UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA
DIRECCIÓN DE POSTGRADO DE
INVESTIGACIÓN E INFORMÁTICA APLICADA

DISEÑO E IMPLEMENTACIÓN DE GUÍA DE MIGRACIÓN DEVOPS DESDE UNA CULTURA DE DESARROLLO DE SOFTWARE TRADICIONAL

ING. JASON RENÉ DÍAZ GONZÁLEZ

GUATEMALA, JUNIO DE 2018

UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA
DIRECCIÓN DE POSTGRADO DE INVESTIGACIÓN E INFORMÁTICA APLICADA

DISEÑO E IMPLEMENTACIÓN DE GUÍA DE MIGRACIÓN DEVOPS DESDE UNA CULTURA DE DESARROLLO DE SOFTWARE TRADICIONAL

TRABAJO DE GRADUACIÓN PRESENTADO POR:

ING. JASON RENÉ DÍAZ GONZÁLEZ

PREVIO A OPTAR EL GRADO ACADÉMICO DE:

MAGISTER ARTIUM EN INFORMÁTICA
CON ÉNFASIS EN BANCA ELECTRÓNICA
Y COMUNICACIONES

GUATEMALA, JUNIO DE 2018

AUTORIDADES DE POSTGRADO Y DEL TRIBUNAL QUE PRACTICÓ EL EXAMEN DEL TRABAJO DE GRADUACIÓN

DIRECTOR DE POSTGRADO: DR. EDGARDO ALVAREZ

PRESIDENTE DEL TRIBUNAL EXAMINADOR:

ING. M.A. ROBERTO MARIO LÓPEZ SAJQUIN

SECRETARIO: INGA. M.A. SHEYLA YADIRA ESQUIVEL

VOCAL: ING. M.A. GERBER GUSTAVO FLORES DUEÑAS

UNIVERSIDAD MARIANO GALVEZ DE GUATEMALA
DIRECCION DE POSTGRADO DE
INVESTIGACION E INFORMATICA APLICADA
3a avenida, 20-52 zona 2, interior Finca El Zapote
Tel. 2411 1800, ext. 2011, E-mail: ealvarez@umg.edu.gt
www.umg.edu.gt

POSTGRADO DE: INVESTIGACION E INFORMATICA APLICADA

PROGRAMA DE: MAESTRIA EN INFORMATICA CON ENFASIS EN BANCA ELECTRONICA Y COMUNICACIONES.

SE AUTORIZA LA IMPRESIÓN DEL TRABAJO DE GRADUACION "DISEÑO E IMPLEMENTACIÓN DE GUÍA DE MIGRACIÓN DEVOPS DESDE UNA CULTURA DE DESARROLLO DE SOFTWARE TRADICIONAL", PRESENTADO POR EL ESTUDIANTE JASON RENÉ DÍAZ GONZÁLEZ QUIEN PARA EFECTO DEBERA CUMPLIR CON LAS DISPOSICIONES REGLAMENTARIAS RESPECTIVAS.

DESE CUENTA CON EL EXPEDIENTE A SECRETARIA GENERAL DE LA UNIVERSIDAD, PARA LA CELEBRACION DEL ACTO DE INVESTIDURA Y GRADUACION PROFESIONAL CORRESPONDIENTE DE CONFORMIDAD CON EL REGLAMENTO DE TESIS.

GUATEMALA, CUATRO DE JUNIO DE 2018.

UNIVERSIDAD MARIANO GALVEZ POSGRADO DIRECCIÓN INVESTIGACIÓN E INFORMÁTICA APLICADA GUATEMALA

Edgardo ALVAREZ
Director del Postgrado

CONOCEREIS LA VERDAD Y LA VERDAD OS HARA LIBRES

REGLAMENTO DE TESIS

Artículo 8º: RESPONSABILIDAD

Solamente el autor es responsable de los conceptos expresados en el trabajo de tesis. Su aprobación en manera alguna implica responsabilidad para la Universidad.

Índice

Índice de Figuras

Índice de Tablas

Introducción

El presente trabajo de graduación tiene como objetivo Diseñar e implementar una guía práctica y concisa para la adopción de la cultura DevOps en equipos de desarrollo de software; para esto se emplean una serie de partes que permitirán al lector no solo entrar en el contexto del problema sino también de la solución.

En el origen de la producción de software se crearon algunas metodologías, ciclos de vida, técnicas, etc. para desarrollar software de manera profesional, sin embargo, con el surgimiento de los nuevos modelos de negocio, estas herramientas y filosofías quedaron obsoletas por las exigencias en tiempo/costo que estos nuevos modelos de negocio trajeron, por tal razón la primera parte del presente trabajo de graduación es el marco conceptual, que permitirá al lector entrar en contexto con el problema; se presentan los antecedentes, el planteamiento y descripción del problema, los objetivos de la investigación, alcance, limitaciones y la justificación.

Producto de los problemas previamente mencionados surgieron nuevas culturas de desarrollo de software y nuevos conceptos modernos que permiten dar solución a la problemática actual, por tal razón, la segunda parte del trabajo es el marco teórico que permite comprender los conceptos que giran en torno a DevOps y también conocer el estado actual del arte.

Al tener esto definido, se presenta el marco metodológico, que expondrá al lector una guía de implementación que permita migrar una cultura de desarrollo de software tradicional hacia una cultura DevOps por medio de la implementación de técnicas y herramientas DevOps.

Finalmente se realizará un análisis crítico y objetivo de la guía que permite presentar las conclusiones y recomendaciones producto de la investigación y creación de la guía. Acompañado de esto se presentarán las bibliografías, e-grafías, glosario y anexos que apoyan y dan propiedad al presente trabajo de graduación.

Resumen

El ciclo de vida del desarrollo de software ha evolucionado en el transcurso del tiempo; múltiples metodologías de desarrollo han surgido mientras dejan a su paso obsoletas a sus antecesoras. En el origen del software los programadores eran encargados de ejecutar todo el ciclo de desarrollo, sin embargo, conforme los proyectos crecían, se percataron que algunas tareas de implementación y mantenimiento debían ser realizadas por otro equipo; en ese momento surgió la clara división entre el equipo de Desarrollo (Development) y de Operaciones (Operations). Estos equipos han convivido a lo largo del tiempo, no obstante, han tenido serios problemas de comunicación, que provoca lentitud en la entrega de nuevas versiones, mejoras y corrección de errores en servicios de IT. En la actualidad las compañías de desarrollo de software, tienen el reto de entregar aplicaciones modernas, útiles y eficientes, que sean capaces de ajustarse a un entorno cambiante y exigente; debido a los problemas previamente mencionados y a las exigencias actuales, surgió la cultura DevOps, que establece una entrega continua de servicios IT entre Desarrollo y Operaciones basándose en tres filosofías; integración continua, testeo continuo y despliegue continuo por medio de una metodología ágil. El presente trabajo de graduación pretende proveer una guía efectiva y concisa, para migrar desde una cultura de desarrollo tradicional hacia una cultura DevOps, por medio de pasos claros y prácticos que permitan generar una mejora de rendimiento en equipos de entrega de servicios de software.

1. Marco Conceptual

1.1 Planteamiento del Problema

El desarrollo de servicios de IT de forma profesional trajo consigo la creación de diversos departamentos que debían estar en comunicación para la implementación de servicios y proyectos de software. Entre los dos departamentos más notables estaba el de Desarrollo y el de Operaciones. Uno dedicado completamente al desarrollo de nuevo software y el otro con un enfoque más técnico, responsable de darle mantenimiento y soporte al software producido por Desarrollo.

En los proyectos que eran pequeños no existían problemas en cuanto a la comunicación y alineamiento de objetivos de ambos departamentos, ya que en algunos casos eran las mismas personas involucradas en ambas áreas; sin embargo, con el pasar del tiempo, el software pasó a ser una parte determinante de las empresas, los proyectos crecieron y los requerimientos exigían no solo el involucramiento de más personas, sino la especialización en una sola área de las mismas.

Esta separación trajo consigo una serie de problemáticas comunes que eran causadas precisamente por la falta de comunicación entre lo que debía de hacer y cómo debía ser cada producto entregable de desarrollo hacia operaciones, que provocó que muchas veces estos últimos, gastaran demasiado tiempo en trasladar e implementar en ambiente de producción correcciones, mejoras y nuevas características de desarrollo.

Por estas razones, las metodologías tradicionales de software y los ciclos de vida del mismo, cambiaron con el transcurso del tiempo, desde tradicionales hasta ágiles, y en los últimos años, se creó una nueva cultura conocida como DevOps, en donde pretende romper esa separación entre los departamentos involucrados. No obstante, esta cultura es relativamente reciente, y en países de habla hispana no se tiene una documentación técnica y práctica para la implementación de la misma, mismo que provoca que una cultura tan útil no sea tan común en los equipos de desarrollo; es por ello que es necesaria la generación de una guía que permita la implementación de DevOps desde una cultura tradicional para los equipos de desarrollo de software.

1.2 Descripción del Problema

1.2.1 Formulación del Problema

La entrega de servicios de software es un proceso complejo que ha evolucionado con el transcurso del tiempo, que ha dejado como producto varios departamentos involucrados, donde dos son los principales, por una parte Desarrollo conformado por un equipo de analistas, arquitectos y desarrolladores de software, enfocado en el análisis, diseño y producción del mismo; por otra parte tenemos a Operaciones también conocido como Sistemas, que es un equipo conformado por administradores de sistemas enfocados en la implementación, mantenimiento y soporte de los servicios de software.

La complejidad y la exigencia de los servicios de software modernos, han causado que una incorrecta comunicación entre estos dos equipos, tenga como consecuencia una lentitud en el despliegue de nuevas versiones, poca flexibilidad para la aplicación de mejoras y una ineficiente capacidad de respuesta al cambio, que provoca un fracaso de los servicios prestados por ambos equipos y, como consecuencia aún mayor, pérdida del valor agregado en los productos producidos. Debido a esto, hace algunos años surgió una cultura de desarrollo denominada DevOps, misma que pretende optimizar la comunicación entre ambos departamentos por medio de la automatización de tareas repetitivas, dando paso, a una mejora circunstancial en el rendimiento de los equipos de entrega de servicios de software. No obstante, la migración hacia esta cultura puede ser un proceso largo y rodeado de incertidumbre por lo relativamente nueva que es la cultura. Esto provoca el surgimiento de la interrogante sobre ¿cuáles son las etapas para la migración de una cultura DevOps desde una cultura de desarrollo de software tradicional?

1.2.2 Preguntas de Investigación

En torno a la problemática surgen algunos cuestionamientos que pretenden ser resueltos como producto del presente trabajo.

1. ¿Cuáles son los modelos y ciclos de vida del desarrollo tradicional?
2. ¿Cuáles son los beneficios y desafíos que tenemos en el uso de metodologías ágiles contra metodologías tradicionales?
3. ¿Cuáles son los diferentes roles que intervienen en un equipo de desarrollo?
4. ¿Cuáles son las áreas que intervienen en DevOps?
5. ¿Qué es Despliegue Continuo y cuál es su relación con DevOps?
6. ¿Cuáles son las herramientas básicas para tener una implementación DevOps efectiva?
7. ¿Cuáles son los requerimientos mínimos para la implementación exitosa de DevOps en un equipo de desarrollo?

1.3 Objetivos de la Investigación

1.3.1 Objetivo General

Diseñar e implementar una guía práctica y concisa de migración DevOps desde una cultura de desarrollo de software tradicional.

1.3.2 Objetivos Específicos

- ✓ Describir las metodologías de desarrollo y su importancia al desarrollar Software.
- ✓ Identificar y clasificar los diferentes roles en un equipo de Desarrollo de Software.
- ✓ Comparar las metodologías de Desarrollo de Software ágiles y las tradicionales.
- ✓ Definir la cultura DevOps describiendo beneficios y desafíos para su implementación.
- ✓ Describir las áreas involucradas en un equipo que implemente una cultura de desarrollo de software con DevOps.
- ✓ Establecer las herramientas utilizadas para la automatización de los procesos de software mediante DevOps.
- ✓ Determinar los principios y aspectos que se deben tener presentes al implementar DevOps.
- ✓ Definir una serie de etapas para la implementación de la cultura DevOps en un equipo de desarrollo de software.
- ✓ Listar los beneficios y mejoras al implementar DevOps.

1.4 Justificación de la Investigación

Al tener entendimiento de la problemática surgida entre los departamentos de Desarrollo y Operaciones, es natural que se busque una solución concreta y correcta que abarque de manera integral todos los posibles aspectos que pudieran surgir; producto de esto nació el enfoque cultural DevOps, no obstante, en la actualidad no existe una guía completa de pasos bien definidos que permitan la migración de una cultura tradicional hacia una moderna con DevOps, esto hace necesaria la creación de la misma para que sea de utilidad para los equipos de desarrollo.

Por otra parte, al ser una cultura relativamente reciente, la mayoría de literatura científica sobre el tema se encuentra en idioma inglés, en donde se crea otra barrera para países de habla hispana. Además, el panorama actual del desarrollo de software es producido sobre metodologías tradicionales, que hace aún más necesaria la elaboración de documentación que pueda apoyar al aprovechamiento de las técnicas y posibilidades que provee DevOps.

No tener implementada una cultura DevOps conlleva un incorrecto alineamiento de objetivos de un equipo de desarrollo de software, ya que el área de Desarrollo tiene un enfoque de importancia y el área de Operaciones otro, generando un choque directo entre las actividades y responsabilidades de todos los miembros involucrados, mismo que produce lentitud para producir una tarea que sería considerablemente más sencilla si se tuviera implementada la cultura DevOps. El principal beneficio que obtiene un equipo de desarrollo de software con DevOps es tiempo, ya que por medio de la automatización los procesos fluyen de manera constante, sin embargo, se obtienen también reducción en costos de mano de obra ya que se producen menos errores, y al tener un ciclo continuo, el corregir errores es también un proceso menos costoso.

1.5 Alcance y Limitaciones

1.5.1 Alcance

Diseñar e implementar una guía DevOps no es una tarea sencilla, debido a que esta, no es una metodología práctica si no un enfoque cultural, por lo tanto es un proceso complejo y cuidadoso en el que tener presente cada aspecto importante, es una tarea transcendental.

El presente trabajo de graduación pretende cubrir los siguientes factores:

- De Ingeniería de Software, se describen las diferentes metodologías de desarrollo tanto tradicionales como ágiles.
- De los equipos de desarrollo de software, se mencionan sus roles, retos y problemáticas con énfasis en los equipos bajo las metodologías ágiles como programación extrema y Scrum.
- De DevOps, se definen los conceptos básicos para comprender todas las partes involucradas, así como las tecnologías y técnicas necesarias para su correcta implementación.
- De la Guía de implementación DevOps, se estructura el proceso completo desde las definiciones hasta la implementación final.

1.5.2 Limitaciones

Existen algunos aspectos que se salen de los límites fijados para el presente trabajo de graduación, entre los que podemos mencionar:

- Análisis detallado de las metodologías tradicionales y su mapeo correspondiente con metodologías ágiles de desarrollo de software.
- Descripción y definición profunda de microservicios y su manejo en contenedores.
- Uso avanzado y especializado de herramientas para control de versiones, de integración continua, testeo continuo y despliegue continuo.
- Relación de mapeo entre DevOps y estándares internacionales para la entrega de servicios como ISO/IEC 20000 o ITIL.
- Implementación en la nube de una cultura DevOps.

2. Marco Teórico

2.1Ingeniería de Software

2.1.1 Definición e Historia

Antes de hablar sobre lo que es Ingeniería de Software, se considera importante descomponer sus elementos para ser analizados por separado. El Software de computadora es el producto que construyen los programadores profesionales y al que después le dan mantenimiento durante un período de tiempo. Incluye programas que se ejecutan en una computadora de cualquier tamaño y arquitectura, contenido que se presenta a medida de que se ejecutan los programas de cómputo e información descriptiva tanto en una copia dura como en formatos virtuales, en donde engloban virtualmente a cualesquiera de los medios electrónicos (Pressman, 2010). El software es exclusivamente desarrollado por ingenieros de software que pueden ser arquitectos y programadores de distintos niveles. Actualmente el software juega un papel muy importante por su delicada incidencia en nuestras vidas. Se ha registrado el mismo en el comercio, cultura, entretenimiento, salud, deporte y demás actividades de nuestro diario vivir.

Por otra parte, tenemos que la ingeniería es el conjunto de conocimientos científicos y tecnológicos para la innovación, invención, desarrollo y mejora de técnicas y herramientas para satisfacer las necesidades de las empresas y la sociedad. La revolución industrial contribuyó enormemente al desarrollo de casi todas las ingenierías y generó las condiciones necesarias para que la tecnología y la ciencia avanzaran en forma mancomunada y produjeran efectos espectaculares en el siglo XX. (Grench Mayor, 2001)

La ingeniería de software es un término que no surgió hace mucho tiempo, el mismo fue usado por primera vez por el profesor y científico de computación alemán Friedrich Ludwig “Fritz” Bauer en la primera conferencia sobre Ingeniería de Software celebrada en 1968 en la ciudad de Múnich Alemania; dicha conferencia fue financiada por la OTAN, misma que le dio cierta transcendencia que permitió que un año más tarde la misma OTAN financiara una segunda conferencia en Roma. El terminó descrito por Fritz Bauer fue “La Ingeniería de Software es el establecimiento y uso de principios fundamentales de la ingeniería, con objeto de desarrollar en forma económica software que sea confiable y que trabaje con eficiencia en máquinas reales”. (Randell, 1996)

Este concepto como bien indica Pressman, “dice poco sobre los aspectos técnicos de la calidad del software; no habla directamente de la necesidad de satisfacer a los consumidores ni de entregar el producto a tiempo; omite mencionar la importancia de la medición y la metrología; no establece la importancia de un proceso eficaz” sin embargo, al mencionar “principios fundamentales de la ingeniería” deja entrever la importancia de lo omitido. Con el afán de poder enriquecer el concepto, tenemos la definición del IEEE (Institute of Electrical and Electronics Engineers por sus siglas en inglés) que nos dice que “La ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software” (IEEE, 1990)

Según Pressman, la ingeniería de software es una tecnología de varias capas, en donde cualquier enfoque de ingeniería (incluso la de software) debe basarse en un compromiso organizacional con la calidad. La administración total de la calidad, Six Sigma y otras filosofías similares alimentan la cultura de mejor continua, y es esta cultura la que lleva en última instancia al desarrollo de enfoques cada vez más eficaces de la ingeniería de software (Pressman, 2010).

2.1.2 Metodologías Tradicionales de Desarrollo de Software

En la década de 1960 estalló el desarrollo de sistemas tradicionales a gran escala. Esto conllevó la creación de técnicas que permitieran un desarrollo estructurado y metódico, que produjo el nacimiento de las Metodologías Tradicionales de Desarrollo. No es tan simple encontrar un concepto de metodología de desarrollo de software, sin embargo, se ha descrito de diferentes maneras el concepto, y esto acompañado de la experiencia de algunos profesionales, ha permitido tener conceptos bastante aceptados. Una metodología de desarrollo de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo de un sistema de información. Una amplia variedad de estos marcos de trabajo ha evolucionado a lo largo de los años, cada uno con sus propias fortalezas y debilidades reconocidas. Una metodología de desarrollo de sistemas no es necesariamente adecuada para ser utilizada por todos los proyectos. Cada una de las metodologías disponibles se adapta mejor a tipos específicos de proyectos, basados en diversas organizaciones, proyectos y equipos (CMS, 2008).

A su vez podemos definir a una metodología de desarrollo como “un proceso mediante el cual un proyecto de software es completado o desarrollado a través de procesos o etapas bien definidas” (Chandra, 2015). También existen personas que prefieren conocer una metodología de desarrollo de software como una filosofía de desarrollo de programas de computación con el enfoque del proceso de desarrollo de software. Todos los conceptos van encaminados a hacer uso de diversas herramientas, técnicas, métodos y modelos para el desarrollo de software.

Existen diversas metodologías que surgieron como producto de la experiencia de ciertos proyectos, ya que como se mencionó anteriormente, no existe una metodología ideal que cubra toda la amplia gama de proyectos existentes; con esto, los encargados de proyectos se percataron que algunas metodologías no se adaptaban de manera ideal a ciertos proyectos, mismo que dio paso a la creación de una nueva metodología. Las metodologías que fueron creadas primero son también conocidas como tradicionales. A continuación, se listan las más importantes con énfasis en sus ventajas y desventajas.

2.1.2.1 Metodología en Cascada

La metodología en cascada es uno de los modelos de desarrollo de software más clásicos y primitivos. En este, existen siete fases para desarrollar software los cuales son Análisis de Requisitos, Diseño del Sistema, Diseño del Programa, Codificación, Ejecución de Pruebas, Verificación y Mantenimiento. Básicamente, el estilo del modelo en cascada, es que no se podrá avanzar a la siguiente fase, si la anterior no se encuentra totalmente terminada, pues no tiene por qué haber vuelta atrás (Ok-Hosting, 2016). Esta metodología se basa en principios como la división secuencial de fases, énfasis enérgico en la planificación, plazos, fechas límite y presupuestos, así como la completa documentación de cada fase con la debida autorización de los interesados al finalizar cada fase y antes de iniciar la siguiente.

La metodología en cascada presenta varias fortalezas y ventajas entre las que se encuentran:

1. Muy buena opción para proyectos donde los objetivos son simples y fáciles de entender.
2. Ideal para proyectos en donde los miembros del equipo son poco experimentados.
3. Es bastante simple.

4. Sus fases están bien definidas y documentadas, lo que permite que el riesgo de cambio de los miembros del equipo sea menor.
5. Los usuarios finales conocen bien el proyecto debido a la autorización de cada etapa.
6. El progreso del desarrollo del proyecto es mesurable.
7. Ayuda a conservar los recursos disponibles debido a sus estrictos controles.

Por otra parte, esta metodología presenta algunas debilidades y desventajas las cuales son:

1. Inflexible, lenta, costosa y puede llegar a ser engorrosa debido a su estricta estructura.
2. El proyecto debe avanzar hacia adelante y la corrección de fases anteriores debe ser prácticamente nula, esto perjudica considerablemente la casi segura corrección de errores.
3. Depende de la identificación y especificación inicial de los usuarios, cuando se ha documentado que los usuarios son capaces de definir esto hasta fases posteriores.
4. Los errores son descubiertos hasta la fase de pruebas.
5. Difícil y lenta respuesta a los cambios.
6. Produce documentación excesiva la cual puede llegar a ser imposible de actualizar y mantener a medida que avanza el proyecto.

2.1.2.2 Metodología Iterativa o Incremental

Esta metodología es la modificación de la metodología en cascada tradicional. Tiene la facilidad de dar retroalimentación a la etapa anterior, con esto permite rectificar un error con facilidad, antes de pasar a la siguiente etapa. Dado que el error se propaga hacia atrás es por eso que se llama modelo iterativo en cascada o incremental (Chandra, 2015)

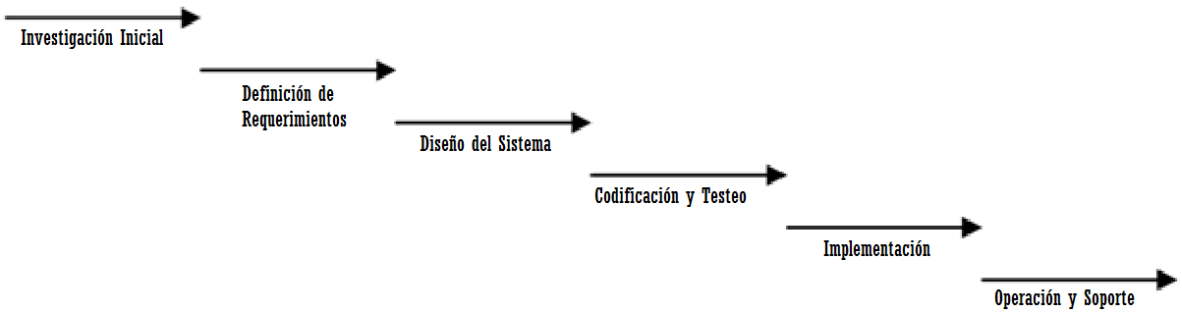

Figura 1 - Metodología en Cascada, traducción personal Fuente: (CMS, 2008)

El Modelo Incremental repite el modelo de cascada una y otra vez, pero con pequeñas modificaciones o actualizaciones que se le pueden agregar al sistema. De este modo el usuario final se ve sumamente sumergido en el desarrollo y puedes proporcionarle un resultado óptimo (Ok-Hosting, 2016). Este modelo iterativo, es la base para muchas metodologías cíclicas que fueron creadas posteriormente. La metodología iterativa o incremental cuenta con tres fases que son Inicialización, Periodos de iteración (incluyen la metodología linear) y Lista de Control. Entre sus fortalezas y ventajas encontramos las siguientes que se detallan a continuación:

1. Más flexible que la metodología en cascada.
2. Es bastante simple de implementar.
3. Las etapas al igual que el modelo en cascada, están bien definidas.
4. Mayor posibilidad de rectificación de errores de una etapa a otra.

Por otra parte, las debilidades del modelo incremental o iterativo son:

1. Falta de consideración general del problema del negocio y los requisitos técnicos para el sistema en general.
2. No es la mejor opción para proyectos complejos y de misión crítica.
3. No hay iteraciones después de la implementación.
4. En algunos casos puede llegar a consumir mucho tiempo.

2.1.2.3 Metodología de Prototipos

Antes de iniciar a hablar de esta metodología, es necesario definir lo que es un prototipo. Un prototipo es una representación limitada de un producto, permite a las partes probarlo en situaciones reales y explorar su uso (Lacalle, 2016). Un prototipo es útil debido a que los clientes o usuarios finales, a menudo no saben exactamente lo que quieren, no obstante, por medio de un prototipo visual, pueden acercarse de mejor manera a la solución de sus necesidades, por otra parte, un prototipo es usado para comunicar, discutir y definir ideas entre los desarrolladores y los involucrados. Además, como bien indica Paloma de Sedekia Ingeniería, un prototipo “es un primer modelo que sirve como representación o simulación del producto final y que nos permite verificar el diseño y confirmar que cuenta con las características específicas planeadas” (Paloma, 2017).

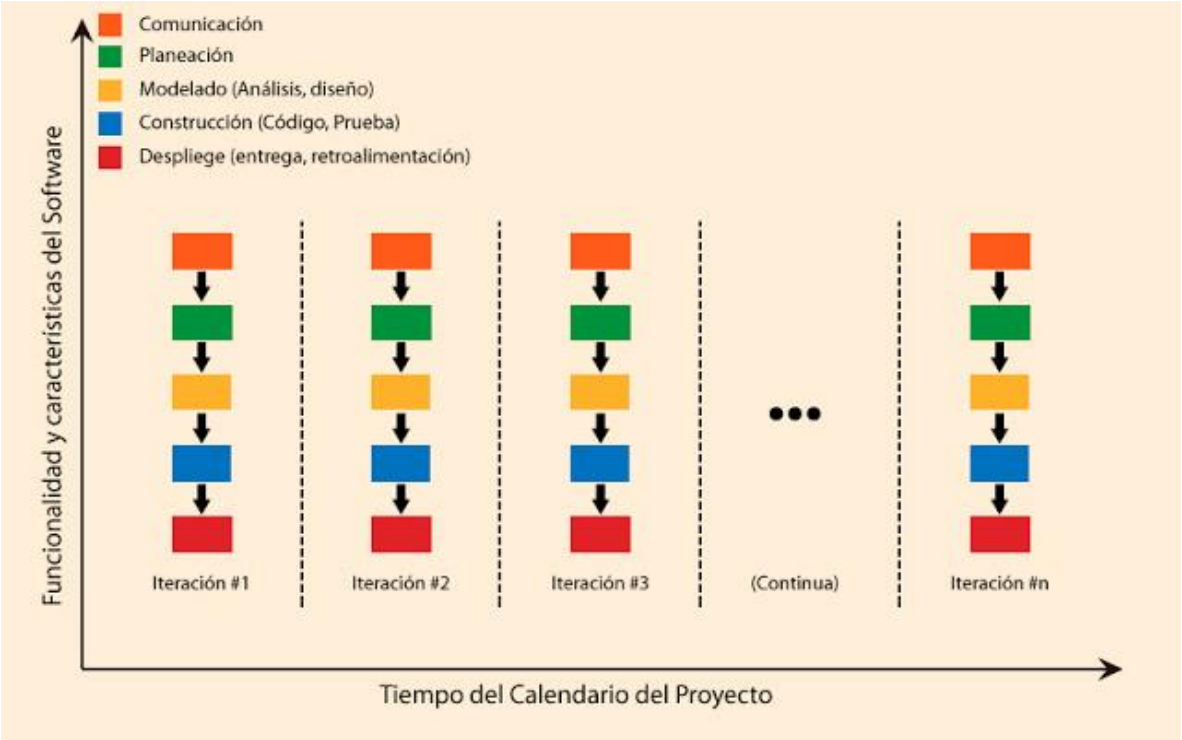

Figura 2 - Modelo Iterativo e Incremental Fuente: (Ortiz, 2012)

Una metodología de desarrollo de software basada en Prototipos consiste en la entrega de una versión inicial del software, misma que no es completamente funcional pero que sirve de base para llegar a una versión estable del producto final. Esto lo realiza por medio de siete etapas las cuales son Planeación, Modelado, Elaboración del Prototipo, Desarrollo, Entrega y Retroalimentación, Comunicación con el Cliente y Entrega del Producto Final.

Esta metodología es distinta en comparación con las anteriormente mencionadas, la misma, cuenta con las siguientes fortalezas y ventajas:

1. Reduce el margen de error debido a que el producto final está basado en los prototipos con la retroalimentación del cliente o usuario final.
2. Involucra de manera activa a los usuarios finales.
3. Es ideal para proyectos en los que no se tienen bien definidos los objetivos inicialmente.
4. Más requerimientos pueden ser añadidos mientras se desarrolla el software.
5. Es mucho más flexible que los modelos tradicionales.

Además de las fortalezas mencionadas, el modelo basado en prototipos cuenta con las siguientes debilidades y desventajas:

1. No es una buena elección para proyectos de gran tamaño.
2. En algunas ocasiones, el prototipo genera requerimientos excesivos y no útiles por parte del usuario y el prototipo inicial consume mucho más tiempo de lo planificado.
3. La documentación del proyecto puede en gran medida no representar al mismo.

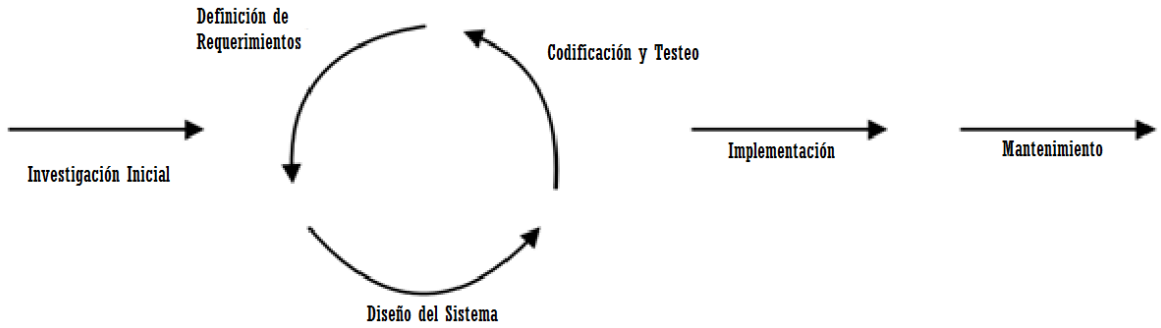

Figura 3 - Modelo basado en prototipos, Fuente: (CMS, 2008) traducción personal

2.1.2.4 Metodología de Espiral

El modelo en espiral, fue utilizado y diseñado por primera vez por el ingeniero informático estadounidense Barry Boehm en el año 1986. A este modelo se le conoce también como meta modelo debido a que ha definido bien su estructura para el desarrollo de software, el mismo, consiste en cuatro cuadrantes que se ejecutan en modo de espiral, los cuales son Determinación de objetivos, alternativas y constantes de la iteración, Evaluación de alternativas, identificación y resolución de riesgos, Desarrollo y verificación de entregables de la iteración y Planificación de la siguiente Iteración. Como se puede apreciar, en este modelo ha incluido la gestión de riesgos, algo que no fue abordado por las metodologías previas. Según Boehm cada ciclo debe comenzar con una identificación de las partes interesadas y sus condiciones de satisfacción, y también debe terminar con revisión y entrega.

Entre las fortalezas y ventajas de esta metodología podemos encontrar las siguientes:

1. El desarrollo de software es dividido en pequeñas partes de riesgo.
2. El cambio de requerimientos durante el desarrollo es aceptado.
3. Es una buena alternativa para desarrollo crítico de software.
4. Esa más de un prototipo.

Por otra parte, entre las debilidades encontramos las siguientes:

1. Puede llegar a ser bastante complicado manejar de manera correcta cada viaje de la espiral.
2. Los objetivos deben ser bien entendidos.
3. Requiere desarrolladores bien calificados y experimentados para este tipo de metodología.

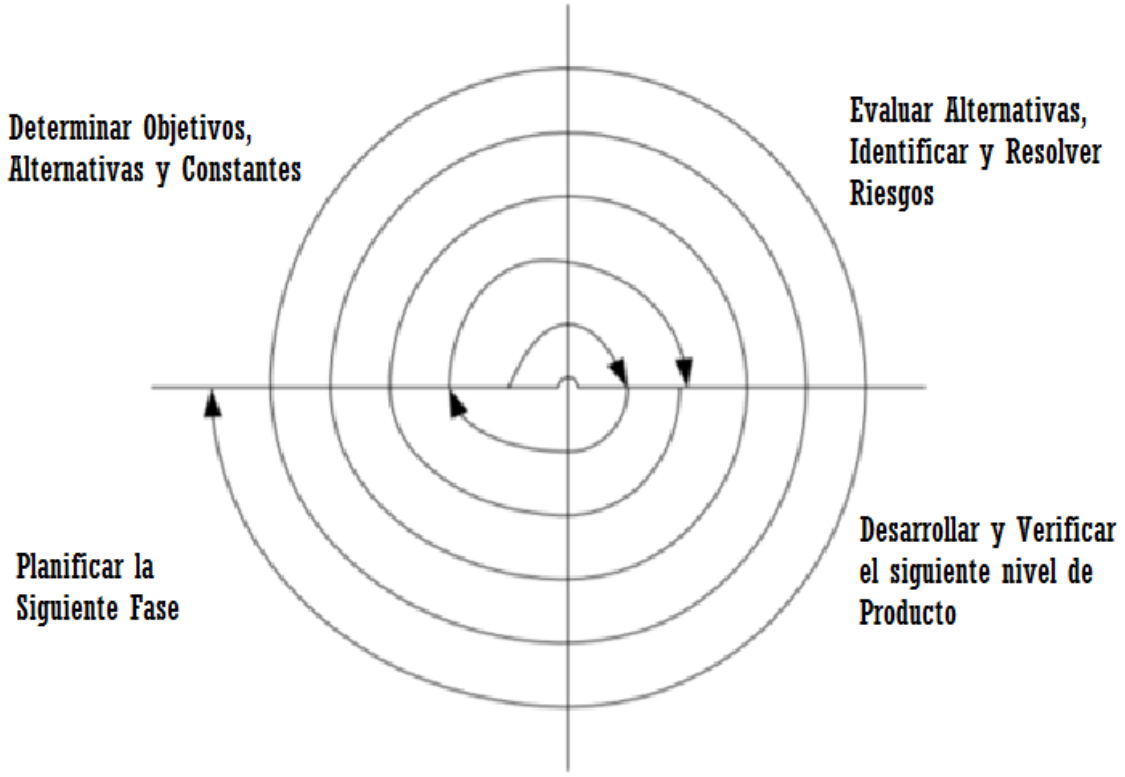

Figura 4 - Metodología de Espiral, Fuente: (Ok-Hosting, 2016) traducción personal

2.1.2.5 Desarrollo Rápido de Aplicaciones

El desarrollo rápido de aplicaciones (Rapid Application Development por sus siglas en inglés) describe un método de desarrollo de software que enfatiza fuertemente el prototipado rápido y la entrega iterativa. El modelo RAD es, por lo tanto, una fuerte alternativa al típico modelo de desarrollo de cascada o linear, que a menudo se centra principalmente en la planificación y prácticas de diseño secuencial. Introducido por primera vez en 1991 en el libro de James Martin con el mismo nombre, el desarrollo rápido de aplicaciones se ha

convertido en uno de los métodos de desarrollo más populares y poderosos, que pertenece a la categoría de técnicas de desarrollo relativamente ágiles (Powell, 2016).

Según Andrew Powell, el desarrollo rápido de aplicaciones presenta las siguientes ventajas y fortalezas:

1. Progreso mensurable: Con frecuentes iteraciones, componentes y prototipos que bajan por la tubería, el progreso en el proyecto en su conjunto, así como en segmentos menores, puede ser fácilmente medido y evaluado para mantener los horarios y los presupuestos.
2. Generación Rápida de Código Productivo: A medida que un mayor porcentaje de desarrolladores de software activos se mueven en roles multidisciplinarios, una metodología RAD permite a los miembros de equipo capacitados, producir rápidamente prototipos y código de trabajo para ilustrar ejemplos que podrían tardar semanas o meses para ver la luz del día si se usa una técnica de desarrollo más lenta.
3. Permite Componentes Reutilizables e Independientes: Debido a que se entregan prototipos, los desarrolladores están obligados a crear componentes independientes que pueden ser reutilizados en diferentes partes del sistema.
4. Respuesta rápida y constante del usuario: Las metodologías RAD permiten la interacción y la retroalimentación casi constante del usuario a través de frecuentes iteraciones y lanzamientos de prototipos.
5. Integración temprana de sistemas: Mientras que la mayoría de los proyectos de software de método de cascada deben, por su propia naturaleza, esperar hasta el final del ciclo de vida para comenzar las integraciones con otros sistemas o servicios, una aplicación rápidamente desarrollada se integra casi de inmediato.

Por otra parte, menciona algunas de sus debilidades y desventajas tales como:

1. Requiere Sistemas Modulares: Dado que cada componente dentro del sistema debe ser iterable y comprobable por sí mismo, el diseño general del sistema al usar RAD requiere que cada componente sea modular, lo cual permite que los elementos sean intercambiados y alterados por una variedad de miembros del equipo.

2. Dificultad en proyectos de gran escala: Si bien los métodos de desarrollo rápido de aplicaciones conducen a una mayor flexibilidad durante el proceso de diseño y desarrollo, también tenderán a reducir el control y las restricciones.
3. Demandas de Interconexión Frecuente del Usuario: Obtener información y retroalimentación del usuario temprano y a menudo es sin duda un beneficio desde una perspectiva de diseño, pero esta espada de doble filo requiere que el equipo esté dispuesto y capaz de comunicarse con los usuarios sobre una base mucho más frecuente, en comparación a un método típico de desarrollo de cascada.
4. Depende de los desarrolladores expertos: Si bien muchos desarrolladores en estos días son multidisciplinados, vale la pena señalar que el uso de técnicas RAD requiere una mayor habilidad general en todo el equipo de desarrollo, con el fin de adaptarse rápidamente a medida que el sistema y los componentes evolucionan.

2.1.3 Metodologías ágiles de Desarrollo de Software

Los métodos ágiles son una reacción a las formas tradicionales de desarrollar software y reconocen la "necesidad de una alternativa a los procesos de desarrollo de software de peso pesado impulsados por la documentación" (Beck, 2005). Como se pudo apreciar en las secciones anteriores, en la implementación de metodologías tradicionales para el desarrollo de software, el trabajo comienza con la obtención y la documentación de un conjunto de requisitos "completo", seguido de diseño, desarrollo e inspección arquitectónicos y de alto nivel. A partir de mediados de la década de 1990, algunos practicantes consideraron frustrantes estos pasos iniciales de desarrollo y, tal vez, imposibles. La industria y la tecnología se mueven demasiado rápido, los requisitos "cambian a tasas que inundan los métodos tradicionales", y los clientes se han vuelto cada vez más incapaces de declarar definitivamente sus necesidades por adelantado y, al mismo tiempo, esperan más de su software (Highsmith, 2002). Como resultado, varios consultores desarrollaron métodos y prácticas de forma independiente para responder al cambio inevitable que se experimentó. Estos métodos conocidos como ágiles son en realidad una colección de diferentes técnicas (o prácticas) que comparten los mismos valores y principios básicos. Muchos se basan, por ejemplo, en la mejora iterativa, una técnica que se introdujo en 1975. (Cohen, 2015)

El 17 de febrero de 2001 en Snowbird Utah, el ingeniero de software estadounidense Kent Beck, convocó a 17 críticos de los modelos de mejora del desarrollo de software, para tratar técnicas y procesos para desarrollar software. Como producto de esta convocatoria surgió el Manifiesto Ágil, firmado por los 17 expertos, en donde se resumieron los 4 principios básicos sobre los cuales están creados los métodos ágiles. Estos principios exponen los elementos que son más valorados por las metodologías ágiles en contraste con las metodologías tradicionales. "El movimiento Ágil no es anti-metodología, de hecho, muchos de nosotros queremos restaurar la credibilidad de la palabra metodología. Queremos restaurar un balance. Aceptamos el modelado, pero no para archivar un diagrama en un polvoriento repositorio corporativo. Aceptamos la documentación, pero no cientos de páginas de tomos nunca mantenidos y raramente utilizados. Planificamos, pero reconocemos los límites de la planificación en un entorno turbulento" (Beck, 2001).

Los cuatro elementos en los que están basadas las metodologías ágiles son:

1. Individuos e interacciones sobre procesos y herramientas.
2. Software funcional sobre documentación extensiva.
3. Colaboración con el cliente sobre negociación contractual.
4. Respuesta ante el cambio sobre seguir un plan.

En el sitio web oficial del Manifiesto Ágil, se menciona que los elementos de la derecha de los principios, permanecen con importancia, sin embargo, los elementos de la izquierda son más valorados (Beck, 2001). Además de esto, existen doce principios que son muy importantes en las metodologías ágiles, ya que, en estos, es donde residen las grandes diferencias con las metodologías tradicionales:

1. La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Se acepta que los requerimientos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Se entrega software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

4. Los responsables del negocio y los desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcional es la medida principal de progreso.
8. Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Derivado de esto, surgieron diversas metodologías ágiles de desarrollo, creadas en base a los anteriores principios. Entre las metodologías ágiles más usadas y efectivas en la actualidad podemos encontrar Scrum, Extreme Programming, Kanban, Desarrollo de Software Adaptativo, Crystal, Agile Inception, Dynamic Systems Development Method (DSDM), Proceso Unificado Ágil, entre otras. Las dos principales serán descritas en las siguientes secciones.

2.1.3.1 Scrum

Scrum es la metodología para el desarrollo de software más usada en la actualidad debido a sus altos niveles de efectividad. Tiene su origen sobre un estudio realizado por el profesor de la Universidad de Harvard Hirotaka Takeuchi en el año 1986, mismo que trata sobre el éxito de los procesos realizados en Japón y Estados Unidos por empresas como Xerox, Honda, Canon, HP, entre otros. Estas empresas debían desarrollar productos con requerimientos muy generales y novedosos, no obstante, tenían la peculiaridad de salir al mercado en mucho menos tiempo comparado con otro tipo de productos anteriores. La manera en que los equipos altamente productivos trabajaban, fue comparada con la colaboración que tienen los equipos

de Rugby y su formación Scrum. Fue en el año de 1993 en el que Jeff Sutherland, John Scumniotales y Jeff McKenna concibieron, ejecutaron y documentaron el primer Scum para desarrollo ágil de software. En el año de 1995 el Scrum para software fue normalizado por Ken Schwaber, creador de los primeros libros acerca de Scrum. En la actualidad, Scrum es utilizado en diferentes tipos de negocio y, especialmente, en el desarrollo de software, sin importar el tamaño del equipo que lo implemente, desde startups (negocios pequeños) hasta multinacionales. La Scrum Alliance es la organización sin ánimo de lucro que se encarga de difundir Scrum (Scrum Alliance, 2017).

Según Ken Schwaber, Scrum no es una metodología propiamente dicha, sino un marco de trabajo, esto en otras palabras quiere decir que Scrum no define exactamente qué es lo que se tiene que hacer sino más bien que se debe tomar en cuenta y cómo debe hacerse. Scrum es un proceso en el que se aplican un conjunto de buenas prácticas para trabajar colaborativamente en equipo, y con esto obtener el mejor resultado posible de un proyecto de software. Scrum está basado en entregas de software parcial y regular del producto final, que se generan de acuerdo a la importancia y relevancia que tengan para el usuario final. Scrum como se mencionó anteriormente, es ideal para proyectos en donde hay altos niveles de complejidad y cambios constantes en los requerimientos en donde la innovación es esencial. Los equipos de desarrollo de software que implementan Scrum, tienden a ser auto-organizados, multidisciplinarios y autónomos. Al ser un equipo multidisciplinario, Scrum obliga a que todo el equipo esté involucrado en el proceso de convertir requerimientos a un software entregable. Por otra parte, al ser un equipo auto-organizado y autónomo, Scrum obliga a que no exista un líder específico que coordine, quién debe resolver cada problema, sino que, al contrario, cada miembro del equipo está capacitado, empoderado y autorizado a resolver por sí mismo un problema (Mountain Goat Software, 2017).

Scrum está conformado por pequeñas iteraciones denominadas Sprints, que son el corazón de Scrum, son un intervalo prefijado durante el cual se crea un incremento del producto potencialmente entregable, es decir, un Sprint se caracteriza por finalizar con un entregable, fácil de transferir al usuario final. La duración de los Sprints va desde 1 hasta 6 semanas, se recomienda, usar el menor tiempo posible. Cada Sprint se puede considerar un mini-proyecto, el cual, al igual que los proyectos, se utilizan para lograr algo. Cada Sprint cuenta con una

definición de lo que se va a construir, un diseño y un plan flexible que guiará la construcción del plan, el trabajo, y el producto resultante (Bara, 2015). Un Sprint está conformado por cinco etapas que se detallan a continuación:

1. Reunión de planificación de Sprint: Es una reunión en la que se define la funcionalidad en el incremento planeado y cómo el equipo de desarrollo creará este incremento y la salida de este trabajo es definir el Objetivo del Sprint. La duración varía en relación a la duración total del Sprint, para Sprints de 2 semanas, la reunión de planificación dura 4 horas.
2. Scrum Diario: Es un pequeño evento diario de 15 minutos, cuyo objetivo esencial es que todo el equipo sincronice actividades y se cree un plan para las próximas 24 horas. Algunos equipos optan por realizar de pie el Scrum Diario para asegurar la menor duración.
3. Trabajo de desarrollo durante el Sprint: Es todo el trabajo que se realiza para cumplir los objetivos del Sprint el cual finaliza con un entregable tangible para el usuario final.
4. Revisión del Sprint: Se lleva a cabo al final del Sprint, para inspeccionar el incremento y adaptar, si es necesario, el Product Backlog. El Equipo Scrum y las partes interesadas colaboran durante la revisión de lo que se hizo en el Sprint.
5. Retrospectiva del Sprint: Es una oportunidad para el Equipo Scrum de inspeccionarse a sí mismo y crear un plan de mejoras para ejecutar durante el siguiente Sprint. (Bara, 2015)

Scrum posee tres artefactos por medio de los cuales, la metodología adquiere sentido. Los artefactos están diseñados específicamente para facilitar la transparencia de la información clave y unificar los criterios de compresión de dicho artefacto.

1. Product Backlog: también conocida como Pila de Producto, es básicamente, una lista priorizada de requisitos del cliente, que son descritas con uso de la terminología del mismo (Kniberg, 2008). El Product Backlog contiene las historias de usuario, que están conformadas por todas características, funciones, requerimientos, mejoras y correcciones que constituyen los cambios que deben introducirse en el producto en futuras versiones. Los elementos del Product Backlog deben contender los siguientes

atributos: descripción, orden, estimación y valor. El Product Backlog es dinámico; cambia constantemente de acuerdo a lo que el producto requiere para mantenerse adecuado, competitivo y útil. (IDS, 2014)

2. Sprint Backlog: es el conjunto de elementos seleccionados del Product Backlog para el Sprint, además de un plan para la entrega del Incremento y el cumplimiento del objetivo del Sprint. El Sprint Backlog es una proyección realizada por el Equipo de Desarrollo sobre la funcionalidad que estará en el próximo Incremento y el trabajo necesario para convertir esa funcionalidad en un incremento “Terminado". El Sprint Backlog hace visible todo el trabajo que el equipo de desarrollo ha identificado como necesario para cumplir con el objetivo del Sprint. (IDS, 2014)
3. Scrum Task Board: una manera simple de manejar el Sprint Backlog, es colocarlo de manera visible. El Scrum Task Board no es más que un tablero que ordena las tareas del Sprint Backlog, clasificándolas por tareas pendientes de hacer, en curso y las realizadas. Además de tener una sección para los impedimentos con sus respectivos pendientes, en curso y resueltos, así como un espacio para los puntos de retrospectiva. No existe una forma definitiva de hacer el Scrum Task Board, cada equipo puede ajustarlo como considere más conveniente, no obstante, colocar las tareas no planificadas, que se ejecutan por urgencia como hotfixes pueden ayudar a saber qué tan reactivo es el equipo, además se puede agregar las tareas de retroalimentación, para saber que tanto está aprendiendo de las retrospectivas anteriores el equipo.

En Scrum sobresalen tres roles que son el Scrum Master, Product Owner y el Equipo Scrum. El Scrum Master es el entrenador del equipo de desarrollo de software, es la figura que lidera los equipos en la gestión ágil de proyectos. Su misión es que los equipos de trabajo alcancen sus objetivos hasta llegar a la fase de “sprint final”, con la eliminación de cualquier dificultad que puedan encontrar en el camino (Canal, 2015).

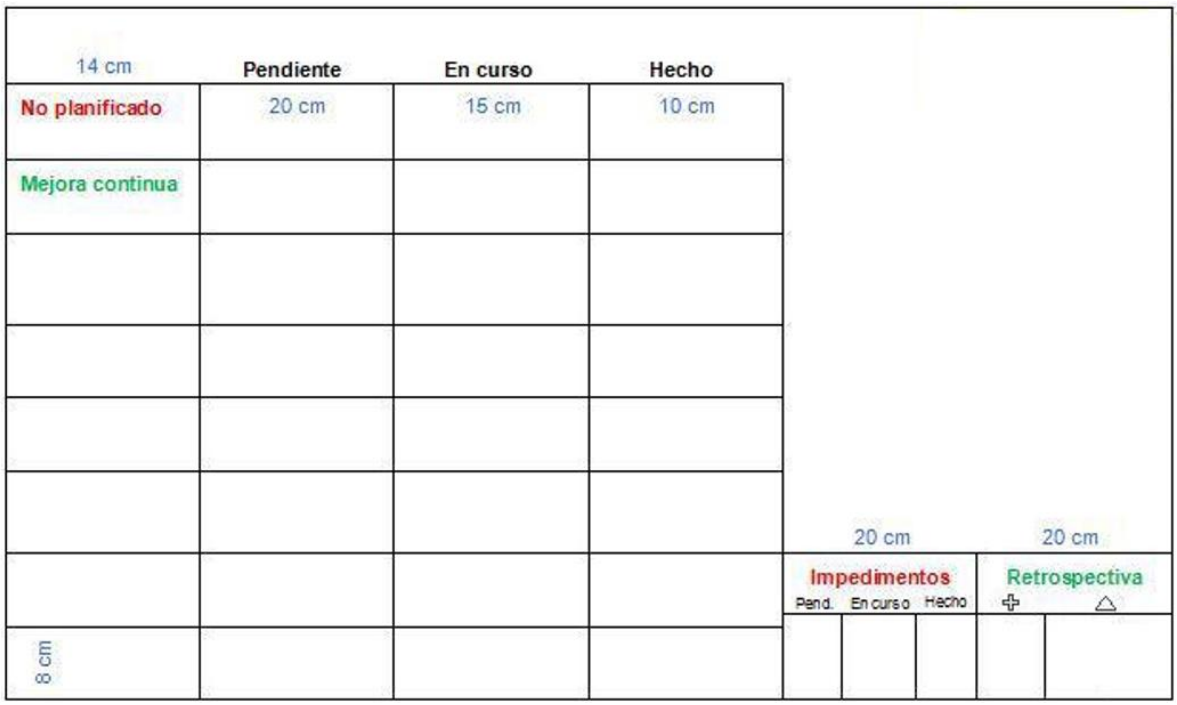

Figura 5 - Scrum Task Board, Fuente: (Albaladejo, 2010)

Como facilitador de proyectos, es el encargado de las siguientes actividades:

1. Seguimiento del Product Backlog, Sprint Backlog y Sprint
2. Eliminación de impedimentos del equipo de desarrollo de software
3. Explicación o detalle de las historias de usuario del Product Backlog
4. Coordinar las 5 fases de cada Sprint
5. Fija objetivos de cada Sprint y es el primer representante del equipo de desarrollo

El otro rol sobresaliente, el Product Owner, como su nombre lo indica es el dueño del producto, es responsable de maximizar el valor del producto y el trabajo del equipo de desarrollo. La forma de hacer esto puede variar ampliamente entre las organizaciones, los equipos de Scrum y las personas (Maeso, 2017). El Product Owner es la persona que sabe exactamente que quieren los usuarios, por lo tanto, es el encargado de elaborar el Product Backlog, y llevar el mismo actualizado, validado y priorizado (Garzas, 2014). Entre sus tareas principales se encuentran:

1. Recoger y tener claros los requerimientos del producto
2. Definir, estructurar y validar las historias de usuario

3. Establecer métricas de aceptación de las historias de usuario
4. Ordenar y priorizar el Product Backlog
5. Asegurarse que el Product Backlog sea claro y validado para todo el equipo de desarrollo
6. Elaborar y definir en base a prioridades el Sprint Backlog

Adicionalmente está el Equipo Scrum que más que un Rol es resto del equipo. Estos roles son más detallados en la sección de Roles de los Equipos de Desarrollo de Software. Existen diversas certificaciones sobre Scrum otorgadas por la Scrum Alliance entre las cuales están Certificado Scrum Master básico y avanzado, Certificado de Product Owner básico y avanzado, Certificado de Scrum Developer básico y profesional, Certificado de Entrenador de equipo básico y empresarial, Certificado de Entrenador Scrum, Certificado de Liderazgo Ágil, entre otros (Scrum Alliance, 2017).

2.1.3.2 Extreme Programming

La Programación Extrema o eXtreme Programming (XP) es una metodología o enfoque de la ingeniería de software creado por Kent Beck en su libro Extreme Programming Explained: Embrace the change en el año de 1999. XP es un enfoque muy popular y destacado en los procesos ágiles del desarrollo de software. Basado en el Manifiesto Ágil, hace énfasis más en la adaptabilidad que en la previsibilidad, por medio de concebir los cambios en los requerimientos iniciales como algo natural, inevitable y hasta deseable del software. Extreme Programming es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, por medio de promover y mejorar el trabajo en equipo con la incentivación del aprendizaje de los desarrolladores además de propiciar un buen clima entre ellos (Beck, 2005)

La Programación Extrema se basa en doce buenas prácticas que le caracterizan, y aseguran que, con su implementación, XP brindará mejores resultados en la producción de software. Estas son:

1. Integración del equipo: En XP todas las personas involucradas en cualquier medida con el proyecto, son parte del equipo. Desde el usuario final, hasta el empleado comercial.

2. Planificación continua: Se deben de realizar las historias de usuario, que son el conjunto de características y requisitos que debe cumplir el sistema, posterior a esto se priorizan y se estima la duración. Esta planificación es constantemente revisada y corregida para que se tenga actualizada.
3. Versiones pequeñas: Al ser una metodología ágil, XP obliga a que el equipo de desarrollo entregue constantemente versiones pequeñas, que deben ser funcionales para el usuario final y no solo código no utilizable. Estas versiones deben ser lo más pequeñas posibles para que sean entregadas en pocas semanas y constantemente (Withrow, 2017).
4. Test del cliente: XP propone que la fase de pruebas de las mini versiones, no deben ser únicamente del lado del equipo de desarrollo, sino que deben ser constantemente revisadas y probadas por el usuario final.
5. Diseño simple: Al usar XP, los desarrolladores deben de tener siempre presente que el diseño, debe ser lo más simple posible ya que es mejor hacer algo simple pero útil, a un diseño complejo que jamás sea usado.
6. Programación en parejas: Con XP se propone una técnica útil e interesante como lo es la programación en parejas. Esta técnica propone que los desarrolladores trabajen en pares, tales que puedan apoyarse mutuamente y también que puedan testear el código que desarrollan uno al otro. Algunos equipos colocan literalmente a dos desarrolladores en el mismo ordenador, otros de manera casi similar hacen la integración de los dos desarrolladores a nivel de relación, aunque cada uno esté en su propio ordenador.
7. Testeo Continuo: XP hace énfasis en el testeo continuo del producto, y propone que las pruebas sean automatizadas como pruebas unitarias, que puedan ser ejecutadas de forma masiva. Para esto los desarrolladores tienen que iniciar no solo con tener en mente cómo se va a comportar el software que van a desarrollar sino también como van a testearlo, para después programar estas pruebas y que las mismas puedan proveer al Tester, un framework de testeo automatizado.
8. Normas y estándares de codificación: XP no dicta una norma y estándar de codificación específico para emplearse, consciente de que esto evoluciona a través

del tiempo, sin embargo, obliga a que todo el código sea desarrollado con el mismo estilo, para que pareciera que una sola persona desarrolló todo el producto.

9. Refactorización y código para todos: La comunicación entre los miembros del equipo es crucial, XP invita a que todo el equipo de desarrollo comparta lo desarrollado para que se pueda hacer reingeniería de los módulos constantemente y que, esto evite la duplicidad de código. Con XP al surgir un problema, cualquier desarrollador debería poder resolverlo. Si todo el equipo conoce el código, se reduce considerablemente el riesgo de dependencia del programador, que surge cuando solo un programador conoce ciertas funcionalidades del sistema y si el mismo llega a retirarse del proyecto, nadie puede continuar con su desarrollo, o esta continuación, consume mucho tiempo.
10. Integración continua: Aunque XP es una metodología que surgió hace poco menos de 20 años, hace importancia sobre la integración continua por medio de la generación de una nueva versión del sistema de manera constante, con esto se obliga a que el equipo integre el código nuevo al proyecto y que el mismo produzca una nueva versión que puede ser probada por el cliente; con esto se evita a que se tengan versiones congeladas por mucho tiempo y que cuando se integre todo el código de golpe, el mismo tenga muchos errores y no se sepa con claridad, cuál es el problema.
11. Metáforas: El software desarrollado en XP requiere el adherirse a un set de estándares para los nombres de variables, clases, métodos, servicios, entidades, etc. que sean desarrolladas, para que no exista una duda de lo que representa cuando alguien más lo vea. Extreme programming hace énfasis en el código autodocumentado y fácil de entender.
12. 40 horas de trabajo semanales: XP propone 40 horas de trabajo efectivas por semana, ya que más horas no garantizan más software correcto, sino por el contrario, el cansancio excesivo puede generar más errores en el desarrollo del equipo. También, 40 horas es una cantidad de trabajo que motiva al equipo a mantener un ritmo constante.

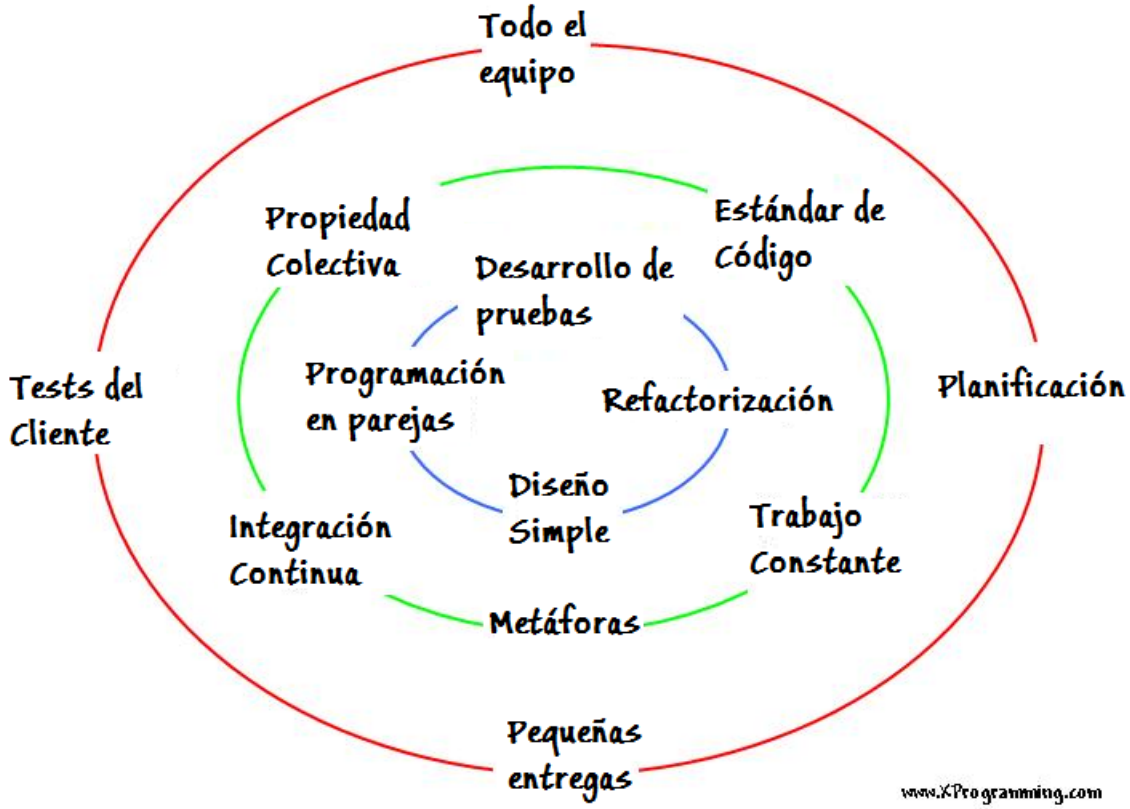

Figura 6 - Prácticas de Extreme Programming, Fuente: (eXtreme Programming, 2014)

Se han descrito dos de las más conocidas metodologías ágiles de desarrollo, que son XP y Scrum. Estas metodologías tienen un enfoque un tanto distinto, mientras XP está enfocado a técnicas explicitas para el desarrollo de software, Scrum tiene un enfoque metodológico para la correcta gestión de proyectos de software, es por esta razón, que muchos equipos de desarrollo optan por implementar Scrum con las mejores prácticas de XP. Es importante hasta este punto que el lector sepa qué, tanto metodologías tradicionales como ágiles, tienen un enfoque únicamente para el equipo de desarrollo, que incluye de esta forma, todo lo que concierne al área de operaciones, seguridad e incluso QA. Esto justifica notablemente la necesidad de una integración completa.

2.1.4 Comparación entre Metodologías Tradicionales y Ágiles

Con el objetivo de realizar un énfasis entre las diferencias entre las metodologías tradicionales y ágiles de desarrollo de software, se presentan a continuación dos comparaciones. La primera comparación va enfocada en las diferencias generales entre ambas metodologías y la segunda tiene un enfoque específico en las etapas de cada una.

Metodologías Tradicionales	Metodologías Ágiles
Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo	Basadas en heurísticas provenientes de prácticas de producción de código
Cierta resistencia a los cambios	Especialmente preparados para cambios durante el proyecto
Impuestas externamente	Impuestas internamente (por el equipo)
Proceso mucho más controlado, con numerosas políticas/normas	Proceso menos controlado, con pocos principios.
El cliente interactúa con el equipo de desarrollo mediante reuniones	El cliente es parte del equipo de desarrollo
Más artefactos	Pocos artefactos
Más roles	Pocos roles
Grupos grandes y posiblemente distribuidos	Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio
La arquitectura del software es esencial y se expresa mediante modelos	Menos énfasis en la arquitectura del software
Existe un contrato prefijado	No existe contrato tradicional o al menos es bastante flexible

Tabla 1 - Diferencias entre Metodologías Tradicionales y Ágil, Fuente (Arevalo, 2011)

Metodologías Tradicionales	Etapa	Metodologías Ágiles
Planificación predictiva y "aislada"	Análisis de requerimientos	Planificación adpatativa:Entregas frecuentes + colaboración del cliente
	Planificación	
Diseño flexible y Extensible + modelos + Documentación exhaustiva	Diseño	Diseño Simple: Documentación Mínima + Focalizado en la comunicación
Desarrollo individual con Roles y responsabilidades estrictas	Codificación	Transferencia de conocimiento: Programación en pares + conocimiento colectivo
Actividades de control]: Orientado a los hitos + Gestión miniproyectos Más artefactos	Pruebas	Liderazgo-Colaboración: empoderamiento +auto-organización
	Puesta en Producción	

Tabla 2 - Diferencias entre etapas y enfoque metodológico Metodologías de Desarrollo, Fuente (Arevalo, 2011)

2.2Equipos de Desarrollo de Software

En las empresas dedicadas al desarrollo de software, así como en los departamentos informáticos empresariales, que desarrollan software, se tiende a cometer un error común, que es asignar a todos los miembros del equipo tareas de cualquier índole. Esto podría ser válido y hasta bueno en algún escenario específico, sin embargo, en la gran mayoría esto puede causar que los miembros del equipo no sean capaces de emitir estimados de calidad, ya que les resulta imposible poder determinar el tiempo y el esfuerzo que van dedican a cada tarea.

Por tal razón, tener roles correctamente definidos y específicos dentro de un equipo de desarrollo de software es muy importante. Sin embargo, como bien indica José León Director de área Business Software Solutions "depende del tamaño de la aplicación, del tiempo y de los recursos disponibles, el equipo podrá ser de un tamaño u otro. Pero hay una serie de puestos que son imprescindibles para que todo funcione adecuadamente. Estos puestos tienen unas responsabilidades bien definidas. En una empresa pequeña, varios roles pueden ser realizados por la misma persona. En organizaciones más grandes, el objetivo debe ser disponer de más células y hacerlas interactuar correctamente." (León, 2017).

2.2.1 Roles y Responsabilidades

La clasificación de Roles y Responsabilidades en los equipos de desarrollo de software tradicionales y ágiles varían en base al autor, del enfoque e inclusive de la metodología, no obstante, la gran mayoría de estos comparten varios en común, que se mencionan a continuación:

2.2.1.1 Cliente

Existen algunos equipos que no consideran al cliente como parte del equipo de desarrollo de software, sin embargo, en las metodologías ágiles se hace mucha importancia en que la incidencia del cliente es transcendental en el desarrollo del software. Por otra parte, es importante entender quién es en realidad "El Cliente", tanto si se desarrolla software para clientes externos, como si se desarrolla para clientes internos, siempre existe un rol de cliente. El cliente, es en esencia, quien pone en marcha el proyecto, paga las cuentas, o define el resultado final. Aun si no se tiene literalmente un "cliente", es bueno entender que aun así

existe un rol “cliente” en el proyecto. Esto puede ayudar a evitar confusiones. “Si hay varias personas con la tarea de describir qué características se necesitan, hay que asegurarse de que exista algún responsable de tomar las decisiones cuando estos requisitos sean contradictorios” (Perez, 2015).

2.2.1.2 Jefe del Proyecto o Project Manager

Es el encargado de la planificación del proyecto, además es el máximo responsable de mantener el proyecto dentro del presupuesto y de la solución del problema; en otras palabras, esta persona es quien debe de resolver cualquier amenaza que ponga en peligro el progreso del proyecto. Muchas de las tareas del jefe o gerente del proyecto tienen que ver con la comunicación, tanto la comunicación al cliente sobre el progreso del proyecto como la comunicación con todos los miembros del equipo. Incluso en los proyectos de desarrollo que no cuentan con un jefe o gerente de proyecto, es conveniente asignar este rol a alguien, para que quede claro quién es responsable de la ejecución del mismo (Perez, 2015). A este puesto también se le conoce por su traducción en inglés Project Manager o Gerente del Proyecto.

2.2.1.3 Analista, Product Owner o Dueño del Producto

En metodologías tradicionales, este rol es conocido como Analista, esta persona es la responsable de entender las necesidades del cliente y asegurarse que la solución que es desarrollada, se ajusta a esas necesidades. En metodologías ágiles como Scrum, este rol es conocido como Dueño del Producto o Product Owner, aunque es importante mencionar que en ellas, el rol de analista en el Product Owner es solo parte de sus responsabilidades (Maeso, 2017). Según 4geeks el Product Owner es el responsable de maximizar el valor del producto, además de ser el responsable del Backlog del trabajo. El Backlog no es más que una lista ordenada de todo el trabajo pendiente. En base al método ágil utilizado, los elementos incluidos en el Backlog se denominan ítems, historias de usuario, unidades de trabajo, etc. El Product Owner además, es el responsable de expresar de manera correcta los requerimientos y vela por que el equipo lo haga, es una sola persona, y es el punto de convergencia de los cambios solicitados por interesados, de igual forma, es una persona altamente comunicativa y transparente (4Geeks, 2016).

2.2.1.4 Arquitecto o Diseñador de Software

Este rol está altamente relacionado con el Analista debido que es el encargado de traducir todos los requerimientos recolectados por éste, en una solución técnica de software. Básicamente el arquitecto de software es el encargado de crear el diseño técnico del software por medio de la creación de su estructura y de la elección de las tecnologías necesarias para que la solución desarrollada cumpla con los requerimientos y el concepto de calidad del cliente (Perez, 2015). El arquitecto también es el encargado de definir las métricas mínimas de rendimiento y calidad que el software debe de tener, además, cuando el proyecto ya va en marcha, el arquitecto también es el encargado de dar un seguimiento con los desarrolladores para velar por la consistencia del diseño inicial. Es de suma importancia recalcar que es el arquitecto del software el encargado de la innovación dentro del producto final.

2.2.1.5 Arquitecto o Diseñador del Sistema

En algunas ocasiones, se incluye como un rol alterno al Arquitecto del Sistema. Este rol está destinado a diseñar la infraestructura de componentes de hardware ideal, capaz de soportar la infraestructura diseñada por el Arquitecto de Software. Como bien indica el Software Architect y Technical Trainer, Mario Pérez "muchas aplicaciones se ejecutan completamente en un único servidor, muchos otros, sin embargo, se ejecutan en grupos de servidores, con servidores dedicados de bases de datos, servidores web y balanceadores de carga; un arquitecto del sistema tiene en cuenta los requisitos de rendimiento y disponibilidad, el número de usuarios / visitantes, etc. y en base a esto, diseña una infraestructura de servidores y una red" (Perez, 2015). Al igual que el arquitecto del software, el arquitecto de sistema es el encargado de la innovación de infraestructura de hardware sobre la cual se ejecutará el software resultante.

2.2.1.6 Desarrollador de Software

En un proyecto de software, normalmente el 20% del código constituye arquitectura y el 80% restante consiste en utilizar esa arquitectura para completar los requerimientos. Los desarrolladores son los encargados de completar ese 80% (León, 2017). Un desarrollador de software es una persona o conjunto de personas, encargada de aspectos que van más allá de codificar software, como lo son la interpretación de requerimientos, realización de pruebas y validación del diseño acordado con el arquitecto de software. Es importante aclarar que existe

cierta diferencia entre un simple programador a un desarrollador de software. Según A. Lores de Velneo un desarrollador es más multifacético ya que debe participar en la definición del producto de software, analizar requerimientos, diseñar y mejorar prototipos, implementar, documentar, testear, dar mantenimiento, etc. y por otra parte un programador está más dedicado a la tarea específica de realizar software por medio de código, tarea que también es realizada por un desarrollador. (Lores, 2008)

Las clases y tipos de desarrolladores varían en dependencia del software que se desarrolla, sin embargo, hay algunos que predominan en la actualidad. Para Juan Lebrijo de Lebrijo.com, existen básicamente tres tipos de desarrolladores, el Backend Developer dedicado al desarrollo de librerías de negocio para máxima reutilización en forma de API's y Web Services, el Frontend Developer, especialista de aplicaciones cliente y el Programador de Base de Datos, también conocido como DBA por sus siglas en el inglés de Data Base Administrator, dedicado a la optimización de los datos gestionados por el software (Lebrijo, 2010). Adicionalmente a estos tres, se debería agregar el Mobile Developer, que está dedicado al desarrollo de aplicaciones nativas para móviles. Es de suma importancia comentar que, en los equipos de desarrollo modernos, los diferentes desarrolladores que conforman el equipo pueden tomar mini roles dentro del mismo equipo, que cuentan con responsabilidades específicas, como por ejemplo el encargado del control de versiones, el integrador de código, el administrador de documentación, el encargado de validar pruebas, el responsable de hot-fixes, entre otros. A los miembros del equipo de desarrolladores se les conoce usualmente como Junior Developer.

2.2.1.7 Jefe de Desarrolladores

En los equipos de desarrollo medianos y grandes, es natural que exista un desarrollador líder, que cuenta con las mismas responsabilidades que el resto de desarrolladores, no obstante, tiene añadidas algunas extras, que son entrenamiento de sus compañeros, entrega de versiones y ayuda inmediata a desarrolladores junior con los problemas que surgen, para que los mismos sean resueltos de la mejor manera. Por tales razones, el jefe de desarrolladores cuenta con un perfil más experimentado y usualmente es el que tiene más influencia en la calidad del código resultante final. En muchos equipos este rol es conocido como Senior Developer.

2.2.1.8 QA Tester (Quality Assurance o Asegurador de Calidad)

El software no podrá evitar los errores en su totalidad, sin embargo, las pruebas son una herramienta indiscutible para reducir los errores al mínimo posible. Por tal razón, es de suma importancia que un equipo de desarrollo de software cuente con el rol de Tester. Esta persona debe poseer un perfil profesional orientado principalmente a la medición de la calidad de los procesos utilizados para crear un producto de calidad. Entre sus funciones se encuentran el diseño y la ejecución integral de pruebas, la medición sistemática, la comparación con estándares y el seguimiento de los procesos, todas ellas encaminadas a la prevención de errores durante el proceso de desarrollo del producto. (iWantic, 2016)

El equipo de desarrolladores, prueba cada componente codificado, no obstante, estas pruebas tomarán únicamente los aspectos con los que el mismo componente fue escrito, por tal razón es necesario que una persona externa al desarrollador audite el software y se asegure que cumpla con los estándares impuestos por el arquitecto de software.

2.2.1.9 Encargado de Metodología o Master

Este rol puede ser considerado independiente o parte del Project Manager, esto va a depender de la importancia que el equipo le dé al complimiento estricto de la metodología a usar, así como al tamaño propio del equipo. La tarea principal de este rol es velar por el cumplimiento de la metodología y coordinar todas las tareas necesarias para el cumplimiento de este fin. Tiene una comunicación estrecha con el Project Manager y el Jefe de Desarrolladores ya que es el principal conocedor de los avances del proyecto en general y el que dará la información importante y necesaria al Project Manager quien informará al dueño de la empresa en el caso de ser una organización dedicada al software o a la alta gerencia en el caso de ser un departamento de tecnología de una empresa.

2.2.1.10 Equipo de Soporte

Estos miembros son los principales que integran el departamento de sistemas o de operaciones. Básicamente son los encargados de mantener en correcto funcionamiento el sistema cuando el mismo se encuentra en ambiente de producción. Entre sus principales tareas es implementar el software que entregan los desarrolladores, previamente validado por el tester, implementar nuevas funcionalidades de software existente, así como reparaciones en caliente del mismo, por otra parte, también son el primer y principal punto de

retroalimentación por parte de los usuarios cuando lo sistemas ya están implementados. Este rol es el principal punto de problemas cuando no se tiene con correcta sinergia del equipo, ya que puede provocar serios retrasos en la entrega de sistemas, correcciones y nuevas funcionalidades. Es por tal razón que las metodologías ágiles tomaron protagonismo en las últimas décadas, ya que, con la nueva manera de entregar software de manera masiva y mundial, la correcta entrega de software y servicios de este equipo es indispensable.

2.2.1.11 Comercial

Este rol no existe en todos los equipos de desarrollo ya que va a depender directamente de la naturaleza del negocio. Si es un departamento informático de una organización, el puesto es cubierto por el Project Manager, sin embargo, si la organización se dedica a vender software, el puesto es totalmente independiente. Entre las tareas principales es ser el primer contacto con el cliente u usuarios finales. Es quien traslada estas nociones iniciales al analista quien continuará el trabajo a través del ciclo de vida del software. Es importante que la personalidad de este rol sea orientada a la entrega de servicios ya que será el encargado de vender el proyecto o producto a los clientes, y de mostrar las bondades y beneficios de implementar el mismo. Para el caso de un departamento informático, las tareas no varían mucho ya que, en lugar de vender un producto, se convencerá a la alta gerencia la aprobación o viabilidad de proyectos.

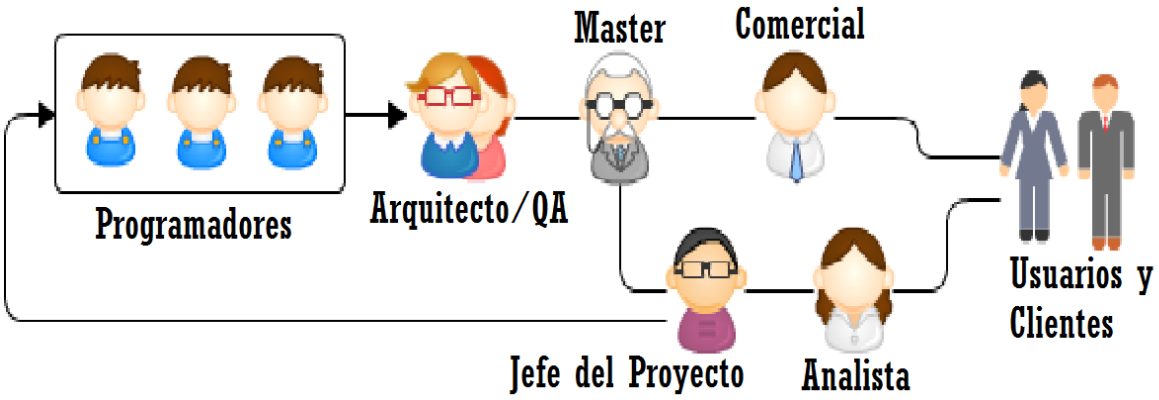

Figura 7 - Flujo básico de un equipo de desarrollo, Fuente: (Lebrijo, 2010) traducción personal

Los equipos de desarrollo de software varían de organización en organización y de metodología en metodología. Es usual que en los equipos con metodologías tradicionales

exista una marcada y estricta asignación de roles y responsabilidades, sin embargo, en equipos con metodologías ágiles es más probable encontrar equipos multifacéticos y multidisciplinarios que puedan realizar diferentes tareas en determinados escenarios. No obstante, es importante tener en consideración que en la gran mayoría de equipos existen roles de desarrollo, roles de sistemas y roles de QA. En DevOps se proponen diferentes roles, sin embargo, existen aspectos que ayudarán a que la comunicación y sinergia permitan percibir un equipo de desarrollo integral y totalmente unificado que trabaje bajo las mismas metas y objetivos que sean entregar software de calidad y prestar servicios útiles.

2.2.2 Retos y Problemáticas

Los equipos de desarrollo de software deben diariamente de enfrentar diversos retos y problemáticas ante los proyectos que desarrollan. Esto va de la mano con el ambiente cambiante que conlleva la tecnología y por ende el desarrollo de software. A diario surgen nuevas plataformas, metodologías, filosofías, lenguajes de desarrollo, frameworks, herramentas, entre otros., que obligan a los equipos de desarrollo para que se mantengan con un constante esfuerzo por renovarse y así mantenerse a la vanguardia en la tecnología.

2.2.2.1 Retos

Los retos de los equipos de desarrollo de software van enfocados hacia todos los aspectos que se enfrentan, y les obligan a renovarse constantemente, mismos que convierten en obsoleto al equipo de desarrollo si no se les maneja con precaución. Según la compañía Wellspring la gestión de innovación en las empresas ha cambiado considerablemente en los últimos años y es necesario que se dedique un porcentaje importante del esfuerzo para este fin. (Wellspring, 2017)

IBM Rational en su reunión técnica online de 2013 para clientes y socios, discutió sobre las problemáticas, tendencias, soluciones y novedades que giran en torno al desarrollo de software en la reciente era; en esta reunión destacaron los siguientes retos para los equipos de desarrollo:

1. Contemplar y aprovechar las nuevas posibilidades de ahorro de costes para el desarrollo y puesta en marcha de aplicaciones en infraestructura Cloud.

2. Gestión de los equipos de desarrollo hacia los equipos de negocio, para crear integración y que los mismos no trabajen de forma independiente.
3. Necesidad de que los equipos de desarrollo sean flexibles y se adapten a las necesidades cambiantes del negocio. Deben contar con agilidad y flexibilidad.
4. La metodología DevOps como marco esencial para superar los retos en agilidad y reducción de recursos que sufren los equipos de desarrollo. Tema esencial del presente trabajo de graduación.
5. Las nuevas exigencias en el desarrollo de software que imprime la creciente "movilización" de los negocios.
6. El amplio abanico de posibilidades que ofrecen las nuevas capacidades de análisis masivo de datos o Big Data.

Los equipos de Desarrollo de Software se convierten hoy en pilar esencial en los procesos de Innovación de las empresas. La Innovación en los productos y servicios que se ofrecen a los clientes se convierte a su vez en elemento imprescindible para competir hoy por hoy. Innovación entendida como la capacidad de poner en el mercado rápidamente nuevos productos que se ajusten a las necesidades cambiantes del negocio. (Alguacil, 2013).

Johan Gaona de Ikonosoft resalta el reto del "efecto novedad" en el desarrollo de software. El menciona que en la entrega de software nuevo, el usuario percibe que todo es nuevo y esto provoca una motivación en él, haciéndose semejante a cuando alguien inicia en un gimnasio, sin embargo, conforme el tiempo pasa, este efecto se pierde, por lo cual un reto muy grande es que los equipos de desarrollo mantengan vivo el efecto novedad, por medio de la entrega constante de nuevas versiones que sean llamativas y funcionales para los usuarios finales y que con esto, el software se mantenga vivo por mucho más tiempo, esto de la mano al gran problema actual que representan las SAAS (Software como Servicio), ya que provoca que el software sea algo globalizado (Gaona, 2017).

2.2.2.2 Problemáticas

Existen diversas problemáticas a las que se enfrentan los equipos de desarrollo de software. El principal problema es la Comunicación, debido a que muchas veces existe fragmentación en el equipo, que impide el correcto flujo de progreso desde los requerimientos hasta la entrega, provocando falta de visibilidad sobre el estatus actual del proyecto.

No obstante, el libro Rapid Development, Steve McConnell introdujo el concepto Errores Clásicos de Desarrollo de Software, que son aquellos que se repiten continuamente en el desarrollo de software y por esa razón, deben ser previsibles y siempre deberían gestionarse. (McConnell, 1996)

En su libro Steve expuso 36 errores clásicos, que fueron actualizados en el 2007 a 42 después de una encuesta a aproximadamente 500 profesionales en el campo, con el objetivo de validar la frecuencia y gravedad con el que cada uno de estos errores suceden (McConnell, 2007). La Tabla 1 muestra los 42 errores de Steve clasificados en 4 distintos enfoques.

Errores relacionados con las Personas	**Errores relacionados con el Proceso**	**Errores relacionados con el Producto**	**Errores relacionados con la Tecnología**
Motivación inadecuada	Planificaciones excesivamente optimistas	Añadir requisitos no solicitados por el cliente	Falta de automatización del control de Código Fuente
Personal Inadecuado	Gestión de riesgos insuficiente	Corrupción del alcance	Cambiar herramientas en mitad del proyecto
Inadecuada gestión de los empleados problemáticos	Planificar con la idea en que se recuperará el retraso acumulado más adelante	Los desarrolladores añaden características no solicitadas	Sobreestimación del ahorro por la reutilización de nuevas herramientas o métodos
Heroicidades	Planificación insuficiente	Negociaciones de tira y afloja	Soluciones mágicas (silver-bullet)
Incorporar más personas a un proyecto retrasado	Abandono de la planificación bajo presión	Desarrollo enfocado a la investigación	*Confiar en el mapa más que en el terreno**
Oficinas abarrotadas y ruidosas	Desperdiciar el tiempo antes del kickoff del proyecto	*Asumir que el desarrollo global tiene insignificante impacto en el esfuerzo total**	
Roces entre los desarrolladores y el cliente	Acortar las tareas de alto nivel	*Visión del proyecto poco clara**	
Expectativas poco realistas	Fallo de los proveedores		
Falta de un patrocinador del proyecto eficaz	Reducción de las tareas de control de la calidad		

Errores relacionados con las Personas	Errores relacionados con el Proceso	Errores relacionados con el Producto	Errores relacionados con la Tecnología
Falta de compromiso por parte de los interesados	*Dejar que el equipo se oscurezca**		
Falta de participación del usuario	Omitir tareas de estimación necesarias		
Anteponer decisiones políticas sobre el objetivo del proyecto	Preparación del producto para su liberación demasiado pronto o demasiadas veces		
Pensamiento ilusorio	Diseño inadecuado		
*Excesiva multitarea**	Programar a cualquier precio		
*Outsourcing para reducir costes**	*Confusión de estimaciones con objetivos**		

*Tabla 3 - 42 errores clásicos del desarrollo de software de McConnell (*actualizados en 2007), Fuente: Personal*

2.3DevOps

2.3.1 Historia

En 2007, al ser consultado sobre la migración de un centro de datos para el ministerio de finanzas del gobierno de Bélgica, Patrick Debois, un ingeniero de software y administrador de sistemas se frustra por los conflictos entre los desarrolladores y los administradores del sistema, ya que el proyecto fue un fracaso y no pudo salir adelante, él inicia a ponderar las soluciones (Fredic, 2014).

Andrew Clay Shafer, creador de Puppet Labs, spin-off de VMware, EMC2 y liberador de la base de datos NoSQL Redis, debía dar una charla en Agile Conference 2008 en Toronto, sin embargo, a la misma solo acudió una persona, Patrick Debois, así que Andrew decidió no dar la charla. Posterior a esto fue asaltado en los pasillos por Patrick, quien había ido a dicha conferencia, a contar su caso de no-éxito “Infraestructura Agile y operaciones: ¿cómo de infra-ágil eres?”, y ante su insistencia, ambos comenzaron a discutir cómo se podría llevar el agilismo al mundo de la infraestructura y la administración de sistemas. Animados por este intercambio de ideas, acordaron crear un grupo en Google para abrir la discusión a la comunidad, el Agile System Administrators Group.

Un año después, el 23 de junio de 2009, O'Reilly organizó el evento Velocity'09, transmitido en streaming. Tras la exposición de "10+ Deploys a Day: Dev and Ops Cooperation at Flickr" por John Allspaw y Paul Hammond, Patrick se lamentaba en Twitter de no haber podido asistir en persona. Entonces Paul Nasrat, responsable del CMS del periódico británico The Guardian y desde 2010 en Google, respondió a su tuit proponiéndole que organizara un evento similar en Europa. Patrick lo tomó literal y sólo cuatro meses después estaba convocado su primer DevOps Day (Ruiz, 2015).

El primer US DevOps Days es organizado en 2010, con la ayuda de Willis y de otros proponentes de DevOps como Damon Edwards y el mismo Andrew Clay Shafer. Los eventos pronto se convierten en una serie global regular de conferencias organizadas por la comunidad y una fuerza importante que impulsará la comunidad DevOps. La repercusión fue enorme, y el hashtag creado para la ocasión, #DevOps, triunfó en las redes sociales de forma viral, que dio nombre a todo un movimiento. En 2011 Cameron Haight, de Gartner, entre otros, predice que para 2015, el 20 por ciento de las empresas globales del 2000 abrazarán DevOps. Otros analistas importantes que surgen en esta época incluyen a Jay Lyman de 451 Research. Fue en el año de 2012 en el que como un jardín en el desierto después de una lluvia, varios DevOps Days aparecen de repente en todo el mundo, desde Bangalore hasta Boston. Se convierten en eventos imprescindibles para conocer las últimas ideas inteligentes e innovadoras en el mundo de DevOps.

En 2013 una voz importante en el universo de DevOps pertenece a Mike Loukides, vicepresidente de estrategia de contenido de O'Reilly Media. Él, junto con Debois, edita algunos de los textos de DevOps más fundamentales. En 2014 el mundo tecnológico en constante evolución presenta nuevos desafíos y oportunidades para el concepto de DevOps. La explosión de nuevos dispositivos, aplicaciones, contenido y transacciones en el entorno móvil brinda un nuevo enfoque tanto a las aplicaciones móviles como a la computación en la nube (Rapaport, 2014).

2.3.2 Definición

¿Qué es DevOps? Esa es una pregunta que se realizaron desde hace algunos años a la fecha, cientos de equipos de desarrollo de software que deseaban mejorar la manera de trabajar, conscientes de los retos y problemáticas. Según IBM, es un enfoque que promueve la

colaboración entre líneas de negocio, desarrollo y operaciones de TI. Es una funcionalidad empresarial que habilita la entrega continua, el despliegue continuo y la supervisión continua de aplicaciones. Reduce el tiempo necesario para tratar el feedback de los clientes. El desarrollo y las operaciones, e incluso las pruebas, que antes se organizaban en silos, esta cultura las reúne para mejorar la agilidad (IBM, 2017).

DevOps es una cultura o filosofía que ayuda a quebrantar las duras barreras entre los distintos departamentos de sistemas, en especial desarrollo y operaciones, por medio de proveer un ambiente de empatía entre las distintas partes, así como de ofrecer una visión integral a todos los involucrados. Engloba algunas tareas relativamente nuevas y otras no tanto, por ejemplo, ayuda a promover una infraestructura como código, pero también toca temas como la integración continua.

DevOps como tal no es una persona, es más una cultura y la utilización de una serie de principios, herramientas y prácticas para lograr ese objetivo de romper barreras entre departamentos y crear una cultura de empresa, y de equipo a distintos niveles. Erróneamente se considera "un DevOps" a alguien de la parte de sistemas que esté más interesado en temas como virtualización, orquestación, monitorización, integración continua y demás, sin embargo, como tal no es un rol.(Garcia, 2014).

Dawn Foster del Sitio Web Linux.com le realizó una entrevista a Patrick Debois, creador del movimiento y también conocido como el Padre del DevOps, en la que le pregunta ¿por qué hay tantas personas interesadas en DevOps? A lo que Patrick contesta que "Algunas organizaciones, por supuesto, lo harán solo porque es una palabra de moda. Espero que la mayoría de las organizaciones lo implementen para ser más competitivos en sus negocios, así como eficientes. En los primeros días, esto iba a estar por delante del resto, pero en la actualidad es una necesidad para sobrevivir, entre otros, implementarlo. El aspecto cultural de la colaboración les da a todos, un asiento igual en la mesa; tanto el desarrollador como el operador son importantes. Y esto significa que hay más respeto mutuo. Esto también crea más empatía por los problemas de cada uno, lo que resulta en un entorno más agradable para trabajar." (Foster, 2016). DevOps promueve la colaboración de un equipo que tira del mismo carro hacia la misma dirección, unifica las partes y trata a todos los interesados por igual.

2.3.3 Objetivos de DevOps

Existen múltiples objetivos que varían en base a la organización que implementa. Según explica Manuel Barroso Camacho, entre los principales objetivos de DevOps están:

a. La reducción del "time-to-market" por la mejora en la eficiencia a través de la realización de actividades comunes (o en la frontera) entre Desarrollo, QA y Operaciones.
b. La automatización de tareas que agilicen la provisión de servicios (infraestructura, cloud, aplicaciones, entre otros).
c. La mejora de los costes operacionales mediante el decremento de los errores humanos que hoy interactúan directamente con infraestructura y aplicaciones o la automatización y sistematización de tareas rutinarias o del día a día.
d. La disminución del tiempo de desarrollo y puesta en ejecución, así como la reparación de errores y entrega de mejoras.
e. El incremento de la satisfacción del usuario y del negocio (Camacho, 2016).

2.3.4 Áreas que intervienen

Como su nombre lo indica, Develpment & Operations nació de la necesidad de romper las barreras existentes entre los departamentos de Desarrollo y Operaciones, no obstante, afecta a toda la institución que lo implementa. Según Patrick Debois, existe usualmente dos implementaciones de DevOps:

a. DevOps: basado en la colaboración y optimización en toda la organización. Incluso más allá de TI (recursos humanos, finanzas, entre otros) y fronteras de la compañía (proveedores).
b. DevOps Lite: basado en la colaboración y optimización únicamente entre Desarrollo y Operaciones.

Si el objetivo es aprovechar completamente los beneficios que se obtienen por medio de su uso, la implementación a elegir debe ser DevOps completa, ya que el uso de esta cultura interviene en todas las áreas principalmente de IT, pero también en el resto de colaboradores de la institución tanto internos como externos. No obstante, es importante mencionar que las tres áreas donde más intervención existe, y dónde inicia el cambio es Desarrollo, Operaciones y Control de Calidad (QA por sus siglas en inglés Quality Assurance).

Figura 8 - Principales Áreas de Intervención de DevOps, Fuente: (Alba, 2016)

2.3.5 Ventajas de DevOps

Existen diversas ventajas que aportan la implementación de DevOps, bajo diferentes enfoques. Según AT Sistemas, las ventajas, se pueden abordar desde tres enfoques que son:

Relacionadas con la Disponibilidad del entorno:

1. Rapidez y flexibilidad en la disponibilidad de entornos
2. Menores tiempos de puesta en producción de las aplicaciones
3. Mayor homogeneidad entre los entornos de desarrollo y producción
4. Sin realización de pruebas en producción

Vinculadas con la gestión de equipos de desarrollo:

5. Más tiempo para crear, menos para corregir
6. Equipos liberados de la resolución de incidencias
7. Mejor coordinación en la asignación de responsabilidades

Orientadas a la entrega y escalado para evitar:

8. Respuesta tardía a picos de negocio e incapacidad de escalar
9. Repetición de incidencias en producción
10. Falta de control en la calidad del software de proveedores externos
11. Dificultades con la disponibilidad y heterogeneidad de entornos

2.3.6 Beneficios de DevOps

DevOps también aporta una serie de beneficios que ayudan a que las organizaciones puedan tener una mejora sensible en su rendimiento. Los principales beneficios son:

1. Frecuencia de Despliegue 200% mayor
2. Paso a Producción 2500% más rápido tras la entrega de código
3. Tiempo medio de puesta en producción menor en 1 hora
4. Tiempo de recuperación 24% más rápido, en promedio 1 hora
5. 3% menos fallos por cambios en el código
6. 22% menos tiempo en resolución de incidencias
7. 29% más tiempo en desarrollo de nuevas funcionalidades (AT Sistemas, 2013)

2.3.7 Desafíos de DevOps

Todo lo que puede producir una serie de beneficios y ventajas, conllevan una serie de desafíos y dificultades que deben ser superados. En el caso de DevOps, según Lucero del Alba profesional en el desarrollo de software, Data Science y DevOps de Site Point los principales son:

1. El primer desafío es a nivel Organizacional, ya que a medida que se eliminan las limitaciones a los departamentos de desarrollo e implementación, los programadores y administradores de sistemas tienen más independencia, y por lo tanto las personas involucradas deben adoptar una mentalidad diferente, y se deben establecer los mecanismos apropiados para un ciclo de retroalimentación de operaciones a desarrolladores, como los rastreadores de problemas, juntas de discusión, entre otros. Este desafío se acentúa bastante cuando el equipo posee una metodología tradicional ya que los roles son restrictivos, por lo que el primer paso para superar este desafío es la implementación de una metodología ágil como lo es Scrum.

2. El segundo desafío clave es con respecto a los procesos. No se desea que los desarrolladores y los administradores de sistemas usen su tiempo para probar individualmente los cambios o nuevas funcionalidades después de ser trasladados. Por lo que se deberá automatizar los procesos de prueba, de modo que se pueda permitir que diferentes equipos realicen cambios y puedan verificar rápidamente que las cosas sigan en su lugar, y revertir esos cambios en caso de que surjan problemas. Además, en las metodologías tradicionales los procesos son secuenciales y usualmente largos y costosos, por lo que se debe hacer una reingeniería de automatización para que los procesos acepten de mejor manera los cambios y se puedan adaptar a un entorno en constante evolución.
3. Finalmente, hay un desafío tecnológico. Una vez que la organización ha revisado cuidadosamente los procesos de principio a fin, es posible que necesite adoptar o crear una tecnología que aborde el tipo de automatización y circuito de retroalimentación que mejor se adapte a sus procesos y organización, además de herramientas monitoreo y orquestación de servicios (Alba, 2016).

2.4 Cadena de Herramientas DevOps

En DevOps existen 7 etapas que ayudan a cubrir por completo el ciclo de vida del desarrollo de software, y para cada una de estas etapas existen herramientas que permiten lograr de una manera automatizada el cumplimiento correcto de las mismas. A estas herramientas se les conoce como Cadena de Herramientas DevOps o en inglés DevOps Toolchain. La cadena de herramientas surge de la necesidad de entregar aplicaciones de manera más ágil y rápida, pero pueden incluir docenas de herramientas no colaborativas, lo que hace que la tarea de automatización sea compleja y ardua *(Williams & Murphy, 2016).*

Como esta cultura es un conjunto de prácticas que enfatiza la colaboración y comunicación de desarrolladores de software y otros profesionales de la tecnología de la información, mientras automatiza el proceso de entrega de software y cambios de infraestructura, su implementación puede incluir la definición de la serie de herramientas utilizadas en varias etapas del ciclo de vida; Debido a que DevOps es un cambio cultural y una colaboración entre el desarrollo y las operaciones, no existe un solo producto que pueda considerarse una

herramienta única de DevOps. En su lugar, se utiliza una colección de herramientas, potencialmente de una variedad de proveedores, en una o más etapas del ciclo de vida.

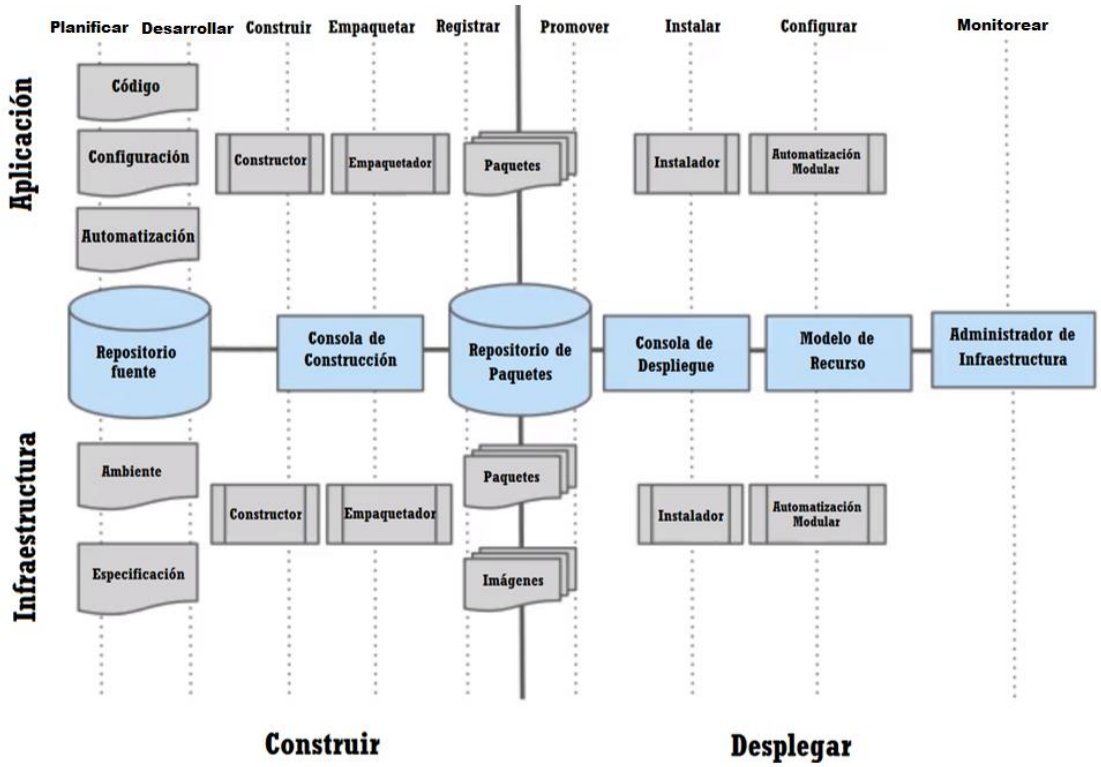

Figura 9 - Ciclo de Cadena de Herramientas DevOps, Fuente: (Edwards, 2012)

2.4.1 Herramientas para Planificación

El plan está compuesto por la definición y la planificación con el enfoque hacia el valor comercial y los requisitos de la aplicación. El objetivo de esta etapa es realizar la planificación de la automatización entre desarrollo y operaciones, además, de la correcta colaboración y sinergia entre ambas partes. Las actividades que son cubiertas en esta etapa son:

1. Determinar métricas de producción y de negocios
2. Actualizar métricas de lanzamiento
3. Administrar y gestionar correctamente Product Backlog
4. Planificar lanzamientos, calendarios y casos comerciales
5. Estudiar y elegir Herramientas de cada etapa
6. Establecer de planes de comunicación, colaboración y sinergia

Una combinación del personal de TI participará en estas actividades: propietarios de aplicaciones comerciales, desarrollo de software, arquitectos de software, administración de versiones continuas, oficiales de seguridad y la organización responsable de administrar la producción de infraestructura de TI. Al ir combinado la mayoría de veces con Scrum, es normal que se utilice software que ayude a implementar los artefactos de Scrum.

Herramientas: En esta etapa se consideran plataformas de definición de requisitos y creación de prototipos de software que puedan ser utilizadas para crear simulaciones de software comercial, lo cual permite a los analistas de negocios, gerentes de producto, gerentes de proyecto y profesionales en usabilidad, montar simulaciones totalmente funcionales de soluciones de software que imitan el aspecto exacto, sentir y comportamiento del producto final propuesto. En esta etapa también se incluye el software para la gestión de la colaboración, en este sentido se necesita software que proporciona seguimiento de errores, seguimiento de problemas y funciones de gestión de proyectos. Herramientas que ofrecen gestión de colaboración y mensajería son Jira, iRise, HipChat, Trello, Team Foundation Server, Rally, StackStorm, ServiceNow, Slack, Deveo, Pivotal Tracker, Flowdock y Wrike.

Atributo	Trello	Jira Software	Slack
Plataformas	Web, Android app, iPhone app	Web, Android app, iPhone app	Web, Android app, iPhone app
Consumidores	Freelancers, pequeñas, medianas y grandes empresas	Pequeñas, medianas y grandes empresas	Freelancers, pequeñas, medianas y grandes empresas
Soporte	Documentación escrita y video tutoriales	Documentación escrita, soporte telefónico y online y video tutoriales	Soporte online y documentación
Precio	Empieza gratis, no es necesario tener tarjeta de crédito, cuenta gratuita y basado en suscripción	Empieza en $10 al mes, hay un trial disponible, no requiere tarjeta de crédito y basado en suscripción	Empieza en $6.67 al mes, hay un trial disponible, no requiere tarjeta de crédito, cuenta gratuita disponible y basado en suscripción
Facilidad de uso	4.5 de 5	4 de 5	4.5 de 5
Seguridad	Encriptación REST, HTTPS en todas las páginas, multi-factor para autenticación	CSA, CCM, PCI y DSS. Encriptación REST, HTTPS para todas las páginas	No.

Tabla 4 - Trello vs Jira Software vs Slack (actualizado en 2018), Fuente: (GetApp, 2018)

2.4.2 Herramientas para Codificación, Construcción y Configuración

Esta etapa se compone de la codificación, construcción y configuración del proceso de desarrollo de software. El objetivo de esta etapa es la correcta configuración del ambiente de desarrollo de software, para que la codificación del mismo, se realice de manera optimizada. Las actividades que son cubiertas en esta etapa son:

1. Diseño del software y la configuración
2. Codificación, incluida la calidad del código y el rendimiento
3. Rendimiento de compilación y desarrollo de software
4. Release candidate
5. Gestión automatizada de la construcción
6. Gestión automatizada de dependencias
7. Control de Versiones del Release

Herramientas: Las herramientas usadas en esta etapa desde el punto de vista de SCM (Software Configuration Managment por sus siglas en inglés, Gestión de Configuración del Software traducido al español), son: Git, Subversion, Bitbucket, Github, Team Foundation Server, GitLab, Crucible, Fisheye, Stash, Deveo, Trac, Kallithea, ISPW, entre otras. Además, para la automatización de la construcción del software están Gradle, Maven, Gulp, Grunt, Broccoli, ANT, SBT, Make, CMake, MSBuild, Rake, Packer, Buildr, NAnt, Jam, Visual Build, Meister, BuildMaster, LuntBuild, QuickBuild y FinalBuilder. También en esta etapa se incluyen las herramientas para la gestión de repositorios de dependencias, entre las que encontramos como principales NPM, Maven, Docker Hub, NuGet, Nexus, Artifactory, Archiva, Pulp y MyGet.

2.4.3 Herramientas para Verificación y Testeo

La etapa de Verificación o Testeo está directamente asociada con garantizar la calidad de la versión del software; esta etapa incluye actividades diseñadas para asegurar que la calidad del código se mantenga y la más alta calidad se implemente en la producción. El objetivo principal es la optimización de las pruebas desde diferentes enfoques para que el software entregado en desarrollo sea automáticamente probado y notificado a los interesados ya sea que la versión entregada por software apruebe o fracase las pruebas.

Atributo	Github	GitLab	Bitbucket
Uso	Cloud gratuito y de paga. servidor local en una máquina ya configurada VMware	Cloud gratuito y de paga, servidor local en cualquier tipo de servidor	Cloud gratuito y de paga, servidor local únicamente como Stash, anterior SCM de Attlassian
Consumidores	Freelancers, pequeñas, medianas y grandes empresas	Freelancers, pequeñas, medianas y grandes empresas	Freelancers, pequeñas, medianas y grandes empresas
Soporte	Email y por medio de Foros	Foros	Email y por medio de Foros
Precio	Cuenta gratuita con código público. Cuentas Premium con repositorios privados, $7 por mes para Developers, $9 para equipos, $21 para negocios	Totalmente gratuito para repositorios privados y públicos. Cuenta con una versión Premium con algunas características no imprescindibles, $39 por usuario por año para starter, $199 premium y cotización para ultimate	Gratuito con repositorios privados hasta 5 usuarios. Posterior a esto es pagado, $2 por usuario por mes $5 premium
Cantidad de Repositorios Privados y Públicos	Ilimitados	Ilimitados	Ilimitados
Facilidad de uso y experiencia UI	4.5 de 5	4.5 de 5	4 de 5

Tabla 5 - Github vs GitLab vs Bitbucket (actualizado en 2018), Fuente Personal

Las actividades principales asociadas a esta etapa son:

1. Test de aceptación
2. Pruebas de regresión
3. Análisis de seguridad, rendimiento y vulnerabilidad
4. Prueba de configuración

Herramientas: Los vendedores notables y las soluciones para verificar actividades relacionadas generalmente caen bajo cuatro categorías principales: Automatización de prueba (ThoughtWorks, IBM, HP), Análisis estático (Parasoft, Microsoft, SonarSource), Laboratorio de pruebas (Skytap, Microsoft, Delphix) y Seguridad (HP, IBM, Trustwave, FlawCheck). Las herramientas más notables son: JUnit, FitNesse, Protactor, Cucumber,

Selenium, JMeter, Karma, Jasmine, TestNG, QUnit, SpecFlow, Bosun, Gauntl, Sahi, Codacy, Pytest, SonarQube, Visual Studio, Gatling, TestComplete, Mocha, Parasoft Environment Manager, Parasoft API Test, Parasoft Service Virtualization, Parasoft Development Testing Plataform, HP UFT, Tosta, Locust, entre otros. La elección de las herramientas a usar, dependerá directamente de las tecnologías que se usan para el desarrollo.

2.4.4 Herramientas para Empaquetado

El empaquetado se refiere a las actividades involucradas una vez que la versión está lista para el despliegue, a menudo también denominada etapa o Preproducción/preprobado. El Empaquetado involucra también la construcción de contenedores, misma que permite la creación de imágenes virtualizadas a nivel de sistema operativo para la clonación exacta de ambientes de aplicación. El objetivo principal de esta etapa es el trasladar el software desde las máquinas de los desarrolladores, hasta los servidores de preproducción/certificación, de manera que se pueda garantizar que el funcionamiento se mantiene desde un ambiente hacía otro. Es importante que este proceso se realice de manera automatizada. Esto a menudo incluye tareas y actividades tales como:

1. Aprobación/preaprobaciones
2. Configuración del paquete
3. Lanzamientos disparados
4. Lanzamiento de puesta en escena

Herramientas: En esta etapa se asocian las herramientas de Integración Continua como Jenkins, Hudson, Bamboo, Visual Studio, Team Foundation Server, Solano CI, BuildMaster, Continuum, CruiseControl, QuickBuild, Buildbot, Continua CI, Shippable, TeamCity, Snap CI, TravisCI, Codeship, CircleCI y UrbanCode Build, Además de las tradicionales herramientas de integración continua, también se mencionan las herramientas para la administración del ciclo de vida de bases de datos, como DBmaestro, Datical, Liquidbase, Flyway, Redgate, Idera y Delphix.

También en esta etapa se incluyen las herramientas de construcción y orquestación de Contenedores como lo son Docker, Rocket, Rancher, Containership, Kubernetes, Packer,

Messos, Swarm, Linux Containers, CloudSlang, Solaris Containers, Fleet, Marathon, OpezVZ, Nomad y Tectonic.

2.4.5 Herramientas para Despliegue y Liberación

Las actividades relacionadas con el despliegue y liberación incluyen programación, orquestación, aprovisionamiento e implementación de software en producción y entornos específicos. El objetivo de esta etapa es la automatización en desplegar y liberar el software a ambiente de producción, y como dictan las tendencias actuales, a su correcta colocación en la nube. Por otra parte, también se tiene como objetivo garantizar la seguridad de la aplicación desde todas las posibles entradas. Las actividades de lanzamiento específicas incluyen:

1. Coordinación de lanzamiento
2. Implementación y promoción de aplicaciones
3. Retornos y recuperación
4. Lanzamientos programados/temporizados
5. Configuración de la Seguridad en Producción

Herramientas: Existen varios tipos de herramientas en esta etapa, las primeras son las encargadas de la implementación y despliegue de aplicaciones, entre ellas están Otto, Deployment Management, SmartFrog, Capistrano, Juju, Rundeck, RapidDeploy, CodeDeploy, Octupus Deploy, CA Nolio, XL Deploy, ElasticBox, Deploybot, UrbanCode Deploy, entre otros. El segundo tipo de herramientas son las encargadas en la liberación en las que podemos encontrar XL Release, UrbanCode Release, BMC Release, CA Release Automatization, Automatic, Plutora Release, Serena Release, entre otros. El tercer tipo de herramientas son referentes a la Computación en la Nube IaaS (Infraestructura como Servicio) y PaaS (Plataforma como Servicio) tales como Amazon Web Services, Microsoft Azure, Google Cloud, Rackspace, OpenStack, Heroku, OpenShift, Cloud Foundry, Jelastic, vCloud Air, Deis, Apprenda, entre otros, en combinación con orquestadores de contenedores como lo son Docker Swarm, Kubernetes, Fleet, etc. Adicionalmente en esta etapa se involucran herramientas concernientes a la seguridad y loggin, esta etapa se encuentra ya en los ambientes de producción y puesta en marcha final. Para ese caso tenemos diferentes herramientas centradas en la seguridad como lo son Snort, Tripwire, Fortify, Gauntlt, CyberArk, SecurityAssist, Vault, Veracode, entre otros. Por otra parte, utilidades para el

testeo de Login y de las diferentes entradas a sistemas puestos en producción, tenemos herramientas como Sentry, Stackify, Splunk, Logstash, Sumo Logic, Loggly, Graylog, Logentries, Sensu, entre otros.

2.4.6 Herramientas para Configuración

Las actividades de configuración se incluyen en el lado de la operación de DevOps. Una vez que se implementa el software, es posible que se requieran actividades adicionales de configuración y configuración de la infraestructura de TI. El objetivo principal es implementar Infraestructura como código para que la configuración de la infraestructura en donde opera el software sea automatizada y programada como código tanto para que los desarrolladores se involucren con infraestructura, como para que los operadores se involucren con desarrollo. Esta es una parte clave para DevOps. Las actividades específicas que se encuentran en esta etapa son:

1. Aprovisionamiento y configuración de almacenamiento de infraestructura, base de datos y red.
2. Provisión y configuración de la aplicación.

Herramientas: En esta etapa las herramientas se encuentran marcadamente claras, las de mejor calidad y popularidad son Chef, Puppet, Ansible, Vagrant y Salt, algunas otras son BladeLogic, Terraform, Consul, Bcfg2 y CFEngine.

2.4.7 Herramientas para Monitoreo

El monitoreo es un enlace importante en una cadena de herramientas. Permite a la organización de TI identificar problemas específicos de versiones específicas y comprender el impacto en los usuarios finales. La información de las actividades de monitoreo a menudo impacta las actividades de la Planificación requeridas para los cambios y para los nuevos ciclos de lanzamiento. Por otra parte, el monitoreo suele ser la etapa que enlaza un nuevo ciclo de mejora. Las actividades más importantes relacionadas con el Monitoreo son:

1. Rendimiento de la infraestructura de TI
2. Respuesta y experiencia del usuario final
3. Métricas y estadísticas de producción

Herramientas: Las principales herramientas de monitoreo son: Kibana, New Relic, Dynatrace, Nagios, Zabbix, Datadog, ElasticSearch y AppDynamics, Además, las herramientas mencionadas en Testing, están en íntima relación con el monitoreo, y algunas de ellas pueden lograr también tareas de monitoreo como Bosun, jKool, Splunk, entre otros. Existe una amplia información actualizada, acerca de las más populares y recientes herramientas de DevOps, y que es realizada y mantenida por Xebia Labs; el nombre es Tabla Periódica de Herramientas DevOps *(Xebia Labs, 2017)*

3. Marco Metodológico

3.1Tipo de Investigación

El presente trabajo de graduación es una investigación documental de tipo narrativa e investigativa, no experimental. Implica la elaboración de una guía que permita implementar DevOps de manera eficiente en un equipo de desarrollo que utilice una metodología de desarrollo tradicional, recolectando los conocimientos básicos que implican las metodologías ágiles.

3.2Árbol de Problemas y Objetivos

3.2.1 Árbol de Problemas

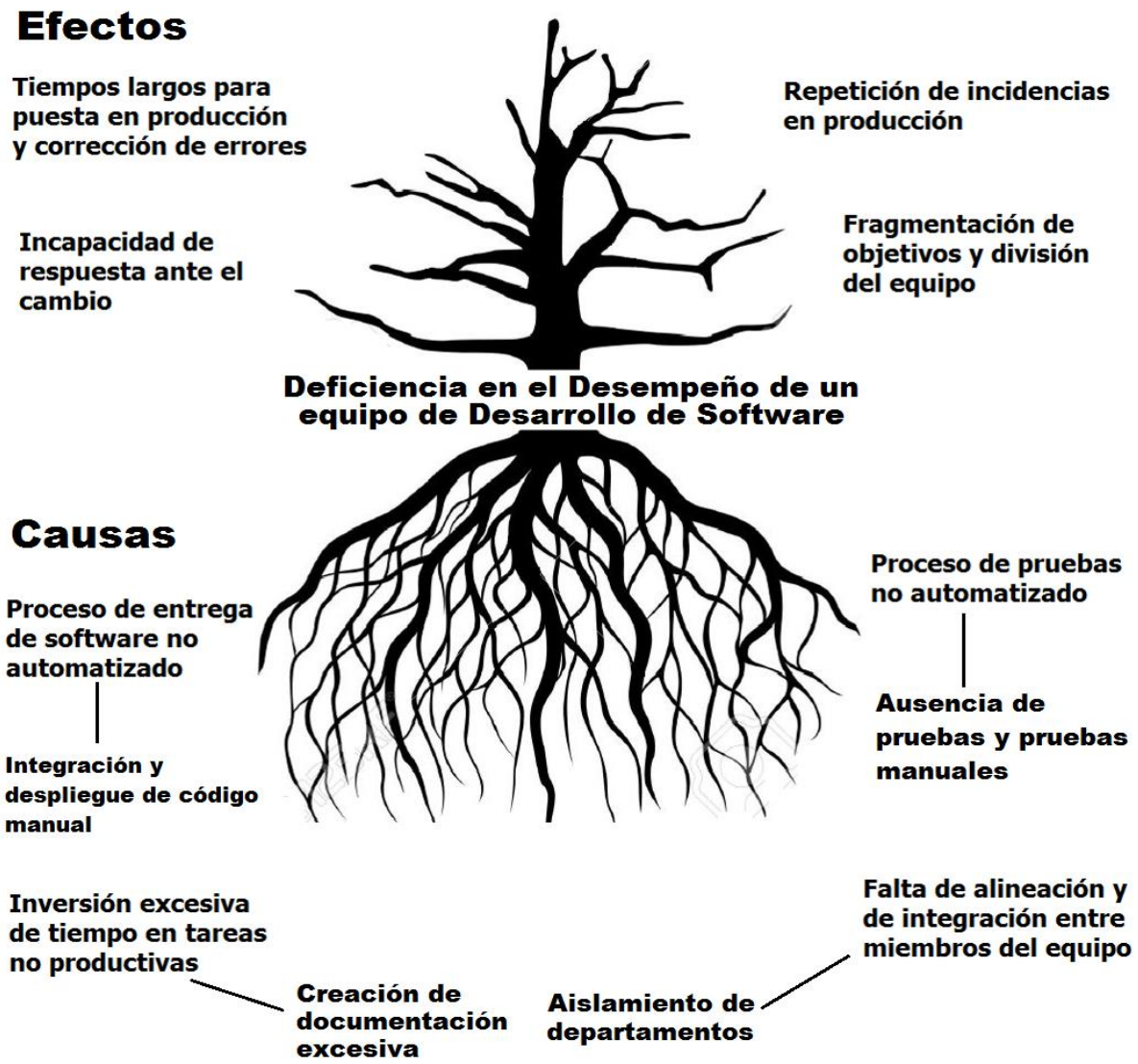

Figura 10 - Árbol de Problemas, Fuente: Personal

3.2.2 Árbol de Objetivos

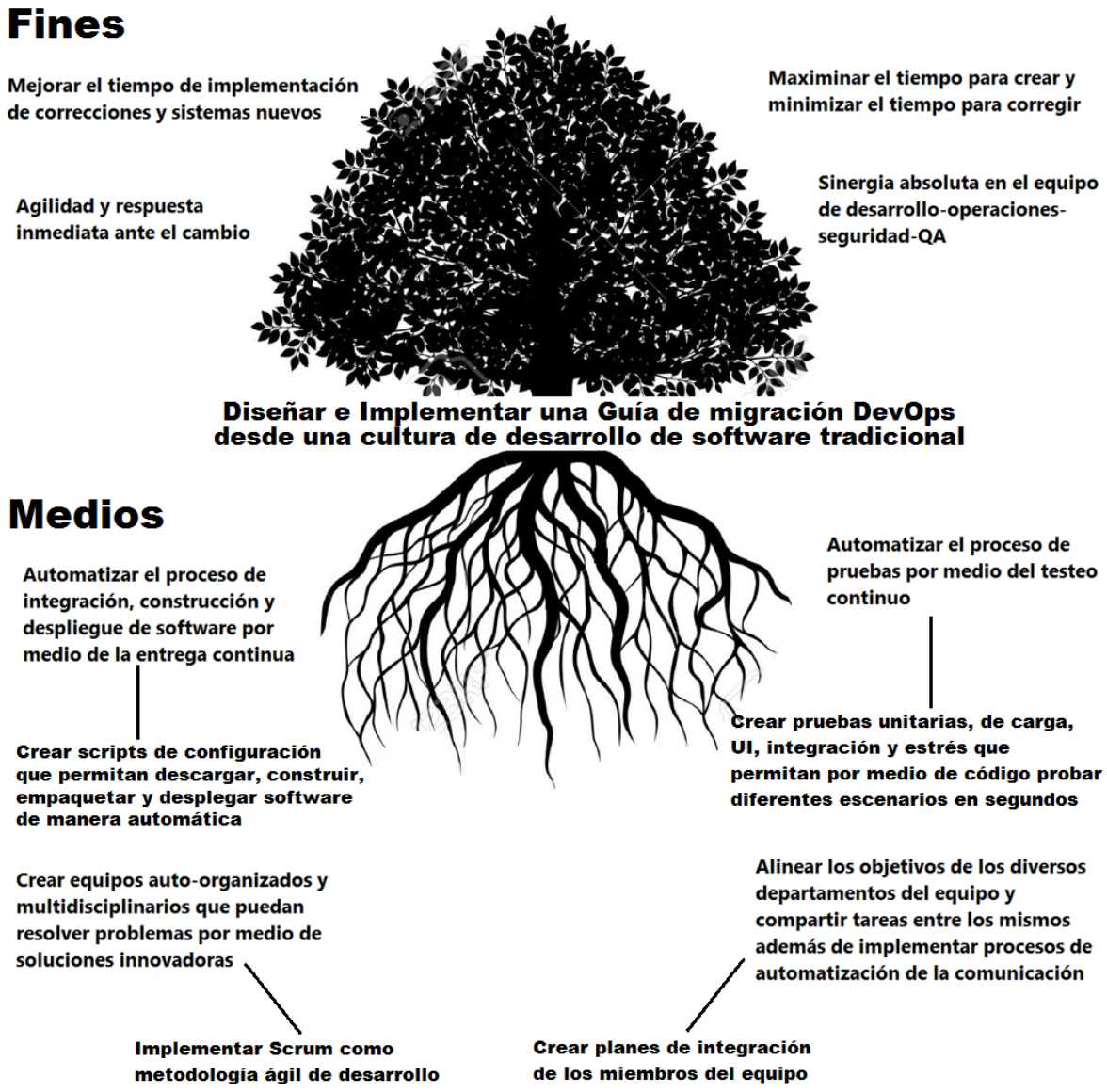

Figura 11 - Árbol de Objetivos, Fuente: Personal

3.3Diseño de la investigación

El diseño de este estudio se fundamenta en una guía para la implementación de DevOps en equipos de desarrollo de software, por medio de una serie de aspectos que serán importantes en cada etapa para la implementación de herramientas, procesos, roles y responsabilidades.

3.3.1 Instrumento de recolección de datos

Se utilizará una Guía de implementación, que será alimentada de una investigación y recolección de información sobre diversas implementaciones y exploraciones de DevOps. Análisis documental, procesos, roles y responsabilidades que han sido implementados por compañías productoras de software de vanguardia, que realizaron exploraciones informáticas para optimizar el rendimiento en equipos de desarrollo, cuya credibilidad es respetada y su veracidad tiene validez en el medio informático.

Christopher Null, CEO de Null Media en su artículo “10 companies killing it at DevOps”, comenta que Amazon, Netflix, Target, Wallmart, Nordstrom, Facebook, Etsy, Adobe, Sony Pictures Entertainment y Fidelity Worldwide Investment, son claros ejemplos de éxito de DevOps (Null, 2017). No obstante, según Gartner el 38% de las compañías utilizaban DevOps en el 2015 en al menos un proyecto, se espera que este número haya incrementado para el 2018 a más del 60%. Otras compañías que han tenido éxito con DevOps son Apple, Microsoft, Starbucks, Linkedln, NASA, Airbnb, Uber, y más (Xebia Labs, 2015).

4. Guia DevOps

4.1 Introducción de la Guía

La presente guía proporciona las pautas necesarias para la implementación exitosa de DevOps —cultura de desarrollo de software que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales de sistemas en las tecnologías de la información. — Esta guía proporciona un marco integral que incluye los principios, aspectos y procesos que conforman la implementación. DevOps es una cultura que no dicta específicamente métodos, procedimientos o incluso herramientas específicas que se deberían de usar, es por tal razón, que esta guía pretende encaminar de una manera más certera a los equipos que deseen implementarla. Es importante que se mencione que existe una íntima relación entre DevOps y las metodologías ágiles, sin embargo, el primero, extiende el enfoque y los límites de las metodologías ágiles por medio del involucramiento de todo el ecosistema a través de la automatización correcta de los procesos. DevOps no son herramientas, ni automatización, un puesto de trabajo o incluso agilidad, no obstante, es una cultura que involucra a todos estos aspectos para entregar de mejor manera el software y los servicios que éste conlleva.

Esta sección tiene como fin mostrar la estructura de la Guía para la implementación de DevOps además de brindar una pequeña introducción para el entendimiento de la misma. La siguiente sección representada como Fase I, explica los pasos para la implementación de principios, mismos que son cimientos fundamentales en los que se basa DevOps; están compuestos por CALMS Culture (Cultura), Automation (Automatización), Lean (Infalible/Delgado), Measurement (Medición) y Sharing (Compartir). En esta misma Fase se detallan los pasos para el establecimiento de los aspectos claves de DevOps, que son Flujo Continuo, Autogestión y Visibilidad, Planificación y Gestión de tiempo además de Herramientas Ideales. La siguiente sección representada como Fase II detalla el proceso DevOps completo, listando las etapas secuenciales para la implementación del mismo. Estas etapas son Planificación, Control de Versiones, Desarrollo, Construcción, Pruebas, Empaquetado, Configuración, Despliegue y Liberación, así como Monitoreo. En todas las etapas se mencionan los aspectos importantes con una recomendación de herramientas que podrían emplearse. Finalmente, en esta misma Fase se describe la Revisión final en la que se evalúa el Ciclo de mejora continua, así como las métricas de Desempeño que justifican de

manera fiable la implementación DevOps. Para tener un mejor entendimiento de la presente guía y su contenido, se presenta una gráfica que explica el flujo lógico para el establecimiento de DevOps.

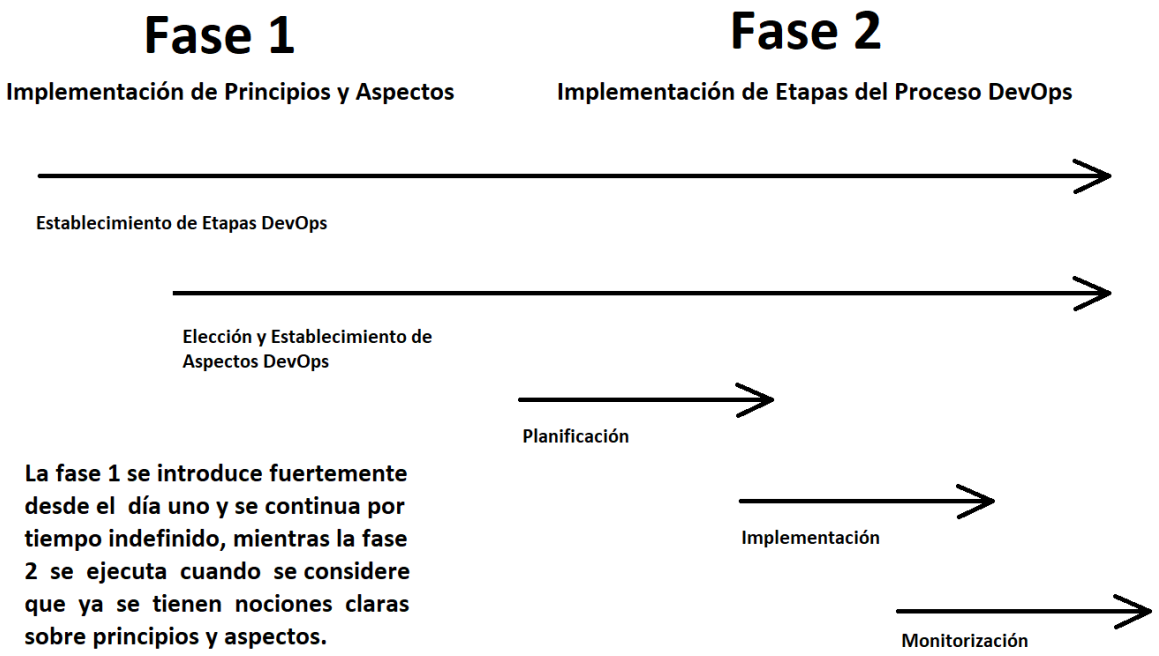

Figura 12 - Guía DevOps, Fuente personal

4.2Fase 1: Implementación de Principios y Aspectos

4.2.1 Etapa 1: Establecimiento de Principios de DevOps

Los principios son las filosofías básicas para aplicar la cultura DevOps, y deben de utilizarse obligatoriamente en todas las implementaciones. Los cinco principios conforman el framework CALMS presentado primeramente por Jez Humble, coautor del libro "The DevOps Handbook", y proviene del acrónimo en inglés Culture, Automate, Lean, Measurement y Sharing, y traducidos al español son:

1. Cultura
2. Automatización
3. Infalible/Delgado
4. Medición
5. Compartir

Estos principios son aplicables a todas las implementaciones y deben cumplirse con el fin de garantizar la aplicación efectiva de la filosofía. Los principios no están abiertos a la discusión, elección o combinación, sino que deben aplicarse de forma individual y completa. El mantener los principios intactos e implementarlos de manera correcta, ayuda a difundir confianza con respecto a cumplimiento de los objetivos del proyecto. Todos estos principios deben respetarse y tenerse como algo muy presente en el equipo de desarrollo. Se sugiere tener físicamente impresos el DevOps Manifiesto y los principios CALMS en un lugar visible por todo el equipo, para recordar los cimientos de la manera en que se trabaja con DevOps.

4.2.1.1 Paso 1: Establecimiento del DevOps Manifiesto

DevOps tiene una íntima relación con las metodologías ágiles. Dicha relación proviene del Manifiesto Ágil que fue creado en 2001 por 17 profesionales en el desarrollo ágil que deseaban crear un estándar como alternativa a las metodologías tradicionales de desarrollo (ver sección 2.1.3 de las Metodologías ágiles de Desarrollo de Software). En el manifiesto ágil se detallaron doce principios en que se basan todas las metodologías ágiles implementadas. A continuación, se presenta una propuesta de alineación para la creación de un Manifiesto DevOps, además de los principios en los que se basa el mismo, con un enfoque en los 12 principios del Manifiesto Ágil con algunas modificaciones, mismas que son resaltadas en cursiva y negrita. Estos enunciados deben tenerse en plena consideración al inicio de la implementación de la Cultura DevOps.

Se han descubierto mejores formas de ejecutar sistemas por medio de nuevas maneras de trabajar, que se ha llegado a valorar:

1. Individuos e interacciones sobre procesos y herramientas
2. Sistemas de trabajo sobre exhaustiva documentación
3. ***Interacción entre cliente y desarrollador*** sobre negociación contractual
4. Respuesta al cambio sobre seguir un plan

Se valoran los elementos de la derecha, pero se valoran aún más los de la izquierda. Al final de cada enunciado, en paréntesis, se encuentran las observaciones de las diferencias entre principios de ambos Manifiestos si las existe.

1. La máxima prioridad es satisfacer al cliente mediante la entrega temprana y continua de **una funcionalidad valiosa** (se cambió a una palabra más general que "software").
2. Se aceptan los cambios en los requisitos, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente (Idéntico).
3. ***La infraestructura es código y debe desarrollarse y administrarse como tal*** (Nuevo).
4. Se entrega ***la funcionalidad de trabajo*** frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible (se usa la palabra funcionalidad de trabajo en lugar de software funcional).
5. Los responsables de negocio, ***el equipo de operaciones*** y desarrolladores, ***así como personal de QA y seguridad*** trabajan juntos de forma cotidiana durante todo el proyecto (Se agregó el equipo de operaciones, QA y seguridad para ser inclusivo y no exclusivo para desarrollo).
6. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo (Idéntico).
7. ***Se entregan funcionalidades de mayor valor cuando todo el equipo involucrado trabaja sobre objetivos alineados y con tareas compartidas*** (Nuevo).
8. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es por medio de ***métodos ágiles digitales o personales*** (Agregado métodos ágiles digitales o personales en lugar de conversación cara a cara, ya que la respuesta pronta es muy importante y la conversación cara a cara no es siempre posible).
9. El software funcionando ***entregado por operaciones*** es la principal medida de progreso (se agregó entregado por operaciones, ya que el valor se le entrega al cliente hasta que está en ambiente de producción).
10. Los procesos Ágiles promueven el desarrollo sostenible. Promotores, usuarios y el equipo de desarrolladores y ***operaciones*** deben ser capaces de mantener un ritmo constante de forma indefinida (se agregó operaciones por la misma razón del punto 5).
11. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad. (Idéntico)

12. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial. (Idéntico).
13. ***La funcionalidad del software solo puede ser realizada por el cliente cuando es entregada por los sistemas implementados. Los requerimientos no funcionales son tan importantes como la funcionalidad deseada por el resultado del usuario (Nuevo).***
14. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados (Idéntico).
15. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia (Idéntico).

Se realizó el cambio de software funcional a funcionalidad valiosa debido a que se puede entregar valor al cliente sin escribir líneas de código por medio de gestionar otros aspectos importantes. Además de estos cambios, se hace énfasis en agregar operaciones, ya que, aunque el Manifiesto Ágil es completamente acertado, carece de completitud, debido que se olvidó agregar al resto de miembros del equipo.

4.2.1.2 Paso 2: Inyección de Cultura

El primer principio es la Cultura. DevOps se podría resumir en la palabra Colaboración ya que precisamente es una actitud de sinergia entre Desarrollo y Operaciones. El entorno que lo rodea se encuentra comprendido de herramientas y automatización, sin embargo, si no existe una actitud realmente genuina de parte de Desarrollo y Operaciones para trabajar juntos, es imposible hacer sinergia, y esto es tan importante incluso indispensable, debido a que no se pretende solucionar problemas relacionados con herramientas, sino más bien con seres humanos. Esta cultura no se limita a Desarrollo y Operaciones, sino que incluye también a Seguridad, Gestión de Productos, Diseño, Gestión de Proyectos, Marketing, etc. ya que todos al final están involucrados en el proceso de producción de software. Es realmente la cultura el punto más importante ya que no hay otro ingrediente que produzca más Colaboración que el compartir metas y objetivos. Cuando un equipo de desarrollo implementa DevOps, deja de ver partes separadas en los departamentos, y empieza a ver a un equipo multidisciplinario que tiene los mismos objetivos. DevOps provee visibilidad

suficiente acerca de procesos y productos para todos, y esto lo hace también por medio de una cultura de Comunicación abierta y continúa que fluye hacia todas direcciones.

El objetivo de DevOps es que cada individuo no tenga únicamente puesta una camiseta del área en la que se desenvuelve, sino que todos los individuos porten la misma camiseta, y que el ambiente colaborativo, la actitud de apoyo de los unos con los otros y sobretodo la armonía y sinergia fluya entre los procesos de comunicación de todas las áreas no sólo de tecnología, sino de la organización en general, para que no solo el software sino además los servicios puedan ser llevados desde el código, hasta el usuario final de una manera rápida y eficiente. Además, las reparaciones, mejoras y depuraciones del mismo, sean conllevadas por un equipo que posee la capacidad de resolver las peticiones de usuarios rápidamente.

Existen muchos métodos para poder inyectar cultura de armonía en el equipo. Se recomienda seguir la corriente popular en la actualidad, en las que corporaciones modernas tanto start-ups como grandes multinacionales, crean procesos de motivación de personal, que apoyan la mejora de las relaciones en el equipo, inclusive acompañándose de actividades fuera del trabajo, en el que se mezclen los equipos de desarrollo, operaciones, QA, seguridad, etc. para que se traten por igual y exista confianza y amistad entre ellos.

4.2.1.3 Paso 3: Identificación e Implementación de Automatización

El segundo principio es la automatización. Según el Informe del estado de DevOps en 2016 de Puppet Labs “los equipos que practican DevOps despliegan 30 veces más frecuentemente, fallan 60 veces menos y se recuperan 160 veces más rápido”; la pregunta es ¿cómo lo hacen? La respuesta es simple, por medio de la automatización. Con este cambio de enfoque, los equipos han aprendido a automatizar muchos aspectos, sin embargo, Atlassian destaca la Entrega Continua y la Infraestructura como Código. La idea de la primera es llevar funcionalidad de valor para el usuario desde el código, hasta los servidores de producción de una manera rápida, fiable y optimizada, por medio de la ejecución de scripts de configuración que detectan los cambios, construyen una nueva versión del software, ejecutan las pruebas y colocan la versión en los servidores de pre-producción, en donde se ejecutan otra serie de pruebas automáticas, mismas que validan esta versión del software y en el caso de ser pruebas exitosas, la misma es colocada en ambiente de producción. Esta automatización le permite una agilidad considerable a los equipos que la emplean, ya que no se trata únicamente de

poner el software en producción, sino de, además, se puedan regresar a versiones anteriores de manera rápida, si ocurriera algo inesperado, se hace un detalle mayor de este proceso en el aspecto de Flujo Continuo en la sección 3.6.1.1 y 3.6.1.2.

La segunda automatización notable, la Infraestructura como código, ayuda a los especialistas de operaciones/sistemas a tratar la configuración de infraestructura como si fuese software, por medio de scripts de configuración que son capaces de preparar un ambiente con tan solo ejecutarse. Este proceso cobró mayor importancia, cuando la computación en la nube tuvo su más grande auge. Se tiene más detalle en la sección 3.6.1.3 del aspecto Flujo Continuo. La automatización sin duda es importante en DevOps ya que un equipo de desarrollo es mucho más ágil cuando todos los procesos repetitivos son ejecutados por computadoras, que pueden ejecutar estos procesos con mayor rigor y exactitud que los seres humanos.

Al implementar esta guía, es importante que se identifiquen y listen todos los puntos de mejora, que incluyan procesos manuales y que, por medio de automatización, se podría mejorar el flujo del proceso. Al tener estos puntos identificados, se deben de encontrar herramientas que ayuden a implementar la automatización necesaria.

4.2.1.4 Paso 4: Reducir todo al mínimo para ser Infalible/Delgado

El tercer principio, Lean puede ser traducido a Infalible o Delgado. Está enfocado a mantener todo al mínimo; herramientas, reuniones e incluso los Sprints deberían ser mantenidos al mínimo. Todo aquello que no aporte el máximo valor para el cliente debería ser suprimido, además todo aquello que si aporta el máximo valor debe ser optimizado y automatizado. Cuando ser delgado es parte de la cultura de un equipo, cada herramienta y proceso que se elija tiene un propósito específico, este principio también aplica para el recurso humano, ya que se deben mantener los equipos a un tamaño efectivo para evitar los rendimientos decrecientes, como el ejemplo de tener demasiados cocineros en la cocina. Por otra parte, si la naturaleza compleja del proyecto, obliga a equipos más grandes, los mismos deben ser divididos en sub-equipos, como se ha tenido éxito al implementar DevOps en casos de éxito, acompañado de Scrum como metodología ágil.

Además, siendo infalible se garantiza el aprendizaje de errores, ya que el equipo sabrá que los errores están asegurados y debido a esto, debe prepararse para asumirlos y recuperarse

después de ellos. Un equipo con DevOps, debe ser infalible, ágil, presto al cambio y dispuesto a corregir y aprender de sus errores.

4.2.1.5 Paso 5: Parametrizar procesos por medio de la Medición

De la revolución en la calidad del siglo XX, se aprendió que no se puede mejorar ningún proceso o producto, si el mismo no se puede medir, y esto llevo a la creciente motivación de toda la industria a establecer métricas para medir las cosas y así, poder establecer si existe mejora. Si se ha mantenido o si se ha desmejorado algún proceso. En DevOps no es diferente, la mejora continua es parte de la filosofía y existen múltiples herramientas que permiten demostrar por medio de datos, que la implementación ha mejorado el rendimiento del equipo. Aunque se puede medir casi todo, no significa que se esté obligado a medir todo; sin embargo, las mediciones tienen que contestar preguntas como estas:

1. ¿Cuánto tiempo consume pasar desarrollo a despliegue?
2. ¿Cuánto esfuerzo conlleva integrar el código nuevo?
3. ¿Con qué frecuencia se comenten errores y fallos?
4. ¿Cuánto tiempo se invierte a la resolución de fallos y errores?
5. ¿Qué esfuerzo conlleva el crear y configurar ambientes de desarrollo y producción?
6. ¿Cuántas pruebas se ejecutan al producto en una fracción determinada de tiempo?
7. ¿Qué esfuerzo se debe invertir para implementar diferentes versiones del producto?

Se deben listar todas las métricas que serán útiles para el equipo que está implementado DevOps. Teniendo ya estas métricas deben implementarse las herramientas de monitoreo necesarias para poder obtener mediciones correctas. Si estas y otras preguntas pueden ser contestadas por los datos que se monitorizan, se puede tener seguridad en que se podrá determinar si existe mejora en el rendimiento o no, así como las puntos fuertes y débiles del equipo, además se podrá ganar información valiosa para la toma de decisiones futuras no solo de tecnología sino de la organización en general.

4.2.1.6 Paso 6: Compartir el trabajo y el conocimiento

DevOps por excelencia no es el trabajo de una persona o herramienta, sino es un trabajo de todos. La división y fricción entre Desarrollo y Operaciones se debe en gran medida a la falta de aspectos en común. Se deben alinear e integrar las metas, objetivos, responsabilidades y

tareas de ambos equipos, para obtener un avance significativo en disminuir o terminar con la división. Los desarrolladores pueden aumentar su buena actitud y disponibilidad de manera instantánea si ayudan a Operaciones con las tareas más pesadas. Por otra parte, puede existir más comprensión de Operaciones en la medida en que entiende la complejidad de los procesos de Desarrollo.

Esto no significa querer que Desarrollo se vuelva experto en Operaciones y Operaciones en Desarrollo, sino más bien que ambos se integren en una fase de ciclo de vida continuo de producción de software. Se podrá saber que un equipo de desarrollo ha realmente implementado DevOps cuando, exista un ambiente de armonía y compartimiento de responsabilidades, tareas, conocimiento, técnicas, etc. entre ambos equipos, y que eso sea trasladado como valor a los usuarios finales del producto. Esto rompe en cierto sentido las limitantes de las metodologías tradicionales donde los roles y responsabilidades eran marcados a tal punto en que se aislaban inclusive las interacciones. Es importante establecer los puntos en donde se puede aumentar el compartir las metas y tareas. Además, es necesario que se establezcan reuniones para compartir conocimiento y de esta forma, poder ir nivelando poco a poco el conocimiento entre los equipos.

4.2.2 Etapa 2: Elección y establecimiento de aspectos DevOps

Los aspectos son enfoques básicos para aplicar la cultura DevOps, y deben de utilizarse y gestionarse en las implementaciones, según sea el caso. Los cuatro aspectos que se presentan en esta sección son:

1. Flujo continuo
2. Autogestión y visibilidad
3. Planificación y Gestión del tiempo
4. Herramientas Ideales

Los aspectos son aplicables a la mayoría de las implementaciones y deben elegirse cuidadosamente con el fin de garantizar el correcto funcionamiento de DevOps. Estos aspectos deben ser aplicados inmediatamente al finalizar el establecimiento de los principios DevOps.

4.2.2.1 Paso 1: Establecimiento del Flujo continuo

Flujo Continuo es el primer aspecto de DevOps, que va de la mano a algunas prácticas ágiles. La idea de este aspecto es que todo el funcionamiento del equipo de software, tanto desarrollo, como operaciones, QA y Seguridad, fluya de principio a fin de manera continua y constante por medio del establecimiento de los siguientes sub-aspectos, los cuales deben ser elegidos e implementados según corresponda a la naturaleza del equipo.

a. Integración y Testeo Continuo

La integración continua más conocida por su traducción al inglés *continuous integration* (CI), es un modelo informático propuesto por primera vez por el ingeniero de software británico Martin Fowler, que consiste en realizar integraciones automatizadas del código de un proyecto, de forma constante para evitar los fallos lo más pronto posible. Las integraciones según Fowler se deben de hacer al menos una vez al día y cada integración se verifica con un build automático, que incluye ejecución de pruebas (testeo continuo). La integración continua provee calidad en el proceso por medio de brindar más visibilidad a todas las partes ya que en cualquier momento se puede ver el estado real del proyecto, así como en que versión está cada ambiente y si se puede o no entregar una funcionalidad útil para el usuario final. También tenemos calidad en el producto por medio de entregar visibilidad en la estrategia de gestión de configuración, política de ramas del control de versiones y tagueos, entre otras cosas. Finalmente se obtiene calidad en las personas ya que los implicados obtienen mejores prácticas en programación y conocimiento en distintos tipos de pruebas, que da como producto código de mayor calidad.

Para hacer Integración y Testeo continuo de manera automatizada, se deben utilizar Servidores de Integración continua, mismos que están capacitados para ejecutar tareas de flujo continuo como el siguiente proceso:

1. Detectar un Commit desde el sistema de control de versiones que puede ser Git, GitHub, GitLab, Bitbucket, Subversion, Team Foundation Server, entre otros. (Ver la sección 2.4.2)
2. Construir de manera rápida y automatizada una versión del proyecto. Usualmente esto se hace por medio de Scripts de construcción en diversos lenguajes según la

naturaleza del software que se encuentra en desarrollo como Gulp, Grunt, Maven, Gradle, Broccoli, entre otros. (Ver la sección 2.4.2)

3. Ejecutar diferentes tipos de pruebas como unitarias, de UI, de carga, de integración, fiabilidad, etc. (Ver la sección 2.4.3).
4. Informar por alguna herramienta colaborativa en el caso de haber un fallo en la construcción o en las pruebas, el software colaborativo y de mensajería puede ser Jira, iRise, HipChat, Trello, Team Foundation Server, Rally, StackStorm, ServiceNow, Slack, Deveo, entre otros (Ver la sección 2.4.1).
5. Si todo se ejecuta correctamente, se versiona el código, y la versión resultante pasa a ambiente de pre-producción o certificación según sean las preferencias del equipo, por medio de empaquetar el release. Para esto se emplean herramientas de empaquetado, construcción y orquestación de contenedores y despliegue (Ver la sección 2.4.4 y 2.4.5).

Existen diversos servidores de integración continúa, los más populares y usados son Jenkins, Bamboo, Team Foundation Server y Hudson (Ver la sección 2.4.4). A continuación, se puede observar un ejemplo de pipeline implementado en Jenkins.

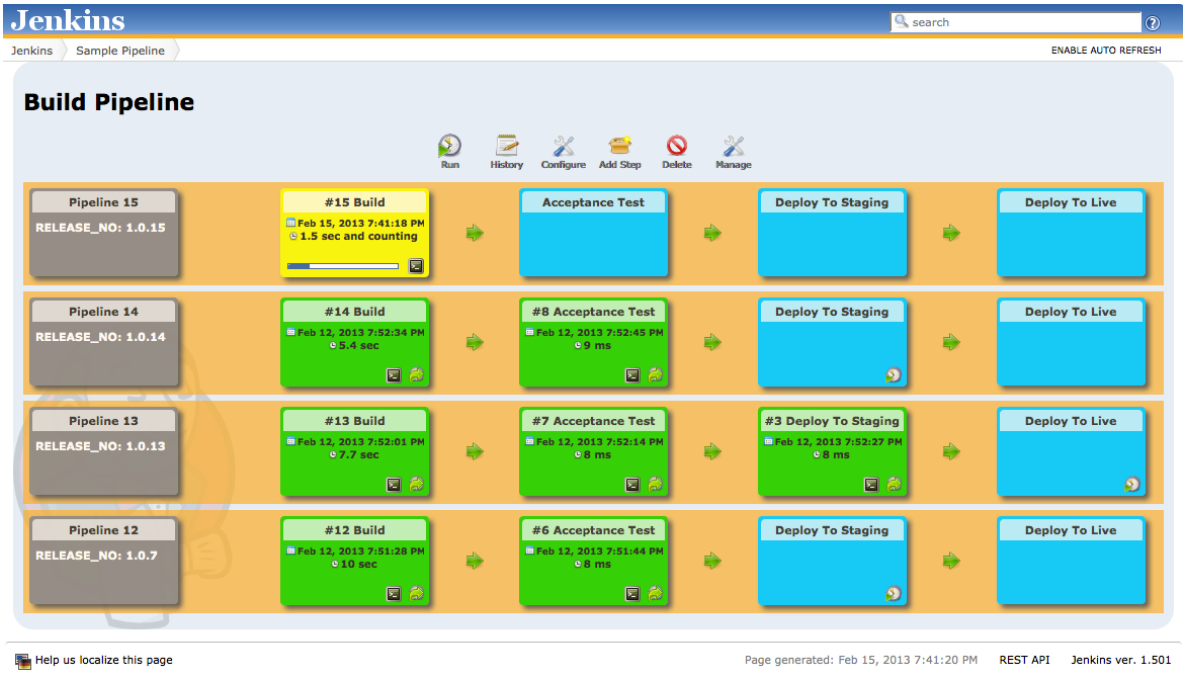

Figura 13 - Pipeline implementado en Jenkins, Fuente: (Garcia A. M., 2014)

b. Entrega y Despliegue/Implementación Continuo

La entrega continua es una práctica de desarrollo de software en la que se crean, prueban y preparan automáticamente los cambios en el código y se entregan para la fase de producción. Básicamente la entrega continua extiende a la integración continua al implementar todos los cambios correctamente integrados en el código, hacia un entorno de pruebas y/o de producción después de la fase de creación. Cuando la entrega continua se implementa de manera adecuada, los desarrolladores dispondrán siempre de un artefacto listo para su implementación que se ha sometido a un proceso de pruebas estandarizado.

Con la implementación o despliegue continuo, las revisiones se implementan en un entorno de producción automáticamente sin la aprobación explícita del desarrollador, con lo que se automatiza todo el proceso de publicación de software. En la Figura 9 se puede observar como la entrega continua automatiza todo el proceso de publicación del software en conjunto con la integración continua y el despliegue continuo.

Este proceso apoya completamente el principio de automatización y puede ser utilizado en la mayoría de procesos de construcción de software moderno.

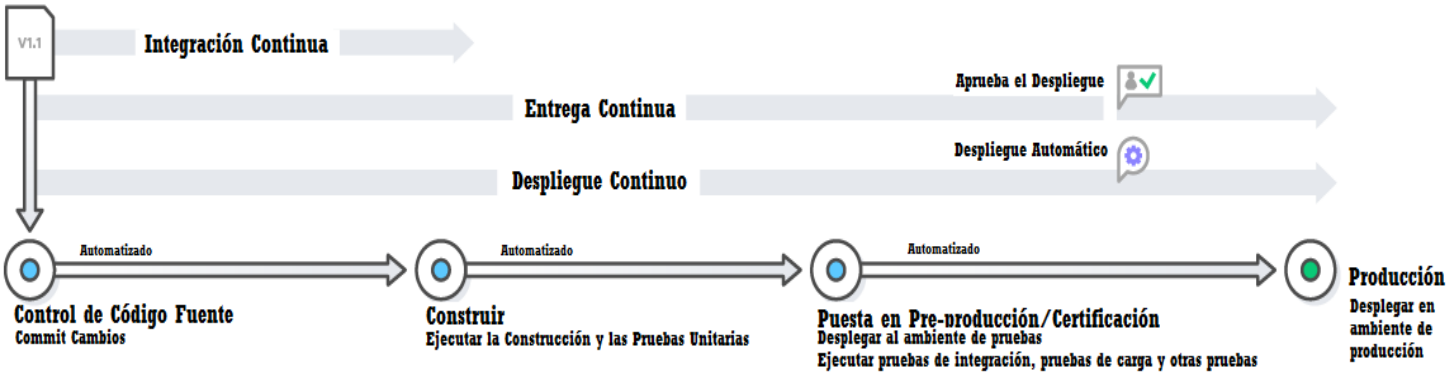

Figura 14 - Entrega Continua, Despliegue Continuo e Integración Continua, Fuente: (Amazon Web Services, 2017) traducción personal

c. Infraestructura como Código

Como se mencionó en el Manifiesto DevOps, la infraestructura no debe ser gestionada de manera manual ad-hoc por parte de operaciones, sino debe ser tratada y mantenida como código, que pueda ser programado y ejecutado como tal, por medio de scripts configurables. Esta es uno de los aspectos pilares en DevOps, ya que integra a los desarrolladores como parte del proceso de operaciones, y a su vez integra a operaciones como parte del proceso de

desarrollo. Según Hewlett Packard “A veces denominada "infraestructura programable", la infraestructura como código (IaC) trata la configuración de la infraestructura exactamente como el software de programación. De hecho, ello comienza a difuminar los límites entre la escritura de aplicaciones y la creación de entornos donde se ejecutan. Las aplicaciones pueden contener scripts que crean y organizan sus propias máquinas virtuales. La infraestructura como código permite a las máquinas virtuales gestionarse de manera programada, lo que elimina la necesidad de realizar configuraciones manuales (y actualizaciones) de componentes individuales de hardware. Esto hace que la infraestructura sea muy "elástica", es decir, escalable y replicable. Un solo operario puede implementar y gestionar una máquina o 1 000 máquinas usando el mismo conjunto de código. Velocidad, ahorros de costes y reducción del riesgo son las ventajas naturales de la infraestructura como código. El concepto de IaC es el marco del que ha surgido DevOps. Una línea cada vez más fina entre el código que ejecuta aplicaciones y el que configura la infraestructura implica que los desarrolladores de aplicaciones y los profesionales de operaciones compartan cada vez más un conjunto de responsabilidades laborales. Además, la infraestructura como código admite IaaS que permite acceder a máquinas virtuales y herramientas de gestión basadas en software para combinarse y venderse como un servicio”. (Hewlett Packard, 2018). Herramientas como Puppet, Chef o Ansible, entre otras, ayudan a tratar a la infraestructura como código y en conjunto con herramientas de Cloud como Amazon Web Services, Google Cloud, Azure o Digital Ocean, se pueden gestionar ambientes completos con scripts de configuración que gestionen las máquinas virtuales y las orquesten. (Ver la Sección 2.4.6). Es de suma importancia que se implementen técnicas como esta para que la automatización no solo afecte de Desarrollo sino también a Operaciones.

d. DevSecOps

DevOps, nos transmite la idea de que todo el equipo debe manejar conceptos de seguridad y están capacitados para comprenderlo e implementarlo en la medida adecuada en sus tareas diarias. Los profesionales en seguridad también tienen un espacio, al tratar la seguridad como código, y la misma mantenerla de esa manera, como algo programable y configurable, que garantice la seguridad y la privacidad de los datos. Según el Manifiesto de DevSecOps los profesionales en seguridad han encontrado por medio de la seguridad como código, una

forma de trabajar y operar con menos fricción, para poderse adaptar a los cambios rápidamente y fomentar la innovación para garantizar la seguridad de los datos. Los profesionales de seguridad se incorporan al ciclo para que el flujo continuo integre incluya los procesos complejos de seguridad como parte de la entrega continua y auto-gestionada de software.

e. Kata

Kata es un ejercicio en Karate que consiste en hacer una forma muchas veces, y en cada una de estas se agrega una mejora. En DevOps, basado en metodologías ágiles y de la mano con Scrum, se deben hacer muchas iteraciones pequeñas, y en cada una de ellas, tal cual manda el Manifiesto Ágil, se debe agregar valor al producto final, y al hacer esto de manera permanente y constante, se incentiva el flujo continuo del proceso.

f. El Pipeline se detiene con errores

El flujo continuo está en constante movimiento por cada iteración que se hace, sin embargo, se debe tener el correcto control sobre el mismo, es por tal razón que el pipeline se detiene cuando hay errores, y no debe de continuar hasta que cada error sea resuelto por medio un pipeline nuevo, con esto tenemos un proceso de integración, despliegue y entrega continua correcta.

4.2.2.2 Paso 2: Implementación de Autogestión y visibilidad

El segundo aspecto tiene que ver con el equipo de sistemas, ya que al igual que las metodologías ágiles, DevOps debe ser formado por un equipo que tenga la capacidad de ser autogestionado, autodidacta y autodisciplinado, entre otras muchas cualidades que deben ser implementadas por cada miembro del equipo. Los sub-aspectos derivados de la Autogestión y la visibilidad son:

a. Empoderamiento directo

El equipo de DevOps debe ser autogestionado y auto-dirigido en cuanto a sus responsabilidades diarias. El empoderamiento directo se refiere que al asignar una tarea específica a un miembro del equipo, esta persona es directamente responsable por el

seguimiento del problema, así como del seguimiento de la solución. También aplica a una plataforma (software) y no solo a individuos.

b. Correcta asignación e información de tickets

Como se mencionó en el aspecto anterior, cada individuo o plataforma es responsable del problema asignado, esto solo se puede lograr cuando existe una correcta asignación e información de tickets, en otras palabras, un determinado ticket debe ser resuelto por cualquier persona, sin embargo, será asignado en la medida de lo posible por la persona más indicada para resolver el problema, esto con el objetivo de que el mismo sea resuelto en el menor tiempo posible, además, cada ticket debe contener una descripción adecuada que permita al individuo que va a resolver el problema, tener un panorama completo del mismo. Además, se debe tener una herramienta de colaboración que permita el seguimiento automatizado de los tickets, así como información oportuna de ellos para la toma de decisión, como tiempo promedio de respuesta, cantidad de tickets reportados y reparados, tiempo menor y máximo de reparación, tickets resueltos, tickets no resueltos, entre otros.

c. Herramientas para empoderar

En la actualidad es fácil encontrar personas que confunden este enfoque cultural con una herramienta o un puesto de trabajo. Sin embargo, DevOps no es específicamente eso. Las herramientas solo sirven para agregar valor al producto, al proceso y a las personas, y bajo este concepto es que se deben implementar. Las herramientas no son DevOps, pero sin duda alguna dan el suficiente empoderamiento para que implementar la automatización y la sinergia de procesos sea posible.

d. Indicadores Actualizados

La visibilidad para todos los miembros del equipo es trascendental, no solo para desarrollo, o para operaciones, sino para todos los involucrados en el proceso. Esto obliga a que los indicadores producto de pruebas estén actualizados en todo momento, esto quiere decir que las pruebas deben dar los resultados en el momento actual, para que se pueda saber el estado de salud de la aplicación cuando se requiera. Además, el monitoreo de las aplicaciones que se encuentran en producción debe ser constante y realista, de tal forma, que tomar decisiones en tiempo real, sea un proceso controlado, estandarizado y hasta normal.

4.2.2.3 Paso 3: Planificación y Gestión del tiempo

Cuando se trabaja en base al Manifiesto Ágil, la gestión del tiempo es algo importante, ya que se trata de entregar la mayor cantidad de funcionalidad en el menor tiempo posible, esto hace que la gestión y planificación del tiempo sea algo crítico. Es importante hacer énfasis que en las metodologías ágiles se debe de realizar planificación continua, esto quiere decir se debe de ser lo suficientemente ágil para planificar adaptándose al cambio, sin embargo, la planificación del tiempo es un proceso menos cambiante. Se debe de tener un adecuado proceso para que planificar y ajustar planificaciones sea rápido. Los sub-aspectos asociados a la Planificación y Gestión de tiempo son:

a. Identificación y Administración de centros de trabajo

Para gestionar de manera efectiva el tiempo, es necesario que se identifique correctamente los centros de trabajo, en otras palabras, los miembros del equipo DevOps deben estar consiente cuales son las tareas que llevan más tiempo para que se busquen correctos métodos de administración para optimizar los esfuerzos en dichas tareas. Esta parte se puede relacionar con la ruta crítica del conjunto de responsabilidades. Tener identificados los centros de trabajo, permite realizar planificaciones más acertadas y más cercanas a la realidad, con tiempos de estimación efectivos.

b. Esfuerzos de trabajo de rutina sobre realizar trabajo de rutina

La automatización, segundo principio de DevOps, no es exactamente el significado, pero si es una acción natural de la cultura. Es importante que los miembros DevOps inviertan más tiempo en el esfuerzo por automatizar y optimizar los trabajos de rutina que, en hacer específicamente una y otra vez el propio trabajo de rutina, es decir, si una tarea de rutina se realiza de la misma forma varias veces, es mejor dedicar un tiempo para automatizar dicha tarea, que invertir tiempo para realizarla “n” cantidad de veces. En otras palabras, el trabajo de rutina, en la medida de lo posible debe ser automático, autocontrolado y correctamente gestionado.

c. Siempre listos ante el cambio

La planificación de tiempo se complica cuando el cambio entra en la ecuación, no obstante, en la medida de lo posible se debe ser lo suficientemente ágil para realizar planificación

continua, por lo tanto, no se debe generar planificaciones de grandes periodos de tiempo, sino por el contrario, se van a re-planificar tareas de manera ágil, corta y constante. Es importante que se lleve un control pequeño (bitácora) pero eficiente de estos cambios, para que en determinado momento justificar atrasos sea algo sencillo.

d. Aprendizaje de errores

Uno de los aspectos más importantes para una correcta planificación y gestión del tiempo es aprender de los errores. Es inevitable cometer errores en el día a día, sin embargo, cuando se implementa DevOps, se debe de aprender de ellos para que la próxima vez que se presente el mismo escenario, los resultados sean distintos. Esto se alinea a la perfección con la Introspección de Scrum, en donde se analiza el ciclo anterior para aprender de los errores surgidos, por medio de soluciones prácticas y eficientes que hagan de la próxima incidencia, un proceso resuelto. DevOps dicta no tener miedo a los errores, sino por el contrario, aprender de ellos.

e. Innovación

En DevOps la innovación es una parte importante del trabajo de cada día, por lo que es indispensable que al menos el 20% del tiempo productivo total, sea dedicado a innovar cualquier aspecto involucrado en el trabajo a modo de poder agregar valor al proceso y producto. La tecnología está en constante desarrollo y cada día nacen nuevos procesos que pueden ser implementados para mejorar el rendimiento del equipo en cualquier disciplina.

4.2.2.4 Paso 4: Establecimiento de Regla de Herramientas Ideales

El último aspecto está enfocado directamente a las herramientas y trata de elegir únicamente las herramientas indispensables. Como se ha mencionado, DevOps tiende a ser confundido con herramientas, sin embargo, estas solo empoderan el proceso, por tal razón, es común que al implementar el cambio cultural, el equipo piense que mientras más herramientas se usen, el funcionamiento será mejor, sin embargo, esto no es completamente cierto. Es importante que se usen únicamente aquellas herramientas que sean útiles para la lógica del negocio. Se revisó detalladamente la cadena de herramientas en la sección 2.4, y se mencionaron muchas herramientas que ayudan a hacer DevOps, sin embargo, no es obligatorio usar ninguna de ella, ni incluso una en cada etapa, no obstante, es muy complicado hacer realmente DevOps

sin usar ninguna; debido a esto, es importante considera solo las herramientas ideales según cada lógica de negocio específica.

Por otra parte, en la medida de lo posible, se deben elegir herramientas que se integren fácilmente entre ellas, para incentivar el flujo continuo, y que la información sea ingresada únicamente por una sola fuente y diseminada al resto de fuentes de manera automática. Cuando el equipo considera que pierde más tiempo utilizando herramientas que haciendo algo de manera manual, el uso de herramientas pierde completamente el sentido y su propósito.

4.3Fase 2: Implementación de Etapas del Proceso de DevOps

Las etapas del proceso DevOps son procedimientos útiles para aplicar la cultura, y deben de seguirse de manera secuencial en las implementaciones, según sea el caso. Las tres etapas que se presentan en esta sección son:

1. Etapa 1 Planificación
    1.1 Identificación y Asignación de Roles
    1.2 Identificación y Elección de Herramientas
2. Etapa 2 Implementación
    2.1 Control de Versiones
    2.2 Construcción
    2.3 Desarrollo
    2.4 Pruebas
    2.5 Empaquetado
    2.6 Configuración
    2.7 Despliegue y Liberación
3. Etapa 3 Monitorización
    3.1 Ciclo de mejora continua
    3.2 Evaluación de métricas de desempeño
    3.3 Análisis de cambio

Estas etapas son aplicables a la mayoría de las implementaciones, los mismos llevan un orden específico y responden al ecosistema de la cadena de herramientas DevOps.

4.3.1 Etapa 1: Planificación

La primera etapa es la planificación; conlleva el proceso de organizar, coordinar y establecer los procesos para que los mismos fluyan de una manera automatizada y ágil. Las tareas principales es la identificación y asignación de roles, así como las responsabilidades que deberá ejecutar cada uno de ellos. Además de esto es importante que se realice una correcta identificación y elección de herramientas a usar. Este proceso difiere completamente a las planificaciones de las metodologías tradicionales, debido que se caracterizan por planificar completamente el trabajo de principio a fin, sin aceptar los inherentes cambios. A continuación, se describen las tareas importantes a seguir en esta etapa.

4.3.1.1 Paso 1: Identificación y Asignación de Roles y Responsabilidades

En DevOps es importante que cada uno de los integrantes del equipo sea multidisciplinario, sin embargo, es importante la identificación de los principales roles, así como responsabilidades. Al ser un derivado ágil, y como se ha mencionado a lo largo del presente trabajo, se sugiere que sea combinado con Scrum, por tal razón, es común encontrar los roles de Scrum Master y Product Owner en los equipos DevOps. Esto más enfocado al cumplimiento de una metodología y no a la automatización de procesos y la culturización de personas. En la sección 2.1.3.1 de Scrum y 2.2 de Equipos de Desarrollo, se profundizan estos roles. Al estar enfocado Scrum explícitamente al desarrollo, la parte de QA, Seguridad y Operaciones se sale del contexto, es por esta razón que es importante realizar una correcta identificación de Roles para agregar a estos departamentos, comúnmente no incluidos en las metodologías ágiles. Además, se sugiere que se evalúe cada equipo por individual, ya que no todos los roles se ajustan a todos los equipos, sino que es necesario establecer cuales roles cumplen para cuales equipos.

Es importante definir los siguientes roles:

a. Sección de Desarrollo y QA:
    1. Product Owner: Encargado del cumplimiento del Product Backlog el cual contiene todos los requerimientos del software que se encuentra en desarrollo. Es importante el uso de software de Colaboración para poder planificar y controlar de mejor manera la asignación de historias de usuario y de tickets de errores.
    2. Scrum Master: Encargado del cumplimiento de la metodología, planificación de Sprints para la ejecución del Sprint Backlog. Encargado de proveer el ambiente ágil, rol importante en la diseminación correcta de cultura DevOps. Es importante el uso de software para Colaboración y Mensajería, que permitan ser ágiles para transmitir información.
    3. Desarrolladores: En metodología Scrum conocidos como Scrum Team, son los encargados de desarrollar el software. Son autodisciplinados, autogestionados y autodidactas. Es un equipo capaz de poder ejecutar procesos de integración continua, testeo continuo y entrega continua, además,

tienen la capacidad de resolver tickets de soporte sin necesidad de la intervención de operaciones. Ejecutan técnicas ágiles de Extremme Programming como el desarrollo en parejas, diseño simple, pequeñas entregas, testeo con el cliente, etcétera. Además, deben tener la capacidad para absorber roles casi obsoletos como el DBA. Se incluyen los diferentes tipos de desarrolladores Frontend, Backend y móviles. Es importante el uso de software de gestión y control de versiones, gestión de repositorios, integración continua, construcción continua, testeo continuo, gestión de contendores, empaquetado, despliegue continuo, gestión de bases de datos, seguridad, entre otros.

4. Testers: Los desarrolladores construyen software con la idea de cómo testearlo, y después de esto realizan diversas pruebas automatizadas, sin embargo, existen roles dedicados a esta tarea, que son encargados de probar el software y de resolver algunos de los errores encontrados en las mismas. Este rol no es únicamente el clásico depurador, ya que debe tener la capacidad de revisar los errores encontrados en el testeo continuo, además deben tener criterio para resolver problemas encontrados en pruebas de estrés, carga, UI, integración, entre otras. Es casi como el rol de un desarrollador especializado en una tarea. Aquí en este rol se sintetiza muchas de las tareas que resuelvan directamente en desarrollo sin pasar por operaciones. Al igual que los desarrolladores, es importante el uso de software especializado, sobre todo en el testeo. Este equipo se encarga de impedir errores en producción.
5. Diseñadores/Arquitectos: Algunos productos específicos, requieren un diseño más especializado y riguroso, que sale de las capacidades de un desarrollador común, por lo que acá se incluyen roles como diseñadores UI y arquitectos de software. Todas las soluciones entregadas por estos roles, deben ser con un enfoque de procesos de automatización y despliegue automático. Deben conocer y tener la capacidad de manejo de software de colaboración, mensajería, integración, control de versiones, testeo, entre otros.

b. Sección de Operaciones:
    1. Plataforma de Ingeniería:

a. Service Owner: El propietario del servicio, cumple un rol similar al del Product Owner, sin embargo, vela por el cumplimiento de los estándares establecidos en los Servicios. Es como incluir el rol del Product Owner en Operaciones, sin embargo, el producto para Operaciones no es Software sino los Servicios de TI. Es importante el uso de software de colaboración y mensajería para facilitar el fluir de información.
b. Encargado de Despliegue: Este rol, puede ser un equipo, todo depende del tamaño de la organización. Es el encargado de velar por el correcto despliegue de las aplicaciones en producción. Debe tener la infraestructura de la organización gestionada como código (ver secciones 2.4.6 y 3.6.1.3) para poder crear ambientes de manera ágil. Los desarrolladores integran el código y lo testean de manera continua, sin embargo, este rol es el encargado de cumplir el despliegue continuo.
c. Encargado de Aplicaciones: Este rol debe automatizar el proceso para remoción de una versión específica de software y despliegue inmediato de otra versión. Es el encargado además de velar por el monitoreo constante del rendimiento de aplicación, así como de las tareas de optimización. El software de gestión de la configuración y despliegue, es indispensable para el encargado de despliegue y de aplicaciones, además es importante el tener conocimiento en el software de gestión de la nube IaaS, SaaS y PaaS.

2. Soporte:
    a. Service/Product Owner: Aquí se encuentra otro Propietario del Servicio/Producto, sin embargo, en este caso tiene un enfoque hacia los usuarios. Esta persona es la encargada de velar porque se cumpla el Product Backlog de tickets reportados por usuarios como errores y mejoras. Tiene entera comunicación con Desarrollo y debe colaborar en las tareas de manera permanente y continua. Es importante el uso de software de colaboración y mensajería.

b. Técnicos de Soporte: Tradicionalmente, son las personas que dan soporte a las aplicaciones. Deben tener conocimiento en el software no solo de manera superficial sino también automatizada. Tienen comunicación constante con desarrollo y comparten algunas tareas básicas. Es importante tener conocimiento en las herramientas que usa el equipo de desarrollo, así como software de mensajería y colaboración.

c. Sección de Seguridad:
    1. Product Owner: Este rol es el encargado de velar por el cumplimiento de los estándares de seguridad de la aplicación, tiene permanente comunicación con todo el resto del equipo y es importante que use software de colaboración, seguridad y logeo,
    2. Especialistas en seguridad: Este rol se ve en los equipos que manejan la Seguridad como Código (ver sección 3.3.1.4). Son los encargados de brindar todo lo necesario y concerniente a la seguridad de la información de aplicaciones.

4.3.1.2 Paso 2: Identificación y Elección de Herramientas

Se ha mencionado en secciones anteriores las herramientas como parte de un proceso de implementación DevOps, y la relevancia que tienen en aportar valor a la cultura. Por tal razón, en la planificación es importante que se decidan cuales herramientas son necesarias en cada proceso del desarrollo de software.

En la sección 2.4 se detallaron los diferentes tipos de herramientas que existen para apoyar cada etapa en DevOps. Es importante que las herramientas que se elijan sean únicamente las que empoderen realmente al equipo. No es indispensable que se implementen muchas herramientas en cada etapa, incluso no es indispensable que se coloquen herramientas en cada etapa, sin embargo, se debe realizar un análisis profundo para decidir cuales herramientas se podrán implementar.

Posterior a la elección de herramientas, se debe proceder a configurarlas con los responsables de cada proceso. En la sección anterior, al describir cada rol y responsabilidad se mencionó

los conocimientos básicos que cada persona debe poseer en las herramientas que ayuden a fomentar la cultura DevOps en su ambiente de trabajo.

4.3.2 Etapa 2: Implementación

La implementación es el proceso iterativo de DevOps. Se realiza todos los días y en todas las partes donde se encuentren personas involucradas. Incluye todo el ciclo de vida y todos los roles de la organización. Esta sección está alineada con la cadena de herramientas DevOps, la cual es Control de versiones, Desarrollo, Construcción, Pruebas, Empaquetado, Despliegue y Liberación, Configuración y Monitoreo.

4.3.2.1 Paso 1: Control de versiones

El control de versiones es el primer segmento que se debe de configurar para hacer DevOps. El software de gestión y control de versiones usualmente conocido como SCM por sus siglas en inglés (Source Control Management) permite tener una gestión correcta sobre el código desarrollado, además de tener una configuración completa para el mismo. El objetivo principal, además de llevar control del código, es poder realizar la integración continua, incluso incrementar el nivel de agilidad al poder regresar en cualquier momento versiones específicas del código. Un software SCM es en la actualidad una herramienta imprescindible ya que entre muchas funcionalidades permite el tracking de archivos, marcado por etiquetas (Tag) de versiones alfas, betas y release oficiales, bifurcación por ramas, integración de versiones distintas, registro de modificaciones por tiempo y usuarios, y la copia en diferentes lugares del código lo cual provee también agilidad, rapidez y seguridad.

Existen en el mercado diversas herramientas que nos permiten la implementación de software de SCM. CSV y Subversion fueron algunas de las herramientas más populares en la última década, las cuales en la actualidad aún son usadas por algunos equipos; el principal problema de ellos, es que son SCM un tanto antiguos que no se ajustan a la realidad de tecnologías que existen hoy en día, específicamente, la implementación DevOps. Por otra parte, existe Git, que fue creado por Linus Torvals en 2005, pero que tiene una mejor integración con DevOps con implementaciones gratuitas como GitLab y también otras de paga como GitHub y Bitbucket, la elección de la misma dependerá de las necesidades de la organización, ya que a algunos equipos les servirá más pagar unas licencias en GitHub y Bitbucket, que permiten depositar el código en la nube y a otros tal vez les parecerá más atractivo tener una versión

local totalmente gratuita con GitLab. En la sección 2.4.2 se mencionaron las principales herramientas de gestión y control de versiones y una comparación precisamente de estos tres gestores de versiones basados en Git.

4.3.2.2 Paso 2: Desarrollo

La segunda etapa que conlleva agilidad es el desarrollo. La o las herramientas de desarrollo será el software con el que más interaccione un desarrollador, por lo que es importante una elección correcta de IDE's o editores de texto que permitan una integración con el software SCM descrito en el proceso anterior. Esto proporcionará una visibilidad mayor a los desarrolladores sobre el código que se encuentra en construcción ya que se podrán tener funcionalidades como:

1. Información de la Rama actual de trabajo.
2. Información del Repositorio actual.
3. Cambios pendientes.
4. Comparación entre versiones de archivos
5. Integración de una terminal propia
6. Ejecución de automatizador de tareas
7. Integración con software de construcción, repositorios, testeo, depuración y empaquetado.

La elección del mejor software para construcción dependerá directamente del lenguaje en el que se desarrolla, para .NET Visual Studio será el más adecuado, Eclipse y Netbeans podrían ser una opción atractiva para Java, Visual Studio Code o Sublime Text podrían ser preferidos para JavaScript o PHP, Ruby Mine es el más conocido para Ruby y de igual manera con el resto de lenguajes de programación existentes. Mientras más integrado esté con la filosofía de entrega continua, mejor será el IDE o editor de texto elegido.

4.3.2.3 Paso 3: Construcción

La construcción es una etapa fundamental en los procesos DevOps, ya que es donde inicia realmente la integración continua y por ende la entrega continua. El software de construcción permite configurar scripts que ejecuten todo el proceso de construcción del software hasta generar una versión completa del sistema que se encuentra en desarrollo. Depende

directamente del lenguaje y el software, ya que en algunos lenguajes se ejecutará un proceso de compilación, en otros de transpiración, en otros de minificación y ofuscación, o en otros simplemente configuración de ficheros de ejecución y traslado de directorios, todas estas tareas repetitivas, no deberán ser realizadas de manera manual sino por medio de ser un ejecutador o gestor de tareas.

La construcción puede ser ejecutada de manera manual, sin embargo, para el proceso específico de la entrega continua deberá ser ejecutada por un servidor de integración continua, como por ejemplo Jenkins, Bamboo o Hudson (ver sección 2.4.4), ya que estos servidores de integración continua son capaces de detectar y actuar ante los cambios en el control de versiones, como por ejemplo commit realizado por un desarrollador, que desencadenará la descarga de dependencias por repositorios de terceros, la ejecución de tareas de configuración para construcción y el traslado del resultante al ambiente de pruebas. Si se obtiene un resultado negativo de la construcción, se debe anular el commit y se puede utilizar la herramienta de mensajería elegida en la etapa de planeación para informar del fallo a los interesados, para que esto sea resuelto de forma manual. Al igual que en el proceso anterior, dependerá específicamente del lenguaje de desarrollo, la elección del lenguaje de construcción y del servidor de integración continua, no hay uno específico que sea la mejor opción para todos los equipos. En la sección 2.4.2 se mencionaron las principales herramientas de para construcción y configuración de repositorios.

4.3.2.4 Paso 4: Pruebas

En el proceso del flujo continuo, la siguiente etapa son las pruebas. Al tener una versión construida y aprobada de forma automática por el servidor de integración continua y puesta por este en ambiente de pruebas, ya se inicia con los diferentes testeos. Estas pruebas deben estar escritas en código para ser ejecutadas de manera automatizada y rápida. Las pruebas pueden ser:

1. Pruebas unitarias: garantizan las menores unidades de la aplicación, clases, validación de tamaño de campos, validación de obligatoriedad de los campos, entre otras.
2. Pruebas de integración: certifican la integración entre las aplicaciones dentro de un ecosistema de los sistemas.
3. Pruebas funcionales: atestiguan el buen desempeño de las funcionalidades.

4. Pruebas visuales/UI: garantizan el diseño y los elementos estáticos de una página o app móvil, por medio de comparar con el prototipo.
5. Pruebas de rendimiento: aseguran el rendimiento del release/paquete a partir de una línea base definida y/o también a través de comparativas con el release o el paquete anteriormente construido.
6. Pruebas de carga y estrés: se encargan de validar el comportamiento de la aplicación cuando la misma es sobrecargada.

Posterior a la ejecución de estas pruebas, se valida que el producto cuenta con las métricas mínimas que se deben cumplir para dar como aprobada la versión construida. Al igual que en la etapa de construcción, si se tienen problemas en alguna prueba, se rechazará la versión construida y se informará por medio de la herramienta de mensajería elegida en la etapa de planeación, acerca del fallo a los interesados, para que sea revisado y posteriormente resuelto. Si la versión construida aprueba todos los test y supera las métricas mínimas, será tomada como una versión correcta y el código se integrará en el control de versiones.

Es importante tomar en cuenta que existen algunos criterios de pruebas que se salen del enfoque del programador, por lo que el rol de Tester es importante en esta etapa, tanto para crear nuevas pruebas automatizadas, que no están consideradas inicialmente por el desarrollador, como también para realizar pruebas externas que son difíciles de automatizar en scripts.

Tener automatizadas las pruebas permitirán que el equipo no necesite realizar de manera manual algo que ya se había probado anteriormente además da la posibilidad de probar cada ambiente, pruebas por pipeline, feedback casi automático, pruebas más amplias que garantizan que lo que ya existe es estable, y esto al final cumple el objetivo del flujo continuo para la producción de releases más rápido. El QA es el dueño principal y mayoritario de este proceso.

4.3.2.5 Paso 5: Empaquetado

Cada etapa de construcción y testeo se finaliza con la producción de un entregable. Con el crecimiento de la computación en la nube y los microservicios, se hizo popular el empaquetado por medio de contenedores. El objetivo de crear contenedores para cada entregable es poder empaquetar una versión estable del software construido con todas sus

dependencias, con el aseguramiento que este contenedor es ejecutado en cada ambiente de infraestructura de la misma manera, esto elimina la clásica frase de los desarrolladores “en mi computadora si funciona” ya que es exactamente lo mismo ejecutándose en un ambiente de desarrollo, en un ambiente de pruebas, en un ambiente de certificación y en un ambiente de producción.

Existen herramientas para construir contenedores con aplicaciones dentro, que se mencionaron en la sección 2.4.4, la más popular de ellas es Docker, misma que crea imágenes por medio de un script de configuración, con microservicios o aplicaciones de manera implícita. Estas imágenes son colocadas en repositorios para que puedan ser descargadas y desplegadas, sin necesidad de ejecutar el script de configuración. El crecimiento en popularidad de esta herramienta ha permitido que los repositorios de imágenes de Docker existan casi cualquier tipo de software necesario para el funcionamiento de las aplicaciones modernas, tales como servidores, gestores de control de versiones, repositorios, dependencias, librerías, etc.

Empaquetar una aplicación en Contenedores permite Portabilidad multiplataforma ya que las imágenes creadas pueden ser ejecutadas en contenedores de Linux, MacOS y Windows, también apoya la Ligereza ya que los contenedores ahorrar recursos, al utilizar solo los necesarios y no un Sistema Operativo completo, proporcionan Aislamiento ya que los contenedores crean una capa de protección entre ellos, incentivan la agilidad en hot-fixes y mantenimiento ya que se puede hacer rollback entre versiones y balanceo de carga lo cual da alta disponibilidad. Es importante tomar en cuenta que el proceso de Empaquetado no lleva una lógica en línea de tiempo, sino que va implícita en la construcción y testeo de la integración y entrega continua.

4.3.2.6 Paso 6: Configuración

El proceso de Configuración al igual que el empaquetado, no va en una línea del tiempo, sino que se configura implícitamente en el proceso. La configuración es básicamente tratar la infraestructura como código, para que los profesionales de Operaciones, puedan configurar por medio de scripts los ambientes desde el punto de vista de Infraestructura. Con el auge de la computación en la nube, es común que toda la configuración, armamento y gestión sea por

medio de consolas remotas, esto ha provocado que la orquestación de los servicios para los entornos de ejecución de la organización sea remota y por ende deba ser automatizada.

El objetivo primordial es proporcionar un modo estándar, consistente y automático de realización de los despliegues y sus configuraciones. En la sección 2.4.6 y 3.6.1.3 se mencionaron las principales herramientas de Infraestructura como código, así como una explicación de en qué consiste. Este proceso es el primero en el que el dueño es principalmente Operaciones.

4.3.2.7 Paso 7: Despliegue y Liberación

Finalmente, al tener una versión del software en ambiente de certificación que ha sido construido, testeado y empaquetado de manera automatizada, se ejecuta el proceso de Despliegue y Liberación, que es ejecutado automáticamente por el servidor de integración continua, por medio de transferir la imagen a ambiente de producción.

Este proceso al igual que el de integración continua, debe ser tomado como algo con total normalidad y regularidad como parte del día a día ya que como filosofía ágil, lo que se pretende es poder responder rápido al cambio. Se pueden hacer entre 1 y 20 releases aproximados en el día, y depende directamente del equipo y la naturaleza del software. Se debe romper la barrera del miedo e incertidumbre que usualmente existe al desplegar cambios en producción, con la total confianza de que en cualquier momento se puede rápidamente hacer un rollback hacia la última versión estable. Es por esta cultura DevOps que muchas aplicaciones en la nube hoy en día se actualizan todo el tiempo. El proceso de actualización es tan constante porque el proceso de construcción, integración, testeo y despliegue es constante.

El servidor de Integración Continua debe de tener todos los accesos, permisos y configuraciones necesarias para que el proceso de pase a producción sea ejecutado por él, con esto se evitan los errores humanos al olvidar colocar una configuración o un fichero. En la figura 10 se puede observar la distribución de los errores al momento de desplegar, vemos que un gran porcentaje es acreditado a los errores humanos y otra parte a las dependencias o parches extraviados, esto se evita completamente con el Despliegue Continuo y con el empaquetado mencionado en el proceso anterior. En la actualidad la mayoría de despliegues se hacen sobre ambientes de computación en la nube, por lo que en esta etapa se consideran

herramientas de orquestación de servicios y multicloud. En la sección 2.4.5 se mencionaron las herramientas para el Despliegue.

Figura 15 - Distribución de Errores al Desplegar, Fuente: (Forrester, 2016), traducción personal

4.3.3 Etapa 3: Monitorización

Uno de los muros que DevOps trata de romper es la falta de feedback sobre las aplicaciones puestas en producción una vez se ha realizado el despliegue por parte del departamento de operaciones. Esta retroalimentación es difícil de conseguir en el momento adecuado, abarca diferentes aspectos del producto, como rendimiento, experiencia de usuario, funcionalidades, entre otros. La monitorización debe ser algo transversal y debe llegar sin filtros al departamento de desarrollo que podrán detectar e interpretar los resultados del día a día de sus desarrollos.

Existen básicamente tres tipos de monitorización que deben de existir:

1. Monitorización del rendimiento de aplicaciones: En este tipo de Monitorización, el principal objetivo es evaluar cómo se comportan las aplicaciones en los ambientes de producción. Se evalúa la potencia del Hardware para exponer el Software, así como el nivel de estrés y carga, y también el comportamiento de los balanceadores de carga y los orquestadores de servicios. Herramientas como Zabbix, Nagios, Influxdb, Telegraf o Grafana son muy populares para esta tarea.
2. Monitorización a través de Logs: Es común que operaciones tenga que estar en constante monitoreo de los Logs de aplicación ya que son ellos los que informan primeramente de lo que sucede en la aplicación. Existen herramientas como GrayLog,

Logstash, Kibana o Elasticsearch, que sirven en buena forma para esta tarea y facilitan este tipo de monitorización.

3. Monitorización de la experiencia de usuario y analíticas web: finalmente el tercer tipo de monitorización que se debe hacer, va enfocado a la percepción de los usuarios de la aplicación. Existen varias herramientas que cumplen con este objetivo, sin embargo, Piwik y Google Web Analytics son las más populares. Dependerá de la naturaleza de la aplicación, las herramientas de este tipo de monitoreo que se empleen. Además, se pueden considerar monitoreos específicos por medio de entrevistas de satisfacción para obtener el feedback deseado.

4.3.3.1 Paso 1: Ciclo de mejora continua

Posterior a la implementación se debe realizar planificación continua para mejorar permanentemente los procesos, esta es una tarea primordial realizada por todo el equipo. Al implementarse metodologías ágiles como Scrum en conjunto con DevOps, se puede utilizar un corto tiempo de los Sprints para mejorar los procesos en cada equipo Scrum. El Product y Service Owner serán los encargados de identificar correctamente los puntos de mejora y agregarlos al Product y Service Backlog como algo que se debe mejorar, y le colocarán la importancia en prioridad que considere necesario. DevOps es agilidad por lo que el equipo debe ser autogestionado para implementar los cambios necesarios en el menor tiempo posible.

4.3.3.2 Paso 2: Evaluación de métricas de desempeño

Uno de los principios de DevOps es ser medible, lo que conlleva definir y monitorizar métricas de desempeño. En la sección 3.2.5 se describieron algunas preguntas que deben ser respondidas por medio de DevOps. La naturaleza de estas mediciones dependerá específicamente del equipo y del software desarrollado, ya que las prioridades serán diferentes en cada uno de ellos. Sin embargo, es importante que se definan las métricas necesarias y que estas sean constantemente monitorizadas para evaluar si se ha cumplido o no con el rendimiento esperado. De igual forma, será el Product y Service Owner, los encargados de la definición de las métricas, así como de velar por su cumplimiento.

Las métricas mínimas que deben ser consideradas son:

1. Tiempo en el desarrollo de nuevas funcionalidades
2. Costos en la detección y corrección de errores
3. Tiempo fuera de línea
4. Esfuerzo en construcción y configuración de nuevos ambientes con réplicas de existentes
5. Tiempo invertido en pruebas vrs. cantidad de pruebas realizadas
6. Costos frente al cambio
7. Número de errores no atendidos
8. Cantidad de clientes insatisfechos
9. Tiempo para integrar y desplegar nuevas versiones

Todas estas métricas deben ser cuidadosamente monitorizadas para garantizar que la implementación de DevOps ha generado los resultados esperados.

4.3.3.3 Paso 3: Análisis de cambio

Como se pudo apreciar, los procesos ágiles en combinación con procesos DevOps dan como resultado una diferencia notable en comparación con los procesos tradicionales, que son por lo general planificados de manera completa y específica, respetados por fases secuenciales, así como por roles y responsabilidades fuertemente acentuados, estos acompañados de una documentación completa y en ocasiones muy difícil de mantener, en donde la recopilación de requerimientos se hace en fases iniciales y es hasta fases finales en el que se inicia la entrega de aplicaciones al usuario final. DevOps por medio de automatización y agilidad, entrega funcionalidades todas las semanas, más de 20 versiones por día, ambientes autogestionados y rápidamente construidos, con pruebas automatizadas y equipos de operaciones que tienen la capacidad de resolver problemas directamente de ser necesario, así como desarrolladores involucrados con los ambientes de producción.

Los procesos deben ser adaptados de manera secuencial, como se describieron en la presente guía y en el menor tiempo posible. Un equipo que trabaja con metodologías tradicionales tiene que tener la disponibilidad para aprender e implementar todos los principios, aspectos, procesos y herramientas descritos en la presente guía para poder hacer DevOps.

4.4 Caso de Éxito

La presente guía fue implementada en el departamento de informática de una institución pública del Gobierno de Guatemala. Se realizó con especial énfasis en los procesos del área de desarrollo y su relación de entrega en operaciones. La iniciativa se formuló posterior a la detección de problemas para la resolución de hot-fixes, así como el desarrollo de nuevas funcionalidades. Además, se detectaron problemas en los sistemas de gestión y control de versiones, así como los releases que entrega Desarrollo a Operaciones. A continuación, se presenta un resumen de las tareas más notables de implementación:

1. En el proceso de **Planificación** se realizó un análisis del funcionamiento del equipo, determinando que se ejecutaba una mezcla de dos metodologías tradicionales, en cascada y de prototipos además de algunos procesos de RUP. Las etapas iniciales del ciclo de vida de desarrollo se hacían por medio de documentos no estandarizados y parte de la diagramación de UML. El diseño era realizado por mockups y posterior a esto se iniciaba con la codificación, pruebas y entrega del software. El proceso de operaciones para la puesta en producción y soporte, era totalmente aislado. Esto fue cambiado por Scrum. Se asignó el Rol de Product Owner y Scrum Master para desarrollo, así como Service Owner y Scrum Master para operaciones. Se asignaron y estandarizaron los documentos a utilizar para análisis y se implementó el Tablero Scrum/Kanban para gestionar el Product Backlog por medio de Trello. La comunicación se estandarizó por WhatsApp Web y la colaboración por medio de HipChat con la implementación de I Done This, técnica que lleva una bitácora de lo realizado todos los días. Se implementaron reuniones diarias, semanales y mensuales como manda Scrum, intentando en la medida de lo posible alinear las metas de los departamentos de desarrollo y operaciones.

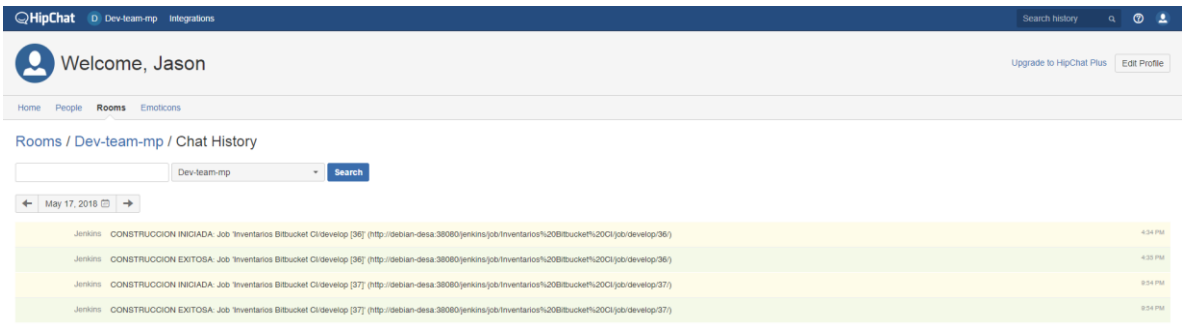

Figura 16 - Integración Jenkins – HipChat Caso de Éxito, Fuente (personal)

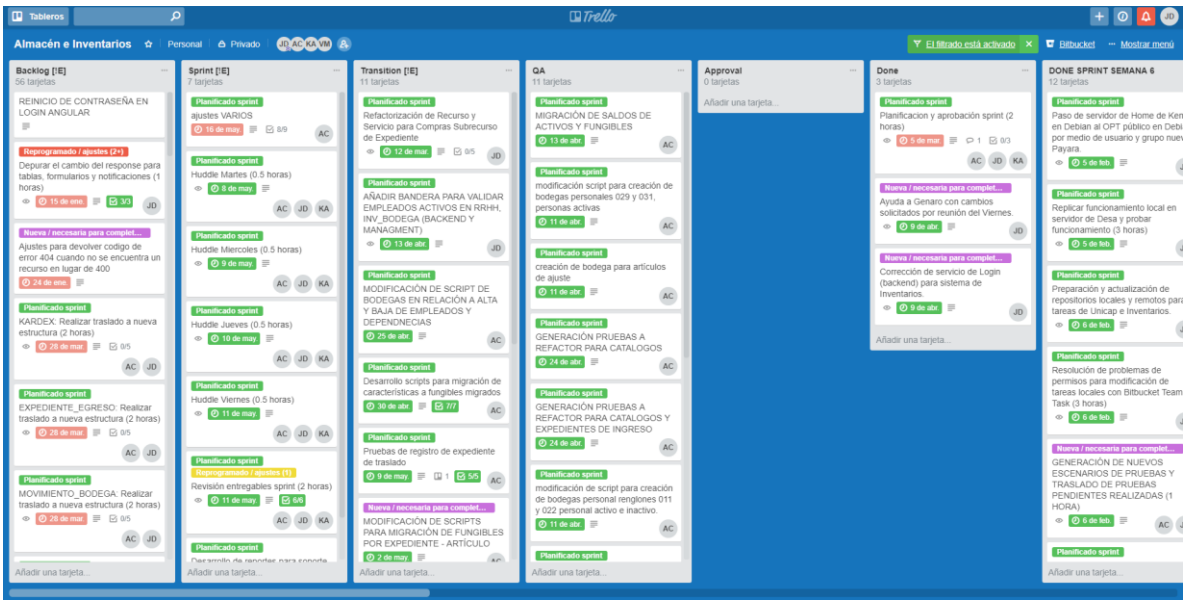

Figura 17 - Tableros Trello Caso de Éxito, Fuente (personal)

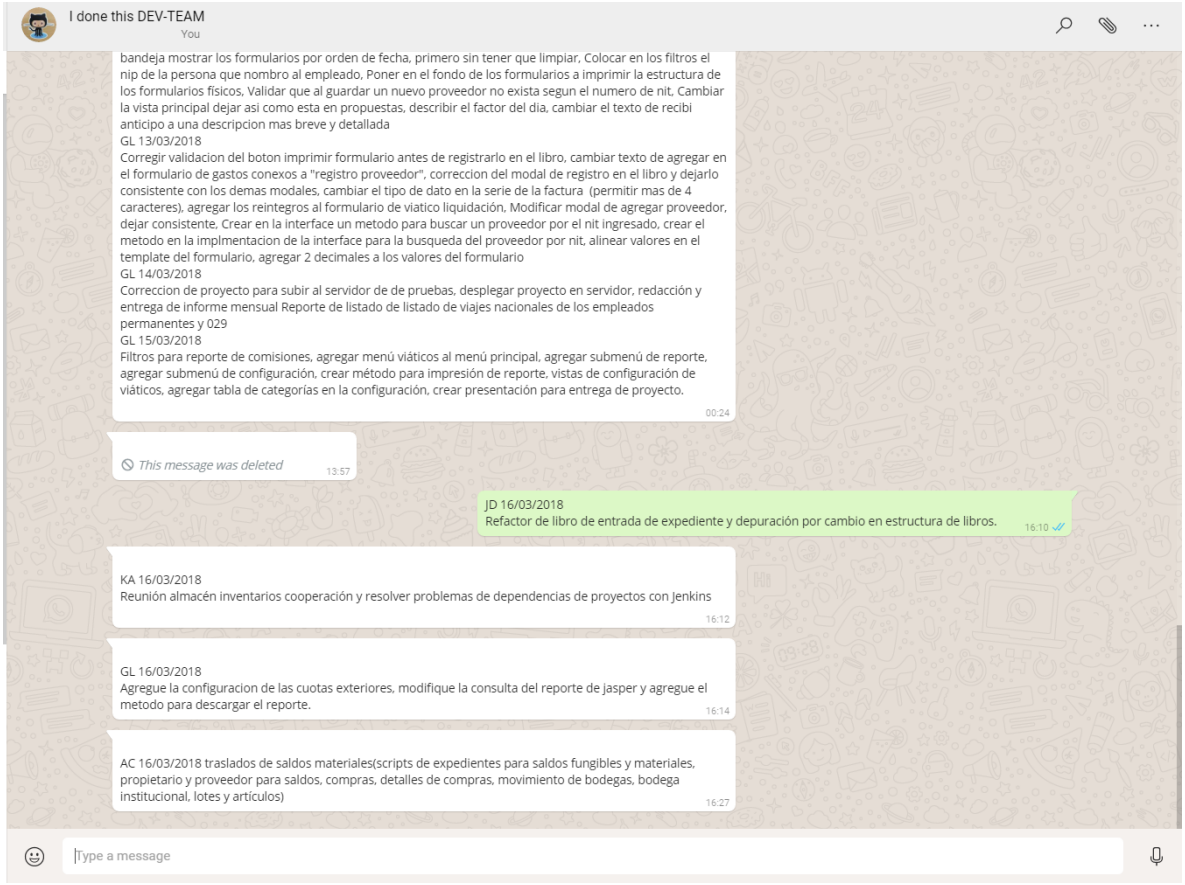

Figura 18 - Implementación I Done This WhatsApp Caso de Éxito, Fuente (personal)

2. En el proceso de **Control de versiones**, se determinó que actualmente se continúa trabajado con Subversion SVN como herramienta SCM para gestionar las versiones del código, sin embargo, se inició con el cambio a Git por medio de Bitbucket para los últimos tres proyectos que se han iniciado. Se planea en los próximos meses finalizar el traslado de proyectos a Git. Se implementó Brach-Flow en Subversion como estrategia de control de versiones y Git-Flow en Bitbucket; esto permite tener un mejor control sobre las ramas en donde se encuentra el código estable e inestable. Para Subversion el Trunk tiene la versión estable y para Bitbucket será la rama Master. La versión con el código de Producción se encontrará en una rama Release con un Tag específico tanto para Subversion como para Bitbucket.

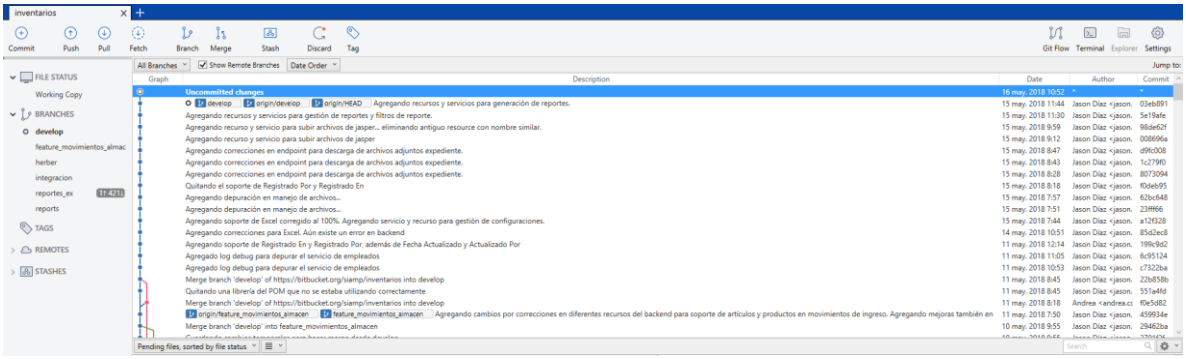

Figura 19 - Source Tree Git y Subversion Caso de Éxito, Fuente (personal)

3. En el proceso de **Desarrollo** se eligió el IDE Netbeans para codificar Backend en Java EE con JPA y se eligió a Visual Studio Code para codificar Frontend en JavaScript con AngularJS. Las dos herramientas permiten un correcto control del código con interacción de Git para el sistema de control de versiones y una consola para la ejecución de comandos.
4. En el proceso de **Construcción** se implementó Jenkins como Servidor de Integración y Despliegue Continuo. Se configuraron Pipelines declarativos por medio de Jenkinsfile que permiten integrar el código de Bitbucket y de Subversion de manera automática y construir versiones por medio de Gulp para frontend y Maven para Backend. Jenkins se encarga de estar pendiente de los commit realizados en los branchs de código estable para realizar el proceso de construcción, testeo y despliegue.

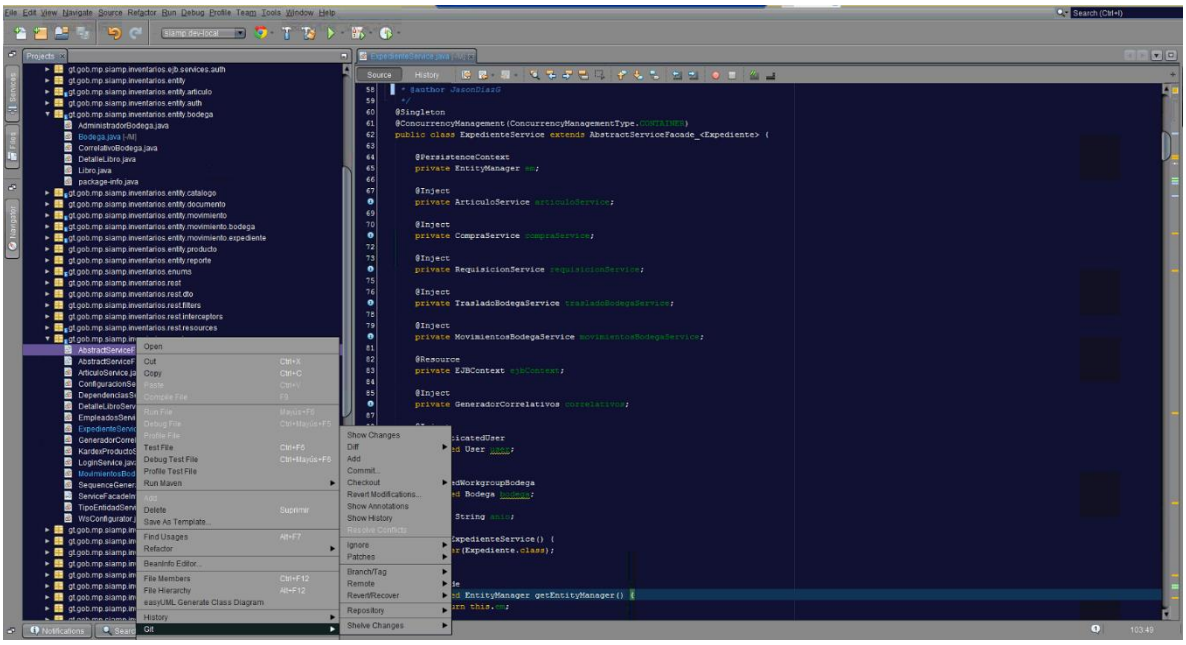

Figura 20 - Netbeans integrado a Subversion Caso de Éxito, Fuente (personal)

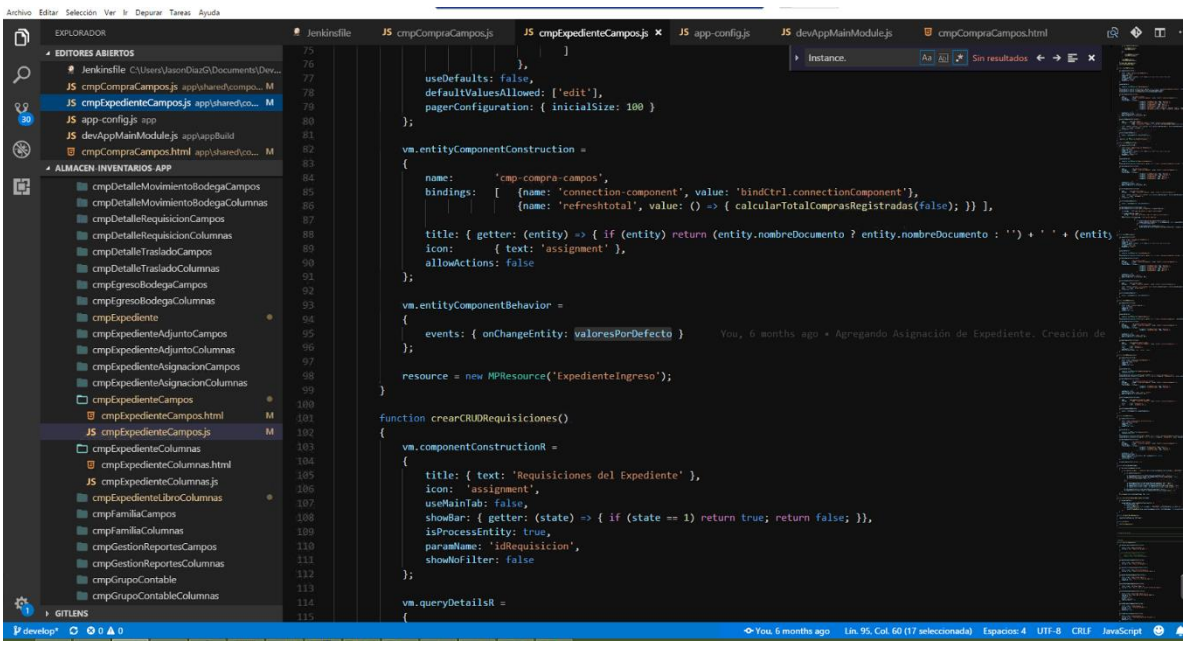

Figura 21 - Visual Studio Code integrado a Git, Caso de Éxito, Fuente (personal)

5. En el proceso de **Pruebas** se realizaron pruebas unitarias en JUnit para el backend y Jasmine para el frontend. Estas pruebas fueron codificadas en scripts y Jenkins es el encargado de ejecutarlas en el momento de realizar la integración continua.
6. En el proceso de **Empaquetado** se usó Docker para crear contenedores por medio de imágenes en los diferentes ambientes. Estas imágenes son creadas por medio de un

dockerfile que ejecuta Jenkins y el cual es configurado en los diferentes ambientes de Desarrollo, Certificación y Producción.

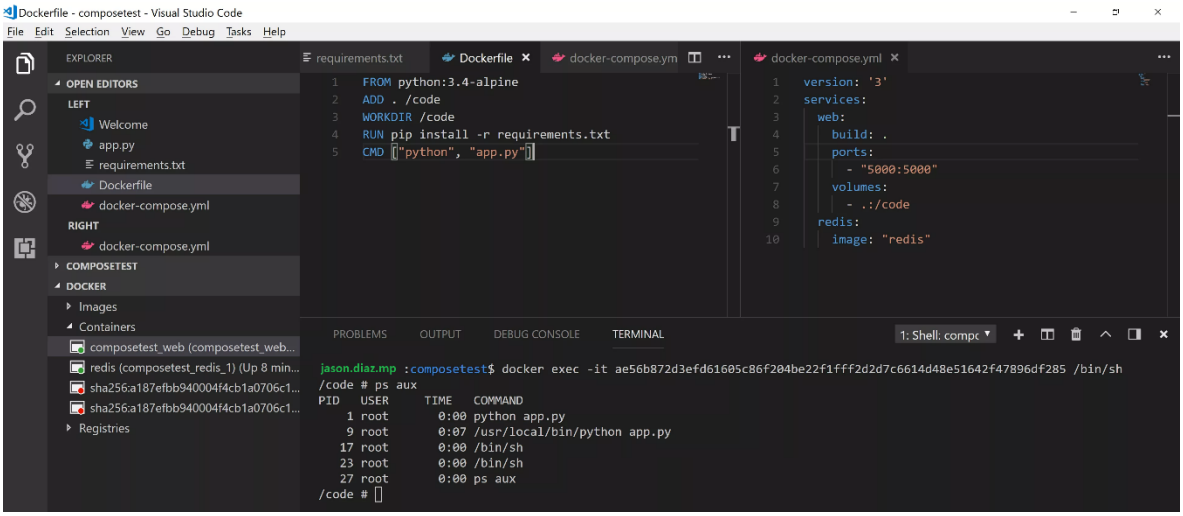

Figura 22 - Docker File Caso de Éxito, Fuente (personal)

7. En el proceso de **Configuración** se trabajó en Operaciones Puppet Labs para crear ambientes de Desarrollo y Certificación por medio de scripts codificados. Los ambientes de producción aún son revisados de manera manual por la complejidad y relevancia de los mismos. Sin embargo, en meses posteriores se tiene planeado pasar un proceso de certificación para tener entera confianza de crear con Infraestructura como Código por medio de los mismos scripts, los ambientes de Producción.

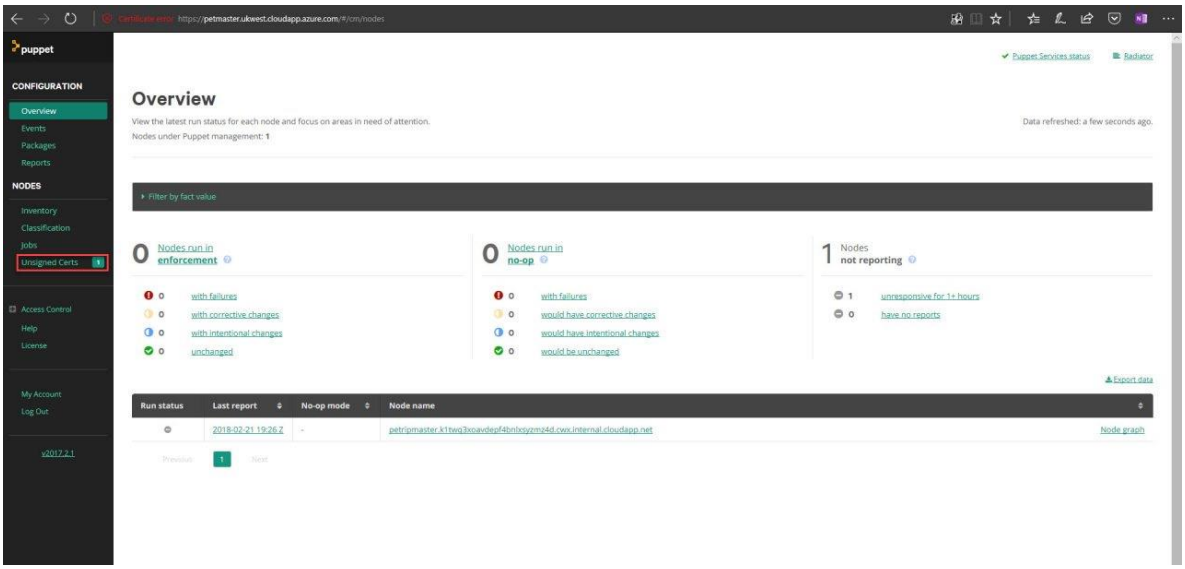

Figura 23 - Puppet Labs Caso de Éxito, Fuente (personal)

8. En el proceso de **Despliegue**, se realizó con Jenkins el alza y baja de aplicaciones. El servidor de Integración Continua es el encargado de colocar los contenedores Docker en los diferentes ambientes, que llevan ya configuradas las API's y App's construidas de manera estable o cualquier otro servicio. Al ser una institución gubernamental, los despliegues son realizados en infraestructura interna, por lo que no se tuvo interacción con ningún proveedor de IaaS con Cloud Computing.

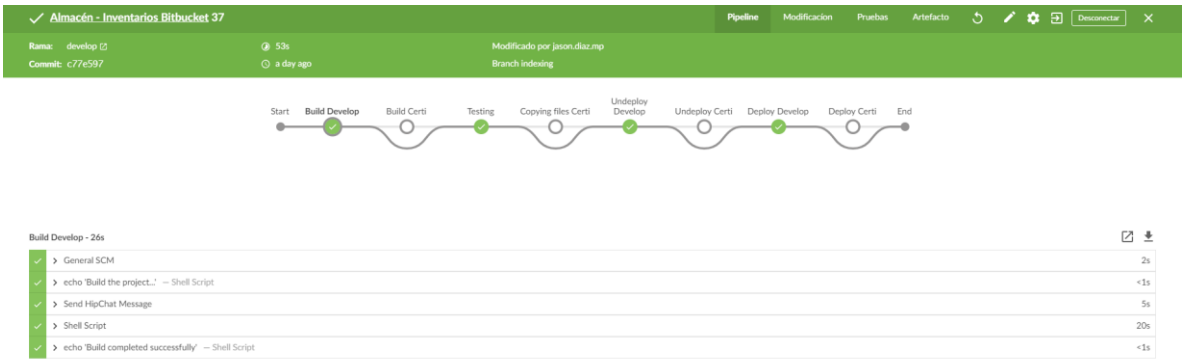

Figura 24 - Jenkins Pipeline Blue Ocean Caso de Éxito, Fuente (personal)

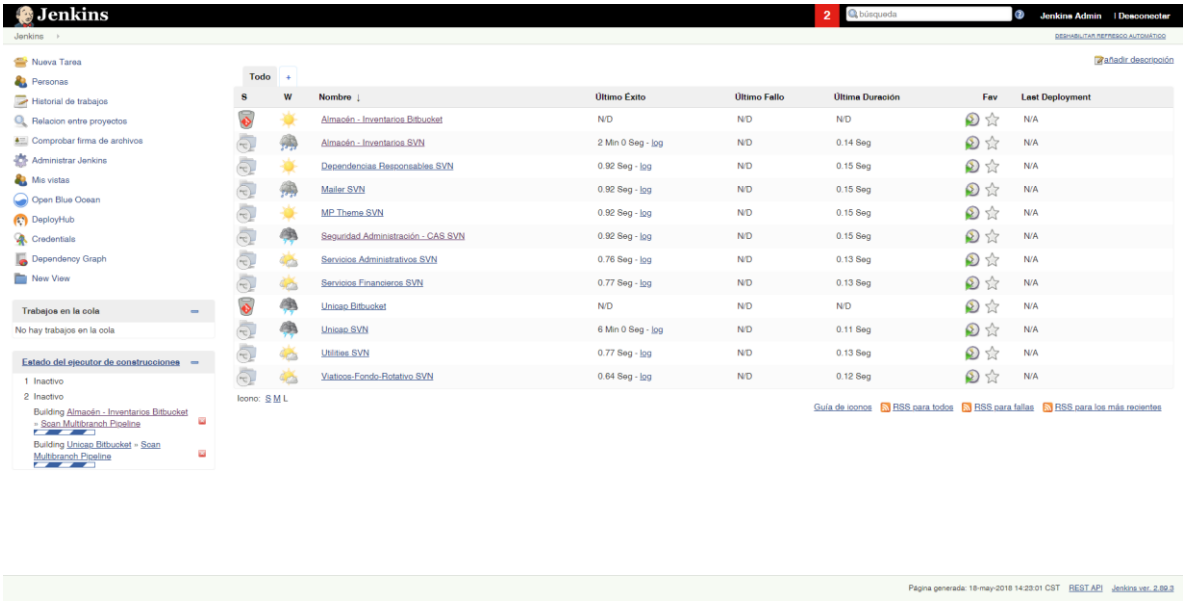

Figura 25 - Administración de Jenkins Caso de Éxito, Fuente (personal)

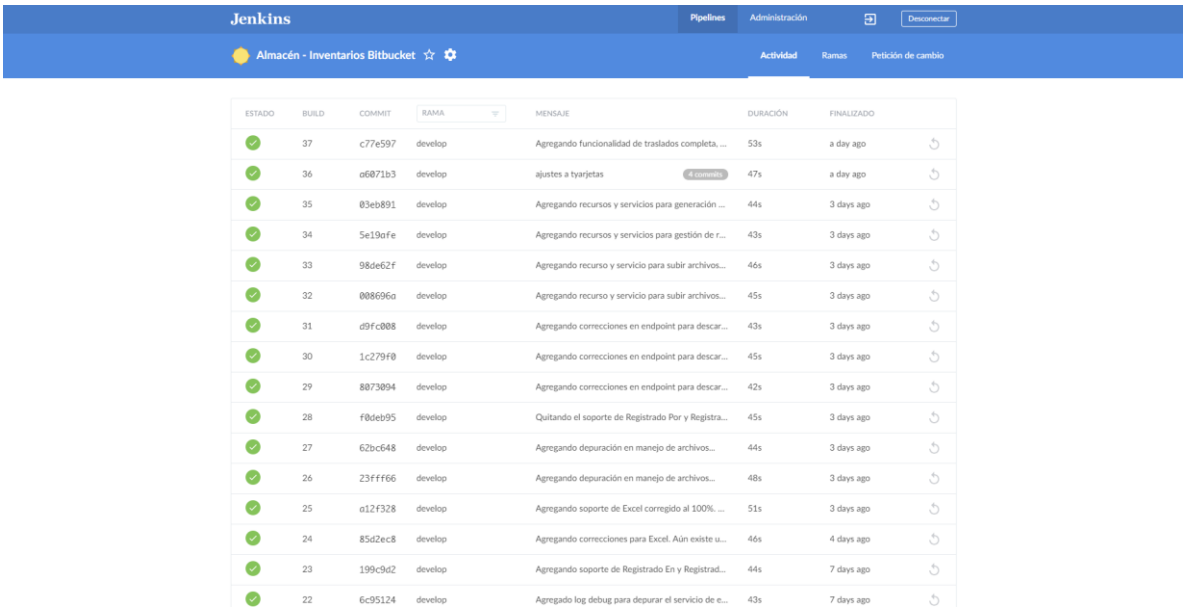

Figura 26 - Historico de Despliegues Jenkins Caso de Éxito, Fuente (personal)

9. En el proceso de **Monitorización**, se implementó la herramienta de revisión y monitoreo Logstash, al ser la herramienta de código abierto más reconocida en el ámbito de monitoreo de backend, en este caso Java. Organiza los logs y los presenta de una manera mucho más útil para el Service Owner de Operaciones. Además, se implementó un sistema interno nuevo, que ayuda con la asignación de tickets de soporte y permite una interacción más eficiente entre Operaciones y Desarrollo, a tal punto que hay tickets que pueden ser resueltos directamente por Desarrolladores.

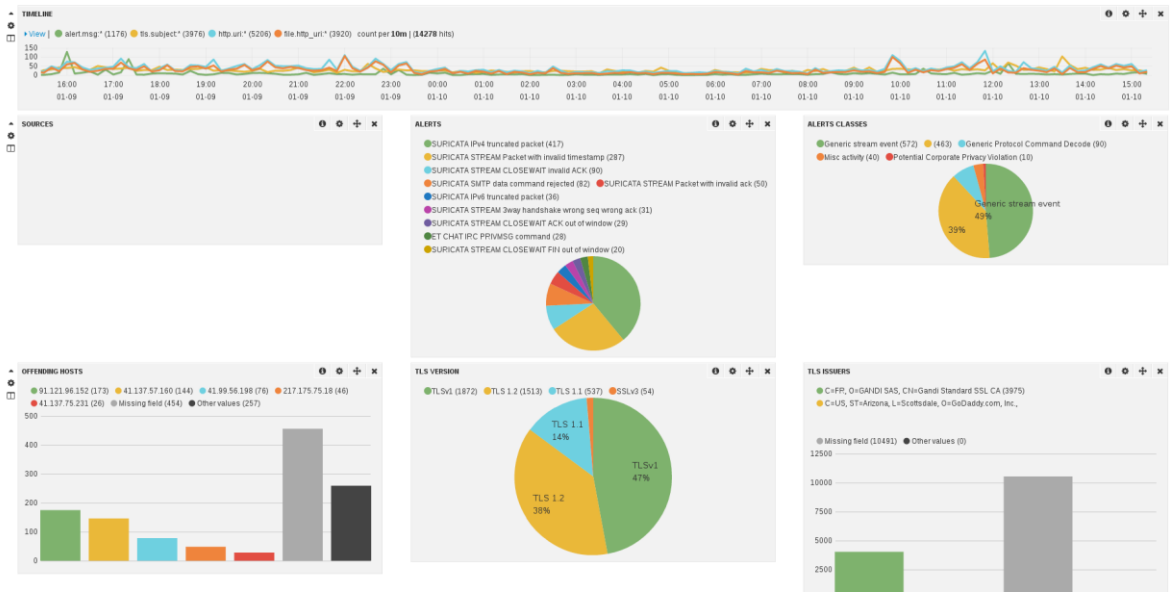

Figura 27- LogStash Caso de Éxito, Fuente (personal)

10. Finalmente, en el proceso de **Mejora Continua**, se implementó la etapa de retrospectiva mensual que sugiere Scrum, en la que cada líder de subequipo analiza con el resto del equipo, todas las oportunidades de mejora que se tuvieron en el mes que acabó, y posterior a esta reunión, cada líder de equipo se reúne con el líder del departamento para proponer nuevas mejoras o soluciones de innovación para mejorar el flujo de los procesos. Además de esto, se dedican unas horas al mes para que se compartan conocimientos adquiridos en todo el equipo, además de 4 pequeños convivios planificados en el año para que se socialice con aquellos miembros del equipo con el que no se tiene mucha interacción por cuestiones de trabajo. Con esto los resultados fueron relaciones más fuertes y mejores entre los miembros.

A continuación, se presenta una tabla con los resultados más notables:

Métrica	**Sin DevOps**	**Con DevOps**
Tiempo en el desarrollo de nuevas funcionalidades	1 mes para generar resultados tangibles	1 semana para generar resultados tangibles
Costos en la detección y corrección de errores	1 o 2 días dependiendo de la complejidad del problema	1 o 4 horas dependiendo de la complejidad del problema
Tiempo fuera de línea	½ día para restablecer	2 horas para restablecer
Esfuerzo en construcción y configuración de nuevos ambientes con réplicas de existentes	1 o 2 días dependiendo de la complejidad del ambiente	2 horas o ½ día dependiendo de la complejidad del ambiente
Tiempo invertido en pruebas vrs. cantidad de pruebas realizadas	2 o 3 días dependiendo de las pruebas	2 horas o ½ día dependiendo de las pruebas
Costos frente al cambio	Altos por reestructuración de análisis y aumenta si es en etapas finales	Bajos porque se realizan entregas semanales a los usuarios finales
Número de errores no atendidos	2 a 4 por semana	2 a 4 por mes
Cantidad de clientes insatisfechos	Media	Baja
Tiempo para integrar y desplegar nuevas versiones	1 o 2 días dependiendo del sistema	1 o 2 horas dependiendo del sistema

Tabla 6 - Métricas de Implementación en Caso de éxito, Fuente: Personal

5. Conclusiones

Como producto del presente trabajo de graduación se obtuvieron las siguientes conclusiones:

1. Los modelos de desarrollo de software son necesarios para tener un marco de trabajo integral y una guía correcta para desarrollar software. El uso de metodologías de desarrollo de software, permite a los equipos manejar una secuencia lógica en los procedimientos desde la concepción hasta la entrega de software. La naturaleza del software determinará que metodología se adaptará más a determinado proyecto.
2. Los roles y responsabilidades se encuentran marcadamente definidos en gran parte de los equipos de entrega de software, en la mayoría de casos son divididos por los departamentos a los que pertenecen; en DevOps se resalta la importancia de formar miembros que sean multidisciplinarios con la capacidad de compartir metas y objetivos de tal manera que la ayuda y comunicación fluya a través de todos departamentos involucrados.
3. Tanto las metodologías tradicionales como ágiles, se enfocan exclusivamente en el departamento de desarrollo y dejan por fuera el resto de funcionamiento de un equipo de desarrollo de software. DevOps permite romper esta brecha divisoria, por medio de la alineación de objetivos y de crear una visión integral para todos los involucrados, diseñando procesos automatizados que cultivan una cultura colaborativa en todo el equipo.
4. DevOps es comúnmente confundido con un rol, herramientas o automatización, sin embargo, es un movimiento que utiliza y ordena estas partes de tal manera que promueve una cultura que ayuda a quebrantar las barreras entre Desarrollo y Operaciones. Presenta beneficios considerables, siendo el principal de ellos el tiempo de respuesta para desplegar software nuevo y mejoras, no obstante, presenta desafíos organizacionales, procedurales y tecnológicos que deben ser tratados y resueltos para lograr una implementación exitosa.
5. DevOps es integral y no excluyente, y conlleva el involucramiento de todo el gobierno de IT como lo son desarrollo, operaciones, seguridad y QA.

6. No existen las herramientas DevOps como tal, sin embargo, si existe la cadena de herramientas DevOps, que clasifica las diferentes herramientas que ayudan a la implementación en cada área particular de producción de software. La elección de estas herramientas va a depender directamente de la manera particular de hacer software en cada equipo. No es indispensable que cada área cuente con una o muchas herramientas, sin embargo, las herramientas ayudan a que los resultados de DevOps sean considerables.
7. No es posible realizar la implementación de una cultura si no se tienen principios. En el caso de DevOps C.A.L.M.S. define exactamente los elementos imprescindibles en cualquier implementación ya que son la esencia de DevOps. Los cinco principios Cultura, Automatización, Intangible, Medición y Compartir deben ser instaurados al inicio de la implementación y perfeccionados a través del tiempo.
8. Para implementar correctamente DevOps se debe Planificar, Ejecutar y Monitorizar. Al definir roles y responsabilidades en la Planeación hay seguridad en la alineación de objetivos. Cuando se implementan herramientas en cada etapa de la concepción del software, en la Ejecución, se automatiza completamente el proceso de inicio a fin. Finalmente, al implementar una etapa de mejora continua en conjunto con el monitoreo de las partes críticas, se tendrá un ecosistema completo que ampliará el rendimiento del equipo y dará fiabilidad a la calidad del software desarrollado.
9. Entre las principales mejoras en rendimiento que facilita DevOps tenemos menos tiempo para desarrollar nuevas funcionalidades y para invertir en pruebas, integración y despliegue de nuevas versiones; también tenemos un menor costo para la detección y corrección de errores, así como menor esfuerzo en la replicación y construcción de ambientes; además, al ser ágiles se reducen los costos frente al cambio, así como el número de errores no atendidos y de clientes insatisfechos.

6. Recomendaciones

Como producto del presente trabajo de graduación se recomienda al lector que desee implementar DevOps:

1. Se recomienda hacer un análisis profundo de la presente guía para quien desee implementar DevOps, así como para quien desee tener más conocimiento en el área del desarrollo moderno y ágil. Es importante que el lector haga énfasis en los principios en los que se fundamenta el cimiento de DevOps, ya que será imposible tener éxito en la implementación sin la esencia clara y presente en el equipo.
2. Se recomienda tener completamente claro el concepto de lo que es DevOps y sobre todo de lo que no es, ya que es común el tender a confundirlo con un rol o con un conjunto de herramientas; es verdad que existen certificaciones para DevOps, sin embargo, el enfoque está orientado a tener la capacidad de poder implementar la cultura y/o formar parte de un equipo que ya la implemente, no obstante, no existe en un equipo el rol DevOps o una herramienta DevOps.
3. Es importante que, si se desea implementar esta cultura, se combine con una o varias metodologías ágiles. La era de las metodologías ágiles ha tenido auge en la última década, el manifiesto ágil fue elaborado y redactado en 2001, y desde la fecha han surgido varias metodologías ágiles de desarrollo. Las técnicas sugeridas por estas metodologías hacen de la implementación de DevOps un proceso más rápido y, sobre todo, más eficiente. El lector debe comprender que DevOps no es una metodología propiamente dicha, pero si se ajusta completamente a los procesos dictados por ellas.
4. El software es de naturaleza cambiante, por tal razón el ser ágiles es una obligación, crear equipos autogestionados, multidisciplinarios y autodidactas puede marcar la diferencia entre el éxito o el fracaso de un proyecto. Se recomienda implementar técnicas para compartir conocimiento nuevo y fresco que cada uno de los miembros adquiera, fomentando la colaboración entre los equipos y compartiendo metas y obligaciones entre los equipos.
5. Se recomienda dedicar un tiempo considerable a la innovación, ya que las exigencias del mundo tecnológico actual, provoca que el tiempo en que el software pasa de ser

nuevo a obsoleto sea menor, y esto a su vez genera que la falta de innovación pueda provocar el fracaso de un proyecto. Se debe innovar implementando nuevas herramientas y técnicas para mejorar los procesos y procedimientos dentro del equipo y también el producto final.

6. DevOps produce visibilidad en todas las iteraciones del proyecto, por medio del flujo continuo, todos los interesados pueden saber el estatus actual, así como tener una versión funcional del software en cualquier momento. Se recomienda monitorizar por medio de métricas todos los procesos, que proporcionen una idea clara del rendimiento del equipo, así como del software mismo. Si no se miden los procesos, es imposible saber si se está mejorando, manteniendo o empeorando.
7. Finalmente, al tener todos los procesos produciendo servicios y productos de calidad, se recomienda certificarlos. Este proceso de certificación sale del alcance del presente trabajo de graduación, sin embargo, se detectó una oportunidad de ampliación o mejora al mismo. Existen certificaciones internacionales para servicios como la ISO 20000 que podría validar las métricas fijadas para procesos, servicios, personas y para el producto mismo, por tal razón, se recomienda indagar y profundizar en este tema para tener una implementación certificada, que provea seguridad a proveedores y clientes externos.

7. Glosario

Ágil: Que funciona de manera efectiva y rápida. Que entiende las cosas con facilidad y piensa y actúa con rapidez. Que implica o denota esta facilidad. En el Software, la agilidad está en términos del funcionamiento del equipo, con la frecuencia en que entrega software funcional, la actitud frente a la aceptación de los cambios, la flexibilidad con respecto a la cantidad de documentación necesaria, entre otros.

Análisis: Examen detallado de una cosa para conocer sus características o cualidades, o su estado, y extraer conclusiones, que se realiza separando o considerando por separado las partes que la constituyen. Documento escrito en que se detalla ese examen. En el Desarrollo de Software, comprende el proceso de recabar requerimientos, descomponerlos, ordenarlos y priorizarlos clasificándolos en funcionales y no funcionales.

Automatización: Aplicación de máquinas o de procedimientos automáticos en la realización de un proceso o en una industria. En software es aplicado para la optimización de procesos internos del software.

Contenedor: Que contiene. Contendor Docker son recipientes virtuales ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

Orquestación de contenedores: Es un método de virtualización de nivel de sistema operativo (nivel OS) para implementar y ejecutar aplicaciones distribuidas sin lanzar una máquina virtual completa (VM) para cada aplicación.

Cultura: Proviene del latín cultus, hace referencia al cultivo del espíritu humano y de las facultades intelectuales del hombre. Es una especie de tejido social que abarca las distintas formas y expresiones de una sociedad determinada.

Cultura de Equipo: El conjunto de normas, de valores y de formas de pensar que caracterizan el comportamiento, posicionamiento del personal en todos los niveles de un equipo, el estilo de dirección, la forma de asignar los recursos, así como el funcionamiento en general del equipo.

Desarrollo de Software: Es el proceso de diseñar, codificar, depurar y mantener el código fuente de programas de computadora. El código fuente es escrito en un lenguaje de programación. El propósito de la programación es crear programas que exhiban un comportamiento deseado. El proceso de escribir código requiere frecuentemente conocimientos en varias áreas distintas, además del dominio del lenguaje a utilizar, algoritmos especializados y lógica formal.

Despliegue: Es el proceso de implementar el software en algún ambiente de desarrollo en específico. Los ambientes comúnmente utilizados para este proceso es Desarrollo, QA/Testeo, Aceptación, Pre-producción y Producción.

Diseño de Software: Es una de las etapas que deben componer el ciclo de vida del software, casi de una forma obligatoria. Su objetivo será armar el cascarón bajo el cual se estará implementando el código o realizando la programación. Se debe definir la arquitectura de la aplicación que resolverán los requerimientos recaudados en la etapa de Análisis de Software. En esta etapa se diagrama la arquitectura de la solución.

Equipo de Desarrollo: En la Ingeniería de Software, se le conoce con este nombre, al equipo conformado por los programadores que codifican software de un sistema.

Equipo de Operaciones/Sistemas: En la Ingeniería de Software se le conoce con este nombre, al equipo que da soporte, mantenimiento y despliegue al software de un sistema.

Feedback/Retroalimentación: En la Ingeniería de Software y en Tecnología en general se le conoce como a la respuesta y reacción del usuario en general ante un producto de software o hardware implementado en producción.

Guía: Cosa o conjunto de indicaciones que sirven para orientarse. Documento que enlista los pasos necesarios para cumplir un objetivo.

Infalible: Adjetivo. Que no puede fallar o equivocarse. Que puede ser asegurado sin ninguna duda.

Innovación: Es un cambio que introduce novedades. Se refiere a modificar elementos ya existentes con el fin de mejorarlos o renovarlos, esta palabra proviene del latín "innovatio" que significa "Crear algo nuevo" está comprendida por el prefijo "in-" que significa "Estar

en" y "Novus" que significa "Nuevo". Además, en el uso coloquial y general, el concepto se utiliza de manera específica en el sentido de nuevas propuestas, inventos y su implementación económica.

Integración: Es la acción y efecto de integrar o integrarse a algo, proviene del latín integration y constituye completar un todo con las partes que hacían falta ya sea objeto o persona. En el desarrollo de software, se refiere a la acción de unir el código disperso en diferentes repositorios, ramas o computadoras, hacia uno solo que contenga todo el código desarrollado.

IT: Son las tecnologías de la información y la comunicación (TIC, TICs o bien NTIC para Nuevas Tecnologías de la Información y de la Comunicación) agrupan los elementos y las técnicas utilizadas en el tratamiento y la transmisión de las informaciones, principalmente de informática, internet y telecomunicaciones.

Metodología: El grupo de mecanismos o procedimientos racionales, empleados para el logro de un objetivo, o serie de objetivos que dirige una investigación científica. En ingeniería de software es el conjunto de procedimientos que dictan la forma de hacer software, usualmente las dos grandes son las metodologías tradicionales y las metodologías ágiles.

Métrica: Que proporciona una medición. En el campo de la ingeniería del software una métrica es cualquier medida o conjunto de medidas destinadas a conocer o estimar el tamaño u otra característica de un software o un sistema de información, generalmente para realizar comparativas o para la planificación de proyectos de desarrollo. Un ejemplo ampliamente usado es la llamada métrica de punto función.

Microservicios o Arquitectura basada en Microservicios: Es una aproximación para el desarrollo de software que consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros (normalmente una API de recursos HTTP). Cada servicio se encarga de implementar una funcionalidad completa del negocio. Cada servicio es desplegado de forma independiente y puede estar programado en distintos lenguajes y usar diferentes tecnologías de almacenamiento de datos.

Obsoleto: Que no se usa en la actualidad, que ha quedado claramente anticuado. Cuando algo se convierte en obsoleto es debido a la caída en desuso de las máquinas, equipos y tecnologías motivada no por un mal funcionamiento del mismo, sino por un insuficiente desempeño de sus funciones en comparación con las nuevas máquinas, equipos y tecnologías introducidos en el mercado.

Programa de Computadora/Informático: Es una secuencia de instrucciones, escritas para realizar una tarea específica en una computadora. El programa tiene un formato ejecutable que la computadora puede utilizar directamente para ejecutar las instrucciones. El mismo programa en su formato de código fuente legible para humanos, del cual se derivan los programas ejecutables (por ejemplo, compilados), le permite a un programador estudiar y desarrollar sus algoritmos. Una colección de programas de computadora y datos relacionados se conoce como software.

Programador: Persona que se dedica a elaborar programas informáticos. Es aquella persona que escribe, depura y mantiene el código fuente de un programa informático. Los programadores también son denominados desarrolladores de software, aunque estrictamente forman parte de un equipo de personas de distintas especialidades (mayormente informáticas), y siendo que el equipo es propiamente el desarrollador.

Proyecto: (Del latín proiectus), es una planificación que consiste en un conjunto de actividades que se encuentran interrelacionadas y coordinadas. La razón de un proyecto es alcanzar las metas específicas dentro de los límites que imponen un presupuesto, calidades establecidas previamente, y un lapso de tiempo previamente definido. La gestión de proyectos es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades de un proyecto para satisfacer los requisitos del mismo. Consiste en reunir varias ideas para llevarlas a cabo, y es un emprendimiento que tiene lugar durante un tiempo limitado, y que apunta a lograr un resultado único. Surge como respuesta a una necesidad, acorde con la visión de la organización, aunque ésta puede desviarse en función del interés.

Requerimiento/Requisito: En la ingeniería de sistemas, un requisito es una necesidad documentada sobre el contenido, forma o funcionalidad de un producto o servicio. Se usa en un sentido formal en la ingeniería de sistemas, ingeniería de software e ingeniería de requisitos. En la ingeniería clásica, los requisitos se utilizan como datos de entrada en la etapa

de diseño del producto. Establecen qué debe hacer el sistema, pero no cómo hacerlo. La fase de captura, licitación y registro de requisitos puede estar precedida por una fase de análisis conceptual del proyecto. Esta fase puede dividirse en recolección de requisitos, análisis de consistencia e integridad, definición en términos descriptivos para los desarrolladores y un esbozo de especificación, previo al diseño completo.

Software: Es el soporte lógico de un sistema informático, que comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos que son llamados hardware. La interacción entre el Software y el Hardware hace operativa una computadora (u otro dispositivo), es decir, el Software envía instrucciones que el Hardware ejecuta, haciendo posible su funcionamiento.

Tecnología: Es la ciencia aplicada a la resolución de problemas concretos. Constituye un conjunto de conocimientos científicamente ordenados, que permiten diseñar y crear bienes o servicios que facilitan la adaptación al medio ambiente y la satisfacción de las necesidades esenciales y los deseos de la humanidad. Es una palabra de origen griego, τεχνολογία, formada por téchnē (τέχνη, arte, técnica u oficio, que puede ser traducido como destreza) y logía (λογία, el estudio de algo).

Testeo: Es básicamente un conjunto de actividades dentro del desarrollo de software, que está destinado a probar el software antes de su entrega. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo. Existen distintos modelos de desarrollo de software, así como modelos de pruebas. A cada uno corresponde un nivel distinto de involucramiento en las actividades de desarrollo.

Versionado de Software o control de versiones: Es el proceso de asignación de un nombre, código o número único, a un software para indicar su nivel de desarrollo.

8. Bibliografía y E-grafía

4Geeks. (2016). ¿Cómo armar un equipo de desarrollo de software? Bogotá, Colombia. Recuperado el 01 de Octubre de 2017, de https://www.4geeks.co/es/b/contratando-desarrolladores/como-armar-un-equipo-de-desarrollo-de-software

Alba, L. d. (27 de Septiembre de 2016). DevOps by Example: Tools, Pros and Cons of a DevOps Culture. Estados Unidos de América. Recuperado el 05 de Noviembre de 2017, de https://www.sitepoint.com/devops-by-example-tools-pros-and-cons-of-a-devops-culture/

Albaladejo, X. (26 de Septiembre de 2010). Ejemplo de uso del tablero o pizarra de tareas (Scrum Taskboard). Madrid, España. Recuperado el 28 de Noviembre de 2017, de https://proyectosagiles.org/2010/09/26/ejemplo-tablero-pizarra-tareas-scrum-taskboard/

Alguacil, D. (04 de Julio de 2013). *IBM explica los nuevos retos de los equipos de Desarrollo de software*. Recuperado el 09 de Octubre de 2017, de ERP-LATINO.com: http://www.erp-spain.com/articulo/73628/erp/todos/ibm-explica-en-innovate-2013-los-nuevos-retos-de-los-equipos-de-desarrollo-de-software-cronica-desde-orlando-usa

Amazon Web Services, A. (2017). ¿Qué es la entrega continua? Estados Unidos de América. Recuperado el 20 de Noviembre de 2017, de https://aws.amazon.com/es/devops/continuous-delivery/

Arevalo, M. (15 de Noviembre de 2011). *María Eugenia Arevalo Lizardo*. Recuperado el 16 de Diciembre de 2017, de https://arevalomaria.wordpress.com/2011/11/15/diferencias-entre-metodologias-tradicionales-y-agiles-metodologiasagiles/

AT Sistemas, C. I. (Noviembre de 2013). *¿Qué es un DevOps?* Recuperado el 05 de Noviembre de 2017, de AT Sistemas: https://www.atsistemas.com/es/novedades/opiniones/Opinion-12122014

Bara, M. (2015). Las 5 etapas en los “Sprints” de un desarrollo Scrum. Barcelona, Cataluña, España. Recuperado el 23 de Octubre de 2017, de https://www.obs-edu.com/int/blog-investigacion/project-management/las-5-etapas-en-los-sprints-de-un-desarrollo-scrum

Beck, K. (17 de Febrero de 2001). *The agile manifesto*. Recuperado el 17 de Octubre de 2017, de The agile manifesto Org: http://agilemanifesto.org/iso/es/manifesto.html

Beck, K. (2005). *Embrace Change with Extreme Programming* (2da. Edición ed.). (C. Andres, Ed.) Westford, Massachusetts, Estados Unidos de América: Addison-Wesley. Recuperado el 16 de Octubre de 2017, de

https://www.amazon.com/Extreme-Programming-Explained-Embrace-Change/dp/0321278658

Camacho, M. B. (18 de Octubre de 2016). ¡Cuéntame más sobre DevOps! Madrid, España. Recuperado el 04 de Noviembre de 2017, de https://www.iecisa.com/es/blog/Post/Cuentame-mas-sobre-DevOps/

Canal, P. (10 de Septiembre de 2015). Definición y características del Scrum Master. López de Hoyos, Madrid, España. Recuperado el 23 de Septiembre de 2017, de http://www.iebschool.com/blog/definicion-y-caracteristicas-del-scrum-master-agile-scrum/

Chandra, V. (09 de Diciembre de 2015). Comparison between Various Software Development Methodologies. *131*. Estados Unidos de América. Recuperado el 22 de Septiembre de 2017, de https://pdfs.semanticscholar.org/e237/f9cb136f494c2bd0ce91525808c5c968b6b4.pdf

CMS, C. f. (27 de Marzo de 2008). SELECTING A DEVELOPMENT APPROACH. *2*(2da. Edición). Estados Unidos de América. Recuperado el 2017 de Septiembre de 22, de https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf

Cohen, D. (2015). *Agile Software Development* (1 ed.). (M. Lindvall, Ed., & P. Costa, Trad.) Fraunhofer Center, Maryland, Estados Unidos de América. Recuperado el 16 de Octubre de 2017, de http://users.jyu.fi/~mieijala/kandimateriaali/Agile%20software%20development.pdf

Debois, P. (Enero de 2017). Áreas Devops - Codificación de prácticas de devops. Bélgica. Recuperado el 05 de Noviembre de 2017, de http://www.jedi.be/blog/2012/05/12/codifying-devops-area-practices/

eXtreme Programming, X. (20 de Septiembre de 2014). Prácticas de Extreme Programming. Estados Unidos de América. Recuperado el 10 de Noviembre de 2017, de http://rchavarria.github.io/blog/2014/09/20/charla-sobre-extreme-programming/

Forrester. (2016). What is Continuous Deployment. Cambridge, Estados Unidos de América. Recuperado el 11 de Diciembre de 2017, de http://electric-cloud.com/resources/continuous-delivery-101/continuous-deployment/

Foster, D. (21 de Junio de 2016). ¿Qué es DevOps? Patrick Debois explica. Recuperado el 04 de Noviembre de 2017, de https://www.linux.com/blog/what-devops-patrick-debois-explains

Fredic, P. (16 de Mayo de 2014). La increíble historia real de cómo DevOps obtuvo su nombre. New Relic, Reino Unido. Recuperado el 02 de Noviembre de 2017, de https://blog.newrelic.com/2014/05/16/devops-name/

Gaona, J. (30 de Marzo de 2017). El reto “efecto novedad” en el desarrollo de software. Bogotá, Colombia. Recuperado el 09 de Octubre de 2017, de http://www.ikonosoft.com/notas-de-interes/159-el-reto-efecto-novedad-en-el-desarrollo-de-software

Garcia, A. M. (01 de Agosto de 2014). Aprende a implantar integración continua desde cero (I): ¿Por qué integración continua? Madrid, España. Recuperado el 19 de Noviembre de 2017, de http://www.javiergarzas.com/2014/08/implantar-integracion-continua.html

Garcia, A. M. (5 de Diciembre de 2014). Vayamos al grano, ¿qué es eso de DevOps? Madrid, España. Recuperado el 01 de Noviembre de 2017, de http://www.javiergarzas.com/2014/12/devops-en-10-min.html

Garzas, J. (13 de Febrero de 2014). Derechos y deberes de los miembros de un equipo Scrum (o ágil en general, o de cualquier tipo). Madrid, España. Recuperado el 26 de Octubre de 2017, de http://www.javiergarzas.com/2014/02/derechos-y-deberes-de-los-miembros-de-un-equipo-scrum-o-agil-en-general-o-de-cualquier-tipo.html

GetApp. (18 de Enero de 2018). Jira vs Slack vs Trello. Estados Unidos de América. Recuperado el 2018 de Enero de 21, de https://www.getapp.com/project-management-planning-software/a/trello/compare/jira-vs-slack/

Grench Mayor, P. (2001). *Introducción a la ingeniería. Un enfoque a través del diseño* (2da. Edición ed.). Bogotá, Colombia: Pearson Education. Recuperado el 20 de Septiembre de 2017

Hewlett Packard, E. (2018). ¿Qué es la infraestructura como código? Estados Unidos de América. Recuperado el 23 de Enero de 2018, de https://www.hpe.com/lamerica/es/what-is/infrastructure-as-code.html

Highsmith, J. (2002). *Agile Software Development Ecosystems.* (P. E. Division, Ed.) Indianapolis, Estados Unidos de América: Addison-Wesley.

IBM. (2017). ¿Qué es DevOps? Estados Unidos de América. Recuperado el 02 de Noviembre de 2017, de https://www.ibm.com/cloud-computing/es-es/learn-more/what-is-devops/

IDS. (02 de Diciembre de 2014). Scrum - Artefactos. Distrito Federal, Ciudad de México. Recuperado el 23 de Octubre de 2017, de http://www.ids.com.mx/desarrollo-profesional/comunidad-ids/blog/scrum-4ta-parte-artefactos

IEEE. (20 de Octubre de 1990). Standard Glossary of Software Engineering Terminology. Estados Unidos de América. doi:ISBN 155937067X

iWantic. (15 de Enero de 2016). ¿Que es un QA Tester? Recuperado el 03 de Octubre de 2017, de http://iwantic.com/que-es-un-qa-tester/

Kniberg, H. (2008). *Scrum y XP desde las trincheras.* Estados Unidos de América: Info Q - Enterprise Software Development Series. Recuperado el 24 de Octubre de 2017, de http://www.proyectalis.com/wp-content/uploads/2008/02/scrum-y-xp-desde-las-trincheras.pdf

Lacalle, A. (Julio de 2016). Prototipos. Recuperado el 28 de Septiembre de 2017, de http://albertolacalle.com/hci_prototipos.htm

Lebrijo, J. (16 de Octubre de 2010). Equipos De Desarrollo De Software. (Lebrijo.com, Ed.) Recuperado el 03 de Octubre de 2017, de http://blog.lebrijo.com/gestion-de-proyectos-de-desarrollo/

León, J. (07 de Marzo de 2017). EQUIPOS DE DESARROLLO DE SOFTWARE. *Director de Área Business Software Solutions*. Madrid, España. Recuperado el 01 de Octubre de 2017, de https://www.sistel.es/equipos-desarrollo-software

Lores, A. (10 de Julio de 2008). ¿Qué diferencia a un desarrollador de software de un programador? Pontevedra, O Porriño, España. Recuperado el 2017 de Octubre de 03, de https://velneo.es/que-diferencia-a-un-desarrollador-de-software-de-un-programador/

Maeso, A. (22 de Junio de 2017). ¿El Product Owner es un Analista de Negocio? Barcelona, Madrid, Bilbao, España. Recuperado el 26 de Octubre de 2017, de https://www.netmind.es/knowledge-center/el-product-owner-es-un-analista-de-negocio/

McConnell, S. (1996). *Rapid Development* (1 ed., Vol. 1). (J. Litewka, Ed.) Washington, Redmont, Estados Unidos de América: Microsoft Press A Division of Microsoft Corporation. Recuperado el 11 de Octubre de 2017, de https://www.amazon.es/Rapid-Development-Taming-Software-Schedules/dp/1556159005

McConnell, S. (15 de Junio de 2007). Classic Mistakes Updated - Actualización de Errores Clásicos. Estados Unidos de América: 10x Software Development. Recuperado el 11 de Octubre de 2010, de http://www.construx.com/10x_Software_Development/Classic_Mistakes_Updated/

McConnell, S. (15 de Junio de 2007). Classic Mistakes Updated - Actualización de Errores Clásicos. Estados Unidos de América. Recuperado el 11 de Octubre de 2017, de http://www.construx.com/10x_Software_Development/Classic_Mistakes_Updated/

McConnell, S. (15 de Junio de 2007). Classic Mistakes Updated - Actualización de Errores Clásicos. Estados Unidos de América. Recuperado el 11 de Octubre de 2017, de http://www.construx.com/10x_Software_Development/Classic_Mistakes_Updated/

Mountain Goat Software, I. (2017). *Scrum*. Recuperado el 23 de Octubre de 2017, de Mountain Goat Software: https://www.mountaingoatsoftware.com/agile/scrum

Null, C. (20 de Enero de 2017). *TechBeacon*. Recuperado el 2018 de Mayo de 2018, de https://techbeacon.com/10-companies-killing-it-devops

Ok-Hosting. (2016). Metodologías de Desarrollo de Software. Ciudad de México, Distrito Federal, México. Recuperado el 26 de 09 de 2017, de https://okhosting.com/blog/metodologias-del-desarrollo-de-software/

Ortiz, M. (01 de Septiembre de 2012). Modelo Iterativo. Recuperado el 26 de Noviembre de 2017, de http://isw-udistrital.blogspot.com/2012/09/ingenieria-de-software-continuacion.html

Paloma, S. (24 de Marzo de 2017). ¿Qué es un prototipo y para qué sirve? Madrid, España. Recuperado el 28 de Septiembre de 2017, de https://sendekia.com/que-es-un-prototipo-y-para-que-sirve/

Perez, M. (07 de Mayo de 2015). Roles y Responsabilidades en un Equipo de Desarrollo de Software. Distrito Federal, México. Recuperado el 01 de Octubre de 2017, de http://www.marioperez.com.mx/equipos-de-desarrollo/roles-y-responsabilidades/

Powell, A. (23 de Noviembre de 2016). Rapid Application Development (RAD): What Is It And How Do You Use It? Recuperado el 29 de Septiembre de 2017, de https://airbrake.io/blog/sdlc/rapid-application-development

Pressman, R. S. (2010). *Ingeniería del Software, Un enfoque práctico* (Séptima Edición ed.). (V. C. Brito, Trad.) San Francisco, Estados Unidos: McGrawHill. Recuperado el 18 de Septiembre de 2017

Randell, B. (26 de Agosto de 1996). *History of Software Engineering.* Recuperado el 20 de Septiembre de 2017, de The NATO Software Engineering Conferences: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/NATOReports/

Rapaport, R. (23 de Diciembre de 2014). Una breve historia de DevOps. Estados Unidos de América. Recuperado el 02 de Noviembre de 2017, de https://www.ca.com/us/rewrite/articles/devops/a-short-history-of-devops.html

Ruiz, J. (16 de Noviembre de 2015). El legendario origen del movimiento DevOps. Madrid, España. Recuperado el 02 de Noviembre de 2017, de https://www.paradigmadigital.com/techbiz/el-legendario-origen-del-movimiento-devops/

Scrum Alliance, S. (2017). Scrum Alliance. Recuperado el 2017 de Octubre de 26, de https://www.scrumalliance.org/certifications/certifications2017

Wellspring. (2017). Innovation Management. Chicago, Illinois, Estados Unidos de América. Recuperado el 07 de Octubre de 2017, de https://cdn2.hubspot.net/hubfs/2204337/Solution%20Overview%20-%20Innovation%20Management%20Software.pdf?t=1488902801470

Williams & Murphy, D. P. (16 de Marzo de 2016). Avoid Failure by Developing a Toolchain That Enables DevOps. Stamford, Estados Unidos de América. doi:G00293223

Withrow, S. (2017). Extreme Programming: Do these practices make perfect? Estados Unidos de América. Recuperado el 28 de Octubre de 2017, de https://www.techrepublic.com/article/extreme-programming-do-these-12-practices-make-perfect/

Xebia Labs, D. (11 de Septiembre de 2015). *Xebia Labs Blog*. Recuperado el 16 de Mayo de 2018, de https://blog.xebialabs.com/2015/09/11/9-companies-you-wouldnt-expect-to-be-using-devops/

Xebia Labs, D. (2017). Periodic Table of DevOps Tools. Estados Unidos de América. Recuperado el 11 de Noviembre de 2017, de https://xebialabs.com/periodic-table-of-devops-tools/