Universidad de las Ciencias Informáticas
Facultad 1

Título: Propuesta de entorno de integración continua para el desarrollo de software en el Centro de Informatización Universitaria

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Yassef Amed Galarraga Grant
Tutor: Ing. Damián Cervantes Rodón

Ciudad de La Habana, junio 2010

La única manera de hacer grandes trabajos es amar lo que uno hace. Si no lo encontraron todavía, sigan buscando.

Steve Jobs.

Declaración de Autoría

Declaro ser el único autor del presente trabajo titulado:"Propuesta de entorno de integración continua para el desarrollo de software en el Centro de Informatización Universitaria", por tanto, reconozco y autorizo a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____ _____

Yassef Amed Galarraga Grant **Ing. Damián Cervantes Rodón**

Datos del contacto

Damián Cervantes Rondón. Ingeniero en Ciencias Informáticas con 3 años de experiencia, graduado en la Universidad de las Ciencias Informáticas. Residencia UCI: Edif. 29, apto 206. Teléfono: 835-8881. Correo Electrónico: dcervantes@uci.cu

Agradecimientos

A todos aquellos que han contribuido de una forma u otra, a las personas más importantes en mi vida. Los que me han apoyado siempre y me han guiado hasta aquí: mi mamá, mi papá, mi hermana, a mi novia Yenelis y mis abuelos. Gracias por ayudarme a convertirme en la persona que soy, gracias por confiar en mí y darme su amor. Todo se lo debo a ustedes.

Agradezco a todas las personas que de una forma u otra contribuyeron a la realización de este trabajo.

A todos aquellos que conocí durante estos 5 años y se han convertido en amigos para toda la vida.

Y por último me gustaría agradecer sinceramente a mi tutor Ing. Damián Cervantes Rodón, su esfuerzo y dedicación. Sus conocimientos, sus orientaciones, su persistencia, su paciencia y su motivación han sido fundamentales para poder realizar este trabajo.

Dedicatoria

Me gustaría dedicar esta Tesis a mi familia.

Para mis padres y hermana Bárbaro, Ana Julia e Indira, por su comprensión y ayuda en los buenos y malos momentos. Me han enseñado a encarar las adversidades sin perder nunca la dignidad ni desfallecer en el intento. Me han dado todo lo que soy como persona, mis valores, mis principios, mi perseverancia, mi empeño y todo ello con una gran dosis de amor y sin pedir nunca nada a cambio.

A mi hermana Indira gracias por estar ahí conmigo y apoyarme siempre, Te Quiero Mucho.

A Yenelis (mi osita), que te pudiera decir muchas gracias por todos estos años que hemos estado juntos en los cuales hemos compartido tantas cosas, a pesar de que estos últimos 4 años no hemos estado juntos físicamente, ha sido como si lo estuvieras, ya que por muy difícil que fuera el problema que tuviésemos el uno o él otro, siempre éramos el apoyo del otro y entre los 2 buscábamos la solución del problema, esta tesis también te la dedico a ti mi Amor.

Quisiera también dedicarle esta tesis a 2 de los seres más queridos por mí en este mundo y aunque uno de ellos no tiene la oportunidad de estar en esta vida, no es razón para que no le dedique el fruto más grande que he obtenido hasta hoy en la vida, este trabajo es para ustedes Abu Yolanda y abuela Celestina donde quiera que estés, este fruto es para ti.

A todos mis amigos que de verdad los nombraría a todos pero no quisiera que se me olvidara alguno, pero tenga presente que me acuerdo de todos y que no me olvidaré de ninguno ya todos llegaron a mi vida los mejores momentos y hemos compartido muy buenos momentos, los quiero mucho mis amigos.

Y no puedo terminar sin antes decirles, que sin ustedes a mi lado no lo hubiera logrado, tantas desveladas sirvieron de algo y aquí está el fruto. Les agradezco a todos ustedes el haber llegado a mi vida y el compartir momentos agradables y momentos tristes, pero esos momentos son los que nos hacen crecer y valorar a las personas que nos rodean. Los quiero mucho y nunca los olvidaré.

Es la hora de partir, la dura y fría hora que la noche sujeta a todo horario.
(Pablo Neruda)

Resumen

Actualmente en la Universidad de las Ciencias Informáticas, existen proyectos que se ven afectados por las pérdidas de tiempo y de recursos, provocadas por las extensas fases de integración del software. Esto se evidencia también en los proyectos del Centro de Informatización Universitaria.

La tendencia hoy en día, es obtener productos de software en el menor tiempo posible y elaborar la documentación necesaria; por lo que para elevar la productividad en el centro de informatización universitaria y los proyectos que lo componen, se propone una práctica de metodología de procedimientos ágil.

El propósito de la presente investigación consiste en proponer un entorno de Integración Continua, en los proyectos del Centro de Informatización Universitaria, con el objetivo de automatizar los procesos de compilación, construcción, pruebas y despliegue en los proyectos de dicho centro. Para esto fue necesario realizar un estudio del estado del arte de la práctica de Integración Continua en el mundo y de las herramientas que permiten crear un escenario utilizando esta técnica. Se realiza también un estudio del proceso de pruebas que se lleva a cabo en el Centro de Informatización Universitaria, por último, después de realizar todo este estudio se hace una propuesta de solución de acuerdo con la situación problémica que se detectó y de las herramientas a utilizar en conjunto con la propuesta de entorno planteado, además de dejar definido los roles y responsabilidades que están inmersos en dicha propuesta y también se describe el funcionamiento de dicho entorno.

Índice

Introducción

Con el avance tecnológico que existe en la actualidad, imaginar un mundo sin el uso de las tecnologías es un gran reto, a tal medida que las tecnologías se utilizan en todas las aéreas del desarrollo de nuestras vidas.

Esto tiene como consecuencias que el avance de la industria de desarrollo de software, cada día sea más eficiente y se trace metas que alcancen satisfacer todas las necesidades que se presenten, esto exige que se utilicen nuevas prácticas y metodologías que aumentan la calidad de los productos de software que estos proveen. Ahora además de exigir la excelente calidad del producto final, también se pide que durante el proceso de desarrollo del producto se trabaje con calidad por parte de todos los integrantes del equipo de desarrollo.

En nuestro país desde los inicios en el campo de la producción de software, se ha tenido como principal matriz la calidad del producto y del proceso de desarrollo. Esas son las bases sobre las que se trabajan en las diferentes entidades que se dedican a la producción de software, una de ellas es la Universidad de las Ciencias Informáticas (UCI).

La UCI, primera universidad de la Batalla de Ideas, a la cual nuestro Comandante en Jefe llamó a que se convirtiera en una “Universidad de excelencia”, desarrolla un modelo de formación especial, distinta de las demás universidades y centros de estudio, ya que este modelo está orientado a que los estudiantes lleven al unísono su formación como profesionales y la producción.

El desarrollo de software se realiza directamente por estudiantes y profesores que dedican sus horas extras docentes a la producción, esto trae consigo que se deban impartir cursos de capacitación para orientar al personal de trabajo de acuerdo con los requisitos del software a producir. En este punto también son importantes las acciones que se toman en función de mejorar la calidad del producto y el proceso de desarrollo.

Para controlar que todo este proceso de desarrollo tenga la calidad requerida, la UCI creó el centro de calidad, entidad responsable de rectorar las acciones que con respecto al proceso de calidad de software que se desarrolla en la universidad.

De las principales tareas que desarrolla esta dirección, está la de tratar de imponer a los equipos de desarrollo, que usen un mismo procedimiento de calidad que cumpla con los

lineamientos de calidad establecidos en la UCI. Pese a esto se ha podido comprobar que no todos los proyectos cumplen con este procedimiento o trabajan de manera precipitada para garantizar la entrega de las tareas cuando se acerca una revisión o auditoria.

Esto se debe a la poca experiencia por parte del personal que interviene en el desarrollo de software, la mala o poca capacitación que se da en los proyectos con respecto a estos temas y la falta de comprensión en cuanto a la necesidad real de garantizar la calidad durante el proceso de desarrollo y el producto final.

Esto trae como resultado, que los errores que se insertan durante el desarrollo, frecuentemente sean detectados cuando se le realizan las pruebas por parte del laboratorio industrial de calidad, perteneciente a Calisoft, que es el centro de calidad. Esto por supuesto trae consigo que ejecuten con mayor esfuerzo las revisiones de calidad y las posteriores etapas de corrección de errores. Muchos proyectos dedican varias iteraciones a la revisión y correcciones, incluso ha ocurrido que después de terminado y entregado el producto al cliente, muchos han detectado errores en las funcionalidades de los mismos.

Todos estos esfuerzos que se dedican a las exhaustivas revisiones y las varias iteraciones de corrección de errores, implican atrasos en los cronogramas de trabajo de los equipos de desarrollo.

La documentación que se elabora para efectuar las pruebas, contienen muchísimas deficiencias, esta es una de las causas por lo cual hay que realizar varias iteraciones de revisión. Esto es precisamente porque los equipos de desarrollo, no tienen en cuenta que parte de su trabajo es mantener un proceso de desarrollo con calidad desde su inicio hasta la culminación del mismo, garantizando de este modo mejores resultados.

Después de la situación problémica antes expuesta, queda planteado el siguiente **problema a resolver:**

¿Cómo lograr una mejora en el proceso de desarrollo de software en el Centro de Informatización Universitaria que propicie el aumento de la productividad? Teniendo como **objeto de estudio** el proceso de desarrollo de software en el Centro de Informatización Universitaria y como **campo de acción** los procedimientos de calidad que se llevan a cabo en los proyectos del Centro de Informatización Universitaria.

Para desarrollar este trabajo se planteó como **objetivo general,** desarrollar una propuesta de entorno de integración continua para el desarrollo de software en el Centro de Informatización Universitaria.

Partiendo de esto quedaron trazados los **objetivos específicos:**

- Realizar un estudio conceptual sobre la integración continua.
- Proponer un entorno de integración continua para el Centro de Informatización Universitaria.
- Validar la propuesta de entorno.

Después de aplicar todo este estudio se esperan como **posibles resultados:**

- Una propuesta de entorno de integración continua para el Centro de Informatización Universitaria.
- Documentación sobre la integración continua para su posterior estudio.

Un tema importante que interfiere en el resultado de los proyectos productivos en el Centro de Informatización Universitaria está relacionado con la necesidad de desarrollar productos de mayor calidad con un proceso de desarrollo que propicie este objetivo, sin afectar los cronogramas de ejecución y de entrega. De tal manera se plantea **la hipótesis** de que si se implantaran las prácticas de integración continua en los equipos de desarrollo de software en el Centro de Informatización Universitaria, se logrará una disminución del tiempo dedicado a las pruebas de calidad y a la corrección de errores.

Proponer un modelo de integración continua adaptado a las particularidades del proceso de desarrollo de software en el Centro de Informatización Universitaria, propiciará una reducción del tiempo de desarrollo de los proyectos, que se detecten los errores que inmediatamente sean introducidos, puede disminuir considerablemente el tiempo de ejecución de las pruebas que hace el laboratorio industrial de calidad, mantener al equipo de desarrollo informado del estado del sistema en desarrollo y tener disponibilidad de la última versión del código para realizar pruebas de despliegue o demostraciones a los clientes.

Los **métodos de investigación científica** utilizados para llevar a cabo el trabajo planteado son:

Métodos Teóricos:

Analítico - Sintético: posibilita el entendimiento de la utilización de la integración continua, partiendo de los materiales analizados se extrae la información más importante relacionada con el tema que se está desarrollando.

Análisis histórico lógico: permite estudiar de forma analítica la trayectoria histórica real de los fenómenos su evolución y desarrollo.

Métodos Empíricos:

Entrevista: esto constituye un medio significante para la obtención de información por parte de expertos en el tema que se está tratando.

Estructura Capitular:

Capítulo 1 - Fundamentación Teórica: en este capítulo se hace un estudio del estado del arte de la Integración Continua, sus avances y utilización en el mundo, en Cuba y en la universidad. Además, se hace un estudio de algunas empresas que utilizan la integración continua como parte de sus modelos de desarrollo.

Capítulo 2: en este capítulo se realiza un estudio analizando cómo se trabaja en función de garantizar la calidad de los productos de software durante el proceso de desarrollo de software en el Centro de Informatización Universitaria, así como se expone la propuesta de modelo de integración continua para el Centro de Informatización Universitaria.

Capítulo 3: se plantea la validación de la propuesta realizada y se especifican las valoraciones finales sobre la misma.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

Integración Continua (en inglés *Continuous Integration*) es una práctica de metodologías ágiles, especialmente de XP, que pone énfasis en el hecho de tener un proceso de construcción y prueba completamente automático, que permita al equipo modificar, compilar y probar un proyecto varias veces en un mismo día.

Todos los integrantes del equipo deben integrar su código con la última versión estable por lo menos una vez al día.

Martín Fowler (Fowler, 2002) afirma que el desarrollo de un proceso disciplinado y automatizado es esencial para un proyecto controlado, el equipo de desarrollo está más preparado para modificar el código cuando sea necesario, debido a la confianza en la identificación y corrección de los errores de integración. Con el proceso de Integración Continua la mayor parte de los errores de un sistema se manifiestan en el mismo día en el que se integran los módulos intervinientes, reduciendo drásticamente los tiempos para determinar el problema y resolverlo. Cada vez que se realiza la integración, los que la hacen deben asegurarse que todas las pruebas se ejecutan correctamente, para que el nuevo código sea incorporado definitivamente.

Hacer compilaciones regularmente permite tener la posibilidad de obtener retroalimentación continua y actualizaciones del producto en desarrollo. Tal monitoreo permite controlar y seguir el progreso del proyecto, y anticipar errores eventuales y/o discrepancias; esto proporciona indicadores de la evolución real del código.

1.2 Integración Continua

Con el surgimiento de las metodologías ágiles, llegó la integración continua, que no es más que una práctica de desarrollo de software donde los miembros del equipo integran su trabajo frecuentemente, usualmente cada persona integra a diario, lo que produce varias integraciones diarias. Cada integración es verificada en una construcción automática (incluyendo pruebas) para detectar errores en la integración tan rápido como sea posible. Muchos equipos encuentran que este enfoque conduce a una reducción significativa de problemas de integración y permite al equipo desarrollar un software coherente más rápidamente. (Fowler, 2006)

El término de integración continua (CI de sus siglas en inglés) fue creado por Martin Fowler en un patrón de diseño. CI se refiere a las prácticas y herramientas que aseguran la compilación y las pruebas de manera automática de una aplicación en intervalos frecuentes, usualmente en un servidor de integración específicamente configurado para esta tarea. La convergencia entre las prácticas de las pruebas de unidad y las herramientas, conjuntamente con las herramientas de compilación automáticas de la CI, se hace una necesidad para cualquier proyecto en la actualidad. (Bartlett, 2009)

La idea principal es que los desarrolladores trabajando en un proyecto puedan integrar los cambios en sus códigos al menos diariamente. Con un potente conjunto de pruebas y un sistema que permita asegurar estas pruebas de forma continua y automática, se puede asegurar que cualquier problema introducido en el sistema puede ser identificado en la próxima compilación. (Sam-Bodden, 2006)

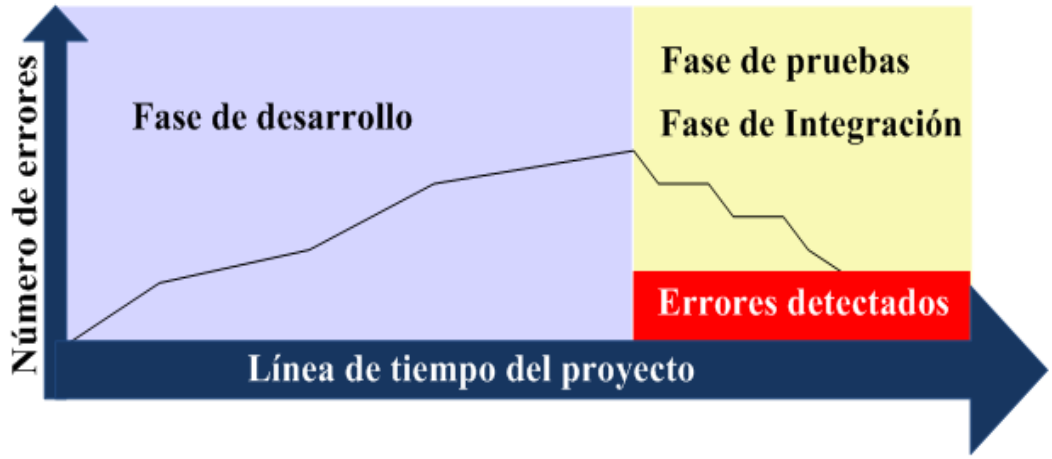


Figura 1. 1 Proceso de desarrollo sin Integración Continua.

A medida que aumenta el grado de independencia entre los desarrolladores, menos compatible se vuelven los diferentes módulos que ellos desarrollan y por tanto aumenta el número de problemas a la hora de integrarlos. Ejemplo de esto son los grandes proyectos de software libre (proyecto GNU) cuyos programadores están dispersos por todo el mundo y que por lo general su interacción solo se resume al correo electrónico, pero que colaboran entre sí

para desarrollar proyectos de gran envergadura. En este tipo de proyectos sería engorroso dejar la integración para el final ya que el tiempo requerido para la búsqueda y erradicación de errores se haría extenso e imposible de predecir. En la figura 1.1 es posible apreciar como se comporta la cantidad de errores a lo largo del tiempo de desarrollo en los proyectos.

Para un buen desempeño de la Integración Continua, primeramente se necesita contar con un equipo disciplinado ya que en muchos proyectos no se trabaja en unión hasta que llega el momento de la integración final. La Integración Continua facilita que el desarrollo de los proyectos sea incremental, y brinda funcionalidades que permiten automatizar la ejecución de pruebas unitarias, esto permite que a medida que se va desarrollando el proyecto, se puedan añadir nuevas funcionalidades sin temor a que las demás dejen de funcionar.

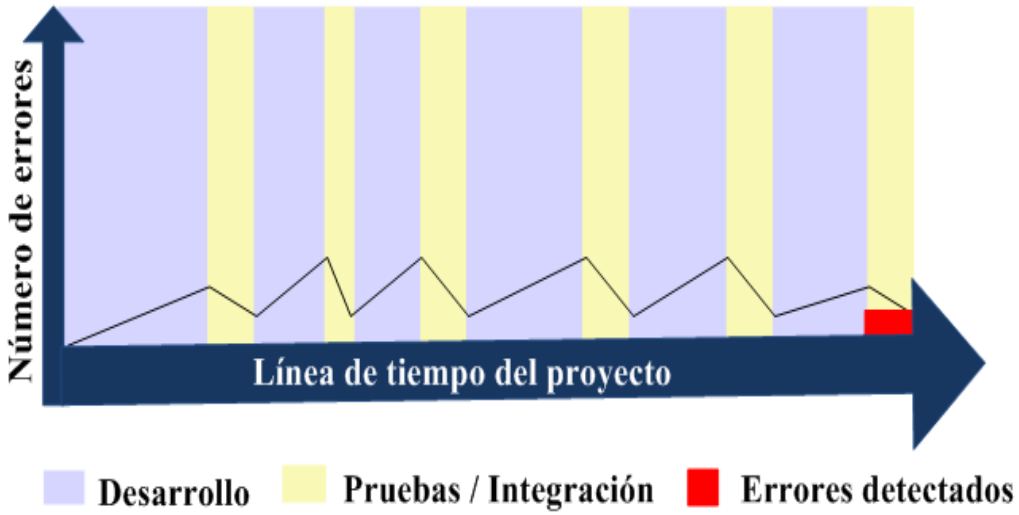


Figura 1. 2 Proceso de desarrollo con Integración Continua.

La Integración Continua es integrar un proyecto de una forma incremental y frecuente, permitiendo así encontrar problemas de integración en etapas tempranas y resolverlos rápidamente, reduciendo drásticamente las fases de integración y prueba (Figura 1.2). El proceso de integrar no es nuevo, constantemente los desarrolladores realizan compilaciones a lo largo del desarrollo del proyecto. Desde mucho antes del surgimiento del concepto de CI se consideraba como una buena práctica la realización de pruebas automáticas, aunque es válido especificar que no gozan de mucha aceptación entre los programadores ya que suelen ser consideradas como un trabajo innecesario. CI puede ser aplicado a cualquier proyecto,

incluso en aquellos que son desarrollados por un único trabajador, pero su importancia se hace más perceptible a medida que aumenta la complejidad y el número de desarrolladores.

La Integración Continua es una práctica que se aplica en todo tipo de entornos de desarrollo de software en la cual para automatizar el proceso de construcción, pruebas, despliegue y para que se realice diariamente es necesario:

- Escoger un repositorio para almacenar las fuentes, donde se lleve el historial de todos los cambios del código y de donde sea posible obtener la última versión del proyecto.
- Automatizar el proceso de construcción de manera que a partir de las fuentes se pueda construir con un único comando todo el sistema.
- Realizar las pruebas de forma automática y de esta manera saber si todo está correcto o si existe algún problema.

¿Qué impide a los equipos de desarrollo el uso de Integración Continua? Demasiados cambios: Algunas personas pueden sentir que son numerosos los procesos que necesitan cambios para poder introducir la Integración Continua al proyecto. Es más efectivo una aproximación incremental, por ejemplo: se puede comenzar con la construcción de la aplicación y pruebas con una frecuencia baja (ejemplo una vez al día), luego incrementar la frecuencia poco a poco mientras el equipo de desarrollo se sienta cómodo con los resultados.

Demasiados fallos en la construcción de la aplicación: Muchas personas creen que el proceso de compilación es solo compilar, pero no es así, ya que consiste entre otras cosas en la recopilación, las pruebas, inspección y despliegue. La compilación o construcción es el proceso que compila los códigos fuentes y verifica que el software funcione correctamente. Generalmente ocurren compilaciones fallidas cuando los desarrolladores no realizan compilaciones privadas antes de subir el código al sistema de control de versiones, lo que provoca una rápida respuesta del sistema CI.

Incrementa los gastos al mantener un sistema de CI: Es una percepción errónea porque independientemente de que se tenga implementado un sistema CI o no, se tiene la necesidad dentro de un proyecto de integrar, realizar pruebas e inspecciones. Administrar un sistema CI es mejor que mantener un proceso manual.

Costo adicional en software y en hardware: Efectivamente, integración continua requiere una máquina independiente ya que a menudo la integración es muy costosa, en lo que a tiempo de ejecución se refiere, pero resulta más costoso encontrar problemas en etapas tardías de un proyecto.

1.3 Modelos de Integración Continua

Esta práctica consiste en llevar al límite el proceso de construcción automática y diaria. Si se logra automatizar este proceso, el modelo de integración continua debe estar bien diseñado, porque más que ventajas crea grandes problemas. En este momento no existen definidos modelos específicos como recetas para ser implementados en el desarrollo de software, pero si hay referencia de los más conocidos y/o los que más se utilizan de acuerdo con sus resultados en dicho proceso.

En el caso de una ejecución de todo el procedimiento de integración continua al concluir cada día de trabajo. Seguramente los desarrolladores no han terminado la tarea en la que trabaja y por ende, el código está inestable, y por tanto, un resultado esperado de la ejecución de las pruebas automáticas sería una gran lista de errores. Esta situación se puede ver representada en la siguiente figura, donde cada flecha es un cambio hecho en el repositorio, y como se puede apreciar, algunos cambios desestabilizan el producto introduciendo errores de compilación.

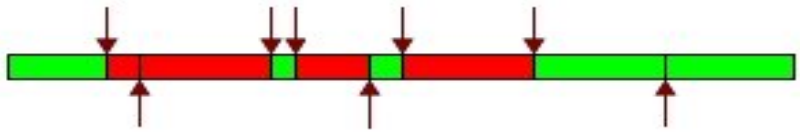


Figura 1. 3 Modelo de desarrollo en una sola rama

Para conseguir que la integración continua tenga su efecto beneficioso, se debe organizar el desarrollo en distintas ramas.

Cada desarrollador tiene su propia rama de código, donde trabaja a diario, hace sus cambios, registra código, etc. Esa rama es privada, y podrá hacer lo que quiera con ella y mantenerla

en cualquier estado. Lo habitual es que esa rama esté muy inestable al comenzar con una nueva tarea y vaya ganando estabilidad conforme avance en el desarrollo.

En el momento en que el desarrollador termina con su trabajo, se compila una versión de esa rama y se pasa a pruebas. El departamento de pruebas validará la funcionalidad sobre la que ha estado trabajando el desarrollador, se reportarán los bugs oportunos, y cuando se dé por correcta, se etiquetará esa rama y se integrará en el repositorio principal. En ese momento se lanza una construcción del producto de la rama principal, del producto ya integrado, para validar que la integración ha sido correcta. En este caso, se obtiene más funcionalidad, reduciendo al máximo los tiempos de inestabilidad de la rama principal. Una vez que la rama principal se da por buena, se etiqueta y se continúa con la siguiente funcionalidad. La siguiente ilustración muestra que en la rama principal tiene muchas menos zonas en rojo:

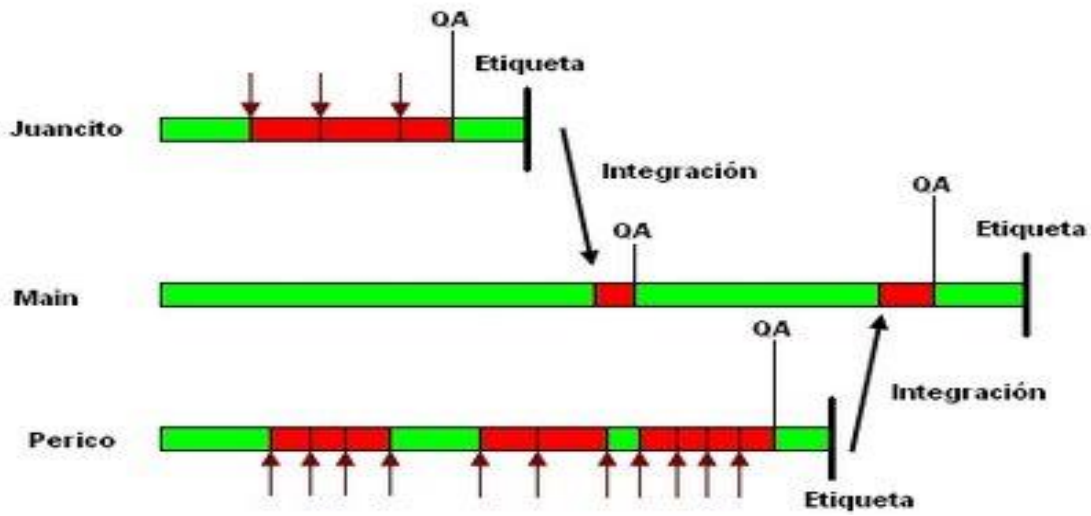


Figura 1. 4 Modelo de desarrollo con varias ramas

Existe todavía otro modelo, muy parecido al anterior, y que consiste en tener una rama de desarrollo para cada nueva funcionalidad. El esquema es el mismo que se muestra arriba, donde los desarrolladores que trabajan en una misma funcionalidad lo hacen sobre una rama separada. La diferencia es que cuando la funcionalidad se da por terminada, se integra en la rama principal y esa rama queda muerta, teniendo que crear otra nueva para la siguiente funcionalidad.

En todos los casos, se recomienda aplicar el proceso de integración continua, solo al código principal, que es donde se insertan los cambios definitivos del producto, que son los más estables. Porque no es de mucho resultado aplicar la integración continua a una rama de código secundaria que no está lista para ser revisada.

Tener en cuenta también, que sin importar el modelo que se aplique, debe ser observado durante su aplicación, porque uno de los objetivos de la integración continua es detectar los errores lo más pronto posible, por esto los cambios al código principal no deben hacerse en intervalos largos de tiempo, se recomienda que se hagan diarios.

1.4 Ventajas y desventajas de la Integración Continua

La Integración Continua constituye un cambio fundamental en las distintas organizaciones en cuanto a la forma de desarrollar software, ya que introduce nuevas reglas dentro del equipo de desarrollo, permitiendo un incremento de la calidad del mismo. Esta práctica no es muy fácil de aplicar pero su implementación trae consigo numerosas ventajas:

- Disminuye la búsqueda de fallos a la hora de integrar el código, fallos que realmente son más difíciles de encontrar pues en muchas ocasiones suelen ser el resultado de un código desarrollado de manera independiente por diferentes desarrolladores.
- Permite identificar fallos en el entorno de producción en etapas tempranas, evitando largos períodos de integración finales.
- Disminuye el tiempo de retroalimentación de errores con el cliente como resultado de la detección temprana de los errores.
- Aumenta la confianza de los desarrolladores al subir el código al control de versiones. El repositorio deja de ser un elemento estático dentro del proyecto, para convertirse en parte de un sistema dinámico. Esto permite al programador tener la seguridad de que el código que descarga del repositorio está correcto ya que ha pasado por una serie de pruebas de validación.
- Constante disponibilidad de las compilaciones realizadas, permitiendo el acceso a cada uno de estos así como sus correspondientes reportes que son almacenados en el servidor de integración. Este permite almacenar cada uno de los proyectos que fueron

integrados correctamente y ponen a disposición de los desarrolladores, cada una de estas compilaciones enumeradas por fecha.

La implantación de la Integración Continua también trae consigo algunas desventajas que pueden relacionarse de la siguiente manera:

- **Sobrecarga por el mantenimiento del sistema:** Si el servidor de Integración Continua es detenido durante un tiempo considerable se acumula código en el control de versiones el cual podría tener errores que no serian detectados durante este período de tiempo. Además de que al comenzar nuevamente el servidor de integración podría sobrecargarse por existir un gran número de nuevos cambios.

- **Necesidad potencial de un servidor dedicado a compilar:** Existen varias razones por las cuales un servidor de integración necesita correr en un sistema dedicado, una es que las pruebas unitarias pueden llegar a ser muy costosas en tiempo de ejecución por lo tanto esto llevaría a conflicto con otros servicios que se ejecuten en el mismo. Otra de las razones es que desde el mismo momento en que se aplica la Integración Continua, este se vuelve el proceso más importante dentro del ciclo de desarrollo ya que todos los actores dependen de su correcto funcionamiento.

- **El impacto inmediato al subir código erróneo provoca que los desarrolladores no suban su código frecuentemente como sería conveniente como copia de seguridad:** Un fallo en el servidor de integración provoca un paro en el desarrollo y un retraso en el proyecto

- **Implica introducir una nueva filosofía de desarrollo:** Para aplicar Integración Continua dentro de un proyecto es necesaria la participación activa de cada uno de sus miembros, es decir, primero se debe preparar a los desarrolladores y concientizarlos de la necesidad de llevar buenas prácticas de desarrollo lo cual a veces puede resultar complicado en grandes grupos de desarrolladores.

1.5 Gestión de Configuración

La gestión de configuración del software es uno de los procesos clave para toda organización dedicada al proceso de desarrollo, ya que posibilita una mejor organización del desarrollo y mantenimiento del producto, facilitando el resto del proceso de producción.

Durante el proceso de construcción de un software, los cambios son inevitables debido a las modificaciones de requisitos y los fallos. En los proyectos de desarrollo se trabaja en equipo por lo que es preciso llevar un control y registro de los cambios con el fin de reducir los errores, aumentar la calidad y la productividad, para así evitar los problemas que puede acarrear una incorrecta sincronización en dichos cambios, al afectar a otros elementos del sistema o a las tareas realizadas por otros miembros del equipo de proyecto.

La gestión de la configuración del software es el medio para conocer, en todo momento, qué componentes y versiones, tanto de un producto como de sus elementos, son las correctas. Es el proceso de identificar y definir los elementos de la configuración para controlar la liberación y los cambios durante todo el ciclo de vida, registrar e informar de su estado y de las peticiones de cambio, además de verificar la corrección y acabado de los elementos. (Cuevas, 2002)

El objetivo de la gestión de la configuración es mantener la integridad de los productos que se obtienen a lo largo del desarrollo de los sistemas de información, garantizando que no se realizan cambios incontrolados y que todos los participantes en el desarrollo del sistema disponen de la versión adecuada de los productos que manejan. Así, entre los elementos de configuración software, se encuentran no únicamente ejecutables y código fuente, sino también los modelos de datos, modelos de procesos, especificaciones de requisitos, pruebas, etc.

La gestión de configuración se realiza durante todas las actividades asociadas al desarrollo del sistema, y continúa registrando los cambios hasta que éste se deja de utilizar. La gestión de configuración facilita el mantenimiento del sistema, aportando información precisa para valorar el impacto de los cambios solicitados y reduciendo el tiempo de implementación de un cambio, tanto evolutivo como correctivo. Asimismo, permite controlar el sistema como producto global a lo largo de su desarrollo, obtener informes sobre el estado de desarrollo en que se encuentra y reducir el número de errores de adaptación del sistema, lo que se traduce en un aumento de calidad del producto, de la satisfacción del cliente y, en consecuencia, de mejora de la organización.

Algunos beneficios de la implementación del proceso de gestión de configuración para la organización son la reducción de riesgos, mejora de la calidad y beneficios de coste en la

entrega y soporte de productos, asegurar la correcta configuración del software, proporcionar la capacidad de controlar los cambios, reducir los sobreesfuerzos causados por los problemas de integridad y garantizar que todo el equipo trabaja sobre una misma línea base de productos.

Si no se realiza una buena gestión de configuración puede incurrir en que no se disponga de un inventario completo de los componentes del sistema cuando necesitemos, que haya que realizar re-trabajo durante las pruebas porque los componentes ya probados no sean los que debieran, o que no se pueda recuperar una línea base anterior para realizar mantenimiento. Todo ello conlleva una pérdida de dinero y recursos.

1.5.1 Control de versiones

Para poder implantar una herramienta de integración continua la primera condición es tener un repositorio con todas las fuentes del proyecto. Utilizar un controlador de versiones es una práctica que se hace especialmente imprescindible cuando se trabaja en entornos colaborativos y en los que se desarrolla de una forma evolutiva, para que el software pueda estar disponible desde las primeras versiones. Estas dos razones han hecho de estos sistemas el corazón de los proyectos.

Los sistemas de control de versiones se basan en mantener todos los archivos del proyecto en un lugar centralizado, normalmente, un único servidor, aunque también hay sistemas distribuidos, donde los desarrolladores se conectan y descargan una copia local del proyecto. Con ella, envían periódicamente los cambios que realizan al servidor y van actualizando su directorio de trabajo que otros usuarios a su vez han ido modificando.

Una revisión sería cada una de las versiones disponibles en el repositorio. Este término se utiliza refiriéndose tanto a ficheros individuales como a conjuntos de ficheros o incluso al repositorio entero. Suelen identificarse mediante un número.

Las ramas permiten que a partir de cierto punto, un mismo fichero o conjunto de ficheros, sea desarrollado de dos formas diferentes, manteniendo sus respectivas revisiones independientemente.

En un proceso de integración pueden producirse conflictos, en el caso de que dos o más clientes hayan realizado cambios de manera independiente sobre un mismo documento. El

sistema puede ser capaz de manejar algunos de estos conflictos, según el caso, pero en general, el proceso de resolución (conciliación) suele requerir la intervención del usuario. (García, 2008)

Al proceso de actualización o escritura de los ficheros bajo control de versiones, se le denomina integración. Esta palabra se utiliza tanto para indicar la escritura en el repositorio de los cambios realizados en la copia de trabajo, como a la actualización de la copia local desde el repositorio para incluir los cambios realizados por otros clientes.

1.6 Compilación automática

Hacer que el código fuente se convierta en un sistema funcionando puede ser un proceso complicado que involucre compilar, mover archivos, cargar esquemas en las bases de datos y más. Pero, como la mayoría de las tareas en el desarrollo de software, puede ser automatizada y, como consecuencia de esto, debe ser automatizada. Pedirles a personas que escriban comandos o pinchen en ventanas de diálogo es una pérdida de tiempo y un campo fértil para errores. (Fowler, 2006)

Un proceso de construcción de software que normalmente contiene diversos pasos manuales y repetitivos. Esto lleva consigo un proceso largo de generación de software y que el mismo sea susceptible de errores humanos. Por esto es preciso garantizar que la compilación siempre sea realizada de la misma forma y conteniendo todos los pasos establecidos inicialmente. La construcción de software a través de un ambiente de desarrollo puede generar diferencias de productos debido a las configuraciones individuales de cada desarrollador, pero una compilación completa y limpia puede demorar mucho tiempo para realizarse.

Para reducir el riesgo de que un desarrollador pierda mucho tiempo para realizar una compilación por cada alteración del código fuente, es posible adoptar diferentes tipos de compilaciones automatizadas. Es posible crear un *script* que realice una compilación incremental, esto es, que se compila y reconstruye códigos nuevos o alterados. De este modo es posible ejecutar una compilación completa apenas cuando es considerado necesario por el paso del tiempo.

La construcción de un software es repetitiva y debe ser hecha frecuentemente en un proyecto. La realización manual de compilación puede generar errores de re trabajo.

La automatización reduce el tiempo que los desarrolladores gastan en tareas repetitivas y aumenta el tiempo de respuesta de todo el equipo de desarrollo. También facilita la integración de todos los códigos, componentes y otras herramientas utilizadas en un proyecto.

Como resultado se obtiene que el proyecto posea un *script* automatizado de compilación, con la aplicación de este patrón. Este *script* permite repetir la construcción de software con un único comando. Esto da confianza al equipo de que siempre podrá realizar la construcción de software de manera efectiva.

1.7 Desarrollo guiado por pruebas sistemáticas

La mayor de las prácticas en la ejecución de las pruebas de calidad a un software, es realizar todo el trabajo luego de implementado el sistema, cuando ya los desarrolladores indican que han terminado. Sin embargo, bajo un desarrollo basado en pruebas, se escriben primero todos los casos de pruebas y luego se implementa el código que permita que las pruebas diseñadas se ejecuten con resultado exitoso.

Esta técnica de desarrollo permite, entre sus principales características, que los casos de prueba se conviertan prácticamente en una especificación de los requerimientos del sistema, por cuanto indican que es lo que el sistema debe hacer para cumplir con las funcionalidades requeridas por el cliente.

Lo que hace tan efectivo el desarrollo basado en pruebas es la automatización de las pruebas y el hecho de que las herramientas para implementar estas técnicas son gratis y totalmente funcionales. Tener un único repositorio para ejecutar las pruebas facilita la ejecución de esta práctica de programación.

En el desarrollo basado en pruebas, se utilizan dos técnicas fundamentales: la refactorización y escribir primero las pruebas. La primera implica que una vez se termine de implementar y que el sistema consiga pasar satisfactoriamente todas las pruebas unitarias, entonces se lleva a cabo la refactorización del código. La segunda más bien lo que le exige al desarrollador que antes de comenzar la implementación de su fragmento o unidad, diseñe y escriba todos los casos de prueba que debe satisfacer las necesidades de dicha unidad.

Para que el desarrollo basado en pruebas tenga resultados satisfactorios se deben tener definidos lo más claros posibles los requisitos del sistema a desarrollar. Después se pone en

marcha un ciclo de desarrollo donde se comienza por seleccionar un requerimiento, revisando siempre que sea un requerimiento que este correctamente especificado y no tenga ambigüedades, así como fácilmente programable.

Teniendo esto se prosigue a elaborar una prueba para el requerimiento, exigiendo siempre que el programador piense como el cliente a partir de la interfaz. Se revisa entonces que la prueba falla, pasándole una entrada que propicie el error. Se escribe el código más sencillo posible en función de que pase la prueba. Se ejecutan las pruebas y se refactoriza el código, eliminando código duplicado. Se ejecutan nuevamente las pruebas hasta estar todo perfecto, luego se actualiza la lista de requerimientos. Este ciclo de desarrollo se ejecuta una y otra vez hasta agotar todos los requerimientos.

Las limitaciones que hoy en día posee esta técnica están relacionadas a la necesidad de que las pruebas puedan ser automatizadas, tema que se vuelve muy complejo en el caso de las interfaces gráficas de usuario y los objetos distribuidos.

1.8 Pruebas automatizadas y sus diferentes tipos

En las pruebas de software es posible aplicar diferentes procesos y metodologías. La mayoría usan como estrategia aplicar pruebas diariamente, sin importar la parte del sistema que se construya o llevar todo este mismo proceso de forma automatizada construyendo los casos antes de comenzar a escribir el código del producto y decidiendo en ese momento que se probará y con qué profundidad.

En integración continua, las pruebas se ejecutan de manera regular con el objetivo de probar que la evolución del software desarrollado funciona perfectamente, tanto para las funcionalidades nuevas como para aquellas que ya existían. Una de las claves es que las pruebas sean automatizadas, porque si se ejecutaran manualmente, el coste de hacerlo “continuamente” sería incosteable.

Para poder ir probando durante el desarrollo, es necesario entonces escribir las pruebas antes que el código. Estas pruebas conforman un banco de pruebas que luego el servidor de integración continua ejecuta automáticamente al software, pasándole datos de entrada en espera de un resultado esperado. (Somerville, 2005)

Esto tiene como ventajas que el equipo de desarrollo se ve en la obligación de definir claramente y antes de comenzar a escribir código de la aplicación, que cualquier duda que exista sobre las funcionalidades a desarrollar en el sistema. Esto asegura que no exista en el proceso de desarrollo un concepto muy presente a menudo en la industria de software retraso de prueba, donde todo el equipo de desarrollo espera por las versiones que libera un equipo de revisores, aspecto que en la actualidad ha ocasionado que cada día se salten mas las pruebas de calidad al software en construcción, en busca de poder ajustarse a los cronogramas de trabajo.

Ahora se debe prever porque existe la tendencia por parte de los desarrolladores a preferir la implementación del producto por sobre la elaboración de pruebas, lo que provoca que en ocasiones se escriban pruebas incompletas que no comprueben todas las posibilidades de entradas, específicamente las excepcionales. Algunas pruebas son más difíciles que otras de escribir, lo que puede dar como resultado que se queden partes cruciales del sistema sin probar.

A continuación se muestran los tipos de pruebas:

- **Pruebas unitarias**: se encargan de probar una clase en concreto, probando cada uno de sus métodos y viendo si dados unos parámetros de entrada, la salida es la esperada.
- **Pruebas funcionales**: como su propio nombre indican, prueban una funcionalidad completa, donde pueden estar implicadas una o varias clases, la propia interfaz de usuario y, en el caso del desarrollo web, llamadas AJAX.
- **Pruebas de regresión**: son aquellas pruebas cuyo objetivo es comprobar por qué ha dejado de funcionar algo que ya funcionaba. El objetivo de las pruebas de regresión es no tener que “volver atrás”.
- **Pruebas de aceptación**: son pruebas funcionales, pero vistas directamente desde el cliente. Son aquellas pruebas que demuestran al cliente que la funcionalidad está terminada y funciona correctamente.
- **Pruebas de integración**: conjunto de pruebas unitarias, funcionales, de regresión y/o de aceptación que se realizan tras probar el software. Incluye también comprobar que

lo programado por los diferentes desarrollados no “choca” entre sí y que funcionará en un entorno real.

1.8.1 Pruebas unitarias

Las pruebas unitarias tienen como objetivo verificar la funcionalidad y estructura de cada componente individualmente una vez que ha sido codificado. Las pruebas iníciales constituyen un sistema, y todas las demás pruebas deben apoyarse sobre ellas. Existen dos enfoques principales para el diseño de los casos de pruebas: el enfoque estructural o de caja blanca y el enfoque funcional o de caja negra. (Vivas White & López Romero, 2009)

El concepto de las pruebas unitarias, o *unit testing* es sencillo. Si se tiene que hacer un programa, e ir comprobando según se avanza haciendo la funcionalidad del programa se corresponde con lo que en un principio se definió hacer, el comportamiento habitual de la mayoría de programadores es o bien ir escribiendo texto por la salida estándar como “llego hasta aquí”, “el valor de esta variable es…”, etc. código que hay que escribir, comprobar, y luego eliminar. Otra posibilidad sería escribir todo el código confiando en nuestra concentración y luego usar algún tipo de depurador si el código no hace lo esperado. En el primer caso, se pierde mucho tiempo. En el segundo quizá la pérdida de tiempo sería menor pero, probablemente se pasan por alto casos de prueba que podrían ser propensos a error.

Por todo lo anterior y para ganar en tiempo y corrección, actualmente la mejor solución es la realización de pruebas unitarias. Además, estas pruebas servirán, no solo para probar el código al programarlo por primera vez, sino para comprobar que al realizar un cambio en una unidad del proyecto, el comportamiento de otras unidades dependientes de esta se mantiene y no aparecen efectos colaterales inesperados. Por esto, las pruebas unitarias son tan importantes en la integración continua. Sin ellas, los cambios realizados en intervalos pequeños y regulares, introducirían probablemente errores difíciles de detectar.

1.8.2 Pruebas funcionales

El objetivo de la prueba funcional es validar cuando el comportamiento observado del software probado cumple o no con sus especificaciones. La prueba funcional toma el punto de vista del usuario. (Beizer, 1990)

Las funciones son probadas ingresando las entradas y examinando las salidas. La estructura interna del programa raramente es considerada. Para realizar pruebas funcionales, la especificación se analiza para derivar los casos de prueba.

La prueba exhaustiva del producto requiere ejercitar todos los caminos posibles del mismo y el tiempo requerido para esto, incluso en los pequeños programas, se vuelve muy costoso. Por tanto, se debe propiciar que se maximice la producción de las pruebas, esto se traduce en maximizar el número de errores encontrados con la aplicación de un determinado número de casos de prueba.

En la etapa de diseño de las pruebas funcionales, la especificación se analiza para derivar los casos de prueba y en la etapa de ejecución es donde se ejecutan los casos de prueba diseñados previamente, se compara el resultado real con el esperado y se reportan los resultados.

1.8.3 Pruebas de regresión

Como norma general, cualquier aplicación debe probarse a fondo. Lamentablemente, en la práctica, las pruebas son insuficientes y a menudo pasadas por alto. Las pruebas de regresión intentan verificar que los cambios realizados no han introducido nuevos defectos y que el resto de la aplicación sigue funcionando correctamente.

Cualquier modificación sobre software ya probado, sea porque se corrige un defecto encontrado, porque se implementa una mejora o una adaptación, puede introducir nuevos defectos. Por tanto, es preciso volver a ejecutar sobre el código modificado los casos de prueba que fueron diseñados y ejecutados anteriormente. A este tipo de actividad se le denomina prueba de regresión. (Cosín, 2007)

Para realizar correctamente las pruebas de regresión, hay que comenzar analizando nuestras necesidades y definiendo una estrategia. Es importante que nuestra estrategia defina los pasos a seguir a la hora de seleccionar las pruebas que se van a ejecutar. Se podría pensar que ejecutando todos, pero en muchas ocasiones esto no es posible por lo ajustado de las planificaciones. Es por ello que se debe hacer una selección para conformar una batería de pruebas.

Para una buena selección se debe realizar un análisis de impacto, es necesario identificar las áreas que se han visto impactadas por los cambios realizados en el código. Identificar los flujos principales de nuestro producto y seleccionar un porcentaje de todas las pruebas que se tienen.

Si existen muchos casos de pruebas a los que hay que regresar ya que nuestra aplicación es de un tamaño considerable, una buena practicar es la automatización. Por esto es importante identificar los flujos principales de nuestra aplicación, pues bien, estos serán nuestros casos de prueba candidatos a ser automatizados ya que al ser flujos principales no deberían de sufrir grandes cambios en las siguientes liberaciones, por eso estaría bien automatizarlos.

1.8.4 Pruebas de aceptación

Una prueba de aceptación puede ir desde un informal caso de prueba hasta la ejecución sistemática de una serie de pruebas bien planificadas. De hecho, las pruebas de aceptación pueden tener lugar a lo largo de semanas o meses, descubriendo así errores latentes o escondidos que pueden ir degradando el funcionamiento del sistema. Estas pruebas son muy importantes, ya que definen el paso a nuevas fases del proyecto.

La planificación detallada de estas pruebas debe haberse realizado en etapas tempranas del desarrollo de proyecto, con el objetivo de utilizar sus resultados como indicador de su validez, si se ejecutan las pruebas programadas tal como el cliente espera en función de sus expectativas, el producto se puede considerar correcto, y por tanto, adecuado para su puesta en producción. (Cosín, 2007)

La validación del sistema se consigue a través de las pruebas de caja negra que demuestran la conformidad con los requisitos y que se recogen en el plan de pruebas, el cual define las verificaciones a realizar y los casos de prueba asociados. Dicho plan está elaborado para asegurar que se cumplen todos los requisitos funcionales especificados por el usuario y teniendo en cuanta también los requerimientos no funcionales relacionados con el rendimiento, seguridad de acceso al sistema, a los datos y procesos, así como a los distintos recursos del sistema.

La formalidad de estas pruebas dependerá en mayor o menor medida de cada organización y vendrá dada fundamentalmente por la criticidad del sistema, el número de usuarios implicados en las mismas y el tiempo del que se disponga para llevarlas a cabo.

1.8.5 Pruebas de integración

La prueba de integración es una técnica sistemática para construir la arquitectura del software, mientras, al mismo tiempo, se aplican las pruebas para descubrir errores asociados con la interfaz.

El objetivo es tomar componentes a los que se aplicó una prueba de unidad y construir una estructura de programa que determine el diseño. A menudo, se tiende a intentar una integración que no sea incremental, se combinan todos los componentes por anticipado, se prueba todo el programa en conjunto.

Las pruebas de integración parten de los componentes individuales previamente probados y tienen como objetivo descubrir errores que se pueden producir en la interacción entre los módulos. En teoría, la combinación de componentes válidos debería dar como resultado un software en el que no se detectan errores, pero en la práctica hay múltiples ocasiones en las que las pruebas de unidad no detectan errores que si se desvelan al ejecutar pruebas sobre el software integrado. (Cosín, 2007)

En cuanto a la estrategia a seguir en las pruebas de integración, existen dos enfoques: descendente y ascendente. En la primera se comienza la prueba por el componente principal, sustituyendo por resguardos los componentes que son llamados por este. Sobre dicho proceso de realiza la primera batería de pruebas. Luego se integran los componentes subordinados al principal y se continúa aplicando pruebas hasta tener integrado todo el sistema.

Por su parte, otros autores referencian que estas pruebas tienen como objetivo verificar el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente con el fin de comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes. (Vivas White & López Romero, 2009)

En la integración ascendente las pruebas se hacen en sentido inverso, comenzando por los componentes que no necesitan llamar a ningún otro y terminando con el componente principal.

1.9 Herramientas utilizadas para implantar Integración continua

La integración continua dentro de todo su proceso tiene una serie de disciplinas en las cuales se apoya para realizar todo este proceso automatizado. Cada una de estas disciplinas tiene consigo herramientas que hacen posible automatizar todo este proceso, para nombrar estas herramientas se recorrerá cada una de estas disciplinas de la integración continua, que son:

Controlador de Versiones (CV): En el proceso de Integración Continua es preciso tener un control de las diferentes versiones del código para de esta forma poder recuperar cualquier versión del proyecto, así como contar con un solo repositorio.

Pruebas Automáticas: Estas constituyen una parte importante del proceso de integración. Incluirlas a un proyecto y programar desarrollando pruebas es uno de los cambios de filosofía necesaria para la implantación de CI.

Construcción y despliegue automatizados: Es necesario realizar las tareas de construcción y despliegue automatizados en un entorno semejante al final.

1.9.1 Control de versiones

Los sistemas de control de versiones utilizan para su funcionamiento un repositorio central que es el lugar donde se almacenan los datos y la información.

Existen tres operaciones básicas en un control de versiones:

- Bajar la versión completa del código.
- Actualizar cambios.
- Subir cambios.

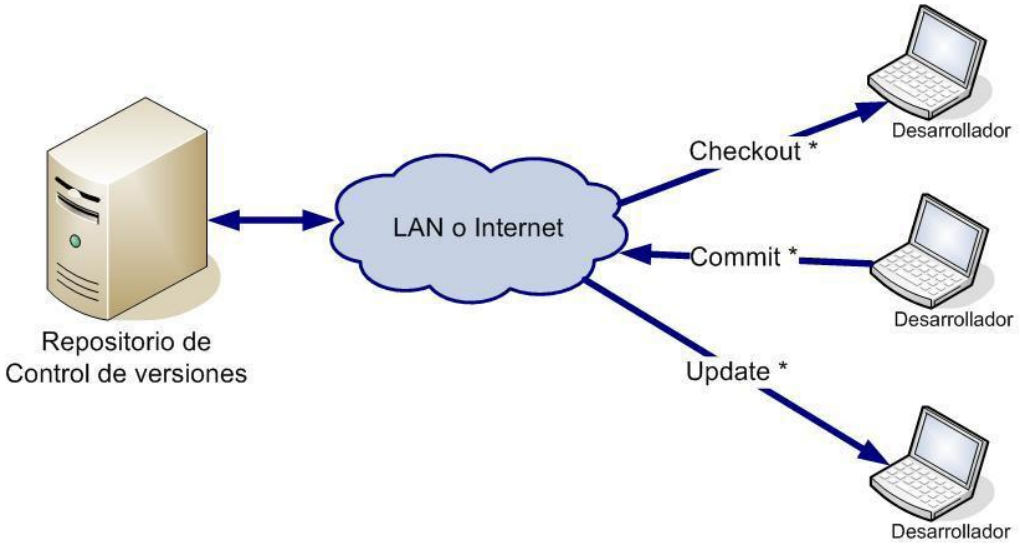


Figura 1. 5 Esquema de un sistema de control de versiones.

El control de versiones es una práctica importante para cualquier grupo de desarrollo de software. Existen varias herramientas para el control del código fuente pero sin importar el que se utilice, al menos deben cumplir con algunas de las siguientes características básicas:

- Proporcionar un lugar para almacenar el código fuente.
- Proveer un historial de lo que se ha hecho a lo largo del tiempo.
- Permitir el trabajo en paralelo de los desarrolladores, uniendo los esfuerzos más tarde.
- Proveer una manera de que los desarrolladores trabajen juntos sin interponerse en el camino de otro.

Un requisito fundamental para que sea posible la implantación de la Integración Continua es que todos los programadores trabajen simultáneamente sobre la misma base del código y esto solo es posible mediante el uso de herramientas de control de versiones, estas se analizan a continuación.

Sistema de Versiones Concurrentes (CVS)

Características:

- Utiliza una arquitectura cliente-servidor.

- Es un sistema de control de versiones bajo una licencia de tipo GNU/GPL, (Licencia Pública General de GNU)
- Permite a un grupo de desarrolladores trabajar y modificar concurrentemente ficheros organizados en proyectos, lo que significa que dos o más personas pueden transformar en un mismo fichero sin que se pierdan los trabajos de ninguna.
- Guarda las antiguas versiones de los ficheros, permitiendo de esta forma recuperar en cualquier momento versiones anteriores a la actual.
- Trabaja con código fuente de programas o con toda clase de documentos siempre que su formato sea completamente de texto, como pueden ser ficheros sgml/html/xml.

Subversion (SVN)

Subversión se creó con el objetivo de mejorar la funcionalidad de CVS, preservando su filosofía de desarrollo.

Características:

- Mantiene versiones no sólo de archivos, sino también de directorios.
- Es un sistema de control de versiones libre bajo una licencia de tipo Apache/BSD (Distribución de Software Berkeley) y se le conoce también como SVN por ser ese el nombre de la herramienta de línea de comandos.
- Es un sistema general que se puede utilizar para administrar cualquier conjunto de ficheros.
- Los archivos versionados no tienen cada uno un número de revisión independiente.
- Mejor uso del ancho de banda, ya que en las transacciones se transmiten sólo las diferencias y no los archivos completos.
- Mayor eficiencia en la creación de ramas y etiquetas que en CVS.
- Realiza el seguimiento de las versiones de proyectos completos.

- Realiza el seguimiento del código modular (un archivo que se reutiliza, o se comparte, en varios proyectos).

Microsoft Visual SourceSafe (VSS)

Microsoft Visual SourceSafe conocida como VSS es una herramienta de control de versiones a nivel de archivos que forma parte de Microsoft Visual Studio.

Características:

- Permite que varias organizaciones trabajen en distintas versiones del proyecto al mismo tiempo y es muy importante en el proceso de desarrollo de software ya que se usa para mantener versiones de código paralelas.
- Ayuda al equipo a evitar la pérdida por accidente de archivos.
- Es un sistema de control de versiones bajo una licencia de tipo Propietaria.
- Permite realizar un seguimiento de las versiones anteriores de un archivo.
- Admite la bifurcación, el uso compartido, la combinación y la administración de versiones de archivos.

1.9.2 Pruebas Automáticas

Cuanto más rápido los errores salgan a la luz, mucho más sencillo será encontrar su origen y erradicarlos. De hecho, se puede decir que el verdadero poder de la Integración Continua está en la realización de pruebas automatizadas, su diseño, elaboración y despliegue es una tarea de gran importancia. Solo sería necesario citar que el tiempo invertido por un programador detectando y corrigiendo errores supera al que utiliza en cualquier otra actividad. Algunos tipos de pruebas, prácticas, y herramientas que los desarrolladores pueden utilizar son:

Automatización de pruebas unitarias: Las pruebas unitarias son una forma de probar el correcto funcionamiento de un módulo de código, mediante el uso de marcos de trabajo como NUnit, JUnit o PHPUnit.

Automatización de pruebas de componentes: Automatizar las pruebas de componente es mucho más complicado que las pruebas unitarias, generalmente estas envuelven más objetos

y toman más tiempo ejecutarlas. Es posible usar herramientas como JUnit, NUnit, DbUnit, y NDbUnit en caso de usar bases de datos.

Automatización de pruebas de sistema: Las pruebas de sistema toman mucho más tiempo que las anteriores ya que envuelven varios componentes. Estas tienen como objetivo verificar que el comportamiento externo del software, satisface los requisitos establecidos por los clientes y futuros usuarios del mismo.

Automatización de pruebas de funcionalidad: Se realizan desde una perspectiva del usuario. Pueden usarse herramientas como Selenium (para Web) y Abbot (para GUI).

Crear categorías de pruebas: Para facilitar las pruebas es posible agruparlas por categorías de acuerdo con complejidad o tiempo de ejecución.

Ejecutar las pruebas más rápidas primero: Ejecutar las pruebas que requieren menos tiempo de ejecución.

Sin las pruebas automatizadas es muy difícil para los diseñadores u otros equipos de trabajo tener confianza en los cambios que se realizan en el software. La mayoría de los diseñadores que utilizan un sistema de integración continua utilizan herramientas para realizar pruebas unitarias como JUnit, NUnit y otros marcos de trabajo del grupo xUnit. También se puede ejecutar categorías diferentes de pruebas en un proceso de integración continua para acelerar sus compilaciones. A continuación se describen dos herramientas para la realización de las pruebas automatizadas, estas son:

JUnit

- Marco de trabajo para pruebas unitarias creado por Erich Gamma y Kent Beck.
- Herramienta de código abierto que se ha convertido en el estándar para las pruebas unitarias en Java y que es soportado por la mayoría de los Entorno de Desarrollo Integrado (IDEs), como Eclipse o Net-Beans.
- Tiene Licencia Pública Común (CPL), que permite usarlo de forma gratuita para cualquier aplicación, sea esta comercial o de código abierto.

- Consiste en un conjunto de clases, fácil de usar y aprender que les permite a los programadores escribir sus pruebas unitarias en el lenguaje Java.
- Posee una comunidad mucho mayor que el resto de los marcos de trabajo de pruebas en Java.

PhpUnit

- PhpUnit nace del lado de conceptos de metodologías ágiles, programación extrema, etc.; sin embargo, esto no significa que deba utilizarse con metodologías ágiles, entre sus características se encuentra:
- Marco de trabajo para las pruebas unitarias en específico a PHP.
- Se encuentra bajo licencia BSD.
- Forma parte del grupo de marcos de trabajo de xUnit
- Almacena los resultados en una base de datos de pruebas.
- Se integra con varias aplicaciones de pruebas.
- Facilita la creación de pequeños *scripts* que ayudan a probar las aplicaciones y analizar los resultados.

1.9.3 Construcción y despliegue automatizados

Automatizar un proceso puede ser una tarea de mucho esfuerzo, se deben automatizar todas las tareas que se realizan y de esta forma conseguir construir el sistema a partir del código que se ha desarrollado para lograr que sea desplegado y esté disponible para poder ser utilizado. Para automatizar todos estos procesos existen algunas herramientas las cuales permiten llevar a cabo la mayoría de las tareas que se ejecutan de forma manual, estas son:

Ant

- Ant es una herramienta gratuita de construcción de dominio público utilizada en la compilación y creación de programas Java.
- Se basa en la ejecución de tareas que se modelan en un fichero el XML.

- Es una de las herramientas J2EE de construcción más usada.
- Es multiplataforma, ya que está escrito en Java. Licencia Apache 2.0.
- Utiliza ficheros de construcción escritos en XML.
- Puede extenderse fácilmente creando nuevas tareas.

Maven

- Maven es una herramienta para la gestión y comprensión de proyectos Java.
- Es capaz de encargarse de la gestión de dependencias, construcción de los artefactos, generación de la documentación, etc.
- Su configuración es más simple, reutilizable y consistente que la de Ant.
- Una vez instalado y configurado, no es necesario mantenerlo.
- Es rápido el acceso a los paquetes ya descargados.
- Licencia Apache 2.0.
- Independencia de la conexión a internet, excepto cuando se necesita algún paquete nuevo.

1.9.4 Mecanismo de retroalimentación

Una de las pautas esenciales que propone la Integración continua es la implementación de un mecanismo de retroalimentación robusto el cual permita la solución de errores de una manera rápida como un punto crítico en el desarrollo del software. La pregunta que debe hacerse todo programador es si es aconsejable seguir programando para saber si se tiene o no un error. En la mayoría de los casos es aconsejable resolver el problema antes de continuar, porque puede darse el caso de tener que reprogramar todo de nuevo hasta ese punto y a veces es inevitable que un error conlleve a otro o que simplemente sea cambiado algo que no tenía que cambiarse. Es por esta razón que la Integración Continua aconseja el uso de medios que alerten lo más pronto posible de la existencia de fallos.

Muchos servidores de CI proveen facilidades para la retroalimentación haciendo uso del correo electrónico, para enviar mensajes acerca del estado del proyecto, incluso mensajes

SMS, a teléfonos celulares donde el responsable es informado en cualquier momento que se está dando un problema y así poder responder con medidas de contingencia previamente establecidas por el equipo de desarrollo.

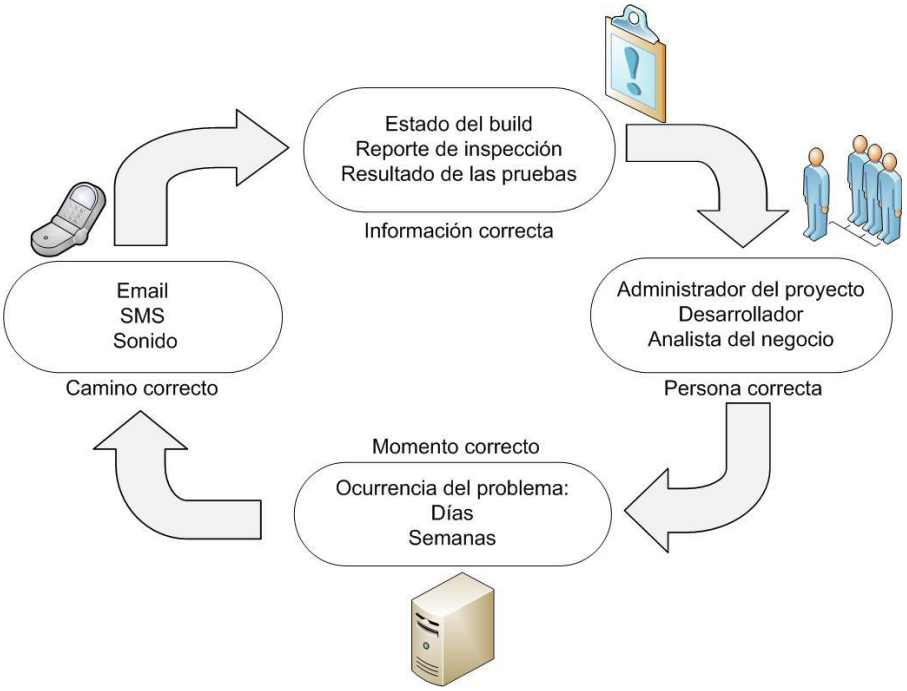


Figura 1. 6 Proceso de retroalimentación continua.

1.9.5 Servidores de integración

La implementación de un entorno de Integración Continua no es una tarea fácil ya que requiere de una gran complejidad técnica. Automatizar el proceso de construcción, pruebas y despliegue sin apoyarse en una herramienta resulta bastante costoso. Existen diferentes herramientas tanto libres como propietarias que permiten la automatización de estas tareas repetitivas, la selección de una de ellas debe basarse según las particularidades de cada equipo de desarrollo. A continuación se muestran un conjunto de servidores de integración y se mencionan algunas de sus características principales.

Cruise Control.

- Cruise Control es una herramienta de código abierto distribuido bajo licencia BSD.
- Es una herramienta que surge en un principio para realizar la Integración Continua a proyectos en Java.

- Basa su funcionamiento en plugins, que se le añaden a un núcleo básico, permitiendo así que los usuarios avanzados puedan extender su funcionalidad.
- Es compatible con algunas herramientas como son, Apache Ant, Maven y NAnt.
- Ofrece soporte para una gran variedad de sistemas de control de versiones incluyendo CVS, SVN y Visual Source Safe (VSS).
- Soporta medios de almacenamiento como por ejemplo servidores Protocolo de Transferencia de Archivos (FTP) o Web.
- Presenta un variado número de plugins que le permiten interactuar con una amplia variedad de mecanismos de retroalimentación, entre los cuales se encuentran el correo electrónico y algunos protocolos de mensajería instantánea como el Jabber.
- Fácil proceso de instalación que no requiere un alto nivel de conocimientos por parte del usuario.
- Configuración a partir de Lenguaje de Marcas Extensibles (XML), en el que se definen, entre otras cosas, las tareas a realizar en el ciclo de construcción para cada uno de los proyectos registrados.

Apache Continuum.

- Continuum es un servidor de Integración Continua para la construcción de proyectos en Java.
- Producto de código abierto y distribuido bajo licencia Apache 2.
- Para la construcción de sus proyectos brinda soporte para el uso de Ant, Mavent 1 y 2 y Shell Scripts como herramientas de automatización.
- Brinda soporte para los sistemas de control de versiones: CVS, Subversion, Clearcase, Perforce,
- En su versión 1.0.3 brinda soporte para el envío de correos electrónicos así como a cuatro mecanismos de notificación, tres de ellos son protocolos de mensajería.

Luntbuild.

- Distribuido bajo licencia Apache 2.0, que permite definir y planificar las distintas tareas a realizar de integración de los proyectos de software; existe también una versión comercial de este proyecto llamada QuickBuild.
- Se integra con diversas utilidades de construcción de software, Ant, Maven1 y Maven2.
- Brinda soporte a los sistemas de control de versiones: CVS, Subversion, ClearCase, y Perforce. Tiene varios métodos para notificar los resultados de la compilación, entre los que se destacan: el envío de correos electrónicos y el uso del protocolo de mensajería de Jabber, etc.
- Toda la configuración, manejo y administración del sistema se hace mediante una interfaz Web, que es ejecutada por el servidor HTTP que viene embebido en el sistema, aunque brinda la posibilidad de utilizar algún otro servidor.

PhpUnderControl.

- Es distribuido bajo las normas de la licencia BSD lo cual supone una desventaja ya que una herramienta que se distribuye bajo la misma permite que la compañía o entidad tenga la libertad de cambiar la licencia a otra más restrictiva cuando lo estime conveniente. O sea, no existe seguridad de que el software sea siempre de código abierto sino que puede la licencia ser cambiada a software privativo. Otras características significativas de PhpUnderControl son:
- Facilidad de uso e instalación por parte del usuario.
- Es configurado a través de archivos de configuración de XML.
- Para el uso de las pruebas unitarias es compatible con PHPUnit.
- Genera automáticamente la documentación a partir de código PHP haciendo uso de PhpDocumentor.
- Realiza las tareas de compilación de forma automática haciendo uso de Ant que está desarrollado en Java y además es extensible a través de este lenguaje.

- Posibilita a través de la herramienta PHP_CodeSniffer implementar reglas de codificación, las cuales tienen gran importancia a la hora de establecer estándares de programación entre los miembros de un equipo de desarrollo.
- Utiliza el correo electrónico y el formato de datos RSS como mecanismo de retroalimentación además usa X10, Jabber entre otros.
- Es compatible con Windows, Unix y cualquier plataforma que soporte el marco de Java.
- PhpUndercontrol es un plugins de CruiseControl por tanto necesita de un entorno Java y una instalación funcionando de CruiseControl.
- Para su mantenimiento es necesario poseer conocimientos de Java.

1.10 Conclusiones parciales

En el presente capitulo se hace un estudio de la integración continua, partiendo de la definición conceptual, se destaca así mismo un grupo de temas de interés para la investigación como: ventajas y desventajas del uso de la Integración Continua, modelos de integración continua existente, la realización del proceso de compilación automática y los diferentes tipos de pruebas que se aplican durante el proceso. Estos temas permiten adquirir los conocimientos necesarios, para realizar posteriormente la propuesta de solución de acuerdo con los objetivos del trabajo.

Capitulo 2: Propuesta de entorno de integración continua para aumentar la productividad en el Centro de Informatización Universitaria.

2.1 Introducción

La calidad es una de las aristas fundamentales en el proceso de desarrollo de software, ya que por muy bueno que sea el equipo de trabajo, si no desarrolla el producto con los parámetros y estándares de calidad requerida, estaría ante un mal uso de los recursos puestos a disposición del desarrollo y construcción del producto pedido por el cliente. Estos resultados y otros tantos serían a gran escala, porque si se analiza a plazos cortos se daría al traste con que habría que dedicar largas jornadas de trabajo e incluso repetidas iteraciones, para la solución de los errores que se arrastran durante todo el desarrollo y en la medida en que se vayan desarrollando las nuevas funcionalidades. Estos errores pueden ser encontrados y solucionados de una forma más fácil y menos costosa para el equipo de desarrollo, aplicando técnicas desarrollo que le dan gran importancia a aplicar la calidad con todo rigor desde el inicio hasta el fin del proceso de desarrollo. Una de estas técnicas es la Integración Continua.

2.2 Estudio del proceso de pruebas en los proyectos del Centro de Informatización Universitaria.

El centro de informatización universitaria de acuerdo con la metodología que utiliza para el desarrollo en sus proyectos, lleva a cabo el proceso de pruebas que dicta la metodología SXP. Esto puede variar y hacerse un poco más flexible de acuerdo con las necesidades particulares de cada proyecto dentro del centro, sin violar lo que plantea la metodología SXP que es la que se utiliza en dicho centro.

Gracias a la metodología XP y la Integración Continua una de las prácticas de esta metodología se hace más fácil el proceso de pruebas. Esto es porque la Integración Continua plantea que las pruebas que se realizaran al código del producto son predefinidas y se confeccionan de acuerdo con las necesidades del producto o las funcionalidades a desarrollar. Esto es lo que plantea las pruebas unitarias que desempeñan un papel fundamental en este proceso de pruebas, ya que estas pruebas dan cierta seguridad de que el código de la funcionalidad escrita por el desarrollador está casi lista para integrar. También se aplican pruebas de regresión las cuales permiten rectificar aquellos errores que aparecen en funcionalidades que estaban funcionando antes de actualizar el código principal y una vez que

se introducen nuevas funcionalidades, pues se crean conflictos, en pocas palabras estas pruebas dan la ventaja de no tener que regresar atrás, si aparece un error en el código que ya se encontraba estable. Otro paso dentro de este proceso son las pruebas de aceptación que tienen un igual peso, pero estas pruebas más bien son para mostrar al cliente el avance del producto, para esto el desarrollador diseña una pequeña interfaz para probar las funcionalidades liberadas y el cliente vea el estado del producto.

Todo este proceso tiene un rol principal el desarrollador, ya que tiene un alto conocimiento del código y es quien además construye las pruebas automatizadas que dirán si el mismo está correcto o si debe rectificar algo. Además de esto existe un sitio de la dirección de calidad, donde se tiene un grupo de planillas y documentos que durante este proceso, que pueden algunos ser llenados de manera opcional y los que estrictamente se completan, es para tener la confirmación de que se está aplicando la calidad según la estrategia planteada en el centro. En la siguiente figura se representa el entorno actual de trabajo en el centro de informatización.

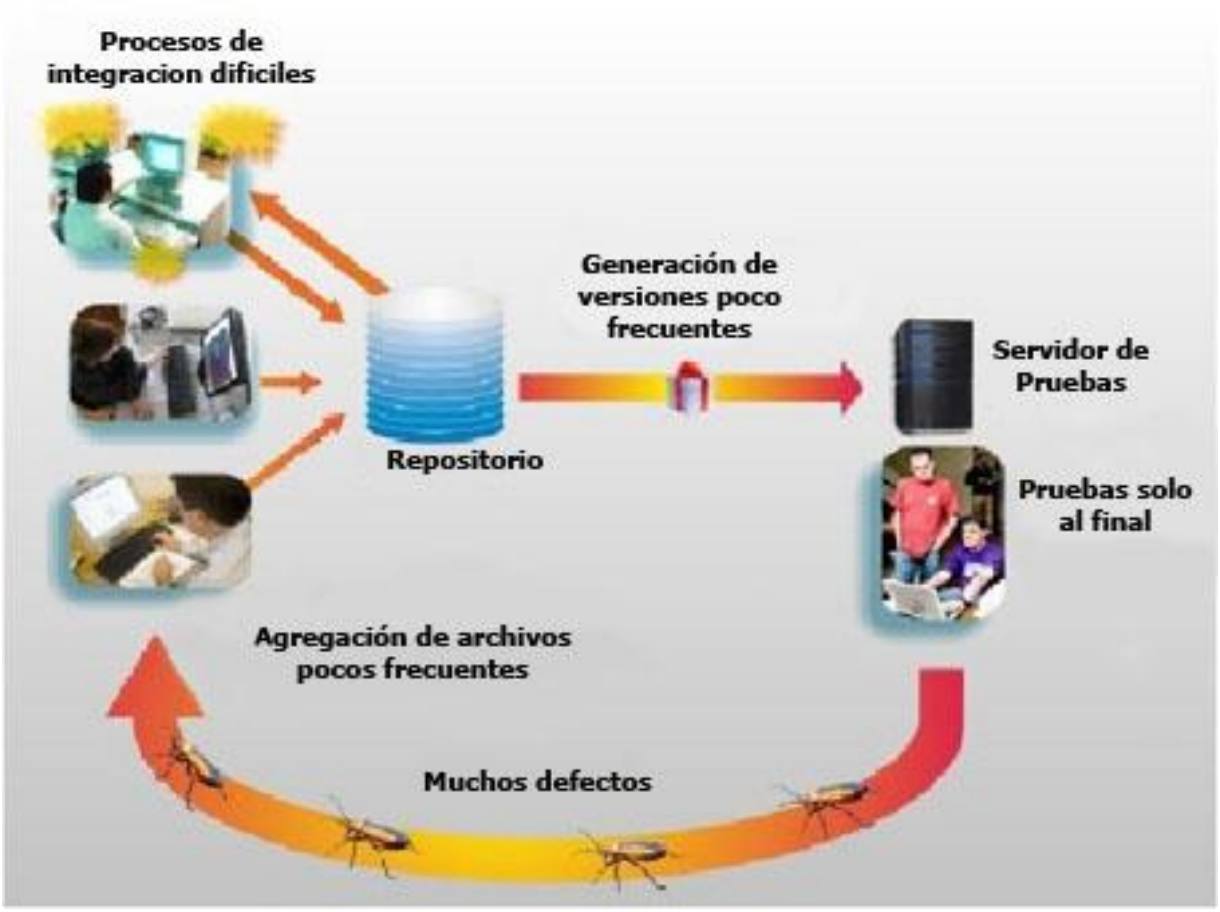


Figura 2. 1 Entorno de trabajo actual del Centro de Informatización Universitaria.

2.3 Propuesta de solución

El modo más fácil para explicar que es Integración Continua, es mostrar un ejemplo de cómo trabaja y a continuación se muestra un ejemplo de como trabajaría un proyecto productivo utilizando dicha técnica. En la siguiente figura se muestra un esquema de cómo sería el desarrollo de software en el Centro de Informatización Universitaria usando Integración continua.

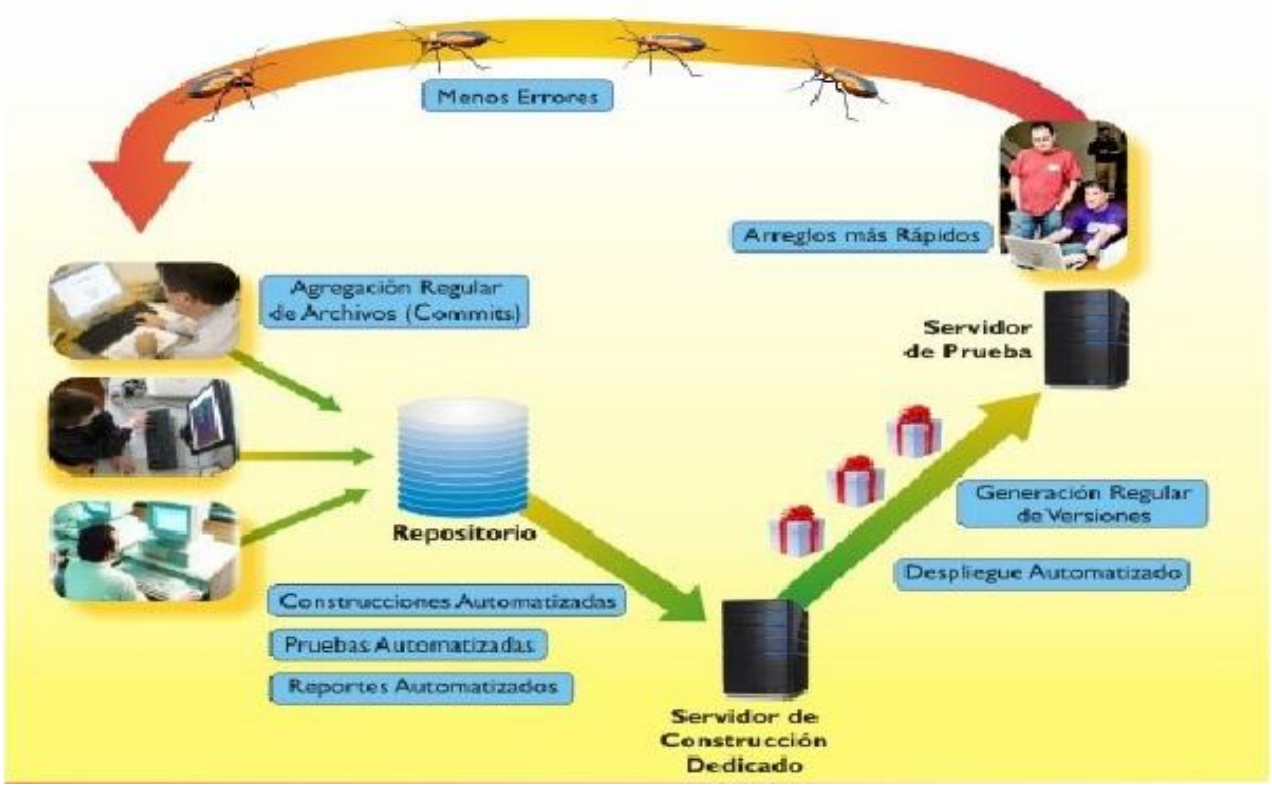


Figura 2. 2 Ejemplo de cómo funcionaría el Centro de Informatización Universitaria usando Integración Continua.

Se toma por ejemplo que se tiene que hacer una funcionalidad de un software. Se descarga una copia del código actual integrado del repositorio a nuestra máquina de desarrollo local. Para esto se utiliza un servidor de control de versiones para verificar una copia de trabajo del código o de la línea principal del código.

Para entender mejor lo dicho anteriormente un controlador de versiones, mantiene todo el código del proyecto en un repositorio. El estado actual del sistema normalmente conocido como línea principal del código. En cualquier momento un desarrollador puede hacer una copia controlada de la línea principal del código en su propia máquina. La copia de la maquina del desarrollador es llamada copia local de trabajo.

Teniendo la copia local, se trabaja en completar la tarea asignada. Esto consistirá tanto en alterar el código de producción, como también ir añadiendo o cambiando las pruebas automatizadas según se vaya desarrollando y sea necesario. Integración Continua asume un alto grado de pruebas ya que están automatizados en el software: una facilidad que se llama código de auto diagnóstico o *self-testing code*.

Una vez terminado se lleva a cabo una construcción automática en la máquina de desarrollo. Esto toma el código fuente de la copia local, compila un ejecutable, y ejecuta pruebas automatizadas. Sólo si todas las construcciones y pruebas no arrojan errores en la construcción total es considerada como buena.

Con una buena compilación, entonces se puede pensar en subir los cambios al repositorio. Lo inesperado, por supuesto, es que otros desarrolladores puedan tener hechos cambios en la línea principal del código antes de que llegue la oportunidad de realizar un *commit o* envío de código al repositorio. Así que primero se actualiza la copia local que tiene los cambios hechos y después se re-construye. Si los cambios de los otros del equipo entran en conflicto con los hechos por nosotros, se manifestará un fallo ya sea en la compilación o en las pruebas. En este caso es nuestra responsabilidad arreglar y repetir hasta que se pueda construir una copia de trabajo local que pueda ser correctamente sincronizada con la línea principal del código.

Una vez que se tiene una correcta construcción sincronizada con la copia local, entonces finalmente se pueden subir los cambios a la línea principal, que a continuación actualizará el repositorio.

Sin embargo, el trabajo con el código transferido al repositorio no finaliza ahí. En este punto se debe compilar de nuevo, pero esta vez en una máquina de integración basada en el código de la línea principal. Sólo cuando la compilación sea satisfactoria se puede decir que nuestros cambios están hechos. Siempre hay posibilidad que haya perdido algún cambio en nuestra máquina de trabajo local y que el repositorio no haya sido correctamente actualizado. Sólo cuando los cambios subidos por nosotros construyan correctamente la integración el trabajo estará bien hecho.

Si hay un conflicto entre las compilaciones de dos desarrolladores, cuando el segundo desarrollador hace la compilación desde su copia local actualizada debe verificar su código. Si no la integración debería fallar, de cualquier modo el error es detectado rápidamente. En este punto la tarea más importante es arreglarlo, y conseguir que la compilación trabaje correctamente otra vez. En entornos con Integración Continua nunca debe permanecer un fallo de integración por mucho tiempo. Un buen equipo debería tener muchos cambios correctamente construidos por día. Malas construcciones ocurren de tiempo en tiempo, pero deberían ser rápidamente arregladas.

El resultado de esto es que hay una pieza estable del software, que funciona correctamente y que contiene algunos errores. Todo el que desarrolla fuera de esta base estable compartida, nunca llega tan lejos de esa base que necesita mucho tiempo para integrar de nuevo con ella. Se pasa menos tiempo intentando encontrar errores porque son mostrados rápidamente.

2.4 Propuesta y configuración de las herramientas seleccionadas

Tres de los pilares de la integración continua son: un repositorio de datos, un servidor de integración continua y una herramienta que permita automatizar la construcción de la aplicación. Además, se pudiera añadir 2 más que serian: un IDE que se integre bien con el resto del entorno y una herramienta que se encargue las pruebas automáticas. De acuerdo con esto se hace la propuesta de herramientas a utilizar en el entorno de integración continua que se propone, además se describe su configuración e integración. En la siguiente figura se muestra un esquema donde se encuentran las herramientas a utilizar en cada parte del proceso.

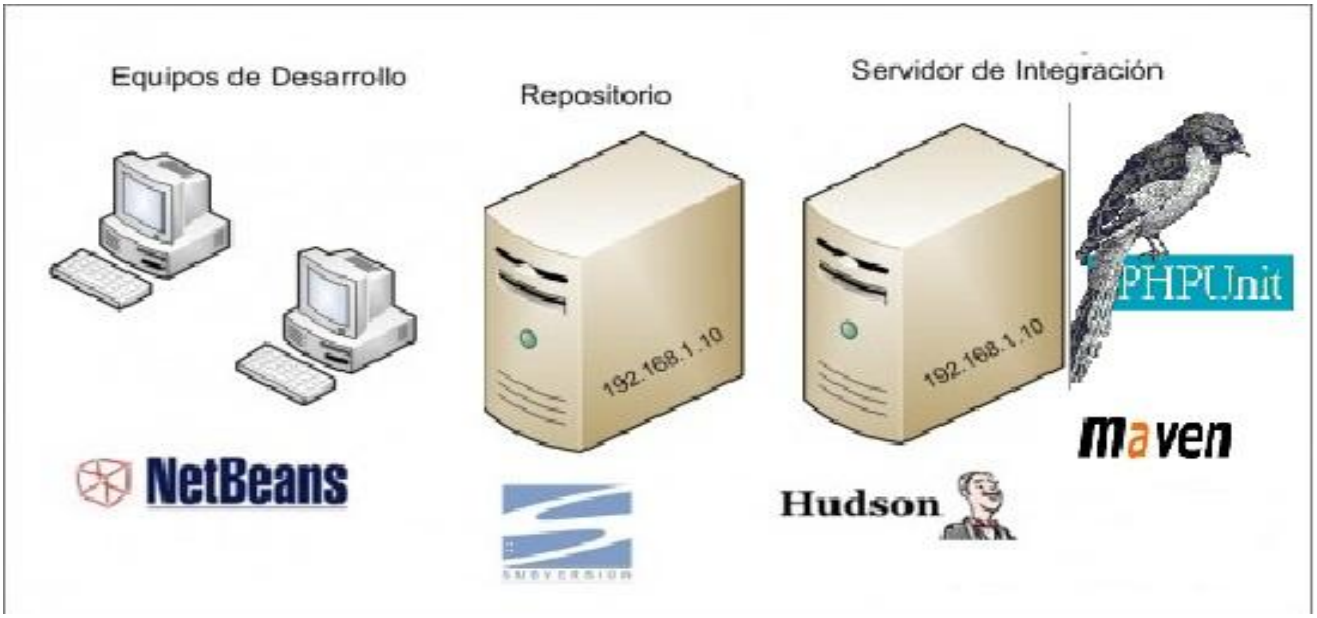


Figura 2. 3 Herramientas a utilizar en cada parte del entorno que se propone

Repositorio de código: SVN

Se elige Subversión porque es un repositorio potente, bastante conocido por la mayoría de los equipos de desarrollo de los proyectos en el centro, el trabajo con esta herramienta es sencillo y sin muchas trabas. Además, se necesita de alguna herramienta administrativa para el trabajo en conjunto con Subversión y queda propuesta Synaptic la cual será de gran ayuda

para manejar el servidor repositorio. Una vez que se tienen las herramientas se pasa a la creación de un repositorio.

Lo primero que hay que hacer es determinar en qué ruta se almacenara el repositorio. Se sugiere almacenar los diferentes repositorios de cada uno de los proyectos bajo una misma raíz y así el servidor se mantiene más o menos organizado. La ruta también debería ser vistosa porque luego formará parte de la configuración que utilizaran los clientes (como los IDE) para conectarse con dicho repositorio. Trabajando con Ubuntu y si se instala el Subversión desde paquetes, se crea un usuario *svn*. Para que todo funcione de manera correcta, hay que asegurarse que el propietario del repositorio es este usuario. Una manera de hacerlo es ejecutar los comandos de administración del Subversión con este usuario y otro es cambiar después el propietario del directorio (y su contenido) con el comando *chown (chown -fR svn.)* En definitiva, para crear el repositorio se hace lo siguiente:

- ✓ svnadmin create /srv/svn/Encuestas-2

Cuando se crea un repositorio en el Subversión es, obviamente, para que los desarrolladores puedan utilizarlo. Por tanto, un paso importante es la parte de seguridad y autorización del mismo: establecer quién podrá acceder y sus credenciales. Subversión tiene diferentes mecanismos de autenticación y de transporte seguro de la información (incluyendo *ssh*).

Como todas las máquinas residen en el interior de mi red de área local, que puede considerarse “zona segura”. Un punto importante es que los mecanismos de seguridad se establecen por repositorio, lo que permite que convivan repositorios con diferentes configuraciones (este es uno de los motivos por los que prefiero tener un repositorio por proyecto). Lo primero es determinar el modo de autenticación.

IDE: Net Beans 6.5

Se escogió este IDE debido al gran conocimiento y experiencia que se tiene con esta herramienta además, se acopla muy bien y es fácil de vincular con las demás herramientas que componen la propuesta de nuestro entorno. Entonces se supone que nuestra aplicación web es típica. Así pues lo primero que se hace es crear un nuevo proyecto web mediante los asistentes del IDE.

Una vez se tiene creado el proyecto web, lo siguiente es enlazarlo con el Subversion. Para ello se hace clic derecho en el nombre del proyecto, y en el menú ***versioning*** o Control de Versiones se selecciona la opción ***Import into Subversion Repository*** o Importar Repositorio Subversion que abrirá un asistente para configurar la conexión. En la versión de Net Beans (6.5) el aspecto que tiene es el que se muestra en la siguiente figura.

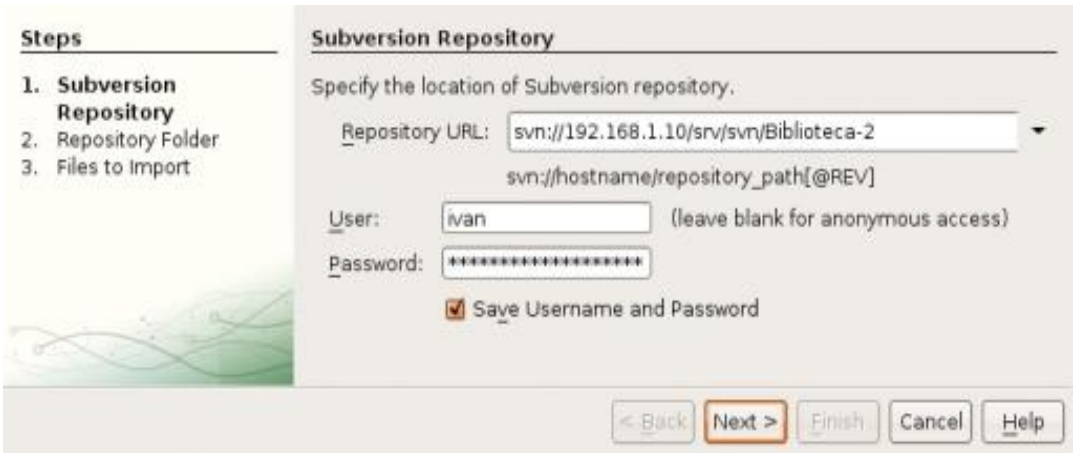


Figura 1. 7 Configurando repositorio en Net Beans 6.5

Los siguientes pasos permiten seleccionar qué ficheros importar al repositorio.

En este momento ya se tiene conectado el IDE con el repositorio. Para utilizar el Net Beans como cliente de Subversion tan sólo hay que hacer clic derecho sobre el recurso correspondiente (un directorio o un fichero) y elegir la opción Subversion del menú contextual, como lo muestra la siguiente imagen.

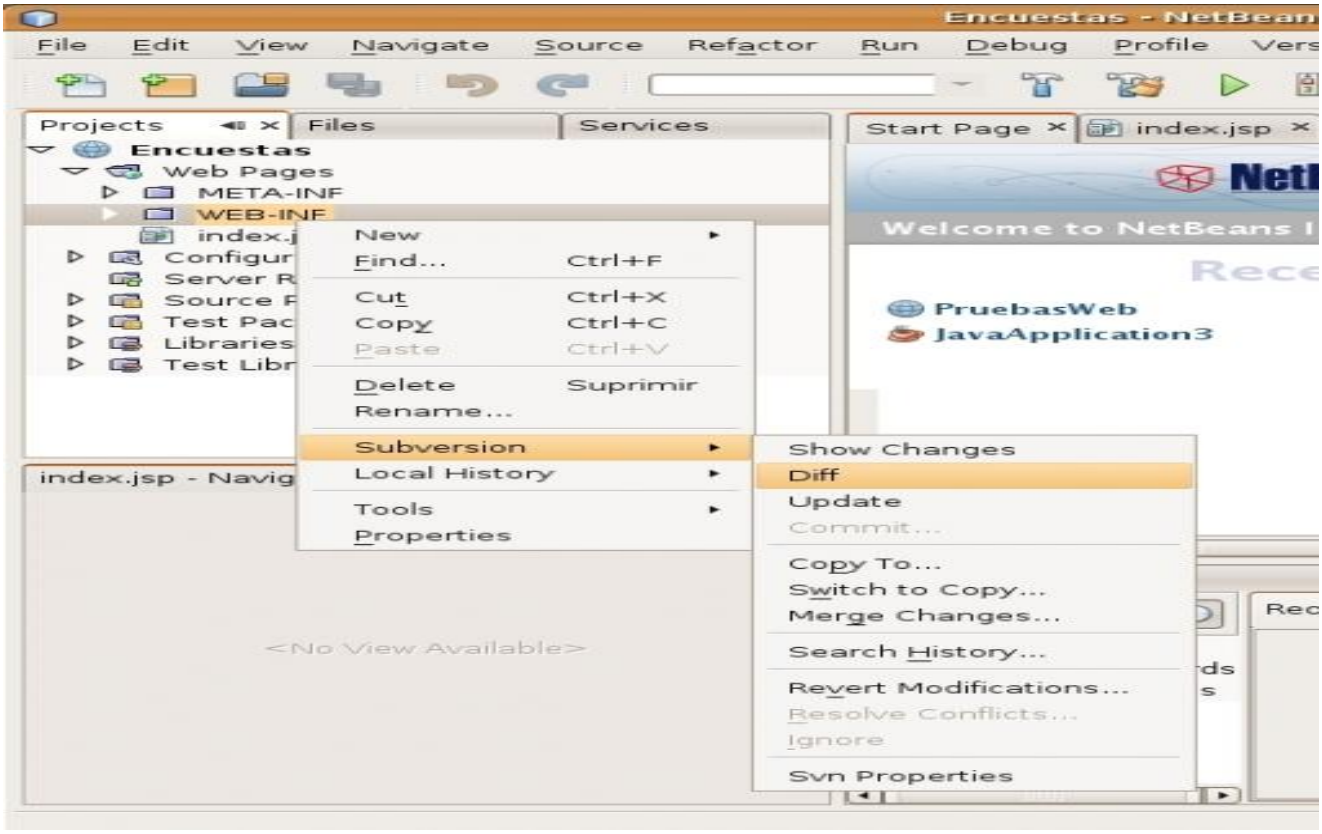


Figura 2. 4 Net Beans conectado con Subversion

Herramienta de pruebas: PHPUnit

PHPUnit es una librería o marco de trabajo que puede ser usada en la fase de pruebas de aplicaciones PHP. Está hecho para correr pruebas y analizar resultados de manera sencilla. PHPUnit es una migración del popular JUnit utilizado en desarrollos Java, integra casos de prueba basadas, pruebas de bases de datos, cálculo de métricas de software, documentación ágil, entre muchas otras características.

Instalar la versión 3.3.0 o posterior PHPUnit. No se necesita configuración especial. Después de PHPUnit está instalado, NetBeans puede reconocerlo. Tenga en cuenta que es necesario tener instalado PEAR con su motor de PHP. Para asegurarse de que el IDE de NetBeans reconoce su instalación PHPUnit, abra Herramientas> Opciones y ver en la ventana de PHP. La ruta de acceso al script PHPUnit debe aparecer en el campo PHPUnit Script, como se muestra en la siguiente figura.

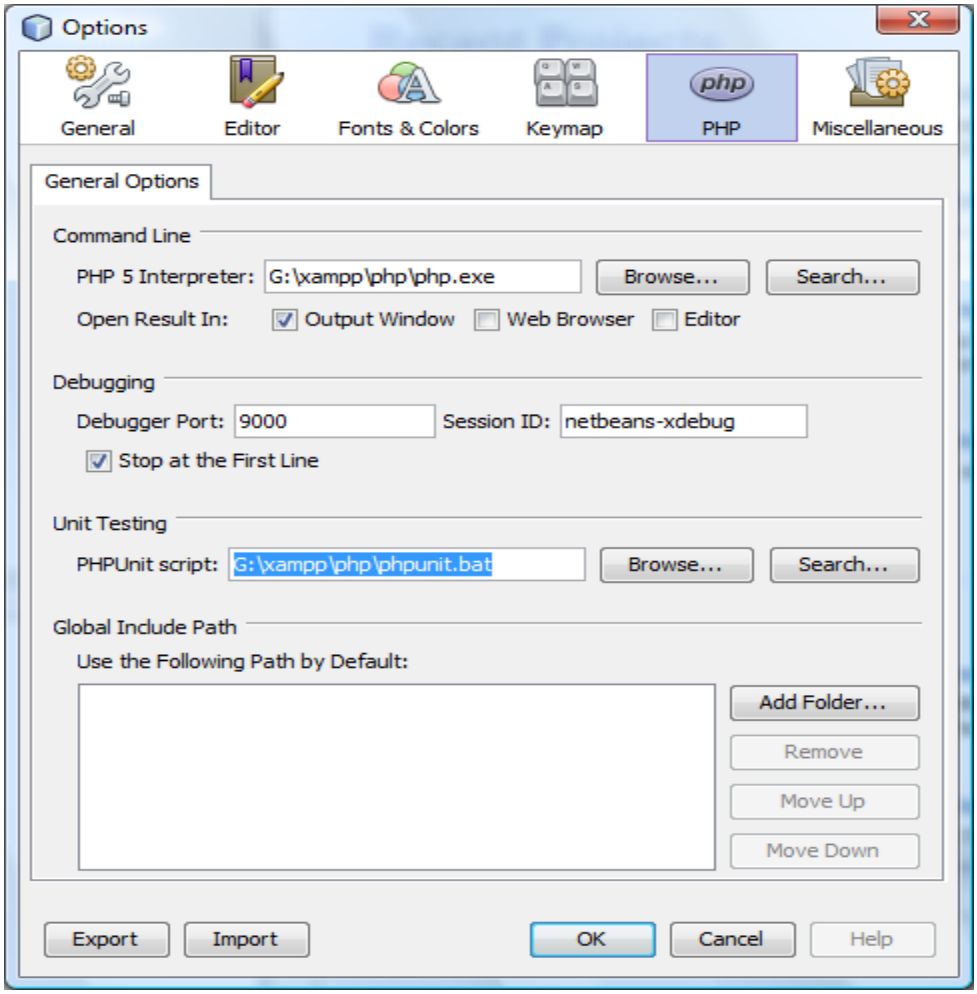


Figura 2. 5 Net Beans reconoce a PHP Unit después de instalado

Herramienta de construcción automática: Maven

Es una herramienta de software para la gestión y construcción de proyectos Java, es similar en funcionalidad a Apache Ant, pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta, pero ahora ya es un proyecto de nivel superior de la Apache Software Foundation.

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado. Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede

dinámicamente descargar plugins de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache. Maven provee soporte no sólo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios. Una caché local de artefactos actúa como la primera fuente para sincronizar la salida de los proyectos a un sistema local.

Maven está construido usando una arquitectura basada en plugins que permite que utilice cualquier aplicación controlable a través de la entrada estándar. En teoría, esto podría permitir a cualquiera escribir plugins para su interfaz con herramientas como compiladores, herramientas de pruebas unitarias, etcétera, para cualquier otro lenguaje.

Maven está construido alrededor de la idea de reutilización, y más específicamente, a la reutilización de la lógica de construcción. Como los proyectos generalmente se construyen en patrones similares, una elección lógica podría ser reutilizar los procesos de construcción. La principal idea es no reutilizar el código o funcionalidad (como Apache Ant), sino simplemente cambiar la configuración o también código escrito. Apache Maven es un marco de trabajo configurable y altamente extensible.

Aunque Maven es configurable, históricamente el proyecto Maven ha enfatizado seriamente que los usuarios deben adherirse a su concepto de un modelo de proyecto estándar tanto como sea posible.

Servidor de integración continua: Hudson

Sólo resta la última pieza: automatizar la construcción del proyecto desde nuestro servidor de integración continua (Hudson) conectándolo al repositorio. También se explica como automatizar el hecho de que se programe una nueva construcción automática cada vez que hay un cambio en el repositorio.

Hudson automatiza la construcción de proyectos en lo que denomina ***jobs***, así pues un mismo proyecto software puede tener diferentes *jobs*. Por ejemplo, tener configurado un *job* que se ejecute cada vez que se haga un cambio en el repositorio y que lo único que compruebe es que el proyecto compila y que se pasan las pruebas unitarias y tener otro *job* programado para ejecutarse tres veces al día para correr pruebas de aceptación o de integración más pesados.

Hudson tiene una interfaz muy sencilla de utilizar. A diferencia de otros servidores de integración, tanto la creación como la configuración de los *jobs* se hacen íntegramente desde la interfaz web.

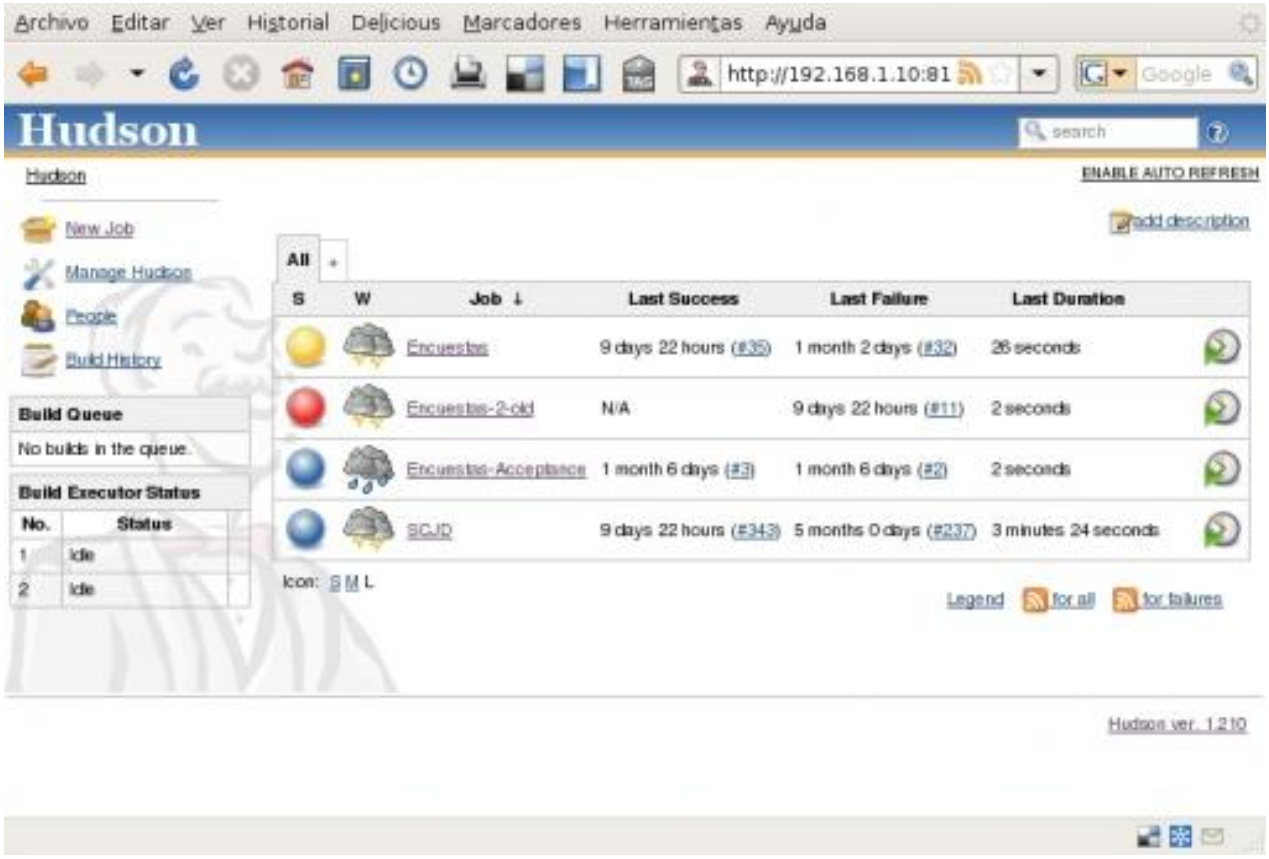


Figura 2. 6 Interfaz de trabajo de Hudson

Para crear los *jobs* se hace clic en ***New Job*** y se iniciará el asistente de creación. Lo primero que se debe hacer es dar un nombre a nuestro *job* que será utilizado tanto en la interfaz e internamente en el servidor de integración. Se puede utilizar cualquier símbolo en el nombre e incluso espacios, pero yo prefiero (manía personal) utilizar nombres con una sola palabra porque cuando luego se quiere navegar "a mano" (mediante la consola) por los *jobs* siempre es más cómodo no tener espacios puesto que el nombre que se usen se corresponderá con el nombre del directorio donde residirá toda la información asociada a dicho *job*. Lo siguiente es elegir el tipo de *job* que se está construyendo; se construirá un *job* estándar que es lo que Hudson llama un ***Build a free-style software Project*** o construir un proyecto de software al estilo libre. Se selecciona y se acepta para ir a la siguiente pantalla del asistente. En esta

nueva pantalla se tienen todos los elementos de configuración del *job* que se rellenan aquéllos que haga falta.

Se podría dejar aquí, pero uno de los requisitos que se quiere es configurar, es que el *job* se construyera automáticamente cada vez que se haga un cambio en el Subversion. Para conseguir este comportamiento se configura el Subversion para que avise al Hudson cada vez que detecte esta situación.

Una de las características que tiene Hudson es que permite programar la construcción de un *job* haciendo una petición *GET* sobre una *url* asociada al proyecto en cuestión.

Por otro lado, el Subversion provee una serie de eventos que pueden ser detectados y asociarles un *script*. Para esto se tiene una serie de plantillas de los *scripts* y eventos que se pueden detectar en la carpeta ***hooks*** del repositorio que se está utilizando. Puesto que lo que se quiere es activar la construcción del proyecto cada vez que se hace un cambio en el repositorio, entonces hay que modificar un *script* que se llama *post-commit*.

El último paso es probar que la integración con el Subversion funciona correctamente. Simplemente se tiene que hacer una transferencia que modifique cualquier fichero y el Hudson debería construir el *job*.

2.5 Roles y responsabilidades

Partiendo de la definición de roles que plantea la metodología XP para Integración Continua y la propuesta antes expuesta, se definen los roles que se vinculan en la misma y las responsabilidades que tiene cada uno.

Jefe de proyecto: Es un rol de administración que debe asegurar que el proyecto se está llevando a cabo de acuerdo con las prácticas y que todo funciona según lo planeado. Su principal trabajo es remover impedimentos y reducir riesgos del producto. Participa en la fase de estudio preliminar (visión general del proyecto, análisis de factibilidad, proyecto técnico). Desarrolla el Plan de Desarrollo de Software. Aprueba las tecnologías a usar en el desarrollo del proyecto. Definir la organización del proyecto. Participa en las revisiones con el cliente de los entregables. Administra la capacitación interna al proyecto.

Programador: Convierte la especificación del sistema en código fuente ejecutable. Desarrolla el diseño teniendo en cuenta la arquitectura. Elabora las pruebas de unidad. Desarrolla el

prototipo de la interfaz de usuario. Integra los componentes que forman parte de la solución. Ejecuta los casos de prueba y generar no conformidades asociadas al mismo. Registra y analiza los resultados de las pruebas.

Probador: Es el encargado de seguir los planes de pruebas. Ejecuta los casos de prueba y genera no conformidades asociadas al mismo. Registra los resultados de las pruebas. Analiza los resultados de las pruebas realizadas. Apoya al cliente en la preparación y realización de las pruebas funcionales. Ejecuta las pruebas funcionales y publica los resultados.

Cliente: El cliente participa en las tareas que involucran la lista de reserva del producto. Revisa y aprueba entregables del proyecto. (Proyecto Técnico). Firma documentación legal del proyecto (actas de inicio, aceptación y fin). Revisa el estado del proyecto.

2.6 Conclusiones Parciales

En el transcurso de este capítulo se ha hecho una propuesta que da solución a la situación problémica planteada, proponiendo también las herramientas a utilizar en dicha solución y además se definen y enumeran los roles que se vinculan y las responsabilidades que debe cumplir cada uno de ellos en esta solución. Se recoge también en este capítulo una propuesta de la organización y funcionamiento global de la solución.

Capitulo 3: Validación de la propuesta

3.1 Introducción

A partir del capitulo anterior donde se hace un estudio del proceso de pruebas que se lleva a cabo en el Centro de Informatización Universitaria y quedando planteada además la propuesta de solución, con su conjunto de herramientas y roles que participan en la misma. Se procede en este capítulo a validar dicha propuesta, la cual fue aprobada por un grupo de expertos, este proceso se explica detalladamente en el presente capitulo.

3.2 Validación de la propuesta

Para la validación y aceptación de la propuesta del entorno de integración continua en el Centro de Informatización Universitaria, se utilizó el criterio de un grupo de especialistas. Además, se hizo uso del Método de Validación por Expertos para evaluar la propuesta y cuantificar el valor de la misma, según el criterio de los especialistas, este panel se conformó con especialistas que dominan los principales aspectos sobre la utilización de integración continua. La eficacia de los resultados depende en su totalidad de la calidad de las preguntas en la elaboración de los cuestionarios y en la selección del panel de expertos. En el presente epígrafe se hace una descripción del proceso de selección del panel de expertos, la elaboración de la encuesta y los resultados arrojados.

El procedimiento a seguir para la validación de la propuesta se centrará en las siguientes fases:

- **Fase preliminar**: Se delimita el contexto, los objetivos, el diseño, los elementos básicos del trabajo y la selección de los expertos.
- **Fase exploratoria**: Elaboración y aplicación de los cuestionarios a los expertos seleccionados en la fase anterior.
- **Fase final**: Análisis de los resultados y presentación de la información.

3.2.1 Selección del panel de expertos

Se entiende por experto, tanto al individuo en sí como a un grupo de personas u organizaciones capaces de ofrecer valoraciones conclusivas de un problema en cuestión y hacer recomendaciones respecto a sus momentos fundamentales con un máximo de competencia. Las características de los expertos influyen decisivamente en la confiabilidad de los resultados obtenidos. Estas características son: calificación técnica, capacidad de emitir una decisión al respecto, conocimientos específicos sobre el tema a evaluar, disposición a participar, entre otros.

Elegir los expertos atendiendo a las características mencionadas anteriormente, propiciará obtener resultados con calidad, junto a otras cualidades propias de éstos como pueden ser: la seriedad, la honestidad, la sinceridad, la responsabilidad y otras en este sentido, que hacen que las opiniones brindadas sean confiables y válidas para el objetivo propuesto. En el desarrollo de este proceso se consideraron tres etapas cruciales:

Determinar la cantidad de expertos: en este trabajo se decidió contar con un número de 7 expertos, teniendo en cuenta nivel de complejidad y profundidad del contenido. Además de su prestigio profesional, los años vinculados a la UCI y su elevado conocimiento.

Elegir los expertos atendiendo a las características mencionadas propician obtener mejores resultados con calidad y que las opiniones brindadas sean confiables y válidas para el objetivo propuesto.

Conformar el listado de los expertos: La confección del listado de expertos se realizó atendiendo a la posibilidad real de participación de los candidatos. Se determinó que los expertos a consultar deben ser profesionales de la UCI, con conocimientos acerca de la integración continua para que puedan emitir un criterio efectivo sobre la propuesta.

Para la selección de los expertos se tuvo en cuenta varias características como son:

- Seriedad.
- Creatividad.
- Disposición a participar en la encuesta.
- Capacidad de análisis.
- Honestidad.

Confirmar la participación de los expertos: Una vez conformado el listado, se invitó personalmente a cada experto elegido para participar en la evaluación. Allí se les explicó en que consistía el trabajo en general, la propuesta a evaluar y el objetivo de la realización de la encuesta, así como el plazo de entrega. Una vez recibida la respuesta positiva, se estableció el listado final de los expertos, informando a cada especialista su inclusión en el proceso a evaluar y las instrucciones necesarias para contestar las preguntas. De esta forma, culmina el proceso de selección, logrando la participación de los 7 expertos escogidos.

3.2.2 Elaboración de la Encuesta

En la encuesta se incluyeron cinco preguntas. En la primera se solicita una opinión general de la solución presentada, facilitando a los expertos la posibilidad de expresarse con libertad, a favor o en contra del procedimiento propuesto. Además, se pidió una evaluación de la propuesta sobre la base de diferentes criterios: de impacto, de implantación y de méritos. La encuesta se llevó a cabo de forma anónima. Se aplicó a cada experto personalmente, explicando detalladamente a cada uno los objetivos y resultados de la propuesta y se les dio un plazo de tiempo para entregarla.

3.2.3 Resultados de la Validación

Para el análisis y procesamiento de los resultados de la propuesta de entorno de integración continua en el centro de informatización universitaria, se escogió un rango de evaluación [1 - 5] para cada una de las preguntas de la encuesta, donde 1 es el mínimo y 5 es el máximo.

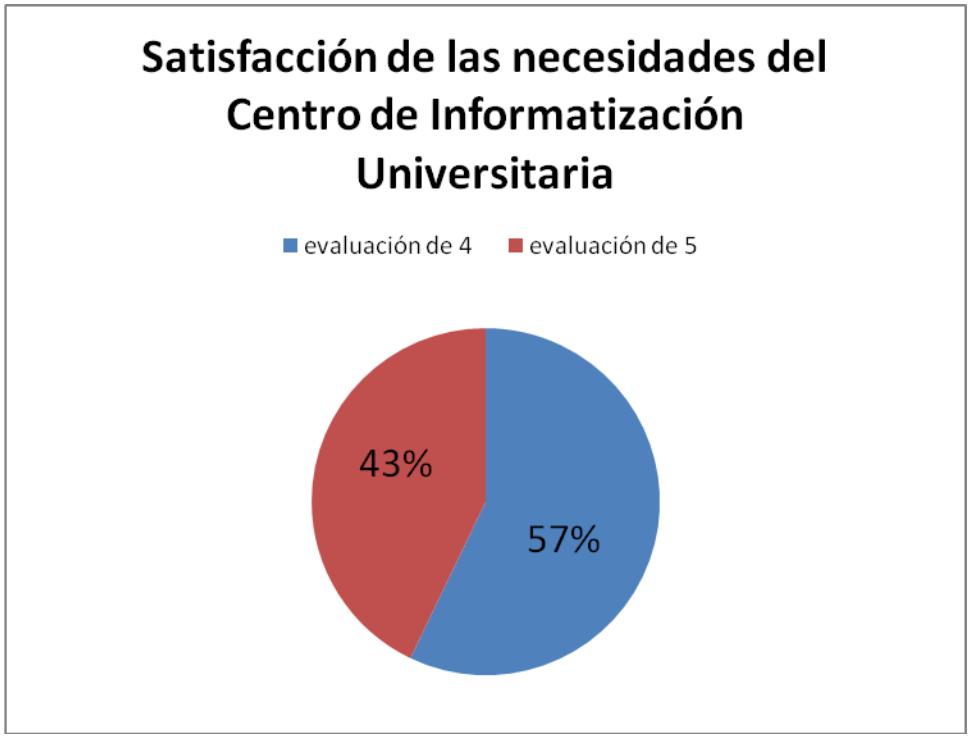
Los resultados de la encuesta aparecen reflejados en la siguiente tabla:

Preguntas	Experto 1	Experto 2	Experto 3	Experto 4	Experto 5	Experto 6	Experto 7
P2	5	4	4	5	5	4	4
P3	4	5	4	4	4	4	5
P4	5	4	5	4	5	5	5
P5	4	4	4	5	5	5	5

Tabla 1 Resultados de la encuesta

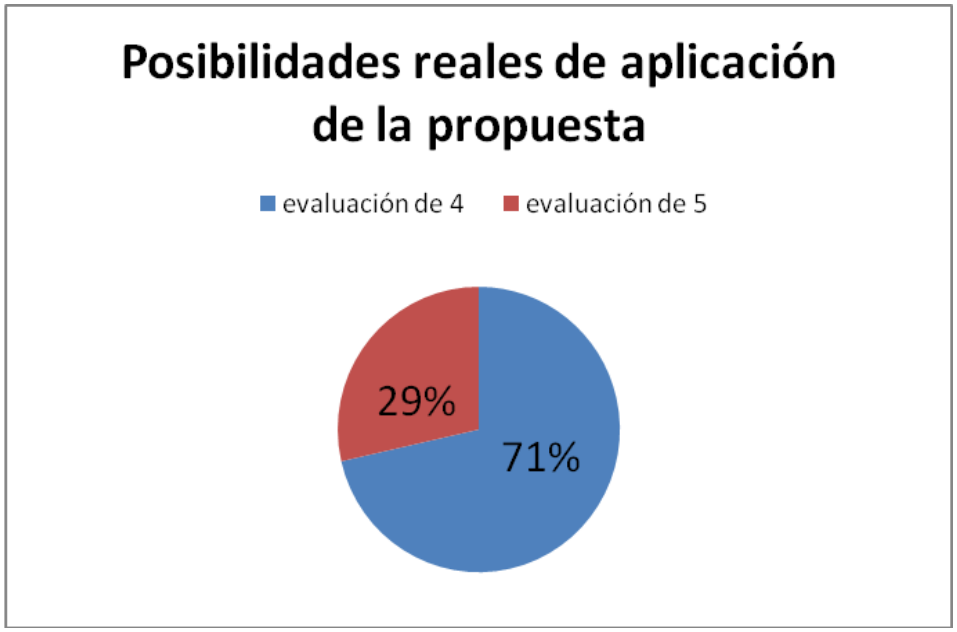
Para la pregunta en la que se solicita una opinión general acerca de la propuesta de solución, la mayoría de los expertos afirma que si se aplica la propuesta cumpliendo con todos los pasos que se describen en el estudio que se realiza en el trabajo, se solucionarán varios problemas que existen actualmente en el centro los largos períodos de integraciones finales, así como identificar fallos en el entorno de producción en etapas tempranas.

Para la pregunta de que si la propuesta satisface las necesidades del Centro de Informatización Universitaria, el 43 % de los entrevistados evaluó de 5, mientras que el 57 % propuso una calificación de 4. (Ver la gráfica 1)



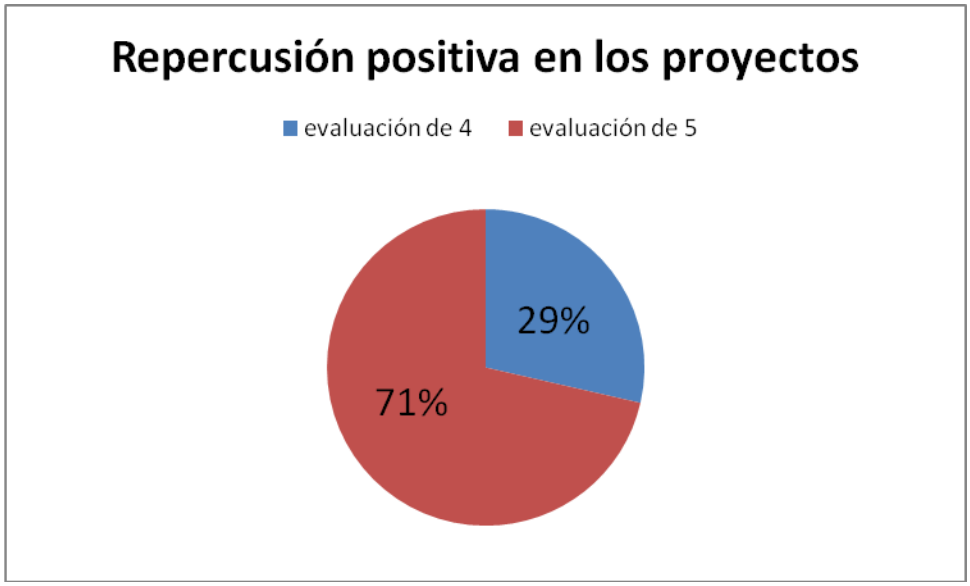
Gráfica 1: Satisfacción de las necesidades del Centro de Informatización Universitaria

Para la pregunta las posibilidades reales de la aplicación de la propuesta, el 29 % de los entrevistados dieron una evaluación de 5, mientras que 71 % dio una evaluación de 4. (Ver gráfica 2)



Gráfica 2: Posibilidades reales de aplicación

Para la pregunta sobre si la propuesta tendrá una repercusión positiva en los proyectos, el 71 % de los entrevistados dio una evaluación de 5, mientras que el 29 % dio una evaluación de 4. (Ver gráfica 3)



Gráfica 3: Repercusión positiva en los proyectos

Para la pregunta sobre si la propuesta ayudará al aumento de la productividad y la motivación al personal del proyecto, el 57 % de los entrevistados dio una evaluación de 5, mientras que el 43 % dio una evaluación de 4. (Ver gráfica 4)

Gráfica 4: Aumentará la productividad y la motivación del personal

Los resultados de la validación arrojaron que el 100% de los expertos considera que la propuesta propiciará el aumento de la productividad en el Centro de Informatización Universitaria. Ante la petición de una evaluación del procedimiento del 1 al 5 sobre la base de los criterios solicitados estos fueron los resultados:

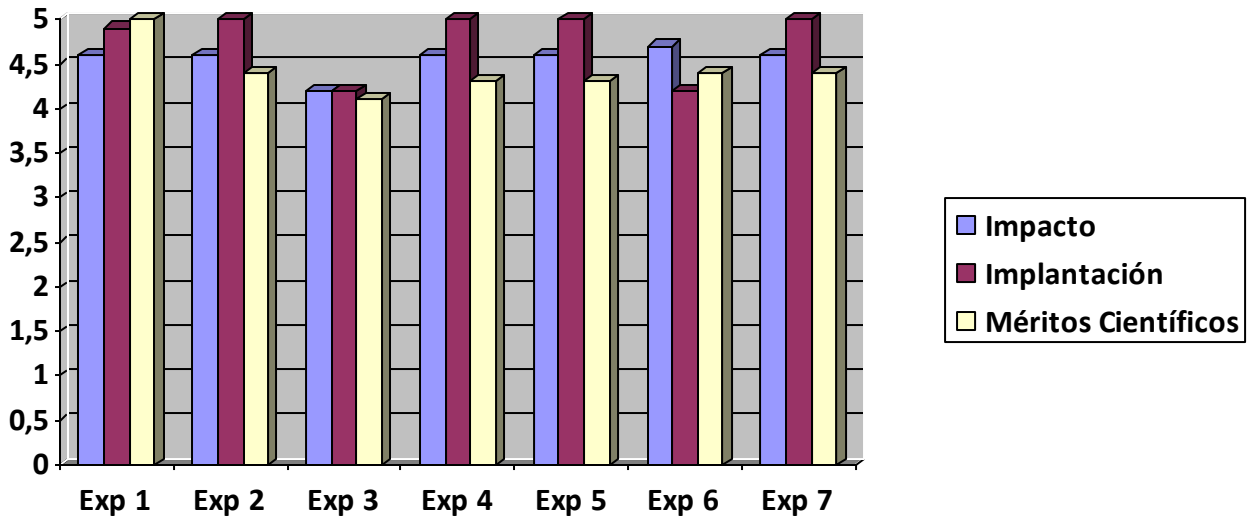


Figura 3.1 Criterio de expertos

De forma general los expertos dieron un alto valor a la propuesta desarrollada ya que en su evaluación los resultados oscilaron entre 4 y 5 puntos. Ante el porqué de esta estimación se recogieron las siguientes opiniones:

- ✓ A grandes rasgos el proceso está bien organizado y pudiera tener éxito, quizás le falte validación práctica para poder evaluar su efectividad.
- ✓ Partiendo de los beneficios que brinda el uso de integración continua, en la propuesta se evidencia, que implantarla en el centro, reducirá los costos de personal y especialmente de tiempo, una reducción y hasta eliminación de los defectos del software y además aumentara la productividad.
- ✓ En estos momentos el proceso de desarrollo en el Centro de Informatización Universitaria se ve afectado por un grupo de deficiencias principalmente con los procesos de integración que son bien difíciles y las pruebas se efectúan solo al final, por lo que el impacto que debe tener esta propuesta debe ser alto, pues debe traer consecuencias positivas al desarrollo del Centro de Informatización Universitaria.

3.3 Conclusiones parciales

En el presente capitulo se realiza la validación de la propuesta, mediante el método Panel de Expertos. Se realiza una descripción de los pasos a seguir para la selección del panel de expertos ya que esto influye decisivamente en los resultados obtenidos. Se recogieron datos cualitativos y cuantitativos que garantizan la validez de la propuesta.

La validación de la propuesta, se realizó mediante encuestas aplicadas a los expertos seleccionados obteniendo un resultado satisfactorio. Lo que determina el cumplimiento de las expectativas de la investigación.

Conclusiones

A partir de la situación problémica por la cual se desarrolla el trabajo investigativo y los problemas que posee el Centro de Informatización Universitaria y con el fin de dar cumplimiento al objetivo general del presente trabajo se definieron los objetivos específicos de la investigación, con los que se arribó a las siguientes conclusiones:

- ✓ Se llevó a cabo el estudio del Estado del Arte de la Integración Continua, lo que permitió tener una mejor visión de los grandes beneficios que puede traer su aplicación en el Centro de Informatización Universitaria y sus proyectos productivos.
- ✓ Se realizó un análisis de algunas herramientas que permiten crear entornos de Integración Continua para diferentes lenguajes de programación, proporcionando datos valiosos para aquellos proyectos que deseen implantarlo.
- ✓ Se seleccionó un conjunto de herramientas de software libre que permitieron proponer un entorno de Integración Continua para el Centro de Informatización Universitaria en el lenguaje de programación PHP.
- ✓ Se validó la propuesta por el método Panel de Experto, donde los expertos luego realizar una encuesta consideraron que la propuesta propiciará un aumento en la productividad en el Centro de Informatización Universitaria, dando un criterio favorable a la propuesta desarrollada.

Recomendaciones

- ✓ Continuar profundizando el tema de Integración Continua.
- ✓ Estudiar como vincular la herramienta Red mine que se utiliza en el Centro de Informatización Universitaria, con la propuesta de solución, para que sea utilizada como mecanismo de retroalimentación instantánea.
- ✓ Aplicar la propuesta en cualquier centro o proyecto con características similares al Centro de Informatización Universitaria.
- ✓ Extender las herramientas propuestas para adaptarlas a las necesidades de la Universidad.
- ✓ Continuar la investigación sobre herramientas de Integración Continua para otras plataformas.
- ✓ Realizar una prueba piloto a un grupo de proyectos del Centro de Informatización Universitaria para darle mayor validez a la propuesta.

Bibliografía

- Artola, L. (24 de Junio de 2009). *Integración continua en PHP: ¿ Xinc o phpUnderControl?* Recuperado el 18 de Septiembre de 2009, de http://www.programania.net/desarrollo-agil/integracion-continua-en-php-%C2%BF-xinc-o-phpundercontrol/
- Artola, L. (16 de Junio de 2009). *Integración continua: objetivos*. Recuperado el 18 de Septiembre de 2009, de http://www.programania.net/desarrollo-agil/integracion-continua-objetivos/
- Artola, L. (5 de Junio de 2009). *Tipos de pruebas automatizadas de software*. Recuperado el 18 de Septiembre de 2009, de http://www.programania.net/diseno-de-software/tipos-de-pruebas-automatizadas-de-software/
- Bartlett, D. (2009). *97 Things Every Software Architect Should Know.*Recuperado el 18 de Septiembre de 2009, de http://www.google.com.cu/books?id=HDknEjQJkbUC&pg=PA40&dq=continuous+integration+software&lr=#v=onepage&q=continuous%20integration%20software&f=false
- Beizer, B. (1990). *Software testing techniques (2nd edition).*Van Nostrand.
- Cañadillas, D. (27 de Abril de 2009). *Elección de herramientas de Integración Continua*. Recuperado el 18 de Septiembre de 2009, de http://www.venera7.com/27-04-2009/eleccion-de-herramientas-de-integracion-continua/
- Christopher Lenz, B. D. (2005). *A framework for collecting software metrics via continuous integration*. Recuperado el 18 de Septiembre de 2009, de http://www.google.com.cu/books?id=nbZWMwAACAAJ&dq=continuous+integration&lr=
- Cosín, J. D. (2007). *Técnicas cuantitativas para la gestión en la ingeniería del software.* Recuperado el 7 de Octubre de 2009, de http://books.google.com.cu/books?id=PZQoZ9KTNaEC&pg=PA58&dq=pruebas+automatizadas+de+software#v=onepage&q=pruebas%20automatizadas%20de%20software&f=false
- Creative Commons. (4 de Febrero de 2007). *TDD: Test Driven Development*. Recuperado el 19 de Octubre de 2009, de Chuidiang.com: http://www.chuidiang.com/java/herramientas/test-automaticos/tdd-test-driven-development.php

- Cuevas, A. G. (2002). *Gestión del proceso software*. Recuperado el 1 de Octubre de 2009, de http://books.google.com.cu/books?id=Pl2HokV7HtIC&dq=gestion+de+configuracion+de+software&source=gbs_navlinks_s
- Curt Hibbs, S. J. (2009). *The Art of Lean Software Development: A Practical and Incremental Approach*. Recuperado el 18 de Septiembre de 2009, de http://www.google.com.cu/books?id=0VsK9cVZauQC&dq=continuous+integration+software&lr=&source=gbs_navlinks_s
- Dan Pilone, R. M. (2007). *Head First Software Development*. Recuperado el 18 de Septiembre de 2009, de http://www.google.com.cu/books?id=QAr82w_eSi8C&dq=continuous+integration+software&lr=&source=gbs_navlinks_s
- Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Recuperado el 19 de Septiembre de 2009, de http://www.google.com.cu/books?id=7dlaMs0SECsC&pg=PA341&dq=continuous+integration+software&lr=#v=onepage&q=continuous%20integration%20software&f=false
- Fowler, M. (2006, Mayo 1). *Martin Fowler*. Retrieved Septiembre 17, 2009, from http://martinfowler.com/articles/continuousIntegration.html
- García, L. (17 de Enero de 2008). *Sistema de control de versiones: SUBVERSION*. Recuperado el 19 de Septiembre de 2009, de http://observatorio.cnice.mec.es/modules.php?op=modload&name=News&file=article&sid=548
- Heinz-W. Schmidt, H. G. (2007). *Component-based software engineering: 10th international symposium, CBSE 2007*. Recuperado el 18 de Septiembre de 2009, de http://www.google.com.cu/books?id=G1jQpyZI2JgC&pg=PA32&dq=continuous+integration#v=onepage&q=continuous%20integration&f=false
- Hunt, J. (2006). *Agile software construction*. Recuperado el 19 de Septiembre de 2009, de http://www.google.com.cu/books?id=P8sWloUWrdIC&pg=PA119&dq=continuous+integration+software&lr=#v=onepage&q=continuous%20integration%20software&f=false
- Joe Walnes, A. A.-B. (2003). *Java Open Source programming: with XDoclet, JUnit, WebWork, Hibernate*. Recuperado el 19 de Septiembre de 2009, de

http://www.google.com.cu/books?id=lf8u8NDNX10C&pg=RA2-PA193&dq=continuous+integration+software&lr=#v=onepage&q=continuous%20integration%20software&f=false

- Larman, C. (2004). *Agile and iterative development: a manager's guide Agile software development series.*Recuperado el 18 de Septiembre de 2009, de http://www.google.com.cu/books?id=76rnV5Exs50C&pg=PA294&dq=continuous+integration#v=onepage&q=continuous%20integration&f=false
- Lee, K. A. (2005, Septiembre 15). *Realizing continuous integration*. Retrieved Septiembre 18, 2009, from IBM: http://www.ibm.com/developerworks/rational/library/sep05/lee/
- Michele Marchesi, G. S. (2003). *Extreme Programming and Agile Processes in Software Engineering: 4th International Conference, XP 2003, Genova, Italy, May 25-29, 2003 : Proceedings*. Recuperado el 18 de Septiembre de 2009, de http://www.google.com.cu/books?id=Zr4mJLaCIQAC&pg=PA114&dq=continuous+integration+software&lr=#v=onepage&q=continuous%20integration%20software&f=false
- Moreno, I. A. (10 de Julio de 2009). *Ambientes de Integración Continua (CI)*. Recuperado el 1 de Octubre de 2009, de http://www.joiz.net/blog/itzcoaltam/?p=409
- Nicolas Guelfi, D. B. (2007). *Rapid integration of software engineering techniques: third international*. Recuperado el 18 de Septiembre de 2009, de http://www.google.com.cu/books?id=x64ocHAaOdEC&pg=PA96&dq=continuous+integration#v=onepage&q=continuous%20integration&f=false
- Paul M. Duvall, S. M. (2007). *Continuous integration: improving software quality and reducing risk*. Recuperado el 18 de Septiembre de 2009, de http://www.google.com.cu/books?id=7700AAAACAAJ&source=gbs_navlinks_s
- Pérez Lamancha, B. (2007). *Gestión de las Pruebas Funcionales.* Recuperado el 19 de Octubre de 2009, de Sistedes.es: http://www.sistedes.es/TJISBD/Vol-1/No-4/articles/pris-07-perez-gpf.pdf
- Rossberg, J. (2008). *Pro Visual Studio Team System Application Lifecycle Management*. Recuperado el 19 de Septiembre de 2009, de http://www.google.com.cu/books?id=iwQ70fojfjoC&pg=PA250&dq=continuous+integration+software&lr=#v=onepage&q=continuous%20integration%20software&f=false

- Russo, B. (2008). *Open Source Development, Communities and Quality: IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software*. Recuperado el 19 de Septiembre de 2009, de http://www.google.com.cu/books?id=lfVAJe6El1IC&pg=PA273&dq=continuous+integration+software&lr=#v=onepage&q=continuous%20integration%20software&f=false
- Sam-Bodden, B. (2006). *Beginning POJOs: from novice to professional.*Recuperado el 18 de Septiembre de 2009, de http://www.google.com.cu/books?id=0y9pgN1rFSkC&pg=PA345&dq=continuous+integration+software&lr=#v=onepage&q=continuous%20integration%20software&f=false
- Smart, J. (2008). *Java Power Tools*. Recuperado el 19 de Septiembre de 2009, de http://www.google.com.cu/books?id=YoTvBpKEx5EC&pg=PA267&dq=continuous+integration+software&lr=#v=onepage&q=continuous%20integration%20software&f=false
- Sommerville, I. (2005). *Ingeniería del software.* Recuperado el 7 de Octubre de 2009, de http://books.google.com.cu/books?id=gQWd49zSut4C&dq=pruebas+automatizadas+de+software&source=gbs_navlinks_s
- Tiako, P. F. (2008). *Designing software-intensive systems: methods and principles*. Recuperado el 19 de Septiembre de 2009, de http://www.google.com.cu/books?id=YTPF08oDb0UC&pg=PA362&dq=continuous+integration+software&lr=#v=onepage&q=continuous%20integration%20software&f=false
- Unkasoft. (5 de Abril de 2006). *Integración continua y gestión de la configuración*. Recuperado el 18 de Septiembre de 2009, de http://eskasiunblog.blogspot.com/2006/04/integracin-continua-y-gestin-de-la.html
- Vivas White, P., & López Romero, E. (Agosto de 2009). *Cuerpo de Profesores de Enseñanza Secundaria. Informática. Temario Vol. II.* Recuperado el 7 de Octubre de 2009, de http://books.google.com.cu/books?id=XYI7i7brEcMC&source=gbs_navlinks_s
- Vivek Chopra, S. L. (2007). *Professional Apache Tomcat 6*. Recuperado el 19 de Septiembre de 2009, de http://www.google.com.cu/books?id=d_FdV6JUZrgC&pg=PA615&dq=continuous+integration+software&lr=#v=onepage&q=continuous%20integration%20software&f=false

Glosario de Términos

Artefactos

En tecnología, es un dispositivo concebido y fabricado, sea de modo artesanal o industrial, por una o más personas. ***29***, ***44***

Calidad

La palabra calidad tiene múltiples significados. La calidad de un producto o servicio es la percepción que el cliente tiene del mismo. Es una fijación mental del consumidor que asume conformidad con un producto o servicio determinado, que solo permanece hasta el punto de necesitar nuevas especificaciones. La calidad es un conjunto de propiedades inherentes a un objeto que le confieren capacidad para satisfacer necesidades implícitas o explícitas. ***1,2*****, 3, 4, 11, 13, 16, 18, 34, 35, 48, 49, 52**

CENIA

Es un acrónimo que significa Centro de Informatización Universitaria, ***54***

Commit

Es cuando subes los cambios del trabajo al repositorio de control de versiones. ***38***, ***46***

Estándares

Es una especificación que regula la realización de ciertos procesos o la fabricación de componentes para garantizar la interoperabilidad. ***33***, ***34***

Herramientas

Son los ambientes de apoyo necesario para automatizar las practicas de Ingeniera de Software. ***VI*** ,***VII***,***6***, ***16***, ***23***, ***24***, ***26***, ***27***, ***28***, ***29***, ***30***, ***31***, ***33***, ***39***, ***40***, ***44***, ***47***, ***48***, ***53***, ***54***, ***55***

Java

Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. ***27***, ***28***, ***29***, ***30***, ***31***, ***32***, ***33***, ***42***, ***43***, ***44***, ***56***, ***58***

Metodologías ágiles

Nuevo enfoque metodológico orientado a la gente y los resultados. ***5***, ***28***

Métodos

Son las maneras que se efectúan las tareas de Ingeniería de Software o las actividades del ciclo de vida. ***4***, ***18***, ***32***

PHP

Es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente en interpretación del lado del servidor. ***28***, ***32***, ***33***, ***42***, ***53***, ***55***

Plugins

Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica. ***31***, ***33***, ***44***

Proceso de desarrollo

Elemento organizativo a través del cual se gestiona el desarrollo de software. El resultado de un proyecto es una versión de un producto. ***1***, ***2***, ***3***, ***4***, ***12***, ***18***, ***26***, ***34***, ***51***

Proyecto de software

Elemento organizativo a través del cual se gestiona el desarrollo de software. El resultado de un proyecto es una versión de un producto. ***43***, ***45***

Script

Guión o conjunto de instrucciones,archivo de órdenes o archivo de procesamiento. ***15***, ***16***, ***42***, ***46***

Software

Es el conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica, en contraposición a los componentes físicos del sistema. ***1***, ***II***, ***VI***, ***1***, ***2***, ***3***, ***4***, ***5***, ***6***, ***8***, ***9***, ***11***, ***12***, ***13***, ***14***, ***15***, ***16***, ***17***, ***18***, ***19***, ***20***, ***22***, ***24***, ***26***, ***27***, ***29***, ***32***, ***33***, ***34***, ***36***, ***37***, ***39***, ***42***, ***43***, ***44***, ***45***, ***51***, ***53***, ***55***, ***56***, ***57***, ***58***

Anexos

Modelo para la recogida de información referente a la encuesta para la validación de la propuesta de entorno de integración continua para el Centro de Informatización Universitaria, mediante la técnica del panel de expertos.

1. **Exponga una opinión general acerca de la propuesta de solución que se realiza para el Centro de Informatización Universitaria.**

2. **¿Cree usted que la propuesta tenga una repercusión positiva en los proyectos del centro?**

   __ Muy alta

   __ Alta

   __ Media

   __ Baja

   __ Muy baja

3. **¿Considera usted que la propuesta de solución para el Centro de Informatización Universitaria satisface las necesidades del centro?**

   __ Muy satisfecha

   __ Bastante satisfecha

   __ Satisfecha

   __ Poco satisfecha

   __ Insatisfecha

4. **¿Considera usted que la propuesta tiene posibilidades reales de ser aplicada en el Centro de Informatización Universitaria?**

   __ Muy alta

   __ Alta

__ Media

__ Baja

__ Muy baja

5. ¿Cree usted que la propuesta ayudará a aumentar la productividad y la motivación del personal de los proyectos del Centro de Informatización Universitaria?

6. __ Muy alta
7. __ Alta
8. __ Media
9. __ Baja
10. __ Muy baja