UTN | FACULTAD REGIONAL CÓRDOBA
INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Ingeniería de Software

Trabajo Conceptual 1

Informe técnico: CI/CD

Docentes:

Judith Meles

Joaquin Robles

Curso: 4k1

Número de grupo: 3

Integrantes:

Julio Achával	74306
Tomás Liendo	70987
Facundo Monteros	69894
Miguel Ovejero	69939
Juan Revello	70085
Joaquín Velasco	72639

Integración Continua
Conceptos claves y su importancia en el desarrollo de software

Achával Vinuesa, Julio Guillermo; Liendo Loker, Tomas; Monteros, Facundo; Ovejero, Miguel; Revello, Juan Manuel; Velasco, Joaquín.
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL CÓRDOBA
{julioachavalvinuesa, TomasLLoker, facundomonteros24, miguelovejero50, juanmanuelrevello, joacovelascob}@gmail.com

Resumen

Para llevar a cabo un proyecto de software bajo los lineamientos de las metodologías ágiles, es imprescindible la aplicación de la CI, en la que los miembros de un equipo integran sus desarrollos de forma frecuente, siendo cada integración validada exhaustivamente de forma automática para detectar de manera temprana cualquier error que pueda ocurrir en el proceso.

Como cualquier otra práctica de la Ingeniería del Software, integrar componentes de software, es una actividad compleja. En este contexto, la CI contribuye con el desarrollo ágil, brindando un esquema que posibilita la realización de integraciones a medida que se lleva a cabo el desarrollo, generando incrementos pequeños y mostrando los resultados obtenidos.

Palabras clave: *software, ágil, integración, frecuente.*

Introducción

En el marco de las Metodologías Ágiles para la gestión de proyectos de software, surge la necesidad de citar los conceptos de CI/CD.

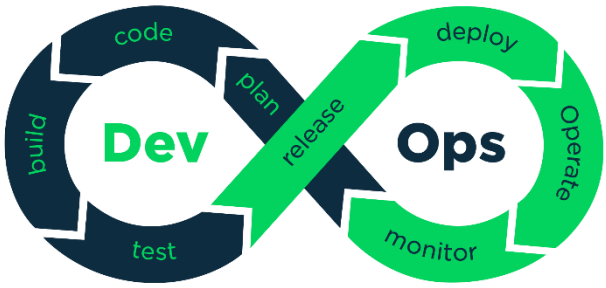

Figura 1. DevOps

Anteriormente, los desarrolladores programaban de manera independiente y al finalizar se realizaba una integración de todo el código. Esta tarea, que naturalmente es compleja, se complejiza aún más cuando la integración comprende a una gran cantidad de código y la misma se realiza de forma manual. La consecuencia de llevar a cabo esta práctica es un incremento en la probabilidad de ocurrencia de errores al momento de integrar, lo que implica a su vez una pérdida de tiempo para resolverlos, lo cual en el contexto del desarrollo ágil es cuestionable.

Esta problemática, impide entregar al cliente incrementos de producto terminados de manera frecuente y eficiente y la integración continua es la practica que permite solventar a la misma.

El presente informe tiene por fin profundizar sobre la Integración Continua, analizando en qué consiste la misma y cuál es su importancia para el desarrollo del software de alta calidad.

Los principios ágiles que brindan contexto a la implementación de esta práctica son principalmente los siguientes:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

Desarrollo

¿Qué es la Integración Continua?

La Integración Continua es una práctica, propuesta inicialmente por Martin Fowler, que consiste en hacer integraciones frecuentes en un proyecto para así poder detectar fallos de forma temprana.

Fowler la define como: "*...una práctica de desarrollo de software en la cual los miembros de un equipo integran su trabajo con frecuencia, por lo general cada persona integra su trabajo por lo menos una vez al día dando lugar a múltiples integraciones por día. Cada*

integración es verificada por una build automática (incluyendo pruebas) para detectar errores de integración lo más rápidamente posible. Muchos equipos encuentran que este enfoque conduce a la reducción de manera significativa de los problemas derivados de integración y permite a los equipos desarrollar software con mayor rapidez." [1].

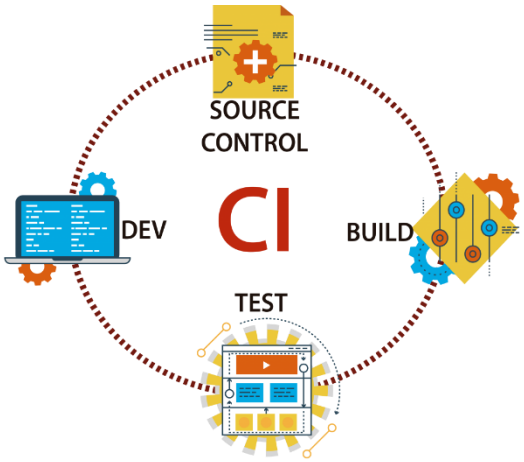

Figura 2: Ciclo de Integración Continua

Prácticas de la Integración Continua

La Integración Continua está basada fundamentalmente el siguiente conjunto de buenas practicas a seguir para su aplicación efectiva

Mantener un único repositorio fuente

Los proyectos de software, usualmente, implican varios archivos que deben organizarse en forma conjunta para conformar a un producto. Una de las tareas más costosas a la hora de desarrollar software es gestionar estos archivos y por consiguiente el código fuente de este. Para ser más eficientes, en la actualidad es indispensable la utilización de herramientas que permitan gestionar el código fuente y el versionado del producto.

La utilización y gestión de un único repositorio fuente brinda transparencia para los involucrados en la construcción del producto, ya que todos pueden visualizar los cambios que introduce el resto.

Una de las características de los sistemas de control de versiones es que permiten crear múltiples "ramas", para manejar diferentes flujos de desarrollo. Generalmente se tiene una línea o rama principal, que se corresponde con la rama de desarrollo actual del proyecto. Los desarrolladores implementan nuevas funcionalidades en ramas paralelas, y una vez que las mismas se finalizan (fueron desarrolladas y testeadas) se integran a la rama principal. Ver figura 4.

Figura 4. Ramas de desarrollo en paralelo

Automatizar la compilación

El acto de compilar un programa, convirtiendo el código fuente en un sistema ejecutable suele ser complejo y tedioso. A su vez, este proceso puede ser automatizable, reduciendo la intervención humana, y por consecuencia los errores que esta trae aparejada.

La compilación automatizada es una de las practicas principales que incluye la Integración Continua y esta puede ser implementada mediante servidores de automatización como Jenkins.

Un error frecuente es no incluir todo en la compilación automática. Con un único comando, encargado de la complicación, debería ser posible obtener un sistema ejecutable.

Hacer que la compilación se auto pruebe

Que un programa pueda ejecutarse, no implica que sea valido, es decir, que haga lo que tiene que hacer. Si bien los lenguajes de programación actuales permiten detectar errores sintácticos, existen otros errores asociados a la funcionalidad del sistema.

La inclusión de pruebas automáticas al proceso de compilación no garantiza una cobertura completa, pero es una forma rápida y eficiente de encontrar errores de manera temprana. Estas pruebas deben poder iniciarse desde un comando simple y autocomprobarse, de forma tal que, si una de ellas falla, entonces la compilación falla.

Compromiso con la Línea Principal todos los días

La integración se trata principalmente de comunicación, la cual permite a los desarrolladores informar a otros desarrolladores sobre los cambios que han realizado.

Si los cambios realizados por un desarrollador fueron compilados de manera exitosa, entonces esta listo para integrarse a la línea principal actualizada.

Este compromiso, permite identificar rápidamente eventuales conflictos que surgen al momento de combinar diferentes trabajos y resolverlos se transforma en una tarea relativamente mas simple utilizando un depurador de diferencias.

Cada commit debe permitir compilar la Línea Principal en una maquina de integración

El requisito previo al compromiso con la Línea Principal es mantenerla actualizada y realizar la complicación previo a la integración, de lo contrario,

probablemente estaríamos introduciendo cambios no validos ya que no pasaron por el proceso de compilación.

Los servidores de integración monitorean continuamente el estado del repositorio. Cada vez que se realiza un commit sobre el mismo, el servidor inicia el proceso de compilación de la Línea Principal e informa el resultado al desarrollador.

Arreglar las compilaciones fallidas inmediatamente

Según Kent Beck "nadie tiene una tarea de mayor prioridad que arreglar la compilación". Usualmente, la forma mas rápida de resolver este tipo de problema es revertir el ultimo commit realizado en la Línea Principal.

La aplicación de la buena practica anterior permite reducir la cantidad de arreglos. En los equipos de desarrollo mas inmaduros, esto ocurre con mayor frecuencia. Con el tiempo, integrar frecuentemente cambios previamente compilados, se convierte en un habito para el equipo.

Mantener la compilación rápida

El objetivo de la CI es obtener una retroalimentación rápida. El proceso de compilación puede implicar varias horas, en función de su complejidad y comprometer el objetivo principal de la CI.

La solución a esto es realizar lo que se conoce como "canalización de compilación" o compilación por etapas. La primera etapa es una compilación rápida, que incluye pruebas unitarias que no afectan a la base de datos. La segunda etapa, incluye pruebas afectan a la base de datos real e implican un comportamiento más extremo a extremo e insume una mayor cantidad de tiempo.

Probar en una copia del entorno de producción

Las diferencias entre los entornos de producción y de aquel en el cual se realizan las pruebas implican un riesgo de que lo que suceda durante la prueba no suceda en producción. Bajo esta premisa, es importante realizar las pruebas en un entorno que sea lo mas parecido a producción.

Si bien cumplir con esta practica implica un costo elevado, el costo que genera la ocurrencia de errores en el entorno de producción es aun mayor.

Facilitar a cualquier persona obtener el ultimo ejecutable

La mejor manera de validar un programa es utilizándolo. De esta manera, cualquier persona involucrada en un proyecto de software debería poder obtener el último ejecutable y poder ejecutarlo: para demostraciones, pruebas exploratorias o simplemente para ver qué cuales fueron los últimos cambios. La CI facilita hacer esto, ya que los desarrolladores integran sus cambios frecuentemente en un repositorio común.

Todos pueden ver lo que pasa

Como mencionamos anteriormente, la CI brinda transparencia, ya que todos pueden ver el estado del sistema en cualquier momento. Esto a su vez facilita la comunicación entre los involucrados, ya que todos conocen los cambios realizados por cada integrante del equipo.

Despliegue automatizado

Dada la existencia varios entornos para realizar los ciclos de prueba, una buena practica es disponer de scripts que permitan desplegar el sistema fácilmente en cualquier entorno. Esto permite acelerar el proceso y reducir los errores que puedan ocurrir.

Automatización de la pruebas

Anteriormente hemos mencionado la importancia de construir pruebas automáticas e incluirlas en el proceso de compilación del sistema. Adoptar esta practica, maximiza los beneficios que provee la CI.

El testing es *"el proceso de analizar un elemento de software para detectar las diferencias entre lo existente y lo requerido (es decir, errores) y para evaluar las características del elemento de software"* [15].

Poder probas el 100% de las funcionalidades del software es utópico. Sin embargo, existen técnicas para maximizar la cobertura de prueba y minimizar el esfuerzo que demanda esta actividad.

Existen diferentes niveles de prueba, algunos mas automatizables que otros.

1. **Prueba unitaria:** es simplemente un fragmento de código que llama a un método (el método a probar) con un entrada predefinida y comprueba si el resultado es el esperado. Si el resultado es correcto, informa de éxito, de lo contrario informa error. La prueba unitaria, como el nombre lo indica, prueba unidades de código pequeñas y aisladas. Estas pruebas son las mas fáciles de automatizar.
2. **Prueba de integración:** permite probar que los diferentes componentes del sistema funcionan correctamente en conjunto, validando las interfaces que los interconectan.
3. **Prueba de sistema:** consiste en probar la construcción final del sistema como un todo. Este tipo de pruebas debe realizarse en entornos similares al de producción, de forma tal que se reduzca el riesgo por diferencia entre entornos.
4. **Prueba de aceptación:** este tipo de prueba es llevada a cabo por el usuario del sistema. Estas pruebas no son automatizadas, ya que no tienen como objetivo encontrar defectos, sino que permiten validar y familiarizarse con el sistema.

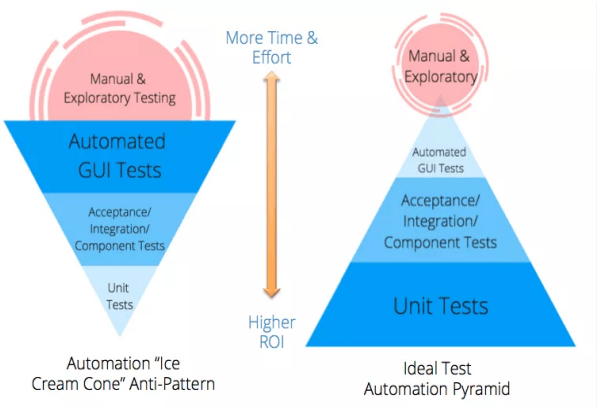

Figura 5. Automatización de pruebas

Condiciones previas para implementar una correcta Integración Continua

A la hora de implementar CI se debe tener en cuenta ciertos criterios previos, entre ellos:

- Concientizar al equipo de realizar integraciones con cierta frecuencia, esto se relaciona directamente con esencia de las metodologías Ágiles.
- Realizar pruebas unitarias completas y correctas a cada porción de código que se integrará.
- La gestión de Configuración y la Gestión de despliegue documentada y divulgada deben estar perfectamente incorporadas al equipo de trabajo para que la integración continua se pueda desarrollar correctamente y para que el proceso que implica la misma, pueda, vaya la redundancia integrarse y estandarizarse dentro de los procesos de la organización y de los equipos de trabajo.
- Los integrantes del equipo de trabajo deben tener conocimientos en Arquitectura, altos estándares de codificación, formación y vocación en metodologías Ágiles con capacidad para el trabajo colaborativo y proactivos y predispuestos a solucionar errores y problemas en cambios y funcionalidades que no desarrollaron.
- Se debe definir quien o quienes se encargarán de la configuración del Servidor de Integración Continua y de cada uno de sus componentes, como también el manejo de grupos, roles y usuarios con sus permisos en la herramienta elegida.

Procedimiento a seguir

Un escenario típico de Integración Continua se compone de la siguiente manera [3]:

1. Un desarrollador realiza un commit de su trabajo al repositorio de control de versiones, mientras que el servidor de CI verifica los cambios a intervalos de tiempo regulares, por ejemplo, cada 5 minutos.
2. El servidor de CI detecta los cambios en el repositorio de control de versión, extrayendo el ultimo commit que ha realizado y ejecuta un build script que se encarga de integrar los distintos componentes del software en desarrollo.
3. El servidor de CI genera la retroalimentación con los resultado del proceso de compilación, el cual es enviado a todos los integrantes que se especifique del proyecto
4. El servidor de CI continúa revisando los cambios en el repositorio de control de versiones.

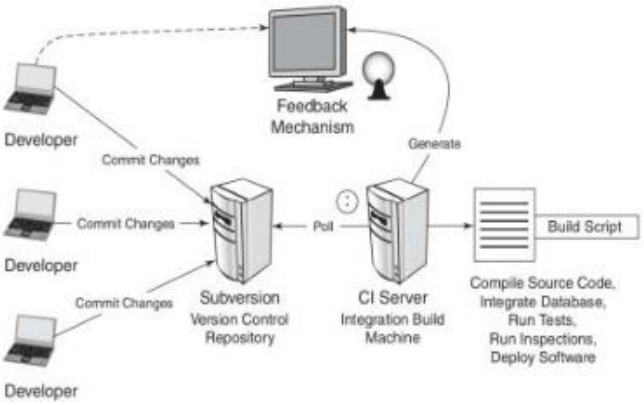

Figura 6. Escenario típico de Integración Continua

En este conjunto de pasos visualiza el proceso de compilación automática y la participación del repositorio de control de versiones. Anteriormente, nos referimos a la importancia de mantener un único repositorio fuente que nos permita el control de las diferentes versiones del producto. Esto contribuye con la rastreabilidad y posibilita la recuperación de versiones especificas en un futuro.

Beneficios de la Integración Continua

1. Mejorar la Calidad del código.
2. Detección de Errores más rápida y fácil.
3. Reduce Tareas repetitivas y manuales.
4. Puede crear Versiones de prueba en cualquier momento.
5. Completa Visibilidad del proyecto.
6. Mayor Confianza y seguridad del equipo de trabajo.

Figura 7. Principales Beneficios de la Integración Continua

1. **Mejorar la calidad del código**

Cuando se utiliza CI se prevé que los cambios insertados en el código poseen mayor calidad e integridad con el resto del software. El servidor CI facilita las tareas de automatización, despliegue, ejecución de pruebas y análisis de código. Existen varias alternativas en el mercado Open source y de licencia comercial integrados con los diferentes ambientes de desarrollo. Estos servidores ejecutan tareas programadas que garantizan la calidad del código y facilitan la detección de errores.

2. **Detección de errores más rápida y fácil**

Integrar de forma periódica permite encontrar los errores de forma temprana y el tratamiento de los mismos es más simple e insume menos tiempo.

Al aplicar CI se reducen los riesgos (uno de los beneficios más importante para Martin Fowler) y tiempos, ya que son menos los cambios que se incluyen en cada integración por ende se requiere menos tiempo y son menos los errores los que pueden surgir, ya que se detectan en un tiempo temprano evitando que se combinen con errores que pueden surgir al agregar cambios posteriores, generando más tiempo y complicaciones para solucionar los mismos. Por otro lado, al realizar integraciones constantemente, los programadores recuerdan y tienen en mente los cambios que se están integrando y, por ende, los errores que pueden surgir y cómo solventarlos de manera más simple.

Según Martin Fowler: "*Como resultado, los proyectos con integración continua tienden a tener dramáticamente menos errores, tanto en producción como en proceso. Sin embargo, debo enfatizar que el grado de este beneficio está directamente relacionado con lo bueno que es su conjunto de pruebas*".

Llevar a cabo esta práctica no garantiza la inexistencia de errores al momento de integrar, pero si estamos reduciendo la probabilidad de que estos ocurran, ya que la tendencia es integrar código que está funcionando, porque los builds y los tests pasaron exitosamente.

Podemos concluir que para llegar a un nivel bajo de errores implica trabajar y mejorar de manera constante en el conjunto de pruebas a aplicar.

3. **Reduce tareas repetitivas y manuales**

Los procesos de instalación manuales, cuando se realizan de manera repetitiva, son una pérdida de tiempo y constituyen una fuente de equivocaciones. Las personas nos equivocamos, los scripts de instalación no. Además, tener scripts que «saben» como hacer la integración continua (y añadidos posteriores, como despliegues, testeo, etc.) evita dependencias de personas.

Los procesos manuales repetitivos son lentos y susceptibles a cometer diversos errores, por lo que, poder automatizarlos implica un gran beneficio para asegurarnos que los procedimientos se completen de manera correcta.

4. **Puede crear Versiones de prueba en cualquier momento**

Permite generar versiones de software en cualquier momento y de manera transparente, clara y entendible para todos los integrantes del proyecto de desarrollo; principalmente sin errores y generalmente con los últimos cambios realizados. Además, dicha versión será correcta, funcional y completa, siempre y cuando la integración se haya realizado correctamente. Por otra parte, otorga independización de una persona específica que realice la integración para recién poder generar una versión del software.

5. **Completa Visibilidad del proyecto**

Al aplicar continuamente la integración, sabemos frecuentemente el estado del proyecto. No se espera al final para detectar funcionalidades incorrectas o errores de carácter grave, que generarían grandes demoras y complicaciones para solucionarlos.

Analizando cómo se va desarrollando la integración continua, se pueden obtener estadísticas y tendencias del estado del desarrollo y plantear mejoras en el proceso de desarrollo, como así también solucionar problemas que surgen en el mismo.

6. **Mayor Confianza y seguridad del equipo de trabajo**

Según Javier Garzás: "*La experiencia demuestra que un equipo que utiliza integración continua logra una mayor confianza interna, al estar comprobando constantemente el estado del desarrollo. Y no es solamente la confianza en el equipo, es también del entorno (usuarios, gestores, etc.).*

Un equipo de desarrollo de software que se siente seguro es un equipo más motivado y productivo" [2].

CI mejora ampliamente el rendimiento y la interacción entre los participantes, ya que no se requiere llegar al final del proyecto para realizar una integración y verificar fehacientemente lo que cada uno de los integrantes desarrolló; si realmente está sin errores, evita peleas y disputas por los cambios que puedan perjudicar a los integrantes.

Herramientas para Integración Continua

La integración continua (CI) permite a los desarrolladores de software evitar una larga y problemática fase de integración al final de un proyecto. En lugar de compilar todos los componentes al final, con la CI se van implementando todas las novedades directamente en el código base. Esto requiere disciplina y un proceso eficiente, pues de lo contrario la CI obstaculizará más de lo que ayudará. El proceso se puede facilitar además con software específico.

A veces de forma totalmente autónoma y otras veces en combinación con otras aplicaciones, las herramientas de integración continua (CI tools) ayudan en la creación de un repositorio, en la ejecución de las pruebas y en la compilación, así como en el control de versiones y, por supuesto, en la propia integración continua.

En la actualidad, Internet ofrece una gran variedad de herramientas para la integración continua. Todas tienen como objetivo ayudar al desarrollador en la implementación de esta metodología, y lo hacen de diferentes modos y con la ayuda de características distintas. Pero estas herramientas no solo se diferencian unas de otras en cuanto a sus características, sino que también existe una gran variedad en lo que respecta a precios y licencias. Mientras que muchas de ellas son de código abierto y se encuentran disponibles de forma gratuita, otros fabricantes ofrecen herramientas comerciales.

Entre las herramientas que priorizamos para mostrar, se encuentran:

Jenkins, una de las herramientas de integración continua más conocidas del mercado. Este software escrito en Java ha continuado desarrollándose constantemente desde el año 2005 y cuenta en la actualidad con numerosas funciones que asisten no solo en la integración continua, sino también en el despliegue y la entrega continua. Jenkins CI no es más que un sistema desplegado en un servidor que nos ayuda en la tarea de hacer integración continua y programar tareas automáticas cuando ocurra una determinada acción.

Travis CI, un servicio de integración continua que se utiliza para construir y probar software alojado en GitHub, ya que se integra a la perfección y realiza automáticamente los pipelines definidos en cada push o pull request. La herramienta es gratuita en proyectos Open Source, aunque cobra una comisión si se utiliza con repositorios privados. Permite testear y buildear aplicaciones escritas en Ruby, Node, Objective-C, Go, Java, C# y F#, entre otras (siempre que corran en Linux). Permite combinarse con Azure para facilitar los deploys automáticos.

A diferencia de Jenkins no utiliza tareas programadas, no tiene costos de mantenimiento (Ya que está alojado como servicio) y posee un setup más simple.

TeamCity, es otro servidor de gestión de compilación e integración continua de la empresa JetBrains. Permite realizar el proceso de integración continua de manera más simple y automática, facilitando diferentes configuraciones que se adaptan a múltiples proyectos y modalidades de trabajo. Reporta errores de manera clara y completa. Consta de una interfaz amigable y con el apoyo de todo el potencial del IDE de JetBrains: IntelliJ, como por ejemplo el analizador de código y reporte de errores del mismo.

Bamboo, otra herramienta para gestión de integración continua muy utilizada en el mercado, propiedad de la empresa Atlassian. Destaca por su completa integridad con Bitbucket y Jira Software. Permite llevar un completo seguimiento durante los procesos de integración, compilación y despliegue de diversos proyectos de software. Posee múltiples configuraciones que se adaptan a cada proceso de compilación creando diferentes fases y planes de compilación como también la posibilidad de crear diferentes pruebas automatizadas, donde los errores que surgen serán reportados y bien detallados. También destaca por su numerosas herramientas y facilidades para desplegar proyectos en diferentes entornos y de manera simultánea.

CodeShip, es una plataforma de integración continua en la nube basado en tecnología de contenedores. Ejecuta las pruebas automatizadas y la implementación configurada cuando se ingresa al repositorio. Se encarga de administrar y escalar la infraestructura para que se pueda probar y lanzar con mayor frecuencia y obtener comentarios más rápidos para crear el producto que necesitan los usuarios. Tiene integración con GitHub y Bitbucket.

Conclusión

En base a lo investigado y expuesto en el presente, no quedan dudas que la CI es una práctica fundamental que contribuye con el desarrollo ágil de cualquier producto de software. Es indispensable su aplicación, ya que facilita al equipo de desarrollo a integrar sus cambios y resolver posibles conflictos lo antes posible. Integrar periódicamente, encontrar errores de forma temprana y corregirlos en el momento, permite optimizar la forma de trabajo, lo cual contribuye al valor que se le entrega al cliente final. Por esto, consideramos que la CI no es una práctica que beneficia únicamente a los desarrolladores, sino también a todas las personas involucradas en el proceso de desarrollo del software.

La elaboración de pruebas unitarias, que permitan validar porciones pequeñas de código, es una inversión de tiempo necesaria para llevar a cabo la CI. La compilación automática y la ejecución de estas pruebas permiten detectar conflictos en los últimos cambios integrados por el equipo de desarrollo. La calidad con la cual se elaboran las pruebas unitarias es un factor para tener en cuenta, ya que luego facilita las correcciones a aplicar para solucionar los errores que puedan surgir de la ejecución automática de las mismas.

Al igual que la Gestión de Configuración y otras disciplinas de la Ingeniería del Software, la CI es una práctica cuya responsabilidad de llevarla a cabo no recae sobre una única persona, sino más bien, es una actividad colaborativa entre los diferentes miembros del equipo de trabajo. Por otro lado, disponer de un repositorio único y de servidores para realizar compilaciones automáticas no garantiza que realmente se esté implementando la CI. Más aun sabiendo que existen diferentes herramientas gratuitas para su implementación, el verdadero desafío de los equipos de desarrollo en la actualidad es interiorizar esta práctica, generar la costumbre y asumir el compromiso de integrar frecuentemente sus cambios.

Por último, queremos destacar el factor humano y comunicativo que componen a esta práctica. La coordinación y el compromiso requerido para llevarla a cabo contribuye y fortalece la sinergia del equipo de trabajo, siendo además esta práctica la base para implementar la entrega y el despliegue continuo, que en definitiva responden a los principios ágiles planteados al comienzo del informe.

Bibliografía

[1] Martín Fowler, "Continuous Integration", 01 mayo 2006. https://www.martinfowler.com/articles/continuousIntegration.html. Consultados por última vez el 10/10/19.

[2] Javier Garzás, "Pero, realmente… ¿Qué beneficios aporta la integración continua?", 25 septiembre 2014. https://www.javiergarzas.com/2014/09/beneficios-integracion-continua.html Consultados por última vez el 10/10/19.

[3] Continuous Integration: Improving Software Quality and Reducing Risk, 2007.

[4] Sander Rossel, Editorial Packt, "Continuous Integration, Delivery, and Deployment", 2017.

[5] Gisell Chávez, "Integración Continua". https://www.smartnodus.cl/integracion-continua/ Consultados por última vez el 10/10/19.

[6] José Ripla, Universitat de Valéncia, Capgemini, "Testing + Integración Continua". https://www.uv.es/capgeminiuv/documents/Presentacion_Testing_IC_2015_11_05.pdf Consultados por última vez el 10/10/19.

[7] Hernández A., Universidad de las Ciencias Informáticas, "Aplicación del proceso de integración continua en el centro de telemática de la universidad de las ciencias informáticas", junio 2014. https://www.researchgate.net/publication/302010909_APLICACION_DEL_PROCESO_DE_INTEGRACION_CONTINUA_EN_EL_CENTRO_DE_TELEMATICA_DE_LA_UNIVERSIDAD_DE_LAS_CIENCIAS_INFORMATICAS Consultados por última vez el 10/10/19.

[8] Alicia Salamon, Patricio Maller, Alejandra Boggio, Natalia Mira, Sofia Pérez, Francisco Coenda, "La Integración Continua Aplicada en el Desarrollo de Software en el Ámbito Científico – Técnico". https://rdu.iua.edu.ar/bitstream/123456789/264/1/IC%20Aplicada%20en%20el%20Desarrollo%20de%20Software%20C-T.pdf Consultados por última vez el 10/10/19.

[9] Pedro Galván Kondo, "Integración Continua". https://sg.com.mx/revista/54/integraci-n-continua Consultados por última vez el 10/10/19.

[10] J. Camilorada, "Integración continua, asegurando la calidad en el desarrollo de software", 10 junio 2014. https://camilorada.wordpress.com/2014/06/10/integracion-continua-asegurando-la-calidad-en-el-desarrollo-de-software/

[11] Travis CI User Documentation. https://docs.travis-ci.com/ Consultados por última vez el 10/10/19.

[12] Principios de las Metodologías Ágiles, "Manifiesto Ágil". https://agilemanifesto.org/iso/es/principles.html Consultados por última vez el 10/10/19.

[13] Julio Alberto Fernández Guerrero, "Implantación de un Sistema de Integración Continua en una metodología consolidada", Trabajo de Fin de Grado, Universidad de Castilla - La Mancha, Escuela Superior de Informática, Junio 2017. https://ruidera.uclm.es/xmlui/bitstream/handle/10578/15411/TFG_Julio_Alberto_Fernandez_Guerrero.pdf?sequence=1&isAllowed=y Consultados por última vez el 10/10/19.

[14] Ana María García Orozco, "La integración continua y su aporte al aseguramiento de la calidad en el ciclo de vida del desarrollo de software", Proyecto de Grado para optar al título de especialista en Proyectos Informáticos, Universidad Distrital Francisco José de Caldas, Facultad de Ingeniería, Especialización Proyectos Informáticos, Bogotá, D.C., 02, diciembre 2015.

[15] 829-1998 - IEEE Standard for Software Test Documentation https://ieeexplore.ieee.org/document/741968/definitions#definitions Consultados por última vez el 10/10/19.

[16] Bamboo https://www.atlassian.com/es/software/bamboo Consultados por última vez el 10/10/19.

[17] Team City https://www.jetbrains.com/teamcity/documentation/ Consultados por última vez el 10/10/19.

[18] CodeShip https://codeship.com/ Consultados por última vez el 10/10/19.

Glosario

- CI: Continuous Integration / Integración Continua.
- CI/CD: Continuous Integration, Delivery y Deployment / Integración, Entrega y Despliegue Continuo.
- SW: Software.
- CI tools: Herramientas de Integración continua.

Plantilla para autores CONAIISI Formato A4 2Columnas

<<Nombre y apellidos – Autor1>><<Nombre y apellidos – Autor2>>
<<Datos de filiación – Autor 1>><<Datos de filiación – Autor 2>>
<<Email Autor1>><<Email Autor2>>

Nota: Los artículos remitidos para su evaluación estarán sujetos a un proceso de evaluación por pares externos y doble ciego. Por lo tanto, para ser evaluados, no debe contener nombres de autores, filiaciones institucionales o ninguna información que pueda revelar la identidad del autor. *Esta información será solicitada en la versión final, de ser aceptado el artículo*. En ambas instancias de deberá enviar el trabajo en formato .pdf.

Resumen

El texto del Resumen debe estar justificado y en cursiva, ubicado en la parte superior de la columna izquierda debajo de la información del autor tal como se lee en este texto. Utilice la palabra "Resumen" como título, con la fuente "Times" tamaño 12 pts en negrita, centrada en relación con la columna, con solamente la inicial en mayúscula. El texto debe estar con la fuente "Times" de tamaño 10 pts en cursiva, el espacio de interlínea simple y sólo podrá extenderse hasta 7,6 cm de largo. Dejar dos líneas en blanco después del resumen, luego inicie el texto principal.

Introducción

Esta plantilla contiene las pautas para la realización de los artículos que se presentarán en el Congreso, incluye una descripción completa de los tipos de tipografía (fuentes), espacios e información relacionada para la producción del Libro de Actas del Congreso. Se recomienda una extensión de 7 a 10 páginas totales por artículo. Debe respetar las pautas aquí detalladas.

Dándole Formato a su Artículo

Todo material incluido en el artículo, tanto el texto, como las ilustraciones, gráficos y tablas, deben mantenerse dentro del *área de impresión* que tendrá 17,5 cm de ancho por 22,54 cm de alto. No debe escribir ni ubicar nada que esté fuera del área de impresión. El texto deberá estar en formato de dos columnas, a excepción del título y los autores. Las mencionadas columnas deberán medir 8,25 cm de ancho, con un espacio entre ellas de 8 mm.

El texto debe estar justificado en ambos márgenes.

En el sitio del congreso podrá descargar una plantilla Word con las especificaciones de formato <Pautas para autores CONAIISI A4 – Plantilla.dotx>.

Título Principal

El título principal (en la primera página) deberá empezar en el centímetro 3,49 desde el borde superior de la hoja, centrado, y con la fuente "Times" tamaño 14 pts en negrita. Ponga en mayúscula la primera letra de los sustantivos, pronombres, verbos, adjetivos, y adverbios; no colocar en mayúscula artículos, conjunciones coordinantes o preposiciones, y no cumple todo ello cuando el título comience con una palabra de este tipo. Dejar dos líneas en blanco después del título.

Nombre(s) de Autor(es) y filiaciones institucionales

Los nombres de autor y filiaciones institucionales deberán estar centrados debajo del título y escrito en fuente "Times" tamaño 12 pts, estilo normal (sin negritas). Con autores múltiples podrán mostrarse en un formato de doble o triple columnas, con sus filiaciones correspondientes debajo de los respectivos nombres. Las filiaciones institucionales estarán centradas debajo del nombre de cada autor, en cursiva. Debe incluir dirección de correo electrónico. Dejar dos líneas en blanco luego de la información del autor y antes del texto principal.

Segunda Página y Siguientes

A partir de la segunda página y las siguientes deberán empezar a la altura del margen superior, de 3 cm. En todas las páginas el margen inferior será de 3 cm.

Estilo de Letra y Fuentes

Debe utilizar la fuente "Times", y en caso de no contar con la misma, podrá reemplazarla por "Times Roman" o "Times NewRoman". Si ninguna de ellas está disponible en su procesador de texto, utilice la fuente que más se parezca a la tipografía "Times" a la cual tenga acceso. Evite el uso de fuentes bitmap. Se prefiere fuentes True-Type 1.

Texto Principal

Escriba su texto principal en "Times" tamaño 10 pts con espacio simple. No utilice espaciado doble. Todos los párrafos deberán tener una sangría de 5 mm. Asegure que el texto esté completamente justificado—es decir, que ambos márgenes queden alineados. No coloque ninguna línea adicional en blanco entre los párrafos. Podrá utilizar la fuente "Courier" para el código fuente.

```
For I = 1 to 10 do
  Print "this figure"
End;
```

Figura 1. Ejemplo de figura.

Pie de Figuras y Tablas

Las descripciones o referencias de figuras y tablas deberán ser escritas en fuente "Helvética" de tamaño 9 pts o fuente similar como "Sans-Serif", en negrita. Coloque en mayúscula solamente la primera letra de las referencias de figuras y títulos de tablas. Las figuras y tablas deberán numerarse por separado. Por ejemplo: "**Figura 1. Ejemplo de figura.**" y "**Tabla 1. Tabla ejemplo**". Las referencias de cada figura deberán estar por **debajo** de las misma (ver **Figura 1**). Los títulos de cada tabla deberán estar centrados por **sobre** las mismas (ver **Tabla 1**)

Tabla 1. Tabla ejemplo

Uno	Dos	Tres

Títulos de Primer Nivel

Por ejemplo, "**1. Introducción**" deberá estar en fuente "Times" tamaño 12 pts en negrita, la primera inicial en mayúscula, alineado a la izquierda, con espacio anterior de 24 pts y espaciado posterior de 12 pts.

Títulos de Segundo Nivel

Como título de segundo nivel, deberá utilizar la fuente "Times" tamaño 11 pts en negrita, alineado a la izquierda, sin sangría y con un espaciado de 6 pts anterior y 6 pts posterior.

Encabezados de Tercer Nivel

Los encabezados de tercer nivel, en caso de utilizarlo, será con la fuente "Times" tamaño 10 pts en negrita, alineado a la izquierda, con espaciado anterior de 12 pts y 6 pts de espacio posterior.

Notas al Pie de Página

Use notas al pie[1] y su ubicación será al final de la página en la cual están referenciados. Debe utilizar fuente "Times" tamaño 9 pts, con espaciado simple.

Agradecimientos

De corresponder, será el lugar para realizarlo. En caso de que el artículo contenga un respaldado, sea una Institución o un Organismo, indicar según corresponda con el nombre completo y las siglas. El encabezado, en caso de utilizarlo, será con la fuente "Times" tamaño 12 pts en negrita, alineado a la izquierda, con espaciado anterior de 24 pts y 12 pts de espacio posterior.

Referencias

Enumere en una lista todas las referencias bibliográficas con la fuente "Times" tamaño 9 pts, espaciado simple, al final del artículo. Cuando haga referencia en el texto, encierre la cita entre corchetes, por ejemplo [2-4], [2, 5], y [1].

[1] Briand, L. C., Daly, J., and Wüst, J., "A unified framework for coupling measurement in objectoriented systems", IEEE Transactions on Software Engineering, 25, 1, January 1999, pp. 91-121.

[2] Maletic, J. I., Collard, M. L., and Marcus, A., "Source Code Files as Structured Documents", in Proceedings 10th IEEE International Workshop on Program Comprehension (IWPC'02), Paris, France, June 27-29 2002, pp. 289-292.

[3] Marcus, A., Semantic Driven Program Analysis, Kent State University, Kent, OH, USA, Doctoral Thesis, 2003.

[4] Marcus, A. and Maletic, J. I., "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing", in Proceedings 25th IEEE/ACM International Conference on Software Engineering (ICSE'03), Portland, OR, May 3-10 2003, pp. 125-137.

[1]Nota al pie, fuente "Times" tamaño 9 pts.