ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE MEDIANTE LA IMPLEMENTACIÓN DE INTEGRACIÓN CONTINUA

GONZALEZ RODRIGUEZ JF CAMILO ANDRES

OLIVER FABIAN LOZANO PINTO

UNIVERSIDAD PILOTO DE COLOMBIA - SECCIONAL ALTO MAGDALENA

FACULTAD DE INGENIERÍA

INGENIERIA DE SISTEMAS

GIRARDOT

2020

ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE MEDIANTE LA IMPLEMENTACIÓN DE INTEGRACIÓN CONTINUA

GONZALEZ RODRIGUEZ JF CAMILO ANDRES

OLIVER FABIAN LOZANO PINTO

MONOGRAFÍA REALIZADA PARA OPTAR EL TITULO DE INGENIERO DE SISTEMAS

DIRECTOR:

ING. EDICSON PINEDA CADENA

UNIVERSIDAD PILOTO DE COLOMBIA - SECCIONAL ALTO MAGDALENA

FACULTAD DE INGENIERÍA

INGENIERIA DE SISTEMAS

GIRARDOT

2020

Nota de Aceptación

_____

_____

_____
Firma del jurado

_____
Firma del jurado

_____
Firma del jurado

Girardot, 2020

Agradecimientos

Gracias a Dios por aquel logro que estoy alcanzando, y aquellas personas que me apoyaron ante cualquier adversidad, y llevar a cabo este sueño. Mi motor por salir adelante son mis padres, mis hermanas, sobrina, y abuela. Han sido parte fundamental como persona y como profesional. Y aquellos profesores que estuvieron en mi proceso de estudiante en la universidad piloto de Colombia seccional alto magdalena.

OLIVER FABIÁN LOZANO PINTO

Quiero agradecerle principalmente a mi familia, en especial a mis padres porque siempre me han apoyado con las decisiones que he tomado desde el comienzo al respecto con la carrera, a mi hermano por darme apoyo en los momentos difíciles. También agradecer a dos personas muy especiales que han formado parte de mi vida, mi hermana, que ya partió de este plano existencial y otra, quien es una mujer maravillosa, la cual estuvo apoyándome, estuvo pendiente de mí, dándome fuerzas, motivándome para continuar, y que hoy me acompaña desde la distancia.

Agradecerles a los docentes que brinda la Universidad, ya que gracias a ellos he aprendido bastante tanto a nivel profesional como personal, a la coordinadora a cargo durante nuestra preparación, ya que nos brindó su colaboración cuando la necesitábamos. De esta manera agradezco a las personas mencionadas anteriormente.

JF RODRIGUEZ GONZALEZ

TABLA DE CONTENIDO

1. RESUMEN

El presente documento pretende evidenciar de manera clara y concisa las diversas ventajas que presenta la puesta en marcha de un sistema o proceso de integración continua.

En el ciclo de vida de desarrollo del software, es frecuente encontrarse con errores de código debido a fallas humanas por más profesional y experimentado que sean los equipos de trabajo. Pero entonces, ¿Cómo podemos disminuir la frecuencia de los errores humanos en la codificación? La respuesta es mediante la integración continua. Pero ¿Y cómo? A través del control de calidad mediante la realización de una serie de ejecuciones de pruebas unitarias validando así que cada entrega o cambio realizado al código fuente funcione correctamente en la aplicación.

La integración continua es una práctica habitual en el mundo del desarrollo actual, usada principalmente por equipos de desarrollo que han adoptado metodologías ágiles de trabajo. Básicamente, consiste en un proceso por el cual cualquier pequeña mejora en el software se integra rápidamente con el software que se llevará producción. Cualquier mejora, debidamente probada, es integrada en los repositorios de control de versiones, junto con las mejoras producidas por otros desarrolladores del equipo de trabajo.

Es por esta razón que se optó por realizar la monografía, lo que a su vez significó realizar una investigación de las tareas imprescindibles para implementar la *integración continua*, con el fin de identificar las herramientas fundamentales que posibiliten dicho proceso, además de realizar el correspondiente análisis e implementación para la integración entre dichos instrumentos.

Además, se realizaron indagaciones sobre las diferentes formas que existen actualmente para ejecutar el proceso de integración continua y la manera en que las empresas realizan dicho proceso, y a su vez los tipos de soluciones informáticas existentes en la actualidad que satisfacen dicha necesidad.

Palabras Clave: Integración continua, Implementación, Automatización.

2. INTRODUCCION

El presente trabajo de monografía abarca la investigación sobre el proceso de integración continúa como herramienta para la automatización de procesos que día a día debe afrontar el equipo de desarrollo en las empresas de software.

En toda empresa relacionada con el campo de las tecnologías de la información y las comunicaciones, el desarrollo de software se ha transformado en un método o procedimiento cada vez más frecuente a través de los años. En el transcurso, se ve reflejado que mientras los proyectos de software incrementan, éstos a su vez se vuelven más complicados. Por esta razón se requieren de expertos en los temas de TIC siendo fundamental que estas personas se colaboren a través del trabajo en equipo de forma armónica y afinada.

Esto genera una gran dificultad y es el proceso de integrar todos los cambios que se realizan en el código fuente por los diferentes individuos que conforman el equipo de desarrollo, el cual aumenta cuando varios desarrolladores están trabajando sobre la misma sección o división del código resultando que parezca inacabable de solucionar.

Actualmente las empresas orientan su trabajo en la obtención de una alta calidad en el software que desarrollan a través del uso de procedimientos, reglas, metodologías y estándares, algunas de estas son: Moprosoft, TSP, PSP, CMMI, SPICE, ISO 14598-1, ISO 9126. Entre otras, idóneas con el fin de realizar y entregar un proyecto de software donde se disminuyan los riesgos, tiempos y costos de entrega además de asegurar una óptima calidad. Es por esta razón que se han definido nuevos modelos y patrones organizacionales que emplean el uso de procesos y estándares facilitando el flujo de trabajo y automatizando diversas mejoras entre la puesta en marcha y el desarrollo.

3. PLANTEAMIENTO DEL PROBLEMA

Desde el pasado y hasta el día de hoy, se aprecia como las empresas dedicadas al desarrollo de software sufren al momento de integrar el trabajo que realizan los diferentes equipos en el área de desarrollo en los códigos fuentes de los proyectos.

Algunas empresas optan por solventar dicho problema a través de la creación de un área que se encargue de dicho proceso, generando carga y trabajo que fácilmente podría evitarse a través de software que permita y/o facilite esta tarea.

En Colombia se puede observar como la mayoría de las empresas en las cuales se desarrollan productos de software para terceros o para su consumo propio, se enfrentan a diario con inconvenientes en la calidad de sus productos. Si bien, algunas cuentan con herramientas que ayudan a mejorar ciertos tropiezos dentro de la cadena productiva del software, como lo son los servidores para controlar las versiones de la aplicación o las aplicaciones que se encuentren desarrollando, siguen existiendo problemas al momento de ejecutar el paso a producción al tratar de desplegar una versión donde se encuentren involucrados más de un desarrollador; también pueden contar con herramientas donde se encuentren automatizadas las pruebas unitarias y de calidad, pero que al solucionar incidentes que se presentan, se pierde el control de la versión o se afectan otras partes de la aplicación, aumentando la posibilidad de encontrar fallos en los despliegues.

Al estar vinculados en un mismo proyecto varios desarrolladores, los cuales tiene formas diferentes de realizar su código, seguramente se enfrentarán a reprocesos más

adelante, en el momento de realizar mantenimientos adaptativos, evolutivos o perfectivos, por no contar con estándares de calidad definidos por la empresa, y realizar inspecciones continuas de código a través de una herramienta para detectar problemas de calidad en él.

Los problemas descritos anteriormente afectan de manera directa los costos de un proyecto de software, al aumentar los tiempos estimados para la fabricación del producto, perdiendo el control en la fase del desarrollo del producto, pasando a través de los diferentes ambientes, de pruebas a producción, devolviéndose de producción a desarrollo, de desarrollo a pruebas, generando tareas repetitivas causando un gran impacto negativo para el ordenamiento del proyecto.

4. OBJETIVOS

5. Objetivo general

Obtener información sobre el proceso de integración continua para su posterior análisis e implementación en las empresas de desarrollo de software, asegurando la calidad en el ciclo de vida del desarrollo de software.

6. Objetivos específicos

- Obtener información acerca de la integración continua.
- Realizar el análisis de las herramientas existentes para la implementación de dicho proceso en el área de desarrollo de la compañía.
- Describir el caso estudio que se obtuvo como resultado en la aplicación de integración continua.

7. JUSTIFICACION

Previamente era muy común que los integrantes de un equipo de desarrollo trabajaran de manera aislada durante un prologando periodo de tiempo, donde procuraban unir o juntar las modificaciones realizadas en la versión final al momento de tener completo el proyecto. A raíz de esto, la unión de cualquier modificación en el código fuente resultaba en un proceso tedioso, además de propiciar y abarcar durante mucho tiempo la generación de errores que no se subsanaban. Todo esto originaba que fuera más complicado suministrar las actualizaciones a los clientes de manera eficaz y eficiente.

La clave de triunfo en la aceptación de todo proyecto está centrado en controlar y disminuir los riesgos e imprevistos; tener un código totalmente documentado, contar con un tablero que indique los estados las tareas del proyecto y del código, realizar las pruebas e integraciones de manera cotidiana, monitoreo, métricas de calidad, automatizar la documentación técnica, auspiciar tranquilidad en el momento en el cual se ponga en producción el software, entregado al cliente o listo para generar réplicas del él.

Aplicando las herramientas y el diseño adecuado, el área de desarrollo dispondrá de un ambiente más organizado y estructurado, aumentando la calidad en el desarrollo de software.

8 MARCOS DE REFERENCIA

a. Marco Teórico

Día a día la realidad de un departamento de sistemas, que por lo general consta de administradores y programadores; es que ambos perfiles realizan sus tareas totalmente independientes uno del otro, y casi siempre llegan a un extremo de "no trabajar colaborativamente", quizás el trabajo colaborativo no es necesario en todos los escenarios, pero cuando el administrador debe mantener un sistema empresarial que usa bases de datos, servidor web, servidor de aplicaciones, y más; debe existir total colaboración entre ambos equipos, o al menos ambos deberían estar completamente consciente de las herramientas, y metodologías que usa cada equipo.

Aun si pareciera que ambos perfiles deberían trabajar de forma independiente porque sus áreas son diferentes, lo cierto es que en la actualidad la tecnología está tan intrínsecamente ligada entre sí que es casi imposible pensar en un proyecto tecnológico sin tener en consideración algunos factores comunes que este involucra, ejemplo: requerimientos mínimos de hardware para soportar el software, seguridad en la transmisión de datos, persistencia de datos, arquitectura de software, frameworks que se van a usar (sean de código abierto o cerrado), etc.

Adicionalmente se suma la administración del proyecto, calcular tiempos estimados, designar responsabilidades, etc.

A lo largo del tiempo las prácticas que antes eran buenas, en algunos casos suele suceder que dejan de ser buenas prácticas, y de la misma manera cambian las formas de planificación, de análisis, y diseño. Pero es gracias a todos estos cambios, que actualmente hay una comunidad grande de científicos revisando, cuestionando y probando cada cambio que hay en el mundo tecnológico, muchas veces ellos son quienes aprueban ciertos movimientos como norma o standard.

Un claro ejemplo de todo esto, en cuanto a proyectos de software se refiere y que es relativamente nuevo en el campo de la tecnología, es que actualmente todo converge en las “Metodologías Ágiles”, difundidas en gran manera por su eficacia y eficiencia.

La integración continua es una práctica que se aplica al desarrollo de software realizado con metodologías ágiles, que se puede ejecutar siempre y cuando se implementen automatizaciones en sus procesos de requerimientos, diseño, desarrollo, pruebas y despliegues, lo que permite realizar cambios en el código, compilar y probarlo el mismo día, las veces que sea necesario, siendo reglamentario que todos los desarrolladores del equipo integren su parte de código por lo menos una vez al día.

Martin Flower (2015), asegura que tener el desarrollo como un proceso disciplinado y automatizado, es esencial para un proyecto controlado, cuando esto se logra el equipo está más preparado para modificar el código cuando sea necesario, debido a la confianza en la identificación de los errores en la integración; una vez se realiza una integración los errores son detectados de inmediato, permitiendo identificar más fácilmente de donde proviene el error, reduciendo el tiempo que se toman los equipos para identificar errores cuando la compilación se hace posterior varias semanas o meses de desarrollo.

La técnica de integración continúa promocionada por Martin Fowler conceptúa sobre la práctica del software estableciendo que con frecuencia diaria los miembros desarrolladores integren su trabajo, que a su vez es validada por un autómata con el objeto de hallar oportunamente, errores que posiblemente hayan ocurrido en la integración en el desarrollo del software ha sido necesario la práctica de la Integración continúa dado a los variados beneficios que trae consigo, entre los que son destacables la disminución de los riesgos y las tareas repetitivas, produciendo un software listo para ejecutar, aumentando la calidad del producto.

La Integración Continua aplica para la supervisión a los Sistemas de Control de Versiones, dado que cuando se encuentra un cambio, muy probablemente una herramienta compile y valide automáticamente estos cambios, notificando inmediatamente de las posibles falencias encontradas.

En Cuba, existe la Universidad de las Ciencias Informáticas, caracterizada por la alta calidad del software producido. Dentro de esta Universidad se halla el Centro de la Telemática (TLM), enmarcado en el desarrollo de sistemas y servicios informáticos en el campo de las Telecomunicaciones y la seguridad informática.

La alternativa de la práctica de la Integración Continua es una herramienta con alta eficacia en los productos de calidad, cumpliendo con las normas recomendadas desde las etapas iniciales del desarrollo del software. Con el objetivo de obtener productos seguros y robustos, para el centro TLM de la Universidad de las Ciencias Informáticas esta práctica de la Integración Continua es indispensable e imprescindible.

En el mundo actual de DevOps, la entrega y el despliegue continuos son fundamentales para entregar productos de software de alta calidad más rápido que nunca. Algunas Ventajas de Jenkins es que es un servidor de integración continua de código abierto escrito en Java. Es, de lejos, la herramienta más utilizada para gestionar construcciones de integración continua y canalizaciones de entrega. Ayuda a los desarrolladores a construir y probar software continuamente. Aumenta la escala de automatización y está ganando rápidamente popularidad en los círculos DevOps. Jenkins requiere poco mantenimiento y tiene una herramienta GUI incorporada para actualizaciones fáciles. Jenkins proporciona una solución personalizada ya que hay más de 400 complementos para ayudar a construir y probar prácticamente cualquier proyecto. Básicamente, Jenkins integra procesos de ciclo de vida de desarrollo de todo tipo, incluyendo compilación, documento, prueba, paquete, etapa, implementación, análisis estático y mucho más.

Con Jenkins, puedes configurar las alertas de varias maneras, por ejemplo, recibe una notificación por correo electrónico, una ventana emergente, etc. y lo automatiza. Al hacer la configuración correcta, obtienes una respuesta rápida. Siempre sabrás, justo después de romper lo construido. Sabrás cuál fue el motivo del error en el trabajo y saber cómo puedes revertirlo.

Caso estudio

Se realizó la implementación de integración continua en uno de los proyectos ya desarrollados con el fin de realizar la puesta en marcha de dicho proceso.

Como servidor de integración continua se utilizó Jenkins y como servidor de control de versiones se utilizó SVN.

Los desarrolladores al momento de realizar las modificaciones de un fichero debían realizar el commit en el control de versionamiento luego el servidor de integración continua recibía los cambios realizados en los ficheros, realizaba la construcción o build de estos, posteriormente acorde a un set de pruebas se realizaban las pruebas automáticas y se obtenían los resultados de dichas pruebas, si el proceso era satisfactorio o exitoso los cambios realizados que eran integrados y se generaba la última versión funcional y se notificaba a los desarrolladores de dichos resultados.

En caso de que las pruebas automáticas no cumplieran con el set de pruebas se regresaba a la versión funcional y se notificaba a los desarrolladores para su posterior corrección y actualización de errores encontrados en los ficheros.

b. Marco Conceptual

- Integración Continua

  El término Integración Continua fue creado por Martin Fowler (2009), en un patrón de diseño. IC refiera a las prácticas y herramientas que aseguran la compilación y las pruebas de manera automática para una aplicación en intervalos frecuentes y se usa un servidor de integración que funciona como el tablero de control de las automatizaciones del proyecto. IC, puede ser implementada en múltiples ambientes de desarrollo, hasta los que cuentan con un solo desarrollador, pero se hace más significativa y notoria la colaboración de esta práctica en ambientes complicados con muchos desarrolladores y probadores de software.

- Se describe a continuación, el proceso de la integración continua:

  - **Comportamiento**

  Cuando se trabaja de este modo, deben acatarse una serie de reglas. En la mayoría de las ocasiones, los programadores se rigen por los principios que Martin
  Fowler describió para llevar a cabo con éxito una integración continua. En primer lugar, se ha de garantizar que todos los implicados se encuentran al mismo nivel, y que no hay nadie cuyo comportamiento pueda causar problemas.

  - **Un único código fuente**

  Aunque parezca obvio, se trata de uno de los factores más importantes, ya que todos los integrantes del equipo deberán utilizar la misma herramienta, es decir, el
  mismo repositorio, cuando trabajen en el código. Y esto no solo se aplica al código fuente. Para que funcionen las aplicaciones, se necesitan otros elementos como, por ejemplo, bases de datos, que también deberán estar contenidos en el mismo lugar.

Por ello, Martin Fowler recomienda construir un repositorio de tal forma que cualquier ingeniero que se incorpore al proyecto con un equipo nuevo encuentre todos los archivos necesarios en un único lugar.

- **Automatizar la compilación del proyecto**

La obtención de un proyecto funcional a partir de un código fuente implica la compilación del mismo, actualizar bases de datos y mover ficheros de un sitio a otro. Todas estas tareas pueden automatizarse y debería ser posible ejecutar la compilación con un único comando.

- **Sistemas que realizan sus propias pruebas**

Un equipo podrá beneficiarse de aún más automatización y rapidez a través de la integración continua si se incorporan mecanismos de prueba en el proceso de compilación (“build”). Al igual que la propia compilación, las pruebas podrán llevarse a cabo en poco tiempo, y lo ideal será implementar un plan completo de mecanismos de prueba.

- **Integración diaria**

Un proceso de continuous integration funcionará únicamente si todos los miembros del equipo respetan el sistema. En el momento en que algún miembro del equipo no integre su código en la línea principal, el resto de compañeros partirá de una premisa falsa. Todos los desarrolladores asumen que trabajan en un sistema estable, pero si alguien tarda más de un día en integrar su código y continúa trabajando en él, al final la búsqueda de errores podría convertirse en un verdadero problema. Asimismo,

la comunicación también es un factor importante en la integración continua, ya que, si los desarrolladores se mantienen al tanto, las pequeñas dificultades podrán aclararse con mayor rapidez.

- **Main line operativa**

El código de la línea principal deberá probarse continuamente y encontrarse siempre operativo, por lo que los desarrolladores deberán construir el proyecto aquí y no solo en su copia local. En este contexto, todos deberán preocuparse también de que sus aportaciones sean válidas y de que su funcionamiento sea correcto, de modo que todos tendrán que comprobar el código y el build. Si aparece algún fallo, tendrán que solventarlo y garantizar que el código no contiene errores.

- **Reparación inmediata**

Lo más importante de la integración continua es que no quede ninguna versión defectuosa en la línea principal, lo que implica que la solución de los fallos no podrá posponerse. En palabras de Martin Fowler, no existe ningún problema si los builds no funcionan y el código debe cambiarse, pero el sistema requiere que la reparación se lleve a cabo de inmediato. Todos los desarrolladores deben poder partir del hecho de que el código de la línea principal funciona correctamente, de lo contrario, podrían estar trabajando sobre un código defectuoso que acabará desencadenando una oleada de fallos.

- **Integración rápida**

La integración completa del proyecto (incluida la fase de prueba) debería realizarse lo más rápido posible. La programación extrema (extreme programming, XP) prevé solo 10 minutos para esto. Puesto que un desarrollador debe realizar varias integraciones diarias, si no se establecieran mecanismos para acelerar el proceso se perdería una gran cantidad de tiempo. Para que no se demore demasiado, debe descartarse la idea de realizar todas las pruebas posibles directamente y aplicarse en lugar de ello un sistema de dos fases: en la primera fase, se realizan pruebas en las que la compilación del proyecto pueda ser rápida. La segunda fase durará varias horas y en ella se llevarán a cabo pruebas más exhaustivas.

- **Pruebas en una réplica del entorno de producción**

Las pruebas deberán realizarse en un entorno seguro y con una configuración exactamente igual que la del entorno de producción. Bajo determinadas circunstancias, esto podría resultar muy costoso, pero la virtualización de los equipos hará que el factor de los costes se reduzca.

- **Accesibilidad**

Todos los involucrados en el desarrollo de un determinado software deberían poder obtener fácilmente el último ejecutable del programa y ejecutarlo. La implementación de este aspecto es relativamente sencilla, ya que la integración continua exige que todos los archivos se encuentren en un único repositorio que todos conocen. De este modo, se puede comenzar a realizar pruebas adicionales en el proceso de programación, los accionistas pueden utilizar archivos ejecutables con fines demostrativos y los directores de calidad pueden examinar las cifras.

- **Buena comunicación**

  No solo resulta importante que todos los implicados tengan acceso al código fuente y puedan ejecutar el archivo, sino que, además, debe quedar constancia de quién ha realizado qué modificación. Asimismo, los desarrolladores deben informar cuando se encuentren en un proceso de compilación, para lo que algunos equipos utilizan elementos visuales que indican que se está trabajando en la integración.

- **Despliegue automatizado**

  Por último, ha de automatizarse el despliegue del software. Para llevar a cabo la integración continua se deben mover ejecutables entre múltiples entornos, lo que puede requerir una gran cantidad de tiempo. Por ello, será mejor hacerlo de forma automática utilizando scripts que faciliten el despliegue de aplicaciones entre entornos. (Integración continua en el desarrollo del software, 2019)

- Ágil

  En el año 2001, se realizó una reunión en Utah-EEUU, en la cual se originó el término "ágil" para ser referenciado con el desarrollo de software. En esta reunión se permitieron generar los principios y procesos que podrían ayudar a los equipos de desarrollo de software a desarrollar aplicaciones de forma diferente a la tradicional, de una forma más flexible en cuanto a documentación y donde se pudiera llevar un control sobre la ejecución de actividades del proyecto, permitiendo tomar decisiones de forma rápida para solucionar problemas y/o cambios que se presenten a lo largo del proyecto. La documentación originada de esta reunión se denomina "Manifiesto Ágil", un documento donde se resume la filosofía del desarrollo "ágil".

- **Metodologías Agiles**
  - **XP**

    Es una metodología trabajo para la construcción de software. Sus primeras aplicaciones datan de mediados de la década de 1990 pero el primer libro que la formalizó fue el de Kent Beck en 1999 [Beck 1999]. Tal vez algo curioso de esta metodología es que como parte de su definición hay un hincapié muy importante en cuatro valores:

    • Simplicidad

    • Comunicación

    • Respeto

    • Coraje. (Paez, 2014, P. 249)

Potencia las relaciones interpersonales es el fundamento central de la metodología ágil, considerada indispensable para el logro efectivo en el desarrollo de software, incentivando el trabajo colaborativo en equipo, destacando el interés por el aprendizaje de los miembros desarrolladores, generando con ello, un adecuado clima de relaciones laborales.

XP basa en la continua evaluación con el cliente y el equipo de trabajo de los desarrolladores, una comunicación pertinente, efectiva, sencilla, entre todos los miembros participantes, procurando la implementación de soluciones simples y bastante gallardía para asumir los retos.

XP está caracterizada por asumir proyectos con requerimientos complejos, variables, específicamente aquellos donde haya un alto grado de inseguridad técnica.

- **Scrum**

  > Si bien sus orígenes se remontan a la década de 1980, su aplicación en el desarrollo de software fue formalizada en 1995 cuando Jeff Sutherland y Ken Schwaber publicaron un trabajo titulado “Scrum methodology” en la conferencia OOPSLA [Sutherland 95]. Siendo estrictos debemos decir que Scrum es un marco de trabajo que propone un proceso, un conjunto de prácticas y roles, dejando a criterio del lector si es suficiente para denominarlo metodología. Una particularidad de Scrum es que es un método de gestión de aplicación general, lo cual permite que sea utilizado más allá del desarrollo de software. (Paez, 2014, P. 246)

  Ken Schwaber, Jeff Sutherland y Mike Beedle establecieron una regulación en la gestión de proyectos, utilizada con bastantes logros durante la última década. Orientada con mayor énfasis en los proyectos que requieren frecuentes ajustes de sus requerimientos.

  Se pueden abreviar en dos la caracterización principal:

  Los denominados sprints, que corresponden a un desarrollo de software mediado por iteraciones, cuyo periodo es de treinta días. El producto obtenido en cada uno de estos sprints, que se da a conocer a los clientes, corresponde a un incremento ejecutable. La segunda, consiste en realizar reuniones durante la ejecución del proyecto, entre estas se realza una reunión del equipo de desarrolladores, de frecuencia diaria con duración de quince minutos, con el objeto de coordinar e integrar el equipo en sus tareas.

- **DSDM**

  El Dynamic System Development Model es un método desarrollado en el Reino Unido a mediados de la década de 1990. En sus comienzos, reunió varias de las prácticas propuestas por el enfoque RAD.

  A largo de los años ha evolucionado incorporando varias lecciones aprendidas y generando nuevas versiones. La versión actual es conocida como Atern. Una particularidad es que su foco es el desarrollo de sistemas de información con restricciones de calendario y presupuesto, permitiendo que el alcance sea variable. Otra particularidad interesante es que define explícitamente ciertos prerrequisitos para su uso. Entre ellos se destaca aceptar que el principal factor de fracaso en los proyectos es de índole humana. Al mismo tiempo toma como fundamentales los siguientes principios:

  • Foco en las necesidades de negocio.

  • Entrega a tiempo.

  • Colaboración.

  • Nunca comprometer la calidad.

  • Construir sobre bases firmes.

  • Desarrollo iterativo.

  • Comunicar continua y claramente.

  • Demostrar control. DSDM está estructurado en siete fases que cubren desde la concepción de proyecto (pre proyecto) hasta su finalización (post proyecto) y donde distintos esquemas iterativos pueden plantearse combinando dichas fases. En cuanto a los roles, los mismos se presentan en dos grupos: Roles de Proyecto y Roles de desarrollo. (Paez, 2014, P. 253)

Se caracteriza principalmente por su proceso iterativo e incremental, y a que desarrolladores y usuarios trabajan en unidad.

Se proponen cinco etapas: estudio de la viabilidad, del negocio, del modelo funcional, del diseño y la construcción, terminando con la implementación. Son iterativas las tres últimas fases listadas, incluyendo la constante realimentación en cada una de las etapas.

- **ASD**

  "ASD tiene como fundamento la teoría de sistemas adaptativos complejos. Por ello, interpreta los proyectos de software como sistemas adaptativos complejos compuestos por agentes −los interesados−, entornos −organizacional, tecnológico− y salidas −el producto desarrollado− "(Cadavid, 2013, P. 36).

  Representante Jim High Smith. Se destaca por ser iterativo, énfasis en los componentes del software sobre las tareas, flexible a los ajustes. Sus etapas fundamentales son la especulación, la colaboración y el aprendizaje. En la especulación se da comienzo al proyecto, se planifican las características del software. El desarrollo de estas características se ejecuta en la segunda etapa, y para culminar en la tercera, evaluando la calidad previamente para entregar al cliente. Se utiliza la revisión de los componentes para el aprendizaje a partir de los errores, reiniciando el ciclo del desarrollo.

- **FDD**

  Feature Oriented Programming (FOP) es una técnica de programación guiada por rasgos o características (features) y centrada en el usuario, no en el programador; su objetivo es sintetizar un programa conforme a los rasgos requeridos. En un desarrollo en términos de FOP, los objetos se organizan en módulos o capas conforme a rasgos. FDD, en cambio, es un método ágil, iterativo y adaptativo. A diferencia de otras metodologías ágiles no cubre todo el ciclo de vida sino sólo las fases de diseño y construcción y se considera adecuado para proyectos mayores y de misión crítica.

  FDD es, además, marca registrada de una empresa, Nebulon Pty. Aunque hay coincidencias entre la programación orientada por rasgos y el desarrollo guidado por rasgos, FDD no necesariamente implementa FOP. (Valverde, 2007, P. 30)

Consiste en un proceso iterativo de cinco fases. Estas iteraciones suelen ser de corta duración, máximo de dos semanas. Basada en las etapas de diseño e implementación del sistema iniciando del listado de características que el software debe contener. Los promotores de esta (técnica, modalidad).

- Build: Es el proceso de transformar el código fuente en artefactos de software que se pueden ejecutar de forma independiente.
- Commit: Es el paso que se realiza al momento de finalizar los cambios efectuados en las clases o ficheros para que estos sean tomados en el servidor principal. En resumen es cuando se finaliza una transacción de manera exitosa.
- Desarrollo: Término que se utiliza para referirse al entorno de desarrollo donde trabajan los programadores.
- Producción: Término que se utiliza para designar la maquina final donde se distribuye y consume el producto final.
- CheckOut: Se emplea este término para referirse a la creación de una copia local del repositorio.

- Update: Se ejecuta, y realiza la actualización mediante los parámetros de la herramienta realiza la respectiva configuración en el direccionamiento local.
- Plugin: Es un programa que incrementa las funcionalidades de un programa principal. Por lo general ha sido creado por una compañía diferente a la que produjo el programa principal.
- Hook: En programación, se denomina así generalmente a una interfaz integrada en un paquete que permite insertar trozos de código personalizados.
- QA: Significa Quality Assurance, o aseguramiento de la calidad. Se trata de un conjunto de actividades de evaluación de las distintas etapas del proceso de desarrollo para garantizar que el producto final sea de calidad.

c. Marco Metodológico

La metodología de investigación elegida es de tipo explicativo. Su objetivo es conocer y describir el contexto de la problemática, esto con el fin de indagar y recolectar la información necesaria en el uso de las tecnologías y el software libre para el apoyo de procesos de integración en el área de desarrollo.

La metodología elegida para el presente proyecto se dividió en las siguientes fases o etapas:

Gestión de la información con el fin de ubicar documentos que abarquen los temas referentes al proceso de integración continua y sus subprocesos, evidenciando así los casos de éxito en las empresas que han decidido implementar dicha práctica, logrando identificar los puntos en común en dichas implementaciones para desarrollar un modelo estándar de implementación.

- Comparación de las distintas herramientas que existen actualmente en el mercado utilizadas en las diferentes etapas de la implementación. Con el objetivo de lograr identificar las ventajas en temas de compatibilidad e integración, con el fin de elegir un software que cumpla con los criterios establecidos por el área de desarrollo.

i. Requerimientos funcionales

Los requerimientos funcionales son los siguientes

- Es imprescindible tener un repositorio de control de versiones.
- Todo lo necesario para realizar la compilación, debe estar en el repositorio (código, scripts de test, librerías de terceros).
- Testear el software.
- Parametrizar las métricas de calidad del software establecidas por el equipo de trabajo A continuación, mencionaremos algunas métricas para el aseguramiento de la calidad del software:
    - Acoplamiento: Es la manera en cómo se comportan los datos con otro entorno, es decir, la forma en cómo trabajan los módulos entre sí. Entre más alto sea el nivel de acoplamiento, más difícil y complicado será su mantenimiento en el tiempo.
    - Code Chrun: Es el número de veces que se realiza modificaciones a una clase.
    - Code Coverage: Sirve para medir el porcentaje de código que se encuentra en prueba, lo que disminuye la probabilidad de que haya errores.
    - Paquetes innecesarios: Es cuando encontramos librerías o clases importadas que no se requieren en el aplicativo.
    - Enviar las modificaciones del software, una vez pasadas todas las validaciones, al repositorio principal.
    - Automatizar la compilación del software o su despliegue, una vez se hayan integrado nuevos cambios en el proyecto.
    - Herramientas para la integración continua, las cuales son: Maven, Git, Jenkins, SonarQube, MySQL, Nexus OSS.

ii. Requerimientos no funcionales

Los requerimientos no funcionales del software son

- El tiempo de aprendizaje del sistema por usuario debe ser menor a 8 horas
- Generar o visualizar la documentación de un proyecto.
- Notificar debidamente a los desarrolladores o al equipo de aseguramiento de calidad cuando se encuentra cualquier tipo de error, ya sea en base a las pruebas del software o a las métricas de calidad definidas.
- Se debe mantener los datos de Jenkins en un directorio separado del sistema.
- Se debe montar una partición HDD dedicada en este directorio, otorgándole permisos que nos permitan administrar el espacio en el disco (unidad virtual, herramienta de cambio de tamaño de partición…).
- Supervisar la actividad para evitar que quede espacio.

  Copias de seguridad constantes (cada 1 o 2 semanas).

iii. Requerimientos técnicos

- Para la implementación de Jenkins existen 2 pre-requisitos:
  - Máquina Virtual de Java (JVM).
  - Contenedor de Servlets (Tomcat o Apache).

  Nota: Los recursos de Jenkins dependen de la capacidad de la jvm (máquina virtual de Java) por lo que se recomienda el almacenamiento dinámico de JVM.
- CPU preferiblemente que maneje entre 2 a 4 núcleos.
- RAM (Entre 4 a 6 GB).
- HDD o Disco Duro (750GB a 1TB).

iv. Requerimientos de reportes

- Navegadores web o editores de texto actualizados para la generación, edición y visualización de archivos en formato XLM (Notepad, Atom, Sublime, …).

d. Marco Legal

Ley 11723 de 1933, diseñada y promulgada para preservar los derechos de autor o de propiedad intelectual, en diferentes ámbitos (artísticos, literarios, científicos, etc).

En esta ley se establecen sanciones económicas y penales a quienes violen la normatividad establecida en la misma.

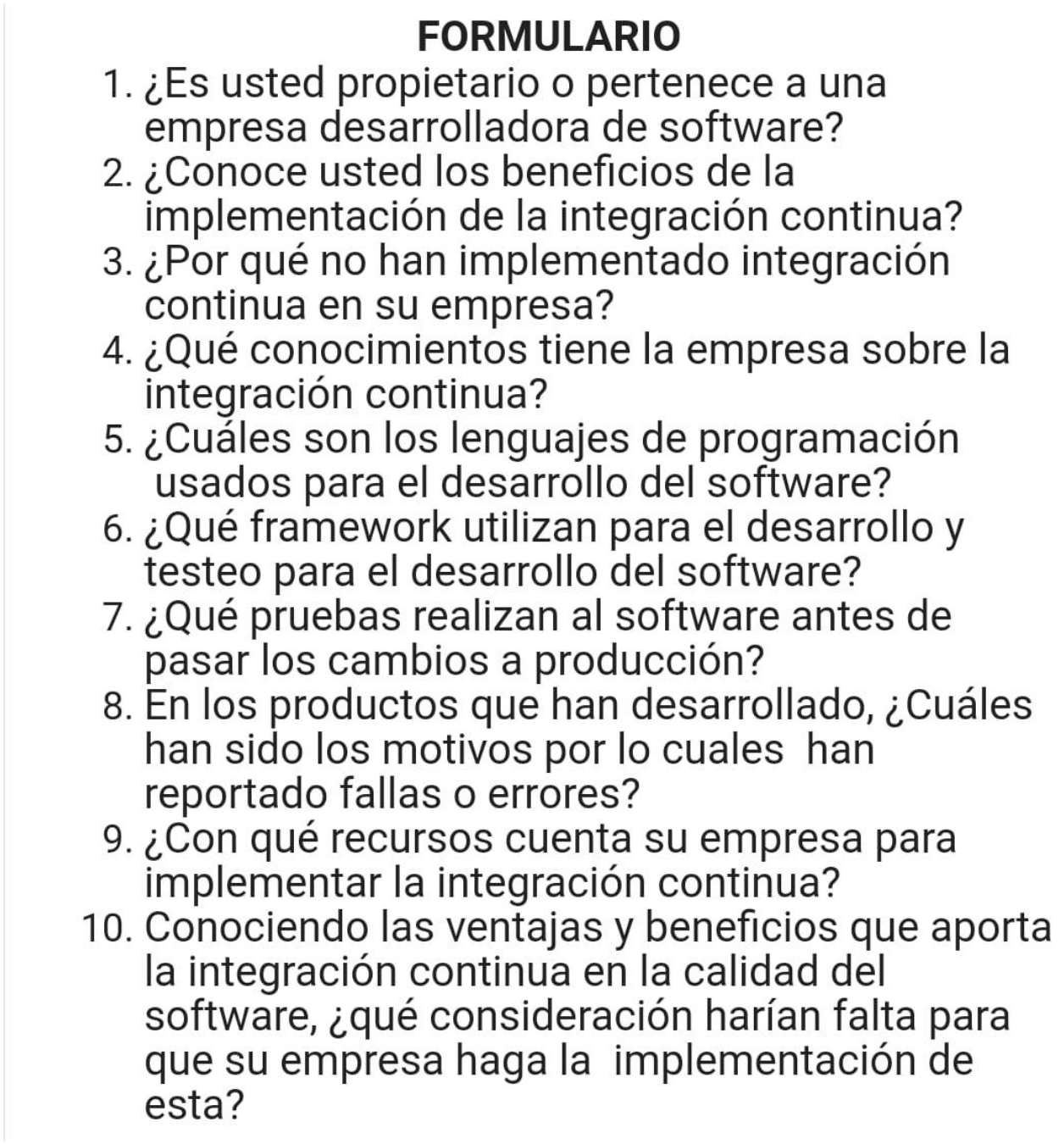
En 1998 aparece la Ley 25036 que realiza algunos ajustes a la ley 11723 de 1993 en lo que respecta a introducir la protección de los derechos de autor al software, programas de computación, contratos de licencia para uso o reproducción de estos.

9. CONCLUSIONES

- A través de la investigación realizada se logró obtener información valiosa y verídica acerca de la integración continua lo que aportó los conocimientos para la realización del presente trabajo brindando así unas bases para su posterior aplicación.
- Hemos podido encontrar como día a día, la demanda de situaciones laborales crece, en consecuencia, genera menor rendimiento de los trabajadores o funcionarios, estrés, frecuentes errores humanos en el código fuente, por ello, para agilizar y facilitar el trabajo de desarrollo, test y puesta en producción del software hace necesario automatizar las tareas. Es por ello por lo que las empresas buscan asegurar la calidad y eficiencia del software mediante la detección de errores en tiempo real y reduciendo los costos de corrección y prevención, con la ayuda de servidores de integración continua. Para el logro de este propósito empresarial, en la actualidad, existen diversas herramientas que ofrecen o permiten la integración continúa acomodándose a las necesidades y a los lenguajes de programación usados por estas.
- En el caso estudio se pudo evidenciar la importancia de implementar la integración continua como soporte en el aseguramiento de la calidad durante el ciclo de vida del desarrollo del software. Esto debido a que si bien es cierto es necesario la toma de requerimiento su posterior análisis y diseño, al momento de desarrollar software contar con un servidor de integración continua reduce el riesgo de errores humanos, además al tener un set de pruebas se automatizan para su posterior ejecución de manera automática, obteniendo como resultado la reducción costos tanto en tiempo, como en dinero puesto que los errores son detectados durante el desarrollo o mantenimiento, y no en producción.

  En resumen, al implementar la integración continua se evidenció mejora en la calidad del código, mejora de productividad, detección de errores, y actualizaciones.

Anexos

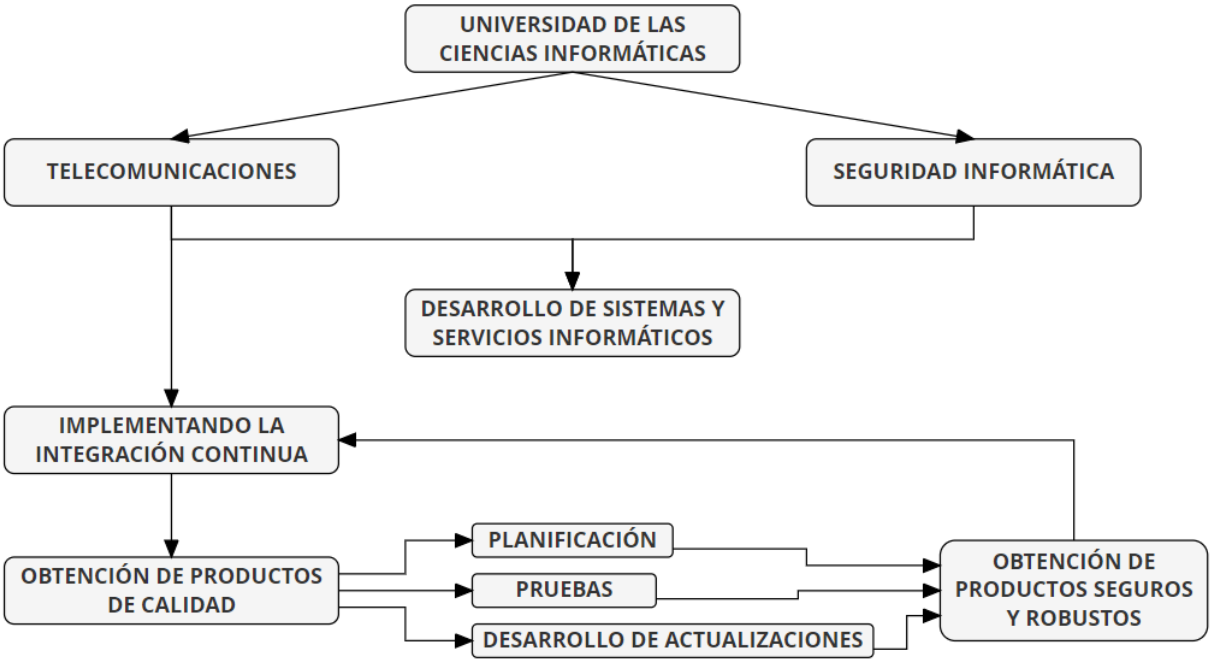


FORMULARIO

1. ¿Es usted propietario o pertenece a una empresa desarrolladora de software?
2. ¿Conoce usted los beneficios de la implementación de la integración continua?
3. ¿Por qué no han implementado integración continua en su empresa?
4. ¿Qué conocimientos tiene la empresa sobre la integración continua?
5. ¿Cuáles son los lenguajes de programación usados para el desarrollo del software?
6. ¿Qué framework utilizan para el desarrollo y testeo para el desarrollo del software?
7. ¿Qué pruebas realizan al software antes de pasar los cambios a producción?
8. En los productos que han desarrollado, ¿Cuáles han sido los motivos por lo cuales han reportado fallas o errores?
9. ¿Con qué recursos cuenta su empresa para implementar la integración continua?
10. Conociendo las ventajas y beneficios que aporta la integración continua en la calidad del software, ¿qué consideración harían falta para que su empresa haga la implementación de esta?

Anexo A. Formulario del focus group

Fuente: Elaboración propia



Anexo b. Integración continua

Fuente: Elaboración propia

BIBLIOGRAFIA

1&1 IONOS España S.L.U. (2019, 4 marzo). *Integración continua*. IONOS Digital Guide.

https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/integracion-continua/

Amazon. (s.f.-b). Integración continua del software | Pruebas automatizadas | AWS. Recuperado de https://aws.amazon.com/es/devops/continuous-integration/

Solving Ad Hoc. (2017, 7 agosto). Qué es la integración continua y cómo aplicarla a la empresa. Recuperado de https://solvingadhoc.com/la-integracion-continua-aplicarla-la-empresa/

Herrera Pérez, S. (2017). Análisis e implementación de la integración continua en empresas de software. Recuperado de http://repositorio.unican.es/xmlui/bitstream/handle/10902/12058/396597.pdf

Canos, J., Letelier, P., Penadés, M. (2012). Metodologías ágiles en el desarrollo de software. Recuperado de http://roa.ult.edu.cu/bitstream/123456789/476/1/TodoAgil.pdf

Navarro Cadavid, A., Fernández Martínez, J. M., Morales Vélez, J. (2013). Revisión de metodologías ágiles para el desarrollo de software. Recuperado de https://www.redalyc.org/pdf/4962/496250736004.pdf

Laínez Fuentes, J., R. (2015). Desarrollo de Software ÁGIL: Extreme Programming y Scrum. Recuperado de https://books.google.es/books?hl=es&lr=lang_es&id=M4fJCgAAQBAJ&ots=1G-MOxU1mm&sig=5h6r7sbMF5jdjMwRBL83yWIHD1w

Melo Vital, L., & Callejas Machacon, Z. 1. Ley De Software Libre En Colombia - Unicesar-legislación. Recuperado de https://sites.google.com/site/unicesarlegislacion/Inf/home/Ley-De-Software-Libre-En-Colombia

Fontela, C. (2020). Construcción de software: una mirada ágil (Spanish Edition). EDUNTREF.

Cadavid, A. (2013, 4 junio). Revisión de metodologías ágiles para el desarrollo de software. Dialnet. https://dialnet.unirioja.es/servlet/articulo?codigo=4752083

Valverde, D. (2007). *Metodologías ágiles*. Universidad nacional de trujillo. https://3699a943-a-e1e09691-s-sites.googlegroups.com/a/unicesar.edu.co/maribelromero2/Metodologias%20Agiles.pdf?attachauth=ANoY7crrOqmDcD20S7r51ZC6ZAJ0b7KPmVUYgDIlHUTtPtAz7E-rTZHP9NgPPT56hFIAnWFijS9zlpEfuCW8HvGafBeDgTbZVo64duaZKPc3YYwmQsbqtGHyEz0J9jmmCzWgVi2o6dqldYnKN1SBiili9ahu3nqR1NUrGrNcBmdqqWxZ9l9xj4bD680tk5oozk2VhVh3p9v8-wDSlfMJj66aHc1T4sbWcI_fzponsqj2bvgMBl__t1I%3D&attredirects=0