para integrarse con su repositorio para ejecutar las respectivas pruebas de manera automática (CloudBees, 2018).

Agilidad y escalabilidad en el desarrollo

Las herramientas de CI necesitan ejecutar cientos o miles de compilaciones por día, dependiendo del tamaño de una organización. Para satisfacer esta demanda, las herramientas de CI ahora admiten la ejecución paralela de trabajos, la administración de dependencias y el escalamiento automático (Forrester, 2017). Las pruebas se pueden ejecutar de manera distribuida en máquinas virtuales separadas, lo que permite agregar la cantidad que sea necesaria según el crecimiento. Además, algunas herramientas poseen capacidades de autocrecimiento, de manera que se incrementen o disminuyen máquinas virtuales para garantizar compilaciones oportunas y de buen desempeño, al tiempo que se optimizan los costos (GitLab, 2018).

Las capacidades de escalabilidad se pueden realizar con tantos procesos paralelos de compilación como sean necesarios, por lo tanto, de acuerdo con los requerimientos de la organización, se puede predecir el crecimiento requerido (CloudBees, 2018).

Soporte a Kubernetes y Docker

La mayoría de estas herramientas soportan nuevas tecnologías como Kubernetes y Docker, de manera que permite que las capacidades del cluster escalen automáticamente, bajo demanda, de forma elástica creciente y decreciente. Además, utilizan controles de verificaciones de salud para identificar nodos defectuosos y reemplazarlos según sea necesario (CloudBees, 2018).

Capacidad de autoservicio

Los líderes de desarrollo desean herramientas que sean fáciles de usar y administrar para respaldar el autoservicio de sus desarrolladores. Esta necesidad de aprendizaje rápido y operación simple para soportar modelos de autoservicio de los desarrolladores, ha resultado en el aumento de herramientas basadas en nube tipo software, como servicios SaaS.

Controles de acceso

Las herramientas ofrecen mecanismos de control de acceso basado en roles y políticas adicionales de seguridad, que permiten definir permisos de acuerdo con los usuarios, equipos, agentes, carpetas o trabajos. La definición de roles posibilita presentar tableros de control con información personalizada, de acuerdo con los roles de los miembros del equipo.

Gestión centralizada del equipo

Esta característica permite administrar de manera centralizada y fácil, tanto el equipo de trabajo como sus proyectos de desarrollo, es posible también configurar grupos de la organización para asignar permisos granulares y entregar una visión general de alto nivel de los proyectos.

Integración y compilación automática

El código fuente que es desarrollado por cada uno de los usuarios del equipo de desarrollo se integra de manera automática. Además, posibilita que todos los desarrolladores colaboren sobre el mismo archivo compartido, sin incurrir en conflictos de código, versionamiento o compilación de la aplicación. Se realiza la compilación automática de fusiones de código de diferentes desarrolladores y en

paralelo, de manera que se garantice que estas sean ejecutadas con la mayor velocidad (Forrester, 2017).

Versionamiento automático de pruebas

Las pruebas ejecutadas son versionadas de manera consistente para no generar conflictos o fallas en las compilaciones, de esta manera se posibilita que los desarrolladores contribuyan de manera continua con sus cambios y modificaciones de código. (GitLab, 2018).

Seguridad del desarrollo

Protege los proyectos de desarrollo al habilitar un único espacio, que es accedido a través de una llave SSH, que requiere autenticación para acceder a la información. Además, la caché se encuentra cifrada, lo que posibilita que los datos confidenciales sean almacenados localmente (CloudBees, 2018).

Alertas y notificaciones

Los equipos de desarrollo desean saber de manera rápida cuándo se presentan fallas en los procesos de compilación, pero no quieren ser invadidos por notificaciones ni correos. Las herramientas realizan clasificación y priorización de las alertas y notificaciones para dirigir los mensajes a la audiencia adecuada.

PRUEBAS CONTINUAS

Conceptos y Definiciones

A lo largo de este documento, uno de los imperativos sobre los cuales se ha hecho énfasis es la calidad del software entregado. Así pues, el proceso de entrega de software debe ser capaz de construir y/o modificar software que satisfaga los requisitos funcionales y No funcionales para el cual es creado. El reto de lograrlo reside en la complejidad inherente al software, pues como se ha mencionado, las aplicaciones ya no son entidades monolíticas, por el contrario, cada vez están más desacopladas en componentes, la arquitectura está basada en Micro-Servicios e interactúan con servicios externos u otras aplicaciones para su funcionamiento. Se

infiere entonces que para que el software opere correctamente los elementos del cual está hecho deben hacerlo de la misma manera y se confirma con lo expresado por (Duvall, Matyas, & Andrew, 2007), quienes afirman que para lograr software o sistemas verdaderamente confiables debe asegurarse confiabilidad a nivel de objetos, por lo cual es necesario realizar pruebas a cada elemento y esto debe realizarse cada vez que se introduzcan cambios en ellos. Tanto por la complejidad como por el número de pruebas, es necesario automatizar las pruebas unitarias y de aceptación cada vez que se envíen cambios al repositorio de control de versiones e incluso los desarrolladores deberían poder ejecutar pruebas automáticas en sus propias estaciones para identificar defectos (Forsgren, Humble, & Kim, 2018).

Para lograr una identificación temprana, las pruebas no deben ser ejecutadas en una fase posterior y/o por un equipo de aseguramiento de la calidad separado luego de haber completado el desarrollo, por el contrario, esto debería realizarse por los mismos desarrolladores (Kim, Patrick, Willis, & Humble, 2016).

Las pruebas son el mecanismo a través del cual se obtiene retroalimentación del impacto positivo o negativo que ha tenido un cambio sobre el estado operacional o funcionamiento del software o uno de sus elementos.

Una taxonomía de alto nivel permite agrupar las pruebas, en: Pruebas Unitarias, Pruebas de Aceptación, Pruebas de Integración. A continuación, una definición de cada una de ellas.

- Pruebas Unitarias. Estas evalúan un solo método, clase o función, de forma separada, sin considerar las interacciones reales de las entidades evaluadas con otros elementos del software (Kim, Patrick, Willis, & Humble, 2016).

- Pruebas de Aceptación. Estas validan la aplicación como un todo para comprobar a alto nivel que una funcionalidad opera de acuerdo con los

requisitos funcionales especificados por los usuarios (Humble & Farley, 2011).

- Pruebas de Integración. Estas revisan que el software interactúa correctamente con otras aplicaciones o servicios en producción.

Otro tipo de pruebas requeridas durante el proceso de entrega de software, son:

- Pruebas de Aceptación de Requisitos No Funcionales (Capacidad, Desempeño, Usabilidad, entre otros)

- Pruebas de Aceptación de Requisitos de Seguridad

- Pruebas de Exploración.

Importancia de las Pruebas Continuas

Cuando las pruebas se realizan de manera automática a nivel unitario o de componentes, es posible identificar rápidamente los defectos que causa un efecto adverso, de modo que puedan ser reparados en las primeras etapas del proceso de creación de software. Esta aproximación no sólo contribuye a la calidad del software sino también a la eficiencia, dado que de este modo es posible suprimir pruebas que se realizan sobre la interface de usuario que con costosas de ejecutar tanto por el tipo de herramientas y licenciamiento que se necesita como por el esfuerzo que requiere la definición de los casos de prueba (Fowler, 2012).

Teniendo en cuenta que las organizaciones en su proceso de transformación digital estarán creando más aplicaciones y modernizando las existentes, el no automatizar las pruebas llevará a las compañías a un circulo de destrucción, propuesto por (Kim,

Patrick, Willis, & Humble, 2016), este es: “Entre más código se escriba, más tiempo y dinero se requiere para probar el software, lo cual para la gran mayoría de las compañías es un esquema inaceptable para el propio modelo de negocio”. Así pues, nuevamente la automatización de pruebas en el largo plazo representa ahorros en la creación, mantenimiento y evolución del software.

Prácticas de Pruebas Continuas

Automatizar las Pruebas tanto como sea Posible

Para lograr los niveles de calidad y confiabilidad esperados es necesario que el software en su conjunto y cada componente sean probados cada vez que se realice un cambio, bien sea en la aplicación, en su configuración y/o en los ambientes y así comprobar su correcto funcionamiento. Dado el alto número de cambios y la frecuencia con los que se introducen modificaciones, deben automatizarse las pruebas de todos los niveles, esto es, las pruebas unitarias, pruebas de componentes, pruebas de aceptación, pruebas de integración, entre otras (Humble & Farley, 2011).

Asegurar Alta Cobertura en las Pruebas Unitarias y de Componentes

Las pruebas unitarias son fáciles y de bajo costo, pues lo que se prueba no tiene dependencias externas ni exige el despliegue e instalación. Como se mencionó antes, en la medida en que se valide el correcto funcionamiento de más elementos o componentes del software, este será más confiable. La forma de hacerlo es probando cada componente.

Una alta cobertura de pruebas unitarias aún en entornos con prácticas de desarrollo orientados a las pruebas (Test-Driven Developement TDD) no reemplazan las pruebas de aceptación, pues el objetivo de estas últimas es validar que la aplicación

hace lo que el cliente requiere, mientras que las pruebas unitarias evalúan que el software hace lo que desarrollador desea que sea haga. Son en esencia dos tipos de prueba con propósitos distintos.

Pruebas de Aceptación Funcionales también deben ser Automáticas

Este tipo de pruebas tienen la capacidad de identificar defectos que no se pueden capturar con las pruebas unitarias (automáticas o manuales). Estas pruebas cuando se diseñan y automatizan con la participación de personas expertas en pruebas, en lugar de dejarlo solo como una responsabilidad del equipo de desarrollo, no sólo se escriben de mejor manera sino que son más eficaces (Humble & Farley, 2011).

Herramientas para las Pruebas Continuas

Descripción General de las Herramientas para Pruebas Automáticas de Aceptación

El mercado de la Automatización de pruebas de software ofrece herramientas, tecnologías, componentes y servicios que constituyen los elementos críticos de la automatización de pruebas para realizar de manera automática los análisis de código, pruebas funcionales, pruebas de carga y de desempeño.

La Automatización de pruebas funcionales (Functional Testing) corresponde a herramientas utilizadas para diseñar, desarrollar, mantener, gestionar, ejecutar y analizar las pruebas funcionales automáticas para las aplicaciones que se ejecutan en diferentes plataformas (incluyendo escritorio, web, móvil y servidor). Estas herramientas posibilitan realizar pruebas funcionales automáticas al manipular la interfaz de usuario (User Interface – UI) de la aplicación, lo que se conoce como Automatización de pruebas de UI, o interactuar con la aplicación a través de interfaces de programación (Application Programming Interface – API), lo que se

conoce como Automatización de pruebas de API. El escaneo de UI y APIs posibilita la construcción de un repositorio de prueba con artefactos reutilizables que se versionan de manera automática (Gartner, 2018).

Necesidad de Herramientas para Pruebas de Aceptación Automáticas

Las organizaciones adoptan prácticas ágiles y DevOps que sólo son posibles con la incorporación de alternativas que introduzcan la automatización de los respectivos procesos.

Las pruebas funcionales corresponden a una de las etapas claves durante el desarrollo de software, pero requieren de tiempo de los equipos y su ejecución manual resulta costosa. Como regla general, las pruebas manuales deberían representar menos del 20% de todas las actividades de pruebas y las pruebas automatizadas deberían representar más del 80% para lograr llegar a los niveles de eficiencia y velocidad requeridos.

Las arquitecturas orientadas a servicios en forma de microservicios junto con las prácticas de desarrollo iterativo e incremental que producen cambios en el software con mucha frecuencia, requieren de pruebas automatizadas para lograr los niveles de cobertura que aseguren que todos los componentes en alta extensión han sido validados.

Funcionalidades y Características Principales de las Herramientas

A continuación se describen de manera general las funcionalidades principales de una herramienta de Automatización de Pruebas Funcionales.

- Diseño de requerimientos de prueba.

- Pruebas multi-plataforma.
- Desarrollo modelado por el comportamiento.
- Automatización de pruebas de interfaz de usuario.
- Análisis de cobertura de prueba.
- Casos de prueba.
- Capacidades de integración.
- Reportes Automatizados.

Diseño de requerimientos de prueba.

Diseño y ejecución de pruebas funcionales sin la necesidad de escribir líneas de código. Lo anterior, puede ser posible mediante funciones sin scripts que requieren la grabación de las actividades de prueba, para su posterior reproducción cada vez que sea necesario ejecutarla. Algunas herramientas utilizan palabras claves para el diseño de los scripts, otras no requieren conocimiento avanzado de programación, mientras que otras se adaptan a las herramientas de preferencia de los testers. Se soportan lenguajes como JavaScript, Python, VBScript, Jscript, DelphiScript, C# y C+.

Pruebas multi-plataforma

Virtualización de servicios para la ejecución de pruebas en las premisas y otras herramientas para las pruebas de aplicaciones basadas en modelos SaaS. La

Automatización de pruebas se puede realizar en una amplia variedad de tecnologías, incluidas las de escritorio, web, móvil y mainframe. Es esencial que las herramientas garanticen la habilidad de orquestar y ejecutar pruebas en plataformas Windows nativas de escritorio (no web), aplicaciones web adaptadas para móvil, móviles nativas, servicios web y aplicaciones empaquetadas.

Las pruebas se pueden ejecutar en entornos de nube, como Amazon Web Services (AWS) y Microsoft Azure, o en máquinas virtuales locales.

Desarrollo modelado por el comportamiento

El desarrollo centrado en el comportamiento (Behavior-driven development - BDD) es una funcionalidad de productividad que posibilita que usuarios no técnicos con pocos conocimientos en programación, sean capaces de desarrollar el código de las aplicaciones sin la necesidad de utilizar script, apoyándose en palabras claves o en el comportamiento. Esta funcionalidad posibilita la colaboración con usuarios que poseen el conocimiento sobre el negocio, para que puedan contribuir a las actividades de diseño de las automatizaciones.

El conocimiento en tiempo real del comportamiento es esencial para simplificar las actividades de diagnóstico y resolución de errores. Los datos históricos son esenciales para entender las tendencias y el comportamiento normal de los ambientes bajo prueba. Lo anterior, acompañado con herramientas de Inteligencia Artificial y Aprendizaje de Máquina ofrece la capacidad de predecir el comportamiento y entregar consejos útiles sobre la gran cantidad de datos analizados.

Automatización de Pruebas de interfaz de usuario

La automatización de pruebas de interfaz de usuario (User Interface – UI) posibilita que los elementos visuales de una aplicación, los cuales interactúan con los

usuarios, sean probados. Es esencial validar que ante cada modificación del código subyacente, los componentes visibles, estén y se comporten según lo esperado. Típicamente utiliza un motor de reconocimiento óptico de caracteres (Optical Character Recognition - OCR) basado en el reconocimiento de objetos, que posibilita diseñar pruebas sobre interfaz gráfica y reutilizarlas para cualquier otra aplicación con una interfaz gráfica de usuario (Graphic User Interface - GUI).

Análisis de cobertura de prueba

Es una funcionalidad que proporciona información sobre la cobertura de la automatización sobre la aplicación bajo prueba. Esta combinación de BDD y pruebas unitarias, así como las pruebas de la capa UI, proporciona una buena base para la calidad continua y para reducir el riesgo. Además, ofrece información sobre la estabilidad de las pruebas, lo que resulta importante en los sistemas con alta tendencia a los cambios.

Casos de prueba

Los casos de prueba se pueden crear con modelos visuales, basado en riesgos o al realizar grabaciones de escenarios en tiempo real. Scripts de prueba pueden generarse a través de código y archivos de configuración. Soporta casos de uso como diferentes navegadores, regresión, paralelo, ágil, regresión completa, entre otros.

Reportes Automatizados

Entrega información centralizada y en tiempo real sobre el progreso y el estado de las pruebas funcionales realizadas. Las herramientas generan reportes detallados que posibilitan profundizar sobre las configuraciones de los ambientes bajo prueba,

para identificar de manera oportuna los errores y las respectivas acciones que se deben ejecutar.

DESPLIEGUE Y LIBERACIÓN DE SOFTWARE

Conceptos y Definiciones

Con frecuencia, particularmente cuando las palabras se utilizan en el idioma, se usa de forma indistinta despliegue (Deployment) y liberación en producción (Release), sin embargo ellas representan dos conceptos diferentes.

Despliegue (Deployment)

Instalación de una versión especifica de software en un ambiente determinado. Ej: Pruebas de Integración, Pruebas Funcionales de Aceptación, Pruebas NO Funcionales de Aceptación, Producción, entre otros (Kim, Patrick, Willis, & Humble, 2016).

Liberación en Producción (Release)

Hacer disponible una o más funcionalidades bien sea a todos los usuarios o un grupo de ellos (Kim, Patrick, Willis, & Humble, 2016). Se puede inferir que una liberación en producción (Release) requiere de un despliegue (Deployment) en este ambiente.

Debe propenderse porque la arquitectura de la aplicación y de los ambientes permita que un despliegue en producción no conlleve de forma automática una liberación en producción.

Importancia de las Prácticas de Despliegue y Liberación en Producción

Realizar de forma automatizada el despliegue y la liberación en producción de nuevo código evita que se creen y acumulen grandes diferencias entre la versión de software que se está ejecutando en producción y la versión que contiene los últimos cambios (Kim, Patrick, Willis, & Humble, 2016) . Con mayores diferencias o número de cambios, la tarea de identificar la causa raíz y realizar la reparación de un error detectado en producción será mayor comparado con el escenario donde las diferencias o número de cambios son pocos. Luego, el despliegue y la liberación automática y frecuente de nuevo código en producción disminuyen los riesgos, tanto la probabilidad como el impacto de afectar la operación y el negocio.

Los procesos de despliegue y liberación en producción manuales son propensos a errores y requieren de la asignación de tiempo de muchas personas. Con procesos automáticos es posible obtener mayor velocidad pues las tareas de uno o más despliegues pueden ejecutarse en paralelo, mayor calidad, dado que habrá menos errores en los procesos de instalación y configuración, y finalmente mayor eficiencia, porque los desarrolladores no deben invertir su tiempo en tareas rutinarias y repetitivas.

Prácticas para el Despliegue y la Liberación en Producción

Automatizar las Actividades del Despliegue y Liberación en Producción.

Para lograr despliegues con alta frecuencia, (Kim, Patrick, Willis, & Humble, 2016), sugieren que las siguientes actividades estén automatizadas:

- Empaquetamiento del código para el despliegue.
- Creación de instancias de máquinas virtuales o contenedores con las configuraciones requeridas.
- Configuración de componentes de capa media de software, como bases de datos, servidores, servidores de aplicación, servicios de integración.
- Copiado de paquetes o archivos en servidores de producción
- Reinicio de sistemas operativos, procesos, servicios, entre otros.
- Ejecución de pruebas de humo (Smoke Tests) para validar la correcta instalación, configuración y funcionamiento.

Para lograr despliegues estandarizados es requerido automatizar el proceso, de este modo la forma como se realiza la instalación en los diferentes ambientes, incluyendo el de producción, siempre será la misma. Lo anterior elimina la posibilidad de errores debido a métodos de instalación y configuración incorrectos introducidos por procesos manuales dependientes de personas.

Crear Arquitecturas y Utilizar Técnicas para el Despliegues Seguros

A continuación se describen arquitecturas de alto nivel y técnicas que contribuyen a minimizar la probabilidad y/o el impacto de errores que puedan introducirse durante el despliegue en el ambiente de producción; estas, son:

Patrones de Despliegue Basados en Ambientes. En este tipo de patrón existe una arquitectura con dos ambientes en los cuales es posible realizar el despliegue, pero solo uno de los entornos está expuesto a los usuarios del software. La técnica aquí consiste en desplegar (Deploy) en varios ambientes de producción al tiempo. Ejemplos de estos patrones, son:

Despliegues Azul y Verde (Blue and Green Deployments). En este tipo de arquitectura y técnica se utilizan dos ambientes de producción idénticos llamados Azul y Verde. El verde es el que se encuentra en producción atendiendo las solicitudes de los usuarios y el azul es el ambiente sobre el cual se hará el despliegue de la nueva versión de software que contiene los cambios. Usualmente una vez desplegado sobre el ambiente azul se realizan pruebas de humo para confirmar la correcta instalación, configuración y funcionamiento del software. Con un resultado positivo para las pruebas se procede a llevar el tráfico de las peticiones de todos los usuarios hacia el ambiente azul el cual a partir de ahora se denomina el ambiente de producción. En caso de encontrar defectos y ser necesario, es posible revertir el tráfico hacia el ambiente verde y dar servicio con la versión anterior de la aplicación.

Despliegues con Canarios (Canary Deployments). Este tipo de arquitectura y técnica deriva su nombre del método empleado por los mineros de carbón donde estos acostumbraban ingresar a las minas junto con canarios en jaulas con el fin de detectar niveles tóxicos de dióxido de carbono, pues estos tienen una tolerancia mucho menor al gas que los seres humanos y los alertarían de dicha situación para iniciar la evacuación. De acuerdo con (Humble & Farley, 2011), este

modelo tiene tres intenciones: a) detectar defectos funcionales en el software con datos y transacciones de producción; b) permitir la experimentación y tener grupos de control para validar la aceptación, adopción y satisfacción de los clientes con los cambios. Esto también se conoce como pruebas A/B; c) ejecutar pruebas de capacidad y desempeño con datos y transacciones reales. En este tipo de despliegue se dispone de varios ambientes de producción (La mayoría de los profesionales que emplean la práctica recomiendan máximo dos) recibiendo y dando respuesta a transacciones de la siguiente manera: a) Un conjunto limitado y seleccionado usuarios es direccionado al ambiente que contiene la nueva versión con los cambios en la aplicación para que generen solicitudes con datos y transacciones y así realizar las validaciones requeridas; b) el resto de los usuarios son dirigidos al ambiente que ejecuta la versión anterior del software. Como ambos ambientes son productivos todos los usuarios reciben respuesta a sus solicitudes. Nuevamente, en caso de encontrarse errores en el ambiente con la nueva versión, todo el tráfico puede ser enviado al ambiente que posee la versión anterior y completamente funcional.

Patrones de Despliegue Basados en Aplicaciones. Estos patrones, a diferencia de los anteriores que se implementan en la capa de infraestructura, se logran realizando definiciones e implementaciones en la capa de software para hacer disponible o no las nuevas funcionalidades que se introducen en la nueva versión de software. Como su manejo es vía código, resultan flexibles y permiten incluso hacer definiciones granulares cuando en un solo cambio se introducen varias funcionalidades y características. De ahí que sea posible habilitar a los usuarios, de forma selectiva, las funcionalidades a liberar en producción (Release). Ejemplo de este patrón, es:

Interruptor de Funcionalidad (Toggle Feature – Flag Feature). En este tipo de arquitectura y técnica se logra con la introducción de un archivo de configuración para tener la posibilidad de cambiar el comportamiento del sistema

sin cambiar el código de la aplicación. Serán entonces los valores del archivo de configuración los que definan si la aplicación exhibe o no la nueva funcionalidad a los usuarios (Fowler, 2010). Debido a que los interruptores de funcionalidad introducen complejidad, es necesario utilizar prácticas y herramientas para administrar la configuración de estos interruptores, asimismo, debe limitarse el número de interruptores en el software (Hodgson, 2017). Como esta implementación se realiza en capa de software utilizando archivos de configuración, es posible también controlar los usuarios o segmentos de estos a los cuales se les presentará una o más funcionalidades (Kim, Patrick, Willis, & Humble, 2016).

Herramientas para el Despliegue y Liberación

Descripción General de las Herramientas para el Despliegue y Liberación

Estas herramientas proporcionan una combinación de capacidades para automatizar el despliegue, administrar el ciclo de vida y los ambientes, y orquestar la liberación en producción. Su propósito es mejorar la velocidad, calidad y eficiencia de las liberaciones de las aplicaciones (Gartner, 2018).

Estas soluciones se utilizan en las empresas para reemplazar o complementar la combinación existente de procesos manuales, scripts hechos a la medida, sistemas de integración continua como Jenkins, que se han extendido para cumplir tareas de despliegue y liberación, sistemas de administración de contenedores, sistemas de automatización continua de la configuración de infraestructura, entre otros (Gartner, 2018).

Estas herramientas también soportan casos de uso emergentes en los cuales se utilizan tecnologías como contenedores y arquitecturas de aplicaciones nativas de nube como “Serverless”, infraestructura como código, entre otras (Forrester, 2018).

Necesidad de Herramientas para el Despliegue y Liberación

Se requiere un proceso estandarizado y predecible para realizar despliegues de la misma manera a los diferentes ambientes, la mejor forma de asegurarlo es a través de la automatización del mismo.

El proceso de creación de software iterativo requiere de múltiples despliegues a múltiples ambientes cada día, estás tareas si bien son críticas, son completamente predecibles y repetitivas, por lo cual es posible automatizarlas, dejando lugar para que los desarrolladores y el equipo de operaciones se dedique a tareas que requieren la aplicación de conocimiento.

La productividad del equipo de desarrollo y operaciones es una exigencia para lograr los niveles de eficiencia. Las herramientas de este tipo permiten que los equipos puedan completar un número más elevado de tareas y de mejor manera.

Funcionalidades y Características Principales de las Herramientas

Automatización de Despliegues

Las herramientas permiten crear automatizaciones personalizadas (por ejemplo, Tareas, runbooks, modelos, plantillas, artefactos de código, componentes y funciones) para eliminar los esfuerzos manuales y los scripts desarrollados internamente para este propósito.

Este tipo de software permite automatizar completamente las tareas de instalación de binarios, paquetes, librerías u otros artefactos en los ambientes seleccionados. Este proceso se inicia de forma autónoma luego que los desarrolladores envían los cambios al repositorio de control de versiones.

Estas tecnologías permiten crear, utilizar, mantener y versionar las automatizaciones utilizando las mismas prácticas y principios empleadas para la creación de código fuente, posibilitando tener también las automatizaciones bajo el control de versiones.

Administración del Ciclo de Entrega y los Ambientes

Las herramientas pueden crear, usar, mantener y versionar los diferentes ambientes que generalmente están compuestos por elementos tanto de infraestructura como de aplicación.

También cuentan con capacidades para crear, aprovisionar, modificar, destruir o desaprovisionar ambientes de pruebas y producción. Incluso incluyen capacidades para validar la correcta configuración y hacer remediaciones a la misma, en caso de encontrarlo necesario.

A su vez cuentan con funcionalidades para crear, utilizar, mantener y versionar diferentes ciclos de entrega de software que pueden ser aplicados a diferentes aplicaciones o incluso ambientes en caso de llegar a requerirse procedimientos diferentes.

Orquestación de la Liberación en Producción

Estas tecnologías cuentan con un motor de flujos de trabajo para definir la secuencia en la cual deben llevarse a cabo las acciones o tareas de automatización durante el proceso de liberación en producción.

Tienen mecanismos de integración con herramientas de comunicación (chat, email, slack and SMS) para realizar las notificaciones requeridas durante el proceso. Asimismo, disponen de capacidades para asegurar el acceso a las mismas, segregar las funciones y otros tipos de controles de seguridad.

Se integran perfectamente con herramientas de seguimiento de defectos, repositorios de código fuente, herramientas de construcción (Build) e integración continua, software para la gestión de incidentes, cambios y configuración, administradores de cloud y contenedores, entre otros.

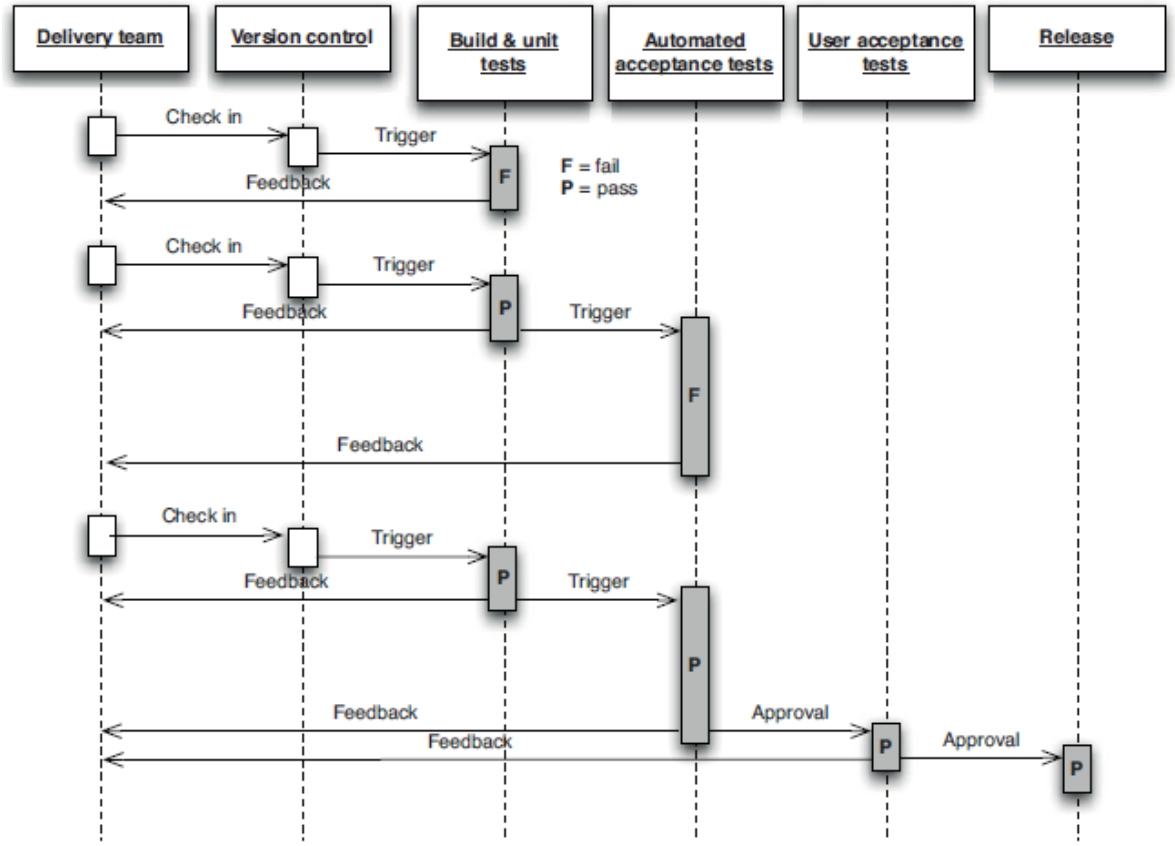
CICLO AUTOMATICO DE ENTREGA CONTINUA DE SOFTWARE

Conceptos y Definiciones

El ciclo automático de entrega de software (Deployment Pipeline) es la representación de la porción del proceso (iterativo) que inicia desde que se obtiene el código desde el repositorio de control de versiones hasta que el software está disponible para su uso (Humble & Farley, 2011).

Como se ha documentado en esta investigación, la creación de software incluye la construcción (Build) y múltiples etapas de prueba (Test) y despliegue (Deployment) hasta la liberación final en producción (Release). En los enfoques de uso de metodologías agiles, los cuales predominan hoy, el software se construye de forma iterativa, y en cada iteración las nuevas versiones del software pasan por las etapas que se acaban de enunciar. Una explicación se proporciona en Ilustración 4: Explicación Ciclo Automático de Entrega de Software:

Ilustración 4: Explicación Ciclo Automático de Entrega de Software



Humble, J., & Farley, D. (2011). CONTINUOUS DELIVERY Reliable Software Releases Through Build, Test, and Deployment Automation

Las etapas típicas del ciclo de automático de entrega de software, según (Humble & Farley, 2011), son las siguientes y se representan en la Ilustración 5: Etapas Típicas Ciclo Automático de Entrega de Software:

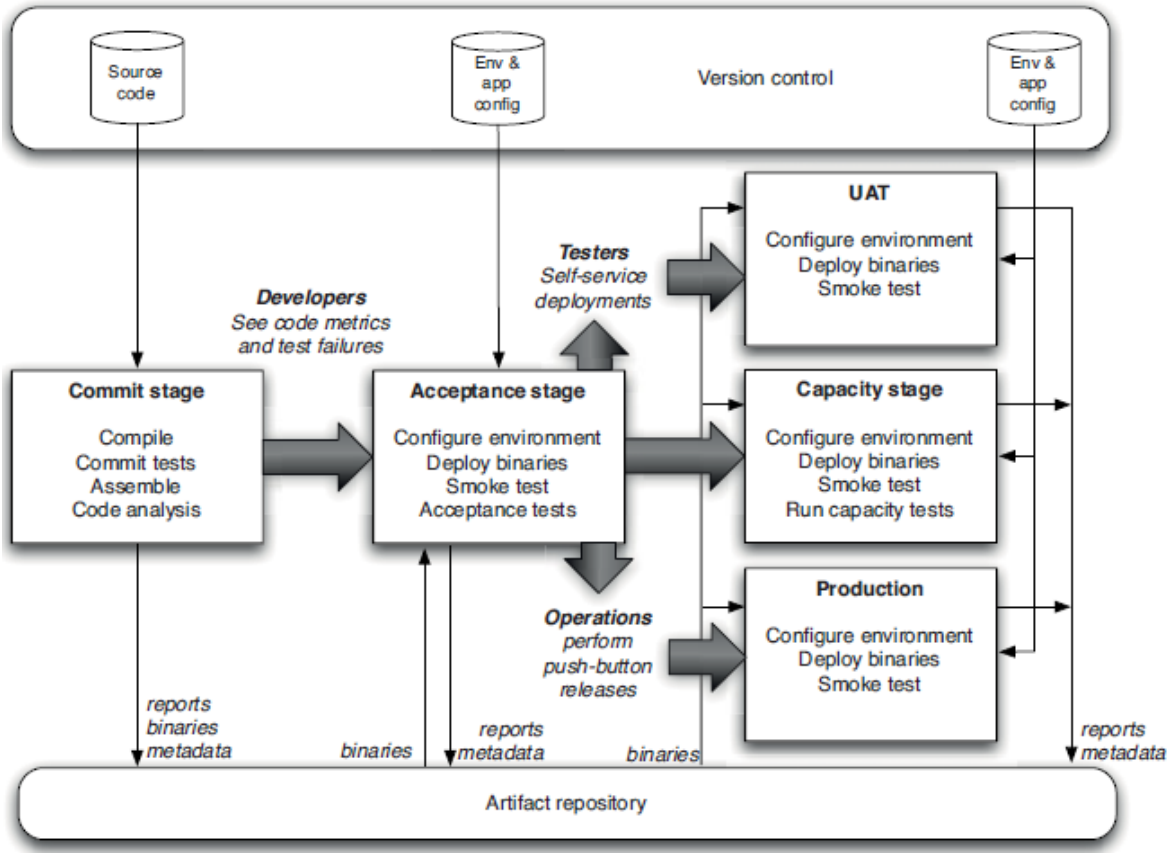
- Etapa de envío de cambios (Commit). Se realiza la integración del código modificado con el código de producción, se realiza la compilación, se realizan pruebas unitarias automatizadas y se ejecutan revisiones para analizar el código. Se confirma que el software funciona a nivel técnico.

- Etapa de pruebas de aceptación automáticas. Se valida que el software satisfice los requisitos funcionales y no funcionales especificados por los usuario de la aplicación.

- Etapa de pruebas manuales. Se hacen revisiones adicionales para encontrar defectos que no se capturan a través de las pruebas automáticas. Entre las pruebas comunes que se realizan están las de aceptación de usuario.

- Etapa de Liberación en Producción (Release). Se pone el software a disposición de los usuarios, instalándolo en el ambiente de producción y haciéndolo disponible para ellos.

Ilustración 5: Etapas Típicas Ciclo Automático de Entrega de Software



Humble, J., & Farley, D. (2011). CONTINUOUS DELIVERY Reliable Software Releases Through Build, Test, and Deployment Automation

Importancia del Ciclo Automático de Entrega de Software

Las mayores ineficiencias en la entrega de software utilizando procesos manuales se encuentran entre las etapas de pruebas y despliegue en producción, principalmente porque las tareas de estas fases ejecutadas manualmente toman tiempo y porque de forma inherente existen dependencias entre los equipos de desarrollo, prueba y operaciones los cuales deben esperar hasta que cada equipo finalice sus tareas. Para acelerar el proceso de entrega de software resulta importante que las tareas de todas las etapas del ciclo de entrega estén

automatizadas para que se completen lo más rápido posible y que los equipos puedan acceder, idealmente en paralelo, a los recursos (ambientes, artefactos, entre otros) en cualquier momento para poder llevar a cabo sus tareas.

La automatización no sólo tiene un efecto positivo en la velocidad, sino también en la estandarización y confiabilidad del proceso. Entregar software de este modo crea un proceso rápido, repetible y confiable (Humble & Farley, 2011).

Prácticas para el Ciclo Automático de Entrega de Software

Construir Los Binarios una Sola Vez

Los códigos ejecutables o binarios, bien sean resultados de una compilación o un conjunto de archivos (.Jar, .War, .Ear) se deben construir una sola vez para evitar que se creen diferencias entre cada compilación debido a cambios en librerías, sistemas operativos, compiladores entre otros (octopus, 2014). Así pues, los binarios obtenidos durante la construcción (Build) deben ser artefactos controlados en un repositorio y deben ser utilizados para desplegar a todos los ambientes (pruebas, producción, otros) durante el ciclo de entrega de software.

Realizar Pruebas de Humo en los Despliegues

Cada vez que se realice un despliegue deben ejecutarse pruebas, idealmente automáticas, para comprobar que la aplicación se encuentre en modo de ejecución y operando correctamente. Para esto las pruebas deben incluir la verificación de la conexión y funcionamiento de componentes como bases de datos, capas de integración y servicios externos (Humble & Farley, 2011) .

Realizar pruebas en ambientes de pre-producción

Las pruebas de aceptación de requisitos funcionales, no funcionales e integración deben realizarse en ambientes cuya configuración sea idéntica al entorno de producción. Esto debe incluir las configuraciones de red, sistemas operativos y sus niveles de actualización y parches, capa media de software.

DISEÑO METODOLÓGICO

Las etapas que se surtieron para completar esta investigación fueron las siguientes:

Etapa 1 – Identificación del problema

Se realiza una primera identificación del problema a través de la observación y las conversaciones que el autor tiene con las empresas de la ciudad en virtud del trabajo profesional que desarrolla en el día a día.

Etapa 2 – Formulación del Problema

Con orientación del asesor se establece con precisión el problema que se quiere resolver, este es: Las organizaciones deben transformar los procesos de Ingeniería de Software para construir aplicaciones y/o modificar las existentes con mayor velocidad, calidad y eficiencia.

Etapa 3 – Definición de los Objetivos de Investigación

Se hizo la definición de los objetivos que debía perseguir esta investigación para que fuera de utilidad a las empresas de la ciudad de Medellín que enfrentan el problema declarado en la etapa anterior.

Etapa 4 – Definición de las Preguntas de Investigación

Para cada objetivo específico se plantearon las siguientes preguntas de investigación, así:

Objetivo Especifico 1

Describir los movimientos DevOps y Entrega Continua de Software, así como su importancia para la transformación digital.

Preguntas de Investigación Objetivo Especifico 1

- ¿Qué es Transformación Digital y cuál es su importancia?

- ¿Qué es el movimiento DevOps?
- ¿Qué es el Movimiento Entrega Continua de Software?
- ¿Por qué son importantes los movimientos DevOps y Entrega Continua de Software para la Transformación Digital?

Acciones para dar respuesta a las preguntas de investigación Especifico 1

Revisión bibliográfica y documentación del estado del arte de las prácticas

Objetivo Especifico 2

Describir las prácticas de Ingeniería del Software alineadas a DevOps para la entrega continua de software.

Preguntas de Investigación Objetivo Especifico 2

- ¿Cuáles prácticas son las prácticas de Ingeniería del Software según el movimiento DevOps?
- ¿Cuáles son las prácticas DevOps que contribuyen de manera directa a la entrega continua de software?

Acciones para dar respuesta a las preguntas de investigación Especifico 2

Revisión bibliográfica y documentación del estado del arte de las prácticas

Objetivo Especifico 3

Identificar y documentar los aspectos claves de las tecnologías requeridas para la implementación de prácticas DevOps orientadas a entrega continua de software.

Preguntas de Investigación Objetivo Especifico 3

- ¿Cuáles son las herramientas de software requeridas para la implementación de cada práctica DevOps?
- ¿Cuál es la importancia de cada herramienta de software para la implementación de la práctica?
- ¿Cuáles son las funcionalidades o características típicas de las herramientas de software para apoyar la práctica DevOps?

Acciones para dar respuesta a las preguntas de investigación Especifico 3

Revisión bibliográfica y documentación para cada tipo de herramienta de software de los siguientes elementos:

- Práctica o prácticas que apoya
- Funcionalidades claves de las herramientas de software que apoyan la implementación de la práctica.

Etapa 5 – Revisión Bibliográfica

Durante esta etapa se procede a recabar información ya existente sobre Transformación Digital, Procesos, Técnicas y Herramientas asociados a los movimientos DevOps y Entrega Continua de Software. Esta información se diferentes tipos de fuentes incluyendo sitios web, informes y reportes de firmas de investigación de mercado, revistas, artículos científicos, libros y otros trabajos

académicos. Esta investigación documental proporciona una visión sobre el estado del tema o problema elegido en la actualidad.

Etapa 6 – Marco Teórico y Documentación del Estado del Arte

Con base a la literatura revisada, se procede a documentar un marco teórico que documenta los conceptos claves, las conclusiones y resultados relevantes hallados por otros autores en sus investigaciones, las similitudes y diferencias de las teorías de los diferentes autores.

Etapa 7 – Desarrollo del Trabajo

Se realiza una propuesta de cómo enfrentar la implementación DevOps para entrega continua de software en las organizaciones. Esto con base en las conclusiones y resultados relevantes encontrados por los autores en sus investigaciones.

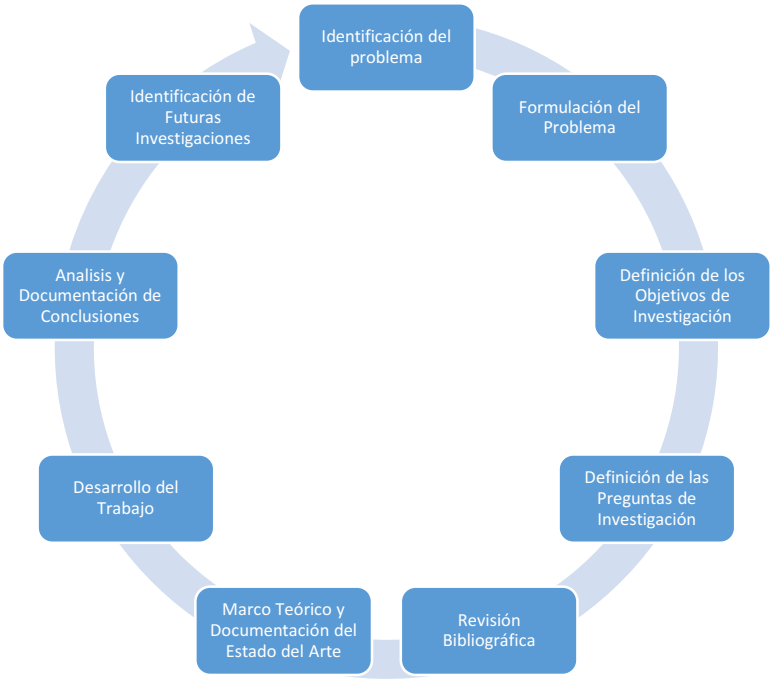
Etapa 8 – Análisis y Documentación de Conclusiones

Se documenta la información más relevante sobre las ideas y las practicas propuestas sobre las prácticas DevOps para entregar software de manera continua.

Etapa 9 – Identificación de Futuras Investigaciones

Entendiendo que: las prácticas DevOps no sólo son aquellas de naturaleza técnica, que las categorías y número de prácticas DevOps son numerosas y que existe interdependencia entre las categorías y sus prácticas, es importante plantear trabajos futuros para entender la correlación entre las diferentes prácticas y su impacto en la entrega de software con criterios de velocidad, calidad y eficiencia.

Ilustración 6: Etapas de la Metodología



Creación Propia

PROPUESTA PARA LA ADOPCIÓN DE PRACTICAS DEVOPS PARA ENTREGA CONTINUA DE SOFTWARE

Durante este capítulo se documentan consideraciones para la implementación de las prácticas DevOps para la entrega continua de software. Se pretende con esto, construir una primera versión de un cuerpo de conocimiento capaz de orientar a los interesados en los aspectos claves para abordar un proceso de implementación. Los elementos aquí expuestos son resultado de:

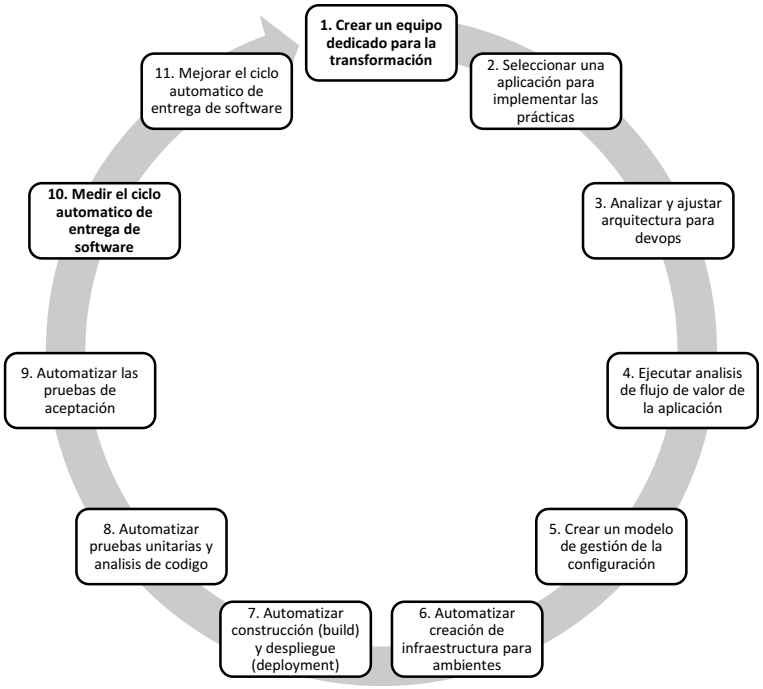
- Inferencias realizadas por el autor de la revisión bibliográfica realizada durante la investigación.

- Experiencias documentadas, publicadas por profesionales y que fueron analizadas la revisión bibliográfica realizada durante la investigación.

- Experiencia del autor implementando modelos de gestión y estándares en áreas de TI

No pretende ser exhaustivo en las actividades o tareas de cada práctica pues existe literatura suficiente en la cual se aborda con detalle, y algunas de ellas se han descrito en el marco teórico de esta investigación. Vale la pena resaltar que esta propuesta solo toma en consideración las prácticas técnicas de DevOps, es decir aquellas orientadas a la entrega continua de software. Quedan por fuera entonces practicas relacionada con arquitectura, gestión de procesos y productos de software, gestión lean y monitorización, y cultura. Futuros trabajos podrían abordar estos aspectos y enriquecer esta primera propuesta.

Ilustración 7: Etapas Modelo Propuesto para Adopción de Practicas DevOps



Creación Propia

CREAR UN EQUIPO DEDICADO PARA LA TRANSFORMACIÓN

Las investigaciones de (Govindarajan & Trimble, 2010) afirman que para lograr transformaciones y lograr innovación disruptiva en las organizaciones es necesario crear equipos dedicados para tal fin. Según los autores estos deben operar de manera separada del resto de los equipos responsables de la operación diaria. A este equipo debe asignarse un conjunto de objetivos específicos, medibles, alcanzables, relevantes y enmarcados en el tiempo, ejemplo: reducir el ciclo de entrega desde envío de los cambios hasta su despliegue en producción en un 50% en seis meses. (Kim, Patrick, Willis, & Humble, 2016), sugieren que adicionalmente se propenda porque el equipo este físicamente ubicado de forma separada de los otros equipos, pero sus integrantes ubicados en el mismo lugar.

SELECCIONAR UNA APLICACIÓN PARA IMPLEMENTAR LAS PRÁCTICAS.

Para lograr implementar estas prácticas a gran escala dentro de las compañías se requiere tanto de grandes esfuerzos de toda índole incluyendo un cambio de cultura. Para vencer el escepticismo y racionalizar los las inversiones en personas y tecnologías, resulta conveniente iniciar por una aplicación en lugar de plantear un proyecto de transformación completo y ambicioso que incluya todas o muchas de las aplicaciones y servicios de una compañía.

El software elegido puede ser de cualquier área de negocio pero se debe asegurar que:

- Haya posibilidad de experimentación y aprendizaje. Un escenario como estos podría encontrarse en aplicaciones que hoy no satisfacen apropiadamente los requisitos del negocio y en los cuales existe urgencia manifiesta de realizar transformaciones.

- Sea posible mostrar logro de resultados en el corto plazo.

- Haya visibilidad para que las personas confíen y luego se puedan extrapolar prácticas y definiciones a otros servicios y/o aplicaciones.

Los nuevos proyectos para la creación de software representan una mejor oportunidad para iniciar con la implementación de las prácticas debido a que en este tipo de iniciativas es probable que todavía no existan arquitecturas, infraestructura, procesos, equipos y tecnologías asociados (Kim, Patrick, Willis, & Humble, 2016). Estos también representan la posibilidad para experimentar las capacidades y servicios tecnológicos que ofrece la nube pública como AWS, AZURE, Google

Cloud Plattform para implementar prácticas de gestión de la configuración, integración continua, pruebas continuas, despliegue continuo.

Lo anterior no significa imposibilidad para hacerlo sobre aplicaciones y servicios ya existentes, de acuerdo con (Kim, Patrick, Willis, & Humble, 2016) más del 60% de las historias de transformación presentadas en el DevOps Enterprise Summit en 2014 fueron sobre proyectos de este tipo. Esto se confirma con las investigaciones de (Puppet Labs, 2015) que establecen que no existen limitaciones para la adopción de prácticas DevOps incluso en ambientes legados como Mainframe. Sin embargo es posible encontrar barreras debido principalmente a: falta de automatización de pruebas y arquitecturas de aplicación de alto acoplamiento (Kim, Patrick, Willis, & Humble, 2016).

ANALIZAR Y AJUSTAR ARQUITECTURA PARA DEVOPS

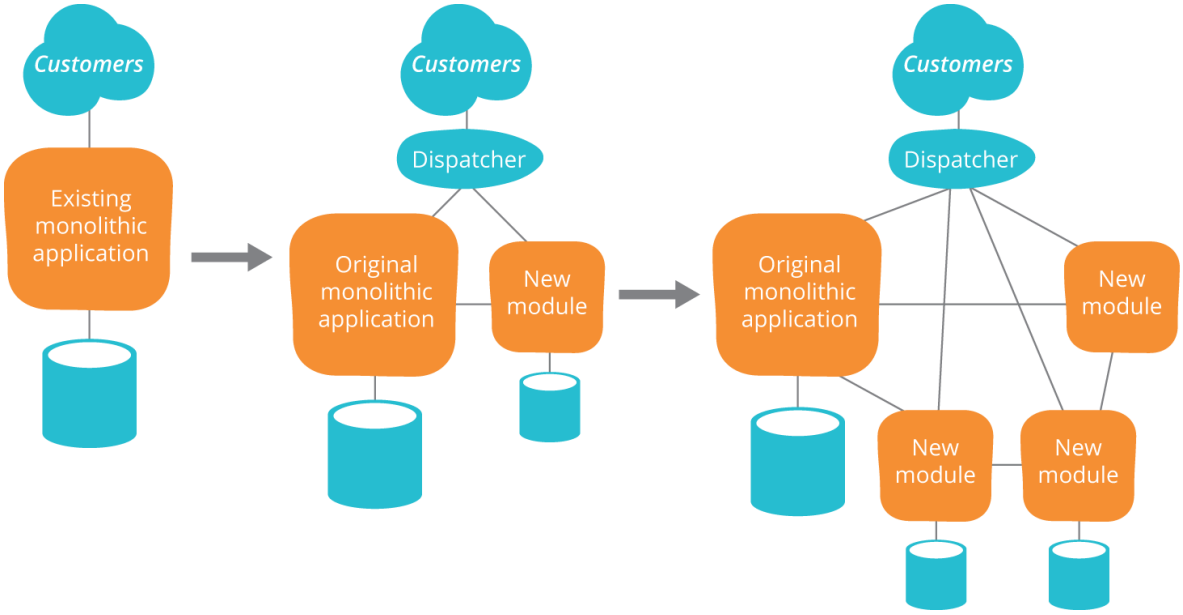
En las arquitecturas de alto acoplamiento cualquier cambio en el software tiene muchas implicaciones por el número de dependencias y/o sistemas interconectados, pues se requiere de un mayor esfuerzo tanto para la construcción (Build) como para la ejecución de las pruebas de integración. Para DevOps se debe propender por arquitecturas orientadas a servicios, es decir, aquellas donde una aplicación está compuesta de múltiples servicios que pueden modificarse, probarse y desplegarse de forma independiente (Kim, Patrick, Willis, & Humble, 2016). De esta manera los equipos de desarrollo y pruebas podrán trabajar en paralelo para desplegar en producción tantos cambios como sean necesarios por el negocio a la velocidad requerida por este.

Si la aplicación no satisface las características de una arquitectura orientada a servicios se sugiere evolucionar o cambiar la arquitectura de la aplicación pues de acuerdo con investigaciones de (Forsgren, Humble, & Kim, 2018) las compañías que tienen un alto desempeño de TI usan aplicaciones que satisfacen lo siguiente:

“Es posible llevar a cabo la mayoría de las pruebas sin requerir un ambiente de integración y es posible realizar el despliegue (Deploy) y liberación en producción (Release) sin depender de otras aplicaciones o servicios”. Lo anterior resulta clave pues los análisis realizados por (Puppet Labs & DevOps Research and Assessment, 2017) revelan que el aspecto que más contribuye a la entrega continua de software incluso, por encima de la automatización de pruebas y despliegue, es la arquitectura de las aplicaciones.

Para modificar la arquitectura de una aplicación e implementar las prácticas de entrega continua de software resulta conveniente hacer uso de un patrón denominado “Asfixiamiento de la Aplicación (Strangler Pattern)”. Este es un patrón útil para transformar una aplicación monolítica o con alto acoplamiento a una orientada a servicios. El método consiste en reemplazar de forma incremental funcionalidades específicas de la aplicación con nuevos servicios (Principalmente, Micro-Servicios) y manteniendo el resto de las funcionalidades en la aplicación actual. Para mantener la misma interface gráfica y experiencia de uso para los clientes se usan mecanismos como intermediarios (Proxy) o fachadas (Facade) para mantener la consistencia deseada de cara a los usuarios. La Ilustración 8: Patrón Asfixiamiento de la Aplicación (Strangler Pattern), ayuda al entendimiento de la explicación proporcionada.

Ilustración 8: Patrón Asfixiamiento de la Aplicación (Strangler Pattern)



http://mycloudcomputing2017.blogspot.com/2017/02/what-is-strangler-application-pattern.html

EJECUTAR ANALISIS DE FLUJO DE VALOR DE LA APLICACIÓN

Un análisis de flujo de valor (Value Stream Map) es una técnica que permite identificar y analizar los diferentes procesos, actividades y/o tareas necesarias para completar algo, ese algo puede ser; un pedido de un cliente en un restaurante, manufacturar un vehículo en una fábrica, desembolsar un crédito solicitado por un cliente en un banco, y en nuestro caso convertir una idea en software liberado en producción para su uso. Este tipo de análisis se realiza para identificar:

- El tiempo total que transcurre desde que inicia el proceso hasta que se ha completado el mismo. En el lenguaje LEAN donde se originan estás practicas esto se denomina como: Tiempo Total del Ciclo de Proceso (Lead Time – Elapsed Time)

- Identificar el tiempo que se emplea en actividades que contribuyen directamente y son estrictamente necesarias para completar la solicitud. En el lenguaje LEAN esto se conoce como: actividades que no agregan valor (Non-Value-Adding Activities), actividades que agregan valor (Value-Adding Actitivies).

- Identificar los tiempos de espera y las actividades que se realizan pero que no son necesarias para completar la solicitud y podrían eliminarse. En el lenguaje LEAN esto se designa como: actividades que no agregan valor (Non-Value-Adding Activities)

Si bien un análisis de valor completo en software debe comprender desde la concepción de una idea hasta su materialización en una aplicación liberada en producción y funcionando, para este caso la atención se centra en los procesos y actividades que va desde envío de los cambios por parte de los desarrolladores (Commit) hasta la liberación de los mismos en el ambiente de producción (Release).

En la metodología propuesta este análisis se ejecuta para encontrar elementos que permitan optimizar el proceso en términos de su eficiencia, para esto se identifican los siguientes aspectos:

- Tiempo que toma liberar una nueva versión de la aplicación en producción a partir del momento que el (Los) desarrollador(es) envían los cambios en el código. Este tiempo servirá como línea base de desempeño y se utilizará para evidenciar las mejoras logradas en términos de velocidad y productividad del equipo.

- Actividades o tareas dentro de los procesos actuales de construcción (Build), pruebas (Test), despliegue (Deployment) y liberación en producción (Release) no agregan valor (Non-Value-Adding Activities) y por lo tanto no deberían considerarse y ser automatizadas.

- Dependencias u otro tipo de situaciones que ocasionan tiempos de espera para iniciar el siguiente proceso, actividad o tarea, por ejemplo: aprovisionamiento y configuración de la infraestructura de los ambientes de prueba, integración, pre-producción; obtención de datos para la ejecución de pruebas de aceptación en los diferentes ambientes; instalación y configuración de los componentes de software de capa media en los ambientes; instalación y configuración de la aplicación en los ambientes; entre otros.

Lograr niveles máximos u optimizados de eficiencia en esta porción del proceso [Construcción (Build), Pruebas (Test), despliegue (Deployment) y liberación en producción (Release)] es particularmente importante. Hoy una porción importante del software se crea o modifica utilizando métodos agiles en los cuales hay iteraciones frecuentes cuyo resultado es la modificación del código de la aplicación para hacer disponible nuevas funcionalidades o corregir errores existentes. Este es un proceso que se repite cada vez que un desarrollador envía un cambio (Commit), como esto ocurre todos los días la ineficiencia del proceso resulta ser exponencial al multiplicar la pérdida del proceso por el número de cambios enviados al día y a su vez por el número de desarrolladores.

Es clave que en la construcción del flujo de valor de la aplicación haya participación de todas las personas que participan en al menos una de las actividades del proceso

para asegurar que la información recolectada y el modelo de flujo de valor creado es completo y confiable. Esto podría incluir la participación de personas a cargo de otras aplicaciones, servicios o interfaces con las cuales interactúa la aplicación objeto de este análisis.

CREAR UN MODELO DE GESTIÓN DE LA CONFIGURACIÓN

Es necesario disponer de un proceso y herramientas para que todos los artefactos del proyecto de software puedan ser identificados, almacenados, accedidos y utilizados por los integrantes del equipo y las automatizaciones generadas. También es importante que los cambios en estos artefactos sean visibles y puedan ser controlados y trazados durante el ciclo de todo su ciclo de vida. El primer elemento que debe ser incluido en este proceso es el código fuente y los archivos de configuración de la aplicación en caso que existan, sin esto no será posible avanzar en la implementación del ciclo automático de entrega de software por lo cual es un imperativo. Gradualmente y como parte del proceso de mejora se deben ir incorporando otros elementos como scripts para la creación de esquemas de bases de datos, scripts de pruebas automáticas, scripts para la creación y configuración de infraestructura y capa media de software, plantillas para la creación de infraestructura y servicios de nube pública, entre otros.

AUTOMATIZAR CREACIÓN DE INFRAESTRUCTURA PARA AMBIENTES

Una de las actividades que más contribuye a incrementar los tiempos del ciclo de entrega de software es el aprovisionamiento y configuración de la infraestructura requerida en cada uno de los ambientes. También, se encuentra con frecuencia que

errores en el funcionamiento de las aplicaciones son resultado de las diferencias de configuración de la infraestructura en los distintos ambientes. En lugar de simplemente especificar en un documento los requisitos de infraestructura de los ambientes, es necesario disponer de capacidades para la creación automática de estos y asegurar de este modo su correcta configuración (Kim, Patrick, Willis, & Humble, 2016). Se recomienda el uso de herramientas de gestión de la configuración como Puppet, Chef, Ansible, para crear capas de abstracción que permitan administrar la infraestructura como un código y acelerar el proceso de entrega de estos recursos a los equipos que lo requieren.

AUTOMATIZAR CONSTRUCCIÓN (BUILD) Y DESPLIEGUE (DEPLOYMENT)

Las actividades de construcción (Build) identificadas en la fase anterior como actividades que agregan valor (Value-Adding Actitivies) deben ser automatizadas utilizando prácticas de integración continua para asegurar que la construcción automática (Build) se inicia cada que se envía un cambio (Commit) al repositorio de control de versiones. Específicamente la automatización debe asegurarse que los binarios resultado del proceso de construcción (Build) siempre son los mismos que se utilizarán en lo sucesivo en los ambientes subsiguientes de pruebas de aceptación funcional, pruebas de aceptación no funcional, integración, entre otros.

El proceso de instalación de los artefactos (Binarios) resultantes del proceso de construcción (Build) debe ser automatizado y debe garantizarse que es el mismo independientemente del ambiente en el cual este se realice. Debe incluirse también la automatización necesaria para probar que el despliegue se ha realizado correctamente en cada ambiente. No debería iniciarse un proceso de pruebas de aceptación hasta tanto no se satisfagan los requisitos especificados por las pruebas creadas para validar el correcto despliegue (Humble & Farley, 2011).

AUTOMATIZAR PRUEBAS UNITARIAS Y ANALISIS DE CÓDIGO

Establecer pruebas automáticas para confirmar que los binarios se construyeron correctamente utilizando las herramientas de xUnit para confirmarlo. Asimismo deben ejecutarse pruebas automáticas de análisis de código estático para asegurar la adherencia a las definiciones de estilo de código, complejidad ciclomática, acoplamiento entre otros.

En caso que las pruebas unitarias automáticas tomen demasiado tiempo (5 Minutos o más) es conveniente separar las pruebas para que estás se ejecuten en paralelo, bien sea utilizando tecnologías de servidor de integración continua que tengan estás características o utilizando varias instancias para hacer el trabajo de modo separado (Humble & Farley, 2011)

AUTOMATIZAR LAS PRUEBAS DE ACEPTACIÓN

La aplicación debe satisfacer tanto los requisitos funcionales como los No funcionales, por esta razón ambos tipos de prueba deben ser automatizados. Deben crearse conjuntos de prueba automáticos para validar cada tipo de requisito (Funcionales y NO Funcionales). Las automatizaciones deben crearse de modo que las pruebas puedan ejecutarse en paralelo, es decir, al mismo tiempo debería ser posible llevar a cabo ambos conjuntos de pruebas.

Herramientas para la “Gestión del Desempeño de Aplicaciones (APM)” son útiles para diagnosticar errores relacionados con esta variable, se recomienda habilitar la visibilidad proporcionada por estas en los ambientes donde se realizan las pruebas.

MEDIR EL CICLO AUTOMATICO DE ENTREGA DE SOFTWARE

De acuerdo con la filosofía LEAN, las optimizaciones globales del proceso en lugar de las locales y focalizadas en actividades o tareas, son las que tienen efecto en el mejoramiento de un proceso. Luego, es importante definir métricas que evalúen el proceso de principio a fin. Para (Humble & Farley, 2011) la métrica global más importante es Tiempo de Ejecución (Cycle Time) pues esta no solo permite controlar y mejorar la velocidad sino también la calidad pues el objetivo de reducir el tiempo de ejecución llevará al equipo a implementar automatización de las pruebas y luego esto derivará en mayor cobertura de las mismas en la medida que se va liberando tiempo de ejecución manual de esta actividad. Esta métrica definida en términos prácticos en es la respuesta a la pregunta: ¿Cuánto tiempo le toma a la organización liberar en producción, de forma segura, un cambio que solo implica crear o modificar una sola línea de código? (Poppendieck & Poppendieck, 2003).

(Forsgren, Humble, & Kim, 2018), coinciden respecto a las métricas con (Humble & Farley, 2011) , los primeros también proponen que las métricas deben ser globales y adicionan que deben ser coherentes y no contradictorias entre los equipos de desarrollo (Dev) y operaciones (Ops). También afirman que las métricas deben evaluar resultados y no cantidad de artefactos que se producen. Sus conclusiones, las cuales recomiendo, establecen que las métricas que satisfacen los criterios descritos y que permiten valorar adecuadamente el desempeño del equipo de entrega de software son las siguientes:

- Tiempo de entrega de software (Delivery Lead Time). Es la misma métrica mencionada arriba nombrada como Tiempo de Ejecución (Cycle Time). Recapitulando, este es el tiempo que transcurre desde el envío de los cambios (Commit) hasta su liberación en producción (Release) con funcionamiento correcto.

- Frecuencia de despliegue y liberación en producción (Deployment and Release Frecuency). Debe entenderse está métrica como la frecuencia con la cual se despliega y libera en el ambiente de producción o en las tiendas de aplicaciones como Apple App Store o Google Play Store.

- Tiempo promedio para restablecer servicio (Mean Time to Restore Service – MTTR). Tiempo que transcurre entre el inicio de una interrupción o degradación relevante del servicio en producción y su corrección.

- Tasa de cambios fallidos (Change Fail Rate).Número de cambios que fallan cuando son liberados en producción porque crean interrupciones o degradaciones relevante del servicio y obligan a tomar medidas como, crear y liberar un parche, retornar a la versión previa de la aplicación, corregir el código con defectos y liberarlo de nuevo).

Para propósitos de referencia competitiva (Benchmarking) en la Tabla 2: Resultados Desempeño de Entrega de Software 2018 se muestran los resultados con las métricas anteriormente expuestas.

Tabla 2: Resultados Desempeño de Entrega de Software 2018

Aspect of Software Delivery Performance	Elite[a]	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code?	On-demand (multiple deploys per day)	Between once per hour and once per day	Between once per week and once per month	Between once per week and once per month
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code commit to code successfully running in production)?	Less than one hour	Between one day and one week	Between one week and one month[b]	Between one month and six months[b]
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?	Less than one hour	Less than one day	Less than one day	Between one week and one month
Change failure rate For the primary application or service you work on, what percentage of changes results either in degraded service or subsequently requires remediation (e.g., leads to service impairment, service outage, requires a hotfix, rollback, fix forward, patch)?	0-15%	0-15%	0-15%	46-60%

DevOps Research and Assessment - DORA. (2018). 2018 Accelerate: State of DevOps, Strategies for a New Economy

Las diferencias entre organizaciones se presentan en la Tabla 3: Comparación de Resultados Organizaciones Elite y de Bajo Desempeño

Tabla 3: Comparación de Resultados Organizaciones Elite y de Bajo Desempeño

DevOps Research and Assessment - DORA. (2018). *2018 Accelerate: State of DevOps, Strategies for a New Economy*

MEJORAR EL CICLO AUTOMATICO DE ENTREGA DE SOFTWARE

Una aproximación iterativa e incremental debe ser el método de elección para la implementación de los ciclos automáticos de entrega de software, porque así es posible, lograr y demostrar en corto tiempo mejoramientos en el proceso. Esto significa que el ciclo o proceso debe mejorar por las siguientes razones:

- Es necesario automatizar aquellas actividades del ciclo [Construcción (Build), Pruebas (Test), despliegue (Deployment) y liberación en producción (Release)] que en la primera iteración no fueron intervenidas y que siguen ejecutándose de forma manual.

- Aun con tareas automatizadas en cada etapa del ciclo se presentan cuellos de botella o restricciones que deben resolverse para asegurar el flujo uniforme a través de este. Entonces debe revisarse los tiempos de ejecución de cada porción del flujo para encontrar la causa raíz de ciclos de ejecución largos y tomar decisiones de mejora como: dividir cargas de trabajo para procesar en paralelo pruebas de aceptación o pruebas unitarias, introducir modificaciones en las automatizaciones para mejorar los tiempos de procesamiento, entre otras.

- El ciclo automático de entrega de software debería tratarse como una aplicación, por lo tanto es importante examinar y tener bajo control la deuda técnica de las automatizaciones, identificar la necesidad de rediseñar y re-

escribir el código de las mismas, implementar tecnologías más eficaces y eficientes para automatizar el proceso, entre otras.

- La evolución de la misma aplicación impone la necesidad de evolucionar y transformar el ciclo automático de entrega de software creado inicialmente, pues es posible que, cambios en la funcionalidad, la arquitectura, los requisitos No funcionales entre otros, requieran la modificación de las actividades del ciclo y sus automatizaciones.

CONCLUSIONES

Conclusión sobre la Importancia de las practicas DevOps de Entrega Continua de Software en la transformación Digital

La exigencia, ineludible, de realizar transformación digital en las organizaciones para adaptarse: al comportamiento de nueva dinámica económica, a los requerimientos del mercado, y sus consumidores; impone la necesidad de adoptar una posición diferente y dar la relevancia adecuada al software. Las aplicaciones móviles, de escritorio y las embebidas en productos, dispositivos o bienes de consumo son el eje central de las iniciativas de los negocios digitales. La velocidad, calidad y eficiencia con la que se cree y se evolucione el software es ahora determinante para la supervivencia y competitividad de las empresas. Se necesitan procesos de ingeniería alineados con los principios y prácticas DevOps de entrega continua de software para permitirle a las empresas jugar en el nuevo entorno de competencia donde la experiencia digital define en gran medida la participación del mercado, la generación de ingresos, la satisfacción y lealtad de los clientes. Por lo tanto es inaplazable para los líderes de la función TI en las empresas iniciar proyectos tendientes a la adopción de prácticas DevOps para entrega continua de software, so pena de poner en riesgo la existencia de sus negocios.

Conclusión sobre el Movimiento DevOps y Movimiento de Entrega Continua de Software

El movimiento DevOps no solo integra principios y prácticas del movimiento ágil sino que las complementa con prácticas de otros movimientos, principalmente los de los movimientos Lean y entrega continua. Con esta convergencia y perfeccionamiento, se logran prácticas de Ingeniería de Software con una visión integral y holística capaz de obtener optimizaciones globales del proceso de creación y evolución del software, desde su concepción hasta su mejoramiento. DevOps es mucho más que

prácticas de entrega continua de software y se ocupa de otros aspectos que son claves para lograr un modelo de trabajo más efectivo. Por ejemplo: la cultura para promover la colaboración entre los equipos Dev y Ops, la mentalidad Lean para buscar constantemente el mejoramiento continuo a través de la automatización, las arquitecturas flexibles y evolutivas para habilitar opciones de trabajo independiente y en paralelo.

Conclusión acerca de la Gestión de la Configuración

Los niveles de automatización requeridos para hacer los procesos de entrega de software repetibles y confiables requieren mantener el control de las versiones de todos los componentes y artefactos requeridos, incluyendo, código fuente, scripts para prueba y despliegue, información de la configuración de la infraestructura y la aplicación, y en general de cualquier paquete o librería del cual se dependa. Luego la gestión de la configuración es una práctica fundamental, sin ella no será posible automatizar la construcción (Build), ejecutar pruebas automáticas (Test), realizar despligue a los ambientes (Deploy) y liberar en producción software confiable (Release). Teniendo en cuenta que la institucionalización de prácticas DevOps es un proceso iterativo de adopción, mínimamente, para iniciar debe asegurarse que el equipo de desarrollo tiene bajo el control de versiones el código fuente de la aplicación, de otro modo no es posible la implementación de la siguiente práctica: Integración Continua.

Conclusión acerca de la Integración Continua

Las prácticas de integración continua permiten que los equipos de desarrollo puedan identificar en etapas tempranas de la creación de software defectos que se están introduciendo con cada cambio que realizan al código, son la primera línea de defensa que tiene un equipo de software para lograr los niveles de calidad. También habilitan un modelo de trabajo donde los equipos y personas de desarrollo pueden

trabajar en paralelo y así aumentar no solo la productividad sino también la velocidad en la creación de nuevo código para nuevas funcionalidades, reparar defectos o pagar deuda técnica. Como tal, la integración continua es una práctica y no meramente un servidor especializado para este propósito, por eso es clave un cambio de cultura y seguimiento frecuente para lograr que los desarrolladores al menos una vez al día envíen sus cambios en el código y tener la menor cantidad de ramas (Branches) abiertas y con trabajo en progreso para evitar conflictos de integración que resultarán en reprocesos que impactan la velocidad y la productividad de los equipos.

Conclusión sobre la Automatización de Pruebas

Considerando que las compañías están todos los días más presionadas para crear software y evolucionar el existente, y que las pruebas son vitales para asegurar la calidad del software y en consecuencia de la experiencia digital de los clientes que lo usan; es indispensable automatizar todos los tipos de prueba (unitarias, funcionales, no funcionales, seguridad, entre otras). Las pruebas son un proceso que está compuesto por actividades individuales que toman mucho tiempo, sin automatización la organización tendrá que ir en detrimento o de la calidad o del costo para poder producir el software requerido en la transformación digital, opción que no es viable en la era de la economía digital. De ahí que las empresas deben entender que aquí deberán invertir gran parte de sus recursos en los proyectos de transformación de sus procesos de ingeniería del software para cambiar el circulo: más software, más pruebas, mas costo, por el circulo: más software, más pruebas, mismo o menor costo, más velocidad, mas cobertura.

Conclusión sobre la Automatización del Despliegue y Liberación

De igual modo que las pruebas, las actividades de despliegue tienden a incrementarse en la medida que se crea y evoluciona más software dentro de la compañía, por lo cual su automatización es necesaria por las mismas razones que se expusieron en el caso de las pruebas. Adicionalmente el despliegue en producción es una operación de sumo cuidado pues una falla ocasiona interrupciones de servicio a los clientes y a los procesos de negocio de la compañía. Para minimizar el riesgo es importante crear mecanismos para minimizar la probabilidad y/o el impacto de errores que puedan introducirse durante el despliegue en el ambiente de producción. El uso de arquitecturas y técnicas como Despliegues Azul y Verde (Blue and Green Deployments), Despliegues con Canarios (Canary Deployments) e Interruptor de Funcionalidad (Toggle Feature – Flag Feature); son hoy frecuentemente usadas para atender la necesidad de disminuir los riesgos.

Conclusión sobre el Ciclo Automático de Entrega Continua de Software

Un ciclo automatizado de entrega de software no sólo tiene un efecto positivo en la velocidad, sino también en la estandarización y confiabilidad del proceso. Entregar software de este modo crea un proceso rápido, repetible y confiable (Humble & Farley, 2011). Con ciclo de esta manera se impacta también la productividad y eficiencia de los equipos, se aumenta la calidad del software pero lo más importante se habilita a la organización para temas aún más relevantes en el contexto de negocio como lo son: el crecimiento, la experimentación, la innovación, la rápida retroalimentación de los clientes sobre las hipótesis de negocio, la experiencia digital, entre otros.

Conclusión sobre la Propuesta de Adopción

Sin dejar completamente de lado otros aspectos como los culturales, los procesos una compañía puede iniciar la metamorfosis de sus proceso de ingeniería del software embarcándose en un proyecto que comprenda solo una aplicación, la cual debe ser seleccionada cuidadosamente para asegurar que se satisfagan ciertos criterios como la posibilidad de experimentación y aprendizaje. Para esto es indispensable crear un equipo dedicado a la transformación y con foco exclusivo en esta.

Queda claro también que la arquitectura de las aplicaciones puede restringir la adopción de las prácticas, así que en principio siempre es necesario evaluarla y hacer las modificaciones a las que haya lugar para lograr que cuando haya necesidad de cambio en el código de la aplicación este puede modificarse, probarse y desplegarse de forma independiente.

Las tareas a automatizar dentro del ciclo [Construcción (Build), Pruebas (Test), despliegue (Deployment) y liberación en producción (Release)] deben ser aquellas que después de realizar el análisis de flujo de valor se hayan clasificado como actividades que agregan valor (Value-Adding Actitivies) o que eliminan tiempos de espera prolongados dentro del ciclo.

Conclusión acerca de las Métricas de Desempeño

Las métricas empleadas para evaluar el desempeño de la entrega de software deben ser globales y no deben crear competencia entre los equipos de desarrollo y operaciones, por el contrario deben fomentar la cooperación entre ambos. Las métricas deben medir los resultados de impacto que se quieren lograr en el caso de entrega de software estos son: Velocidad, Calidad y Eficiencia. Para esto las métricas sugeridas son:

- Tiempo de entrega de software (Delivery Lead Time)

- Frecuencia de despliegue y liberación en producción (Deployment and Release Frecuency).

- Tiempo promedio para restablecer servicio (Mean Time to Restore Service – MTTR).

- Tasa de cambios fallidos (Change Fail Rate).

REFERENCIAS

Ambler, S., & Lines, M. (2012). *DISCIPLINED AGILE DELIVERY.* Boston, MA: IBM Press.

Bass, L., Weber, I., & Liming, Z. (2015). *DEVOPS A Software Architect's Perspective.* Westford, Massachusetts: Pearson Education Inc.

Chen, L. (2015). Continuous Delivery Huge Benefits, But Challenges Too. *THE IEEE COMPUTER SOCIETY*.

CLEVERISM. (2018). *CLEVERISM*. Obtenido de https://www.cleverism.com/software-development-life-cycle-sdlc-methodologies/

CloudBees. (2018). *www.cloudbees.com*. Obtenido de https://www.cloudbees.com/products/cloudbees-core

CMMI Product Team. (2010). *CMMI for Development, Version 1.3.* Pittsburgh, PA: Software Engineering Institute.

DEVOPS AGILE SKILLS ASSOCIATION DASA. (2016). *DASA DEVOPS GLOSSARY.* Obtenido de https://www.devopsagileskills.org/resources/

DevOps Research and Assessment - DORA. (2018). *2018 Accelerate: State of DevOps, Strategies for a New Economy.* DevOps Research and Assessment.

Duvall, P. M., Matyas, S., & Andrew, G. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk.* Pearson Education, Inc.: Boston, MA.

Forrester. (2014). *Software Must Enrich Your Brand:Software Is A Critical Strategic Asset, Not An Operational Nice-To-Have.* Cambridge, MA: Forrester.

Forrester. (2017). *The Forrester Wave™: Continuous Integration Tools, Q3 2017.* Cambridge, MA: Forrester Research, Inc.

Forrester. (2018). *The Forrester Wave™: Continuous Delivery And Release Automation, Q4 2018.* Forrester.

Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.* Portland, Oregon: IT Revolution.

Fowler, M. (2006). *MARTINFOWLER.COM*. Obtenido de https://martinfowler.com/articles/continuousIntegration.html

Fowler, M. (2010). *MARTINFOWLER.COM*. Obtenido de https://martinfowler.com/bliki/FeatureToggle.html

Fowler, M. (2012). *www.martinfowler.com*. Obtenido de https://martinfowler.com/bliki/TestPyramid.html

Fowler, M. (2013). *MARTINFOWLER.COM*. Obtenido de https://martinfowler.com/bliki/ContinuousDelivery.html

GARTNER. (2015). *Market Guide for Software Change and Configuration Management Software.* Stamford, CT: GARTNER.

GARTNER. (2018). *https://www.gartner.com*. Obtenido de https://www.gartner.com/it-glossary/devops/

Gartner. (2018). *Magic Quadrant for Application Release Orchestration.* Stamford, CT: Gartner.

Gartner. (2018). *Magic Quadrant for Software Test Automation.* Stamford, CT: Gartner.

GARTNER. (2018). *www.gartner.com*. Obtenido de https://www.gartner.com/it-glossary/digital-business

GitLab. (2018). *www.gitlab.com*. Obtenido de https://about.gitlab.com/product/source-code-management/

Govindarajan, V., & Trimble, C. (2010). *The Other Side of Innovation: Solving the Execution Challenge.* Boston, Massachusetts: Harvard Business School Publishing.

Hodgson, P. (2017). *MARTINFOWLER.COM*. Obtenido de https://martinfowler.com/articles/feature-toggles.html

Humble, J. (2008). *www.continuousdelivery.com*. Obtenido de https://continuousdelivery.com/foundations/configuration-management/

Humble, J. (2014). *ThouhgtWorks*. Obtenido de The Case for Continuous Delivery: https://www.thoughtworks.com/insights/blog/case-continuous-delivery

Humble, J., & Farley, D. (2011). *CONTINUOUS DELIVERY Reliable Software Releases Through Build, Test, and Deployment Automation.* Boston, MA: Pearson Education Inc.

Kim, G., Patrick, D., Willis, J., & Humble, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations.* Portland, OR: IT Revolution Press.

Lopez, J. (2014). *www.forbes.com*. Obtenido de https://www.forbes.com/sites/gartnergroup/2014/05/07/digital-business-is-everyones-business/#4f9e6e967f82

octopus. (2014). *octopus.com*. Obtenido de https://octopus.com/blog/build-your-binaries-once

Perforce. (2018). *www.perforce.com*. Obtenido de https://www.perforce.com/products/helix-core

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit.* Crowdfordsville, Indiana: Addison-Wesley.

Puppet Labs & DevOps Research and Assessment. (2017). *2017 State of DevOps Report.* Puppet + DORA.

Puppet Labs. (2014). *2014 State of DevOps Report.* Portland, Oregón: Puppet Labs - IT Revolution Press - ToughtWorks.

Puppet Labs. (2015). *2015 State of DevOps Report.* Portland, Oregón: Puppet Labs.

Raskino, M., & Waller, G. (2015). *DIGITAL TO THE CORE.* New York, NY: Bibliomotion Inc.

Stroud, R., Oehrlich, E., LeClair, A., Kinch, A., & Kinck, D. (2017). *A Dangerous Disconnect: Executives Overstimate DevOps Maturity.* Cambridge, MA: Forrester.

VersionONE Inc. (2016). *The 10th Annual State of Agile Report.* Alpharetta, GA: VersionOne Inc.

Willis, J. (2012). *https://itrevolution.com*. Obtenido de https://itrevolution.com/the-convergence-of-devops/

Winston, R. (1970). MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS. *IEEE*, 328-338.