INTEGRACIÓN CONTINUA

Ailín Julieta Asis Martínez - 69639
Athina Bringas - 73126
José María Amiune - 73783
Santiago Iván Prandi - 72144

RESUMEN: *La práctica de desarrollo de software conocida como Integración Continua (**CI**: Continuous Integration), nos permite tener un ciclo de lanzamiento más corto y frecuente, mejorar la calidad del software y aumentar la productividad del equipo de desarrollo. Creada sobre los principios de las metodologías ágiles y encuadrada dentro de la gestión de la configuración de software, la integración continua pone un gran énfasis en la automatización de pruebas a la hora de integrar el software logrando reducir la necesidad de intervención humana minimizando el riesgo de errores.*

La integración continua, hace que la entrega continua tenga éxito y que el software esté "a solo un clic" de la implementación. Es importante saber que para obtener buenos resultados no alcanza con saber la teoría, sino que es necesario concientización y acuerdos por parte del equipo que la van llevar a cabo para poder tener éxito.

PALABRAS CLAVE: Automatización, Integración Contínua, Pruebas, Calidad.

1 INTRODUCCIÓN

Anteriormente, era común que los desarrolladores de un equipo trabajasen aislados durante un largo periodo de tiempo buscando integrar todos los cambios en la versión maestra una vez completado su trabajo. Como consecuencia, incrementó la dificultad de unir los cambios en el código, además de dar lugar a la acumulación de errores que finalmente terminaban sin ser corregidos. Estos factores hicieron que resultase más difícil proporcionar actualizaciones a los clientes con rapidez. Como solución a esta problemática surge una práctica que permite la detección de errores durante las integraciones del proyecto y no al final del desarrollo y es denominada Integración continua.

Al hablar de Integración continua es necesario tener en cuenta que la misma integra lo que se conoce como *Gestión de Configuración de Software (SCM).*

SCM es un método para llevar el control al desarrollo de software y al proceso de gestión de software. Mejora la productividad y aumenta la calidad del software.

En términos generales se utiliza para desarrollar y mantener el software de manera más eficiente, y esto se logra mejorando la responsabilidad, la capacidad de auditoría, la reproducibilidad, la trazabilidad y la coordinación. Dentro de SCM se distinguen tres prácticas íntimamente relacionadas que son la ya mencionada Integración Continua (CI), la Entrega Continua (CDE) y el Despliegue Continuo (CD).

En el presente informe nos enfocaremos en la primera de ellas, describiremos su concepto, como está relacionada con las demás prácticas, sus principios, ventajas y desventajas, y otros aspectos que son considerados fundamentales para exponer el tema. Por último trataremos sobre su aplicación y los resultados a obtener con ella, pudiendo conocer su importancia a la hora de desarrollar software de alta calidad.

2. CONTENIDO

2.1 ¿QUÉ ES CI?

Martin Fowler es uno de los autores más reconocidos en el ámbito de prácticas ágiles y define la integración continua de la siguiente manera:

"La integración continua es una práctica de desarrollo de software en la que miembros de un equipo integran su trabajo frecuentemente, típicamente cada persona integra al menos una vez al día, generando varias versiones por día. Cada versión ejecutable es verificada por un sistema automático de integración y pruebas para detectar errores de integración lo más rápido posible." [2]

La CI reduce riesgos, ya que todos los problemas se pueden identificar y solucionar fácilmente con cada integración. Esto permite ahorrar dinero, tiempo y esfuerzo y además ofrece una visibilidad completa del código del proyecto.

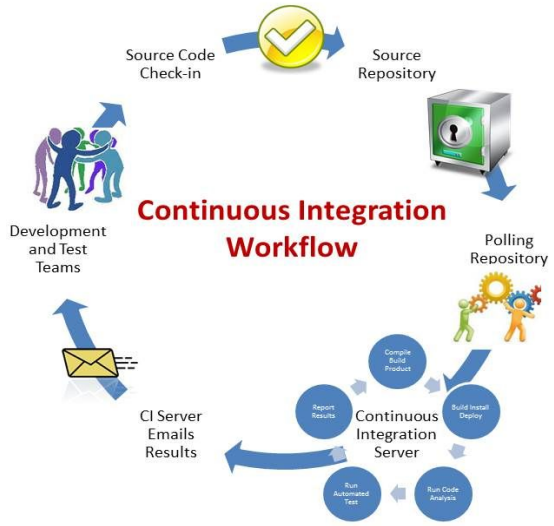

Figura 1. Workflow de Integración Continua.

2.1 RELACIÓN ENTRE CI, CDE Y CD

Con la creciente competencia en el mercado de software, las organizaciones prestan mucha atención y asignan recursos para desarrollar y entregar software de alta calidad a un ritmo acelerado. La integración continua (CI), la entrega continua (CDE) y la implementación continua (CD), llamadas prácticas continuas, son algunas de las prácticas destinadas a ayudar a las organizaciones a acelerar su desarrollo y entrega de funciones de software sin comprometer la calidad. Mientras que CI se enfoca en la integración del trabajo en progreso varias veces al día, CDE y CD tratan sobre la capacidad de liberar valor de negocio de manera rápida y confiable a los clientes al utilizar la automatización tanto como sea posible.

La importancia de la integración continua radica en el hecho de que es la base de las demás, la primera que debe realizarse, por lo cual para obtener buenos resultados es importante que sea confiable, estable y la de más alta prioridad.

El análisis de las prácticas de CI, CDE y CD de manera integrada brinda la oportunidad de comprender qué desafíos impiden adoptar cada práctica continua, cómo se relacionan entre sí y qué enfoques, herramientas y prácticas asociadas existen para apoyar y facilitar cada una. Además, esto ayuda a las organizaciones que se dedican al software a adoptar prácticas continuas paso a paso y pasar sin problemas de una práctica a la otra.

Figura 2. Relación entre CI, CDE y CD.

2.2 LITERATURA EXISTENTE

Se han hallado dos estudios que investigaron la integración continua en la literatura.

Tabla 1. Estudios sobre CI.

Estudio	**Foco**	**# papers incluidos**
Stahl and Bosch [3]	CI	46
Eck et al. [4]	CI	43

El primero ha explorado las diferencias en las implementaciones de la práctica de CI y el segundo está centrado en las implicaciones organizacionales de la asimilación de CI.

2.3 INTEGRACIONES FRECUENTES

Hay un efecto contraintuitivo fundamental en el centro de la integración continua. Es que es mejor integrar a menudo que integrar raramente. Para aquellos que tienen experiencia haciéndolo, esto es obvio pero para aquellos que no la tienen, esto parece una contradicción con la experiencia directa.

Si se integra sólo ocasionalmente, la integración es un ejercicio doloroso por así decirlo, ya que requiere mucho tiempo y energía. De hecho, es lo suficientemente incómoda como para que lo último que se quiera sea hacerlo con más frecuencia. Se suele pensar que en un proyecto muy grande, no es posible hacer una compilación diaria. Sin embargo, hay proyectos que lo hacen. La razón de que esto sea posible es que el esfuerzo de integración es exponencialmente proporcional a la cantidad de tiempo entre integraciones.

Una clave para esto es la *automatización*. La mayor parte de la integración puede y debe hacerse automáticamente. Obtener las fuentes, compilar, vincular y realizar pruebas significativas se puede hacer de forma automática. Se finaliza con una indicación de si la compilación funcionó: sí o no. En caso afirmativo, se ignora, en caso negativo, se debería poder deshacer fácilmente el último cambio en la configuración y asegurarse de que la compilación funcionará esta vez. No se debe pensar para obtener una construcción funcional.

Con un proceso automatizado se puede integrar con la frecuencia que se desee. La única limitación es la cantidad de tiempo que lleva hacer la compilación.

2.4 PRUEBAS AUTOMATIZADAS

El primer paso para alcanzar la integración continua es configurar *pruebas automatizadas***.** Con ellas se logra obtener todos los beneficios de esta práctica.

Existen diferentes tipos de pruebas, entre ellas se encuentran:

- Las pruebas unitarias: tienen un alcance limitado y, por lo general, verifican el comportamiento de métodos o funciones individuales.
- Las pruebas de integración: aseguran que varios componentes se comporten correctamente juntos. Esto puede involucrar varias clases, así como probar la integración con otros servicios.
- Las pruebas de aceptación: son similares a las pruebas de integración, pero se centran en los casos de negocio en lugar de los componentes en sí.
- Las pruebas de IU: aseguran que la aplicación funcione correctamente desde la perspectiva del usuario, es decir, sirve para comprobar que una funcionalidad específica, funciona como se espera.

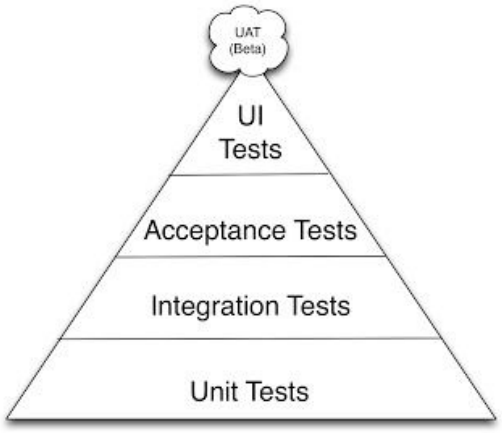

Figura 3. Jerarquía de pruebas realizadas.

Si bien la automatización de pruebas es una parte clave de la integración continua, no es suficiente por sí sola. Se requiere una cultura de trabajo tal que vaya de la mano con la práctica en cuestión, se necesita la concientización en todos los involucrados de la importancia que tiene

cumplir con las normas establecidas para el proceso de Integración Continua. Esto implica, entre otras cosas, que los desarrolladores no trabajen días en una característica sin fusionar sus cambios con la rama principal.

2.5 BENEFICIOS y DESAFÍOS DE LA INTEGRACIÓN CONTINUA

Llevar a cabo esta práctica trae beneficios entre los cuales se encuentran:

- Mejora la calidad del software: la integración continua contribuye a minimizar los problemas en los sistemas por errores de código. Provee un código más robusto mejorando la calidad del software.
- Detección de errores de manera más rápida y fácil: al integrar continuamente, de forma periódica, es mucho más fácil detectar errores lo que permite resolverlos de forma rápida y fácil. La integración continua no elimina los errores lo que hace es detectarlos oportunamente.
- Reduce tareas repetitivas y manuales: los procesos manuales repetitivos son lentos y propensos a cometer errores por lo que reducirlos implica un gran beneficio para asegurar que los procesos se realizarán aplicando los mismos estándares cada vez.
- Puede crear versiones de prueba en cualquier momento: al estar constantemente integrando el código es posible liberar software en cualquier momento, y sin los errores que previamente se detectaron y solucionaron.
- Completa visibilidad del proyecto: se tiene información concreta del avance del proyecto y métricas sobre la calidad del código que se está desarrollando. Ésto permite tomar mejores decisiones a la hora de querer hacer cambios o modificaciones.
- Mayor confianza y seguridad del equipo de trabajo: la integración continua permite obtener un software probado y funcional en todo momento, lo que contribuye a la confianza del equipo de trabajo pudiendo ir viendo y probando los avances del software. Ya no se requiere esperar al final del proyecto para verificar que todo se ha realizado de la forma correcta.

Sin embargo la integración continua presenta desafíos, entre los cuales se encuentran:

- Cambio de procesos habituales.
- Al ser una técnica nueva, cuesta la adopción de esta práctica.
- Precisa recursos adicionales como servidores y entornos nuevos.
- Se requiere la elaboración de planes de pruebas propios lo que conlleva en gastos adicionales no solo económico sino también de tiempo y otros recursos.
- Pueden generarse demoras en la integración del código ya sea por el surgimiento de errores o por conflictos derivados del intento de una integración simultánea.

2.6 PARTICIPACIÓN EN EL DESARROLLO DE SOFTWARE

La integración de un proyecto de software es un proceso largo e impredecible. Esto ha sido así desde hace mucho tiempo. Pero no necesariamente debe ser de esta manera, ya que si se trata a la integración, no como un evento que se produce separado del desarrollo, sino como una parte más del trabajo individual de cada desarrollador, cuyo código está cerca del estado común del proyecto, este trabajo puede ser integrado rápidamente y, en caso de la aparición de errores, los mismos pueden ser encontrados y arreglados en el momento..

Esto no es el resultado de una herramienta compleja y costosa, sino de simples prácticas que consisten en que todos los integrantes del equipo integren frecuentemente sobre un repositorio común y controlado.

2.7 PROCEDIMIENTO

La integración continua es una práctica en la cual los miembros de un grupo de desarrollo integran (compilación y ejecución) los distintos componentes de un proyecto con una frecuencia especificada. Cada integración se realiza de forma automática (incluyendo sus casos de prueba) con el fin de detectar los errores de integración lo antes posible. Según Martin Fowler [2], muchos equipos de desarrollo han encontrado que este enfoque reduce significativamente los problemas de integración y permite que los equipos desarrollen software cohesivo más rápido.

Un escenario típico de integración continua se compone de la siguiente manera:

- Primero el desarrollador realiza un commit de su trabajo al repositorio de control de versión a la vez que el servidor de integración continua (CI) verifica cambios en el repositorio, por ejemplo cada 10 minutos.
- Luego, el servidor de CI detecta los cambios en el repositorio de control de versión, extrayendo el último commit que se ha realizado y ejecutando una build script que se encarga de integrar los distintos componentes del software en desarrollo.
- El servidor de CI genera un feedback con los resultados del building, estos son enviados a los miembros del proyecto previamente especificados.
- El servidor de CI continua revisando cambios en el repositorio de control de versiones y así se completa el ciclo.

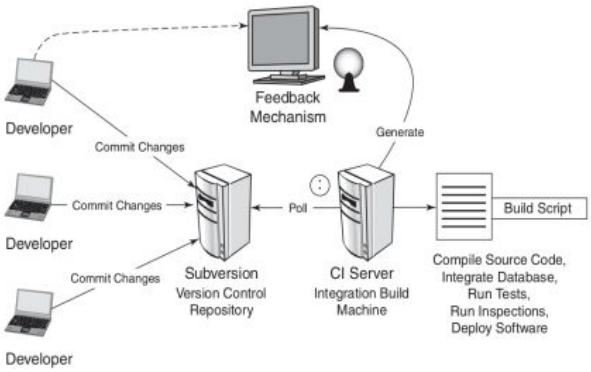

Figura 3: Escenario de Integración continua

2.8 HERRAMIENTAS

Si bien la integración continua no requiere necesariamente de alguna herramienta, estas pueden ayudar a que el trabajo se realice de manera más fácil y rápida.

Hay distintos tipos de herramientas, estas ayudan en la compilación del proyecto y en el control de versiones. Entre ellas se pueden mencionar:

- Jenkins: Es un servidor de código abierto que organiza las acciones para llevar a cabo la integración continua. Tiene como objetivo supervisar las tareas repetidas que surgen durante el desarrollo del proyecto. Admite el ciclo completo de desarrollo, desde la construcción hasta la implementación incluyendo otras etapas del ciclo de vida de desarrollo de software.
- Travis CI: es una herramienta que tiene como particularidad muy importante la compatibilidad con GitHub. Da soporte a 21 lenguajes distintos entre ellos está Java, C, C#, etc.; y se implementa en múltiples servicios de la nube.
- Bamboo: pertenece a la empresa Atlassian y cuenta con interfaz web y tecnología Java. Supone una ayuda a los desarrolladores mediante la automatización de procesos y funciona con diferentes herramientas de compilación.
- Circle CI: proporciona servicios y soporte de clase empresarial, con la flexibilidad de una startup. Existen dos versiones que permiten trabajar directamente en la nube o bien en un servidor local propio.
- CruiseControl: es un software libre basado en Java y compatible con varias plataformas. Provee una interfaz web que proporciona detalles de las compilaciones actuales y anteriores.
- CodeShip: es un servicio de entrega continua que se centra en la velocidad, la fiabilidad y simplicidad. Se basan en la tecnología de contenedores, creándose gracias a esto automatismo fácilmente. Admite distintos lenguajes, entre ellos se encuentra Java, PHP, Node.js, etc.
- TeamCity: es un software comercial y pone mucho énfasis en la interoperabilidad con otras herramientas. Tiene como característica clave la realización de “Pres-tested commits”, es decir, un enfoque que evita que el código defectuoso se convierta en una compilación, por lo que el proceso de todo el equipo no se ve afectado.

2.9 REPOSITORIOS

A lo largo de los años se han desarrollado múltiples herramientas para facilitar la tarea de la integración en donde se involucran muchos archivos de desarrollo que son utilizados por múltiples desarrolladores al mismo tiempo. Una buena coordinación en el uso de repositorios resulta fundamental para una integración rápida y para solucionar inmediatamente los conflictos y problemas que surgen de ella. Esta es una de las herramientas más importantes dentro de CI ya que de se depende de ella durante todo el desarrollo del proyecto y para las tareas del CI.

A pesar de su conveniencia existen todavía varios proyectos que no hacen uso de estas herramientas, las cuales no necesariamente están limitadas bajo una licencia propietaria ni bajo costo, sino que existen múltiples alternativas gratis y disponibles al público en general.

Los usuarios deberían conocer la fuente de donde provienen los archivos que están siendo desarrollados. Todo lo necesario para el proyecto debería estar dentro del repositorio. Para eso se debe guardar todos los archivos que hagan falta para compilar el software. La idea es poder descargar una copia del mismo en una máquina nueva y poder compilar el software que se está desarrollando sin muchas complicaciones. Esto también es ideal para guardar configuraciones sobre el entorno de desarrollo, si es que eso resulta más conveniente para el equipo.

En el repositorio no se debería guardar ningun archivo resultante de la compilación ya que no son necesarios y se pueden volver a construir utilizando el código fuente.

Otra de las funcionalidades de estas herramientas es la posibilidad de crear ramas (copias de la línea base del proyecto o de otras ramas) que permite manejar un desarrollo simultáneo sobre el mismo código. Esto se debería mantener al mínimo ya que se suele abusar de esta característica, lo que trae problemas.

Ejemplos en donde se justifica pueden ser ramas para arreglar algún bug antes de lanzar una Release o experimentos temporales.

2.10 BUENAS PRÁCTICAS

Cuando se comienza a trabajar de este modo, es importante aclarar ciertas reglas. Generalmente los desarrolladores buscan guiarse por los principios establecidos por Martin Fowler [2] para aplicar la Integración Continua.

- Mantener repositorios de una sola fuente: Para la integración continua es clave la utilización de un único repositorio. Los archivos deberían estar correctamente ordenados y al alcance de todos aquellos que requieran de estos. Es de suma importancia que todos los cambios sean incorporados a la brevedad en el repositorio, para que estén a disposición de todos.
- Automatizar la compilación: La automatización es una de las características principales de CI. Un error común es no incluir todo lo necesario en la compilación automatizada. Por ejemplo durante la compilación se debe obtener el esquema de la base de datos desde el repositorio y activarlo en el entorno de ejecución para poder obtener un proyecto funcional. Es ideal que todas las tareas necesarias sean ejecutadas a partir de un único comando, evitando así errores al introducir mal las instrucciones disparadoras de la compilación.
- Hacer que tu compilación se auto-pruebe: Una buena forma de detectar errores de forma más rápida y eficiente es incluir pruebas automatizadas en el proceso de compilación. Si bien no son infalibles, son de mucha ayuda. Estas deben poder iniciar desde un comando simple y autocomprobarse. Lo ideal es implementar un plan completo de pruebas para así poder detectar errores no solo vinculados a problemas en el código sino también en la funcionalidad del sistema en sí.
- Todos hacen un commit en la Línea Base todos los días: Es clave que todos respeten el hecho de la integración de código. Puede suceder que alguno de los desarrolladores no íntegre su trabajo y como consecuencia el grupo en general partirá de premisas falsas, es decir, de código anterior o desactualizado. En este caso, se prestará a inconsistencias y al surgimiento de nuevos problemas.
- Cada commit debería compilar la Línea Base en una maquina de integración: La versión principal se debe mantener en correcto funcionamiento siempre, y cada compilación debe dar como resultado el producto funcionando. En caso contrario, es obligación del desarrollador realizar todas las pruebas que sean necesarias y cambios para dejar el mismo sin errores de ningún tipo.
- Arreglar compilaciones rotas inmediatamente: No es un problema tener que modificar el código por el no funcionamiento de las compilaciones, siempre y cuando las modificaciones se realicen inmediatamente. Todos los desarrolladores deben abocarse al correcto funcionamiento de la Línea Base, en caso contrario se desencadenaran fallos e inconsistencias.
- Mantener las compilaciones rápidas: El propósito de la integración continua es proporcionar una retroalimentación rápida. El desarrollador debe asegurarse que funcione pero, por una cuestión de tiempo y costo no se pueden realizar todas las pruebas posibles, por lo tanto, se aplica un sistema de dos fases donde en la primera se realizan

pruebas en donde la compilación es rápida, mientras que en la segunda se realizarán pruebas más exhaustivas.

- Probar en una copia del entorno de producción: Las pruebas deberán realizarse en un espacio seguro e igual al entorno de producción. Esto puede significar un costo elevado en primera instancia pero reducirá la posibilidad de fracaso, ergo, los gastos en esta puesta en marcha serán menores a una falla a posteriori.
- Facilitar a todos la obtención del último ejecutable: Una de las tareas más difíciles en el desarrollo de un producto de software es predecir el comportamiento que debe tener en base a lo que el cliente desea. Las metodologías ágiles aprovechan esto basándose en la plena comunicación con el interesado. Para ayudar a que esto funcione, cualquier persona involucrada en un proyecto de software debería poder obtener el último ejecutable y ponerlo en marcha para demostraciones, pruebas o para ver qué cambios se produjeron últimamente. Esto debería ser relativamente fácil ya que la integración continua tiene como premisa la utilización de un único repositorio al que todos puedan acceder y obtengan la última versión del ejecutable.
- Todos pueden ver lo que está pasando: La integración continua, como hemos mencionado anteriormente, requiere de una buena comunicación, por lo que debe asegurarse de que todos puedan ver fácilmente el estado del sistema, los cambios que se le han realizado y quién o quiénes son los responsables de los mismos.
- Despliegue automático: Para llevar a cabo la integración continua se requieren de múltiples entornos para ejecutar pruebas de confirmación y uno o más para ejecutar pruebas secundarias. Ha de automatizarse el despliegue del software, lo cual puede llevar mucho tiempo, y para ello será mejor hacerlo de forma automática utilizando scripts que faciliten el despliegue de aplicaciones entre entornos.

2.11 RECOMENDACIONES

Al momento de tomar la decisión de implementar un sistema de integración continua es importante tener en cuenta ciertas recomendaciones que harán que se desarrolle con mayor eficiencia obteniendo lo mejor de esta práctica agilista:

- Hacer commits con frecuencia.
- No hacer commit de código que no funciona.
- Escribir test unitarios automáticos.
- Todos los test y revisiones deben ejecutarse correctamente.
- Ejecutar builds privados.
- No tomar código del repositorio que no funciona.

3 CONCLUSIONES

Luego de recopilar y analizar la opinión de profesionales que compartieron su experiencia en la utilización de un sistema de integración continua, experimentamos de una forma más tangible los beneficios enunciados acerca de la aplicación de esta práctica agilista en los procesos de desarrollo de software. La integración continua no solo disminuye los conflictos a la hora de introducir cambios al código, si no que también mejora ampliamente la comunicación entre los participantes, aceitando así la dinámica del equipo, y abriendo una ventana importante para que el departamento de desarrollo se comunique y realice feedbacks más frecuentes con el departamento de operaciones, eliminando de esta manera los tiempos muertos entre procesos. Los conocimientos adquiridos con la realización de dicho trabajo nos permitió afianzar los conceptos de metodologías ágiles, ya que la integración continua favorece una de las principales características de toda metodología agilista, la obtención de releases frecuentes.

4 REFERENCIAS

[1] Sander Rossel, “Continuous Integration, Delivery, and Deployment”, Birmingham: Packt Publishing, 2017, Cápitulo 1.
[2] Martin Fowler, “Continuous Integration”, 2016 [En línea]. Disponible en: https://www.martinfowler.com/articles/continuousIntegration.html
[3] D. Ståhl, and J. Bosch, “Modeling continuous integration practice differences in industry software development”, 2014.
[4] A. Eck, F. Uebernickel, and W. Brenner, “Fit for Continuous Integration: How Organizations Assimilate an Agile Practice”, 2014.
[5] ThoughtWorks, “Continuous Integration” [En línea].

Disponible en:
https://www.thoughtworks.com/continuous-integration
[6] Amazon Web Services, Inc. “¿Qué es la integración continua?” [En línea].
Disponible en:
https://aws.amazon.com/es/devops/continuous-integration
[7] Ionos, “La integración continua en el desarrollo de software” [En línea].
Disponible en:
https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/integracion-continua/
[8] Alicia Salamon et. al., “La Integración Continua Aplicada en el Desarrollo de Software en el Ámbito Científico –Técnico” [En línea].
Disponible en:
https://rdu.iua.edu.ar/bitstream/123456789/264/1/IC%20Aplicada%20en%20el%20Desarrollo%20de%20Software%20C-T.pdf
[9] Plantilla utilizada para el Informe Técnico, según la IEEE [En línea].
Disponible en:
https://electronicaunisangil.wordpress.com/2011/03/05/plantilla-informes-ieee/
[10] Guía para citar referencias, según la IEEE [En línea].
Disponible en:
http://www2.unavarra.es/gesadj/servicioBiblioteca/tutoriales/Citar_referenciar_(IEEE).pdf
[11] Guru99, “What is Jenkin for Continuous Integration?” [En línea].
Disponible en:
https://www.guru99.com/jenkin-continuous-integration.html
[12] Guru99, “Jenkins vs Travis-CI: What is the difference?” [En línea].
Disponible en:
https://www.guru99.com/jenkins-vs-travis.html#6
[13] Circle CI [En línea].
Disponible en: https://circleci.com/docs/2.0/about-circleci/
[14] Codeship Integration [En línea].
Disponible en:
https://docs.opsgenie.com/docs/codeship-integration
[15] Pres-tested commit (TeamCity) [En línea].
Disponible en:
https://confluence.jetbrains.com/display/TCD9/Pre-Tested+(Delayed)+Commit
[16] Gisell Chávez, “6 Beneficios de la Integración Continua” [En línea].
Disponible en:
https://www.smartnodus.cl/integracion-continua/
[17] Mojtaba Shahin, Muhammad Ali Babar, Liming Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices”, IEEE, 2017 [En línea].
Disponible en:
https://ieeexplore.ieee.org/abstract/document/7884954
[18] Ravi Verma, “Continuous Integration vs Continuous Delivery vs Continuous Deployment” [En línea].
Disponible en:
https://scmquest.com/continuous-integration-vs-continuous-delivery-vs-continuous-deployment/
[19] Tom Price, “Comprobación continua para la entrega continua: ¿Qué significa en la práctica?”, 2017 [En línea].
Disponible en:
https://www.ca.com/content/dam/ca/ar/files/white-paper/continuous-testing-for-continuous-delivery.pdf
[20] it-Mentor- capacitación y guía para el desarrollo de software, “Integración continua” [En línea]
http://materias.fi.uba.ar/7548/IntegracionContinua.pdf