UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERÍA INDUSTRIAL Y DE SISTEMAS

APLICACIÓN DE LA INTEGRACIÓN CONTINUA PARA LA MEJORA DEL PROCESO DE DESARROLLO DE SOFTWARE EMPLEANDO HERRAMIENTAS DE SOFTWARE LIBRE EN UNA EMPRESA CONSULTORA

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO DE SISTEMAS

HUGO SERGIO NÚÑEZ RAMÍREZ

LIMA – PERÚ

2012

A Dios,

a mis padres Sergio y Nilda,

y a mi familia por todo su apoyo.

AGRADECIMIENTO

Doy agradecimientos especiales a mis padres Sergio y Nilda, ya que sin sus palabras de motivación, apoyo incondicional y consejos no estaría donde estoy ahora. Todos mis esfuerzos y logros son dedicados principalmente a ellos.

Y doy muchas gracias a mis compañeros de trabajo Juan, Colin, Andres y Rosario, ya que por su apoyo y esfuerzo el presente trabajo pudo ser realizado.

INDICE

DESCRIPTORES TEMÁTICOS ..... **v**
RESUMEN **1**
INTRODUCCIÓN **2**
CAPÌTULO I. PENSAMIENTO ESTRATEGICO **4**
1.1 DIAGNÓSTICO FUNCIONAL 4
1.1.1 ORGANIZACIÓN 4
1.1.2 OBJETIVOS 4
1.1.3 PRINCIPALES SERVICIOS 4
1.1.4 CLIENTES 6
1.1.5 PROCESOS 6
1.2 DIAGNÓSTICO ESTRATÉGICO 7
1.2.1 ANÁLISIS INTERNO 7
1.2.2 ANÁLISIS EXTERNO 11
1.2.3 ANÁLISIS FODA 12
CAPÌTULO II. MARCO TEÓRICO Y METODOLÓGICO **14**
2.1 DESARROLLO ÁGIL DE SOFTWARE 14
2.2 PROGRAMACIÓN EXTREMA 15
2.2.1 HISTORIAS DE USUARIO 16
2.2.2 ROLES 16
2.2.3 PROCESO 17
2.2.4 PRÁCTICAS 18
2.2 INTEGRACIÓN CONTINUA 18
2.2.1 ESCENARIO 20

2.2.3 VALOR QUE APORTA AL PROCESO DE DESARROLLO.....21
2.2.4 OTRAS PRÁCTICAS QUE COLABORAN CON LA INTEGRACIÓN CONTINUA.....24
2.3 INTEGRACIÓN CONTROLADA.....26
2.3.1 DESARROLLO EN PARALELO.....27
2.4 SOFTWARE LIBRE.....27
2.4.1 FIABILIDAD DEL SOFTWARE LIBRE.....30
2.4.3 SOFTWARE LIBRE CONTRA PROPIETARIO.....32
CAPÌTULO III PROCESO DE TOMA DE DECISIONES 35
3.1 IDENTIFICACIÓN DEL PROBLEMA.....35
3.2 ANÁLISIS DEL PROBLEMA.....36
3.3 PLANTEAMIENTO DE ALTERNATIVAS DE SOLUCIÓN.....41
3.3.1 ADOPTAR LA METODOLOGÍA ÁGIL PROGRAMACIÓN EXTREMA.....41
3.3.2 IMPLEMENTAR UN PROCESO DE INTEGRACIÓN CONTROLADA.....42
3.3.3 APLICAR UN PROCESO DE INTEGRACIÓN CONTINUA.....42
3.4 CRITERIOS DE DECISIÓN.....42
3.5 EVALUACIÓN Y SELECCIÓN DE ALTERNATIVA.....43
3.6 DECISIÓN SOBRE LAS HERRAMIENTAS DE SOFTWARE LIBRE..45
3.6.1 CRITERIOS DE DECISIÓN.....45
3.6.2 HERRAMIENTA DE GENERACIÓN DE SOFTWARE.....46
3.6.3 Herramienta para la Integración Continua.....47
3.6.4 HERRAMIENTA DE REPOSITORIO DE COMPONENTES.....48
CAPÌTULO IV. SOLUCIÓN PROPUESTA 49
4.1 DESCRIPCIÓN DE LA SOLUCIÓN.....49
4.1.1 NUEVO ESQUEMA DE TRABAJO.....49
4.1.2 INFRAESTRUCTURA DEL ENTORNO DE DESARROLLO.....51
4.1.3 HERRAMIENTAS DE SOFTWARE LIBRE PARA LA AUTOMATIZACIÓN DEL PROCESO.....53
4.2 DESARROLLO DE LA SOLUCIÓN.....55
4.2.1 CRONOGRAMA DE PROYECTO.....55

4.2.2 RECURSOS 57
4.2.3 ANÁLISIS DE RIESGOS 58
4.2.4 EJECUCIÓN DEL PROYECTO 58
CAPÌTULO V. EVALUACIÓN DE RESULTADOS 59
5.1 CRITERIOS 59
5.2 CÁLCULOS Y RESULTADOS 60
5.2.1 CALIDAD 60
5.2.2 EFICIENCIA 61
5.2.3 COSTO 62
5.2.4 OTROS RESULTADOS 63
5.2.5 RESUMEN 63
CONCLUSIONES Y RECOMENDACIONES 65
GLOSARIO DE TÉRMINOS 68
BIBLIOGRAFÍA 69
ANEXOS 70

DESCRIPTORES TEMÁTICOS

Proceso de Desarrollo de Software

Desarrollo Ágil

Software Libre

Integración Continua

Calidad del Software

RESUMEN

El presente informe presenta como implementar la integración continua para la mejora del proceso de desarrollo de software mediante la automatización de procesos repetitivos apoyándose en herramientas de software libre.

Un proyecto de desarrollo de software puede generar sobrecostos al emplear esfuerzo adicional para cumplir con los requerimientos del producto en las fechas establecidas.

La integración continua sugiere un esquema de trabajo donde el equipo desarrollador integre su trabajo en un entorno de desarrollo automatizado y controlado, que permite que tareas repetitivas como compilación, ejecución de pruebas, despliegue e inspección de código se ejecuten de manera automática o a demanda, y al mismo tiempo informar al equipo de problemas en el software integrado.

Esto permite una mejora en la calidad del producto software debido a la ejecución y cumplimiento de las pruebas, una reducción del esfuerzo empleado en la construcción dadas las condiciones del entorno de desarrollo de integración continua, lo que trae consigo la reducción de tiempo y costos en las fases de construcción y transición del proceso de desarrollo.

INTRODUCCIÓN

El campo de la ingeniería de software ha evolucionado a lo largo del tiempo, pasando desde el desarrollo de aplicaciones concretas y pequeñas hasta el software de nivel empresarial.

Para poder ir a la par con las exigencias crecientes de software surgió la necesidad de tener una definición del proceso de desarrollo de software, lo que trajo consigo las primeras propuestas metodológicas conocidas hoy en día como metodologías clásicas.

Sin embargo estas metodologías no llegaron a ser suficiente, con el tiempo se exigía un desarrollo mucho más rápido debido a la creciente exigencia en el alcance de los sistemas informáticos. Para ello surgió el desarrollo ágil, el cual vino con nuevas propuestas metodológicas y buenas prácticas.

Entre estás se encuentra la Integración Continua, una buena práctica considerada dentro de la metodología extreme programming, pero que por sí sola puede ser implementada en cualquier entorno de desarrollo, independiente de la metodología o estilo de trabajo.

Al mismo tiempo, el surgimiento del software libre ha permitido que la comunidad mundial de programadores aporten al proceso con herramientas que sirven de apoyo al desarrollo ágil. Software que cualquier persona, natural o jurídica, puede utilizar con total libertad.

Se verá como con la aplicación de esta práctica con apoyo de diversas herramientas de software libre se consigue la mejora en la calidad del software, reducción de riesgos en el desarrollo, así como la automatización de procesos repetitivos, lo que lleva a reducción de tiempos y costos.

En el Capítulo I se dará una vista general de la empresa en estudio, haciendo un diagnóstico funcional y estratégico.

En el Capítulo II se describirán las teorías y metodologías de la cual es objeto el presente informe. Se empezará desde el Desarrollo Ágil siguiendo por la Integración Continua y el tema de Software Libre.

En el Capítulo III se identifica analiza el problema encontrado en la empresa para la cual se plantean un conjunto de soluciones y estas son evaluadas en función a diversos criterios.

En el Capítulo IV se describirá con mayor detalle la alternativa de solución seleccionada, la Integración Continua, indicando los requerimientos de proceso, hardware y software, así como el detalle del plan de acción a seguir.

En el Capítulo V se presentarán los resultados obtenidos de la implantación de la solución, haciendo una comparación entre la situación inicial y final, después de implantar la solución.

CAPÍTULO I

PENSAMIENTO ESTRATEGICO

1.1 DIAGNÓSTICO FUNCIONAL

1.1.1 ORGANIZACIÓN

La empresa es una consultora de desarrollo de soluciones de TI, la cual ofrece servicios de desarrollo de soluciones de TI. Su rubro es el de Consultores Programación y Suministros Informáticos (código CIIU 72202). Ha estado presente en el mercado por alrededor de 15 años.

1.1.2 OBJETIVOS

Dentro de sus principales objetivos se encuentran:

- Mejorar la calidad del proceso y del producto.
- Crecimiento de ventas entre 20 y 25%
- Reducir los costos de las operaciones en 5%

1.1.3 PRINCIPALES SERVICIOS

La empresa ofrece los siguientes servicios a sus clientes:

- **Desarrollo de aplicaciones:** Ofrece la instalación e implementación de aplicaciones ya desarrolladas, así como el soporte en el desarrollo de requerimientos específicos del cliente para que el aplicativo se adapte lo mejor posible al proceso de negocio del cliente. Además ofrecemos el servicio de ventas de software representado por nuestra empresa. Es el servicio más importante ya que representa su principal fuente de ingresos.

- **Educación:** La empresa ofrece también el servicio de training o capacitación de diversas herramientas, principalmente de IBM. Debido a la alianza estratégica que tiene con IBM del PERU, la empresa puede dar capacitaciones sobre los cursos oficiales de IBM.

- **Soporte y Arquitectura**: Comprende el servicio de soporte que ofrece a sus clientes, posterior a un desarrollo o consultoría.

1.1.4 PROVEEDORES

Dentro de sus proveedores se encuentran los siguientes:

- IBM del Perú: Les provee de documentación, herramientas y cursos libres. Poder de negociación alto.

- Microsoft: Un caso análogo para lo que es herramientas de programación de Microsoft (.NET). Poder de negociación alto.

- BITS: Empresa que provee de los equipos de cómputo. Poder de negociación bajo.

Aquí se presenta un caso excepcional en que sus proveedores se convierten también en sus competidores.

1.1.4 CLIENTES

Dentro de los clientes de la empresa se encuentran las principales entidades bancarias, BCP, Scotiabank, Continental, así como empresas de otras industrias como los son Alicorp, Unique, Ransa, por mencionar algunos.

1.1.5 PROCESOS

La empresa no realiza la gestión de procesos, por lo que estos no se encuentran del todo definidos o no se encuentran formalizados como tales. Sin embargo, es posible realizar un mapeo de los procesos principales de la empresa, mostrado en la Figura 1.2.

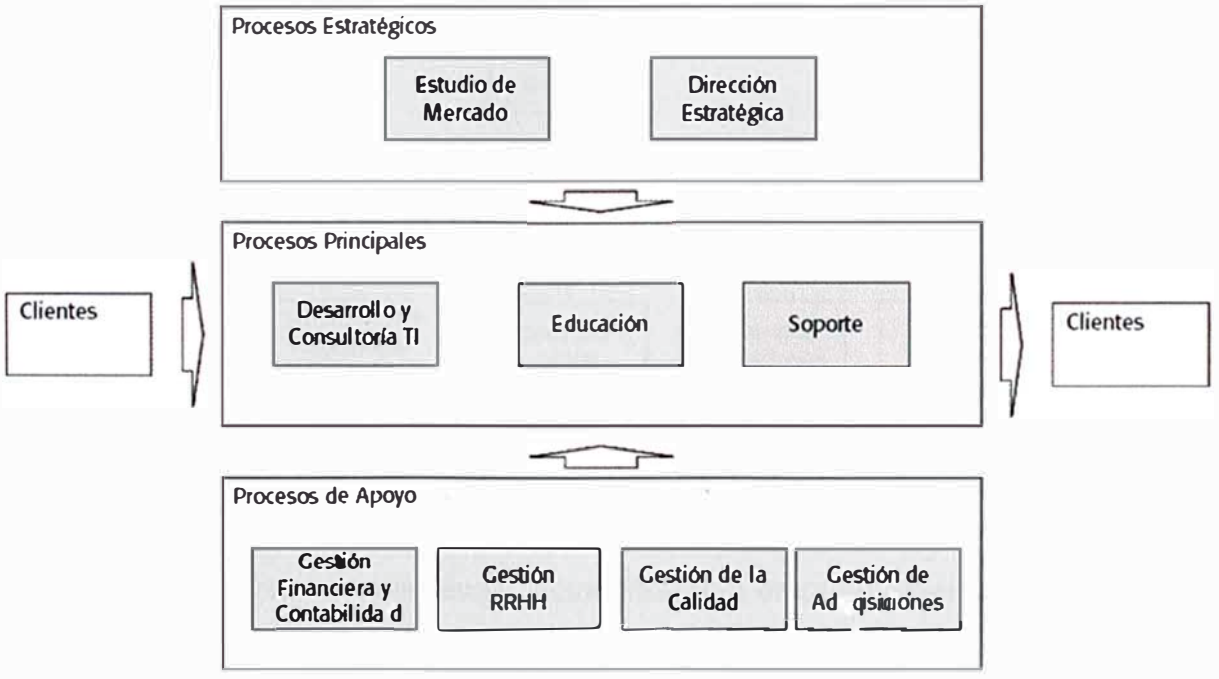


Figura 1.1 – Mapa de procesos de la empresa

Dentro de los procesos principales, el que tiene más relevancia para la empresa es el de Desarrollo y Consultoría TI, ya que esta representa su

principal fuente de ingresos, y es donde tiene asignado la mayor cantidad de recursos.

La empresa no cuenta con un proceso de desarrollo bien definido, pero intentan seguir la metodología de desarrollo RUP, la cual han adoptado para el apoyo en su objetivo de obtener la certificación CMMi.

1.2 DIAGNÓSTICO ESTRATÉGICO

1.2.1 ANÁLISIS INTERNO

1.2.1.1 ORGANIGRAMA

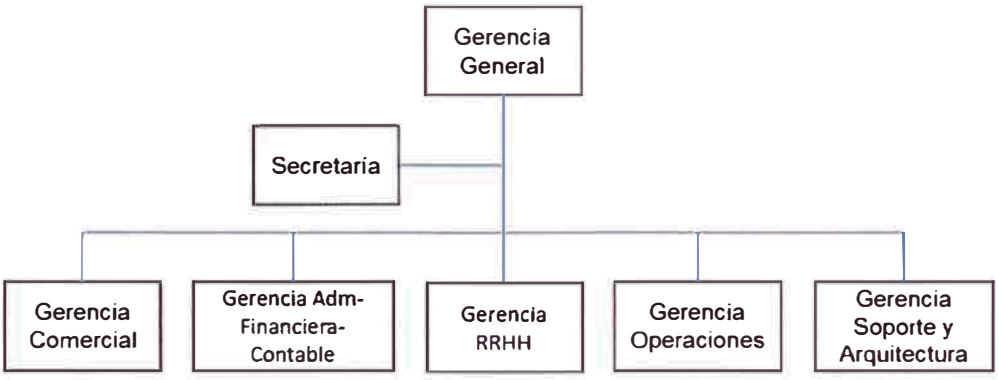


Figura 1.2 – Organigrama de la empresa

El organigrama muestra una estructura funcional con 2 niveles, La gerencia general y las gerencias operativas. Actualmente la empresa cuenta con 110 empleados.

Dentro de estas gerencias, es la Gerencia de Operaciones la que tiene asignada la mayor cantidad de recursos con un 80% de los empleados. Esta está encargada de lo relacionado al desarrollo de proyectos de soluciones software.

1.2.1.2 VISIÓN

"Ser reconocidos como empresa líder en el desarrollo de software, contribuyendo de manera rentable al posicionamiento de nuestro país como una de las mejores alternativas de la industria."

1.2.1.3 MISIÓN

"Experimentar la satisfacción en el aprendizaje y la aplicación de tecnología de punta en el desarrollo de software para beneficio de la sociedad."

1.2.1.4 INFRAESTRUCTURA Y EQUIPOS

La empresa realiza los desarrollos en sus instalaciones o en instalaciones de los clientes que estos acondicionan, según como sea el contrato.

En el último año la empresa amplio sus instalaciones con la compra de una casa al lado de su sede central; así mismo realizo las compras para el equipamiento de los nuevos laboratorios de computo con lo cual han incrementado su capacidad de servicio de educación y de desarrollo in-situ.

Actualmente cuenta con una capacidad de cuatro laboratorios de cómputo con 10 estaciones de trabajo por laboratorio, de las cuales 2 son utilizadas para el desarrollo y los otros 2 para las capacitaciones a los clientes.

1.2.1.5 RECURSOS HUMANOS

Por mucho tiempo la empresa contaba con un pobre proceso de selección el cual no calificaba correctamente a los postulantes lo que no asegura que los

nuevos trabajadores contaran con las capacidades o habilidades necesarias para desempeñar su labor.

Así mismo también carecían de un adecuado criterio para asignar personal a los proyectos, el único que tenían era la disponibilidad del recurso, sin tomar en consideración cuales eran los "skills" del recurso.

En vista de esto la empresa inicio un programa para la mejora del proceso de reclutamiento y selección de personal, así como un programa de capacitaciones para el personal que es orientado a las tecnologías a emplearse en sus futuros proyectos.

1.2.1.6 SISTEMAS DE INFORMACIÓN

La empresa utiliza sistemas ofimáticos para el manejo de su documentación en general, incluyendo manejo de planillas y contabilidad.

1.2.1.7 OPERACIONES

El negocio principal de la empresa es el desarrollo de aplicaciones, para lo cual concentra gran parte de sus recursos en él. Un estimado del 80% de su personal se encuentra asignado a la planificación y ejecución de proyectos.

La empresa maneja un promedio de 17 proyectos a la vez, donde cada equipo de proyecto se compone en un promedio de 5 a 6 personas.

Del histórico de proyectos el 80% de estos son basados en Java mientras que el 20% restante son basados en los lenguajes .NET y PHP.

La empresa está presentando sobrecostos en sus operaciones debido a que en su compromiso con el cliente por cumplir con las fechas acordadas se realiza un esfuerzo mayor al planificado.

Las causas son por problemas internos en los proyectos.

1.2.1.8 GESTIÓN DE LA CALIDAD

En el último año la empresa ha dedicado muchos esfuerzos para prepararse para obtener la certificación CMMi nivel 3, formalizando el proceso de desarrollo, preparando la documentación exigida.

La empresa cuenta con un área dedicada a velar por la calidad del proceso y del producto, para lo cual ha elaborado una serie de indicadores y métricas, así como se encargan de brindar el apoyo para el proceso de desarrollo mediante la validación y verificación de las soluciones desarrolladas.

1.2.1.9 ALIANZAS ESTRATÉGICAS

La empresa consiguió el título de IBM Business Partner, lo que le da un respaldo de IBM del Perú y esto es considerado como una ventaja competitiva ya que esto le permite el acceso a recursos bibliográficos oficiales y exclusivos, licencias de uso de las herramientas de IBM.
Un caso análogo sucede con Microsoft, otorgándoles el título de Microsoft Certified Partner.

1.2.2 ANÁLISIS EXTERNO

1.2.2.1 IMAGEN

La empresa mantiene la imagen ante sus clientes a través del cumplimiento de los tiempos y demás acuerdos establecidos en los proyectos.

1.2.2.2 CAMBIOS MONETARIOS

La empresa mantiene contratos con algunos clientes con un precio establecido en dólares americanos, esto causó que existiera un margen de pérdida debido al incremento del valor de la moneda nacional.

1.2.2.3 FACTORES CULTURALES

Los clientes tienen la tendencia a confiar más en una empresa extranjera, como es el caso de transnacionales como IBM y Microsoft lo que provoca una amenaza.
Sin embargo la experiencia y trayectoria de la empresa hace que esta mantenga a sus clientes.

1.2.2.4 MERCADO

Debido a su limitada capacidad de atención, la empresa cuenta con una cuota de mercado del 2% local. La empresa tiene oportunidades en el mercado a nivel nacional, pero se concentra más en el mercado local.
Sus expectativas de expandir su mercado se encuentran dependientes de un incremento en su capacidad de atención.

1.2.2.5 COMPETIDORES

Los competidores directos de la empresa son aquellas que se encuentran en su mismo rubro, como por ejemplo a nivel nacional:

- GMD
- COSAPISOFT
- Systems Support & Services
- Avatar
- Everest

También tienen las siguientes transnacionales:

- IBM del Perú
- Microsoft del Perú

La empresa también tiene competidores en el extranjero, teniendo que competir con consultoras con sedes en Argentina, Costa Rica, India y España.

1.2.3 ANÁLISIS FODA

En la siguiente tabla se muestra la matriz FODA, que incluye las fortalezas, debilidades, oportunidades y amenazas de la empresa, así como las estrategias sugeridas,

Matriz FODA	**OPORTUNIDADES** • Creciente demanda del sector. • Investigación en tecnologías para dispositivos móviles. • Competencia hace propuestas costosas en comparación con las de la empresa. • Incremento en la demanda laboral del sector.	**AMENAZAS** • Fácil ingreso de nuevos competidores. • Fuga de talentos. • Devaluación del dólar americano frente al nuevo sol.
FORTALEZAS • Tiempo en el mercado. • Programa interno de certificaciones. • IBM Business Partner. • Microsoft Certified Partner. • Profesionales certificados. • Buena imagen en el mercado.	**Estrategias FO:** • Fomentar la investigación en desarrollo para dispositivos móviles. • Incentivar la participación del personal en los programas de certificación.	**Estrategias FA:** • Participar en ferias laborales, presentando las fortalezas de la empresa.
DEBILIDADES • Proceso de selección no muy efectivo. • Sobrecarga de los recursos. • Dependencia en los recursos. • Carece de un área de investigación. • Capacidad de proyectos limitada. • Proceso de desarrollo no está bien definido.	**Estrategias DO:** • Crear el área de investigación y desarrollo. • Participar en ferias laborales, presentando los beneficios de pertenecer a la empresa. • Mejorar el proceso de desarrollo de software, estableciendo metodologías y buenas prácticas del proceso.	**Estrategias DA:** • Mejorar el proceso de selección de personal incrementando la rigurosidad y variando más seguido los exámenes de ingreso.

Tabla 1.1 Matriz FODA

CAPÍTULO II

MARCO TEÓRICO Y METODOLÓGICO

2.1 DESARROLLO ÁGIL DE SOFTWARE

El desarrollo ágil de software es un marco de trabajo conceptual de la ingeniería de software que promueve iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto. Existen muchos métodos de desarrollo ágil; la mayoría minimiza riesgos desarrollando software en cortos lapsos de tiempo.

El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener un demo (sin errores) al final de cada iteración. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto.

Los métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación. La mayoría de los equipos ágiles están localizados en una simple oficina abierta, a veces llamadas "plataformas de lanzamiento" (bullpen en inglés). La oficina debe incluir revisores, escritores de documentación y ayuda, diseñadores de iteración y directores de proyecto.

Los métodos ágiles también enfatizan que el software funcional es la primera medida del progreso. Combinado con la preferencia por las comunicaciones cara a cara, generalmente los métodos ágiles son criticados y tratados como "indisciplinados" por la falta de documentación técnica.

Algunos métodos ágiles de desarrollo de software se nombran a continuación:

- Adaptive Software Development (ASD).
- Agile Unified Process (AUP).
- Crystal Clear.
- Essential Unified Process (EssUP).
- Feature Driven Development (FDD).
- Lean Software Development (LSD).
- Kanban.
- Open Unified Process (OpenUP).
- Programación Extrema (XP).
- Método de desarrollo de sistemas dinámicos (DSDM).
- Scrum.

2.2 PROGRAMACIÓN EXTREMA

La Programación Extrema, también conocida por su acrónimo en inglés "XP", es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define

como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Kent Beck, el padre de XP, describe la filosofía de XP sin cubrir los detalles técnicos y de implantación de las prácticas. Posteriormente, otras publicaciones de experiencias se han encargado de dicha tarea.

2.2.1 HISTORIAS DE USUARIO

Son la técnica utilizada para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

2.2.2 ROLES

Los roles de acuerdo con la propuesta original son:

- Programador. El programador escribe las pruebas unitarias y produce el código del sistema.
- Cliente. Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.
- Encargado de pruebas (Tester). Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, di funde los

resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

- Encargado de seguimiento (Tracker). Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.
- Entrenador (Coach). Es responsable d el proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
- Consultor. Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.
- Gestor (Big boss). Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

2.2.3 PROCESO

El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar

el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

2.2.4 PRÁCTICAS

En la Figura 2.1 se muestran las prácticas sugeridas por la programación extrema.

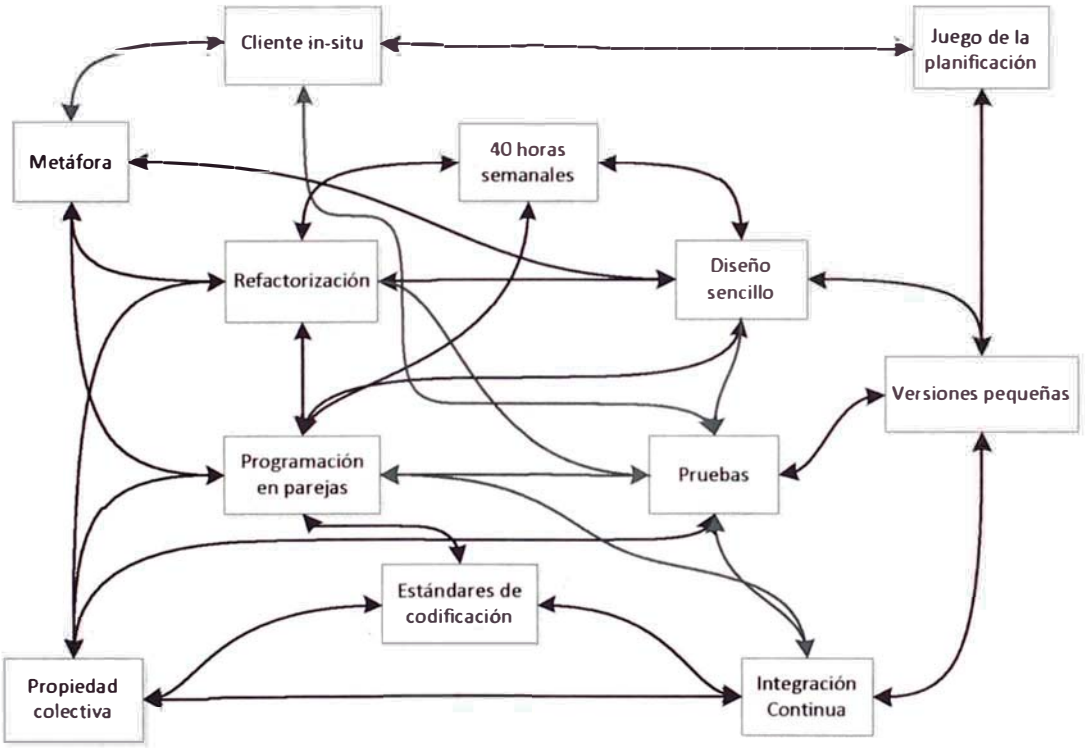


Figura 2.1 – Prácticas XP

2.2 INTEGRACIÓN CONTINUA

Una parte importante de cualquier proceso de desarrollo de software es cada vez se construya software más fiable. A pesar de su importancia, a menudo nos sorprendemos cuando esto no se hace. Se hace hincapié en una construcción totalmente automatizada y reproducible, y las pruebas que se

ejecutan varias veces al día. Esto permite que cada desarrollador integre todos los días lo que reduce los problemas de integración.

La integración continua es considerada como una buena práctica, un proceso o hasta como una metodología por muchos, sin embargo su creador, Martin Fowler, lo define de la siguiente manera:

La integración continua es una práctica de desarrollo de software donde los miembros de un equipo integran su trabajo con frecuencia, por lo general cada persona lo hace al menos una vez al día, lo que conduce a múltiples integraciones por día. Cada integración es verificada por un sistema automatizado de construcción (incluyendo la ejecución de pruebas) para detectar errores de integración lo más rápido posible. Muchos equipos encuentran que este enfoque conduce a una reducción significativa de los problemas de integración y permite a un equipo para desarrollar software cohesivo más rápidamente.[1]

Esto significa que:

- Los desarrolladores ejecutan generaciones de software en sus propias estaciones de trabajo antes de realizar commit su código al repositorio de control de versiones para asegurarse que sus cambios no hagan fallar la integración.
- Los desarrolladores subir los cambios de su código al menos una vez al día.
- La integración sucede varias veces al día en un servidor aparte.
- El 100% de las pruebas deben pasar satisfactoriamente para cada generación de software.
- Un producto es generado (e.g. WAR, ejecutable, etc.) que puede ser probado funcionalmente.

[1] Cf. Martin Fowler, *Continuous Integration*, en: <http://martinfowler.com/articles/continuousIntegration.html>.

- Corregir las generaciones fallidas se encuentra como máxima prioridad.
- Algunos desarrolladores revisan los reportes generados por el build, como los estándares en codificación y reportes de análisis de dependencias, para buscar posibles mejoras.

2.2.1 ESCENARIO

El escenario de IC comienza en el momento en que un desarrollador envía sus cambios al repositorio. En un proyecto común, gente en muchos roles del proyecto pueden enviar cambios que inicien un ciclo de IC: Desarrolladores cambian el código fuente, administradores de base de datos cambian archivos de configuración, y eventos así.

Los pasos en un escenario CI serían los siguientes:

1. Un desarrollador sube su código al repositorio de control de versiones. Mientras tanto, el servidor IC en el equipo de integración está examinando este repositorio en busca de cambios.
2. Tan pronto se sube el código de un programador, el servidor IC detecta que los cambios han ocurrido en el repositorio de control de versiones, entonces el servidor IC obtiene la última copia del código del repositorio y luego ejecuta el script del build, el cual integra el software.
3. El servidor IC genera retroalimentación por correo electrónico los resultados de la generación del software a miembros del proyecto específicos.
4. El servidor IC continúa examinando por cambio en el repositorio de control de versiones.

Estos pasos se ilustran en la Figura 2.1.

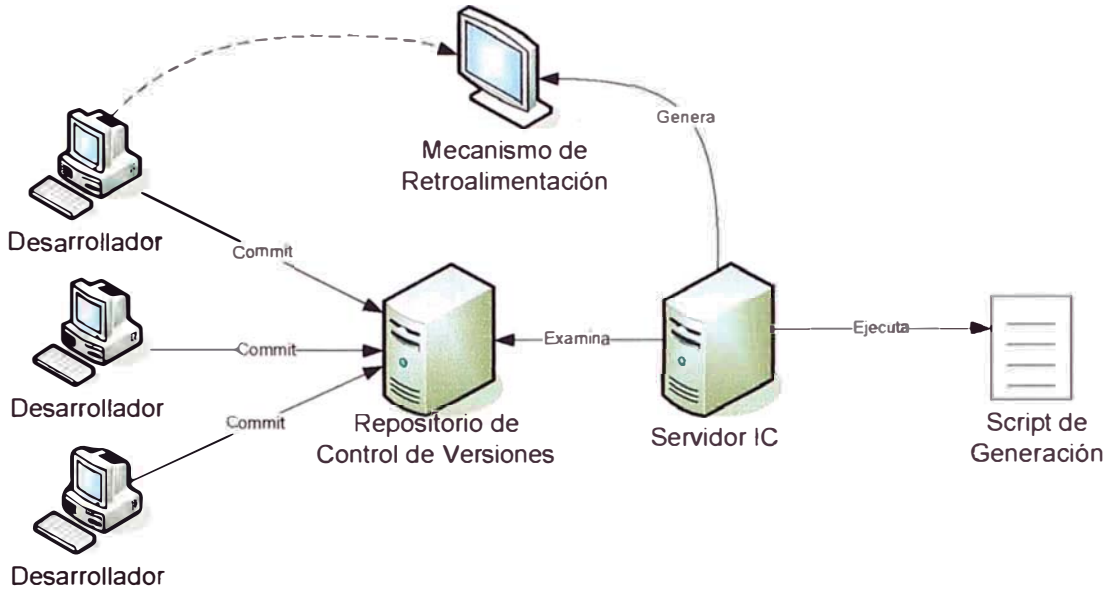


Figura 2.1 - Escenario de Integración Continua[2]

2.2.3 VALOR QUE APORTA AL PROCESO DE DESARROLLO

En un alto nivel el valor que aporta la Integración Continua sirve para lo mencionado en las siguientes secciones.

2.2.3.1 REDUCIR RIESGOS

Cuando se integra varias veces al día, se pueden reducir riesgos en el proyecto. Hacer esto facilita la detección de errores, la medición del estado del software, y la reducción de supuestos.

Los defectos son detectados y corregidos oportunamente. Debido a que la IC integra y ejecuta pruebas e inspecciones varias veces al día, existe una gran probabilidad que los errores sean detectados cuando estos son

[2] Cf. Paul M. Duvall, *Continuous Integration*, Pearson Education, Pg. 5

introducidos (i.e. cuando el código es enviado al repositorio de control de versiones) en lugar que durante un ciclo de pruebas posterior.

El estado del software es medible. Debido a la incorporación de la ejecución de pruebas continuas y a la inspección dentro de un proceso de integración automatizado, los atributos del estado del producto software, como la complejidad, pueden ser rastreados en el tiempo.

Reduce los supuestos. Debido a que realiza el build y se ejecutan las pruebas en un entorno limpio usando el mismo proceso y scripts en forma continua, se pueden reducir supuestos (si se está contando con librerías de terceros o variables de ambiente).

Los siguientes son algunos de los riesgos que la IC ayuda a mitigar.

- Falta de software cohesivo y desplegable.
- Descubrimiento tardío de errores.
- Software de baja calidad.
- Falta de visibilidad del proyecto.

2.2.3.2 REDUCIR TAREAS REPETITIVAS

Reducir tareas repetitivas ahorra tiempo, costo y esfuerzo. Estos procesos repetitivos pueden ocurrir por todas las actividades del proyecto, incluyendo la compilación del código, integración con base de datos, pruebas, inspección, despliegue y retroalimentación. Al automatizar la IC, se tiene una mayor capacidad para asegurar lo siguiente:

- El proceso se ejecuta de la misma forma todo el tiempo.

- Se sigue un proceso ordenado. Por ejemplo, se pueden ejecutar inspecciones (análisis estático) antes de ejecutar las pruebas, dentro del script de generación.
- El proceso se iniciará cada vez que se suba código en el repositorio de control de versiones.

Esto facilita:

- La reducción del trabajo en procesos repetitivos, liberando gente para hacer trabajo que genere mayor valor.
- La capacidad de superar la resistencia (de otros miembros del equipo) a implementar mejoras, al usar mecanismos automatizados para procesos importantes como son la ejecución de pruebas e integración con la base de datos.

2.2.3.3 GENERAR SOFTWARE DESPLEGABLE

La IC puede permitir liberar software desplegable en cualquier momento. Desde una perspectiva de alguien externo, este es el beneficio más obvio de la IC. Se puede decir mucho de la mejora de la calidad de software y reducir riesgos, pero el software desplegable es el activo más tangible para un externo como lo son los clientes o usuarios.

2.2.3.4 MEJORAR LA VISIBILIDAD DEL PROYECTO

La IC ofrece la capacidad de darse cuenta tendencias y tomar decisiones efectivas, y ayuda a brindar el ánimo para innovar en nuevas mejoras. Los proyectos sufren cuando no hay datos actualizados reales parar apoyar a la toma de decisiones, por lo que todos dan su mejor aproximación. Normalmente, los miembros del equipo recolectar esta información manualmente, haciendo de este esfuerzo oneroso e inoportuno.

La IC tiene los siguientes efectos positivos.

- Decisiones efectivas. Un sistema de IC puede brindar información al momento sobre el estado del build más reciente y métricas de calidad.
- Darse cuenta de tendencias. Como la integración ocurre con frecuencia, se hace posible la capacidad de notar tendencias en el éxito o error del build, calidad en general, y otra información pertinente del proyecto.

2.2.3.5 ESTABLECER UNA MAYOR CONFIANZA EN EL PRODUCTO

En general, la aplicación efectiva de prácticas de IC puede ofrecer gran confianza al producir un producto software. Con cada build, el equipo sabe que las pruebas del software se ejecutan para verificar su comportamiento, que los estándares de codificación y diseño se están cumpliendo, y que el resultado es un producto que puede probarse funcionalmente.

Si no se hicieran integraciones frecuentes, algunos equipos podrían sentir inseguridad porque no se conoce del impacto de los cambios en el código. Dado que un sistema de IC puede informar cuando algo resulta mal, los desarrolladores y otros miembros del equipo tienen más confianza al realizar cambios. Y como la IC sugiere tener un único lugar donde genera el software, hay más confianza de su precisión.

2.2.4 OTRAS PRÁCTICAS QUE COLABORAN CON LA INTEGRACIÓN CONTINUA

Sería incorrecto pensar que sólo por tener un sistema automatizado de integración continua ya se encuentra libre de errores de integración. Una

práctica de integración continua efectiva involucra mucho más que sólo una herramienta. Existen otras prácticas necesarias:

- Enviar código al repositorio de control de versiones con frecuencia.
- Corregir generaciones fallidas inmediatamente.
- Usar una maquina aparte para la integración.

También existen otras prácticas que colaboran con la integración continua para obtener mejores resultados.

2.2.4.1 PRUEBAS

La planificación, construcción y ejecución de pruebas unitarias resulta de vital importancia en para la IC. Muchos de los proyectos de desarrollo no se realizan pruebas unitarias debido a que se emplea más tiempo para implementar las funciones del software. Pueden existir pruebas, pero estas resultan insuficientes o pueden no cubrir partes importantes del código.

Para que la IC se dé de manera correcta y se puedan obtener los beneficios de esta práctica es de vital importancia realizar pruebas unitarias al código, pruebas que el sistema de IC ejecute durante la generación.

2.2.4.2 PROPIEDAD COLECTIVA DEL CÓDIGO

Es importante que el código sea de propiedad de todo el equipo, de lo contrario el conocimiento de partes del código se encuentra aislada en individuos. Esta práctica permite que todos tengan visibilidad completa del código y cualquier miembro del equipo asignado a la construcción pueda realizar cambios y correcciones sobre cualquier parte del código.

2.2.4.3 ADOPCIÓN DE UN ESTÁNDAR DE CODIFICACIÓN

Un estándar de codificación resulta de gran ayuda para la IC, provee de los parámetros necesarios para hacer que el código sea más entendible para todos en el equipo, y también permite que

2.3 INTEGRACIÓN CONTROLADA

La integración controlada sugiere, a diferencia de la integración continua, no se haga la integración tan seguida sino que se haga cada cierto tiempo, en puntos de integración bien definidos con anterioridad.

La integración controlada busca explotar al máximo las capacidades de los sistemas de control de versiones empleando técnicas de ramificación y emplear el desarrollo en paralelo.

Dentro de un sistema de control de versiones se tiene la línea principal de desarrollo, que es aquella que contiene el código final para un proyecto de software. Empleando solo esta línea causa que en un equipo de desarrolladores los errores terminen propagándose de manera innecesaria a todo el equipo.

Para evitar este problema, la integración controlada indica que se debe establecer un nuevo rol en el equipo, el Integrador. Este integrador es el encargado de establecer los puntos de control para los ciclos de desarrollo, y de crear dentro del sistema de control de versiones una rama de desarrollo por cada programador. Cuando se alcanza un punto de control es cuando el integrador realiza la tarea de combinar el código (merge en inglés) y agregarlo a la línea principal dentro del sistema de control de versiones. Una vez hecho esto nuevamente el Integrador crea nuevas ramas para los desarrolladores.

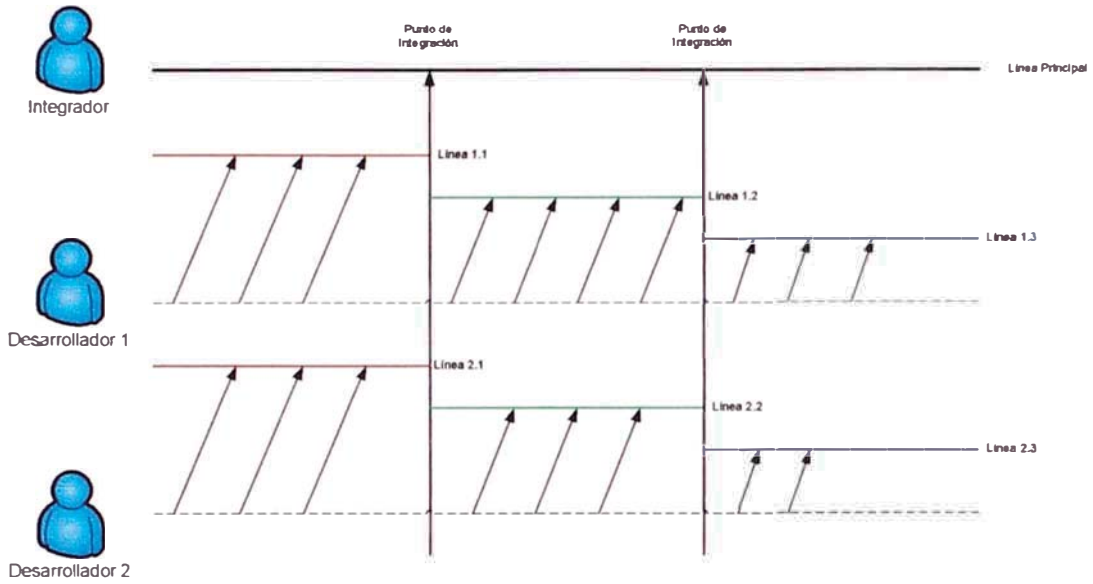


Figura 2.3 - Integración Controlada

2.3.1 DESARROLLO EN PARALELO

La integración controlada sugiere emplear un desarrollo en paralelo, donde cada desarrollador puede realizar la codificación y subir sus cambios al sistema de control de versiones de manera independiente sin entrar en conflicto con el trabajo de otro desarrollador, lo que reduce la posibilidad de propagación de errores en la línea principal de desarrollo.

Para conseguirlo se emplean patrones de ramificación en el sistema de control de versiones.

2.4 SOFTWARE LIBRE

El software libre es una cuestión de la libertad de los usuarios de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Más precisamente,

significa que los usuarios de programas tienen las cuatro libertades esenciales:

- La libertad de ejecutar el programa, para cualquier propósito (libertad 0).
- La libertad de estudiar cómo trabaja el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para que pueda ayudar al prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (la 3ª libertad). Si lo hace, puede dar a toda la comunidad una oportunidad de beneficiarse de sus cambios. El acceso al código fuente es una condición necesaria para ello.

Un programa es software libre si los usuarios tienen todas esas libertades. Entonces, debería ser libre de redistribuir copias, tanto con o sin modificaciones, ya sea gratis o cobrando una tarifa por distribución, a cualquiera en cualquier parte. El ser libre de hacer estas cosas significa, entre otras cosas, que no tiene que pedir o pagar el permiso.

También debería tener la libertad de hacer modificaciones y usarlas en privado, en su propio trabajo u obra, sin siquiera mencionar que existen. Si publica sus cambios, no debería estar obligado a notificarlo a alguien en particular, o de alguna forma en particular.

La libertad de ejecutar el programa significa la libertad para cualquier tipo de persona u organización de usarlo en cualquier tipo de sistema de computación, para cualquier tipo de trabajo y propósito, sin estar obligado a comunicarlo a su programador, o alguna otra entidad específica. En esta libertad, el propósito de los usuarios es el que importa, no el propósito de los

programadores. Como usuario es libre de ejecutar un programa para sus propósitos; y si lo distribuye a otra persona, también es libre para ejecutarlo para sus propósitos, pero usted no tiene derecho a imponerle sus propios propósitos.

La libertad de redistribuir copias debe incluir las formas binarias o ejecutables del programa, así como el código fuente; tanto para las versiones modificadas como para las no lo están. (Distribuir programas en forma de ejecutables es necesario para que los sistemas operativos libres se puedan instalar fácilmente). Resulta aceptable si no existe un modo de producir un formato binario o ejecutable para un programa específico, dado que algunos lenguajes no incorporan esa característica, pero debe tener la libertad de redistribuir dichos formatos si encontrara o programara una forma de hacerlo.

"Software libre" no significa "que no sea comercial". Un programa libre debe estar disponible para el uso comercial, la programación comercial y la distribución comercial. La programación comercial de software libre ya no es inusual; tal software libre comercial es muy importante. Puede haber pagado dinero para obtener copias de software libre, o puede haber obtenido copias sin costo. Pero sin tener en cuenta cómo obtuvo sus copias, siempre tiene la libertad de copiar y modificar el software, incluso de vender copias.

Si una modificación constituye una mejora es un asunto subjetivo. Si sus modificaciones se limitan, en esencia, a los cambios que otra persona considera una mejora, eso no se trata de libertad.

No obstante, las reglas acerca cómo empaquetar una versión modificada son aceptables si no limitan substancialmente su libertad para publicar versiones modificadas, o su libertad para hacer y usar versiones modificadas en privado. Así que es aceptable que una licencia le obligue a cambiar el nombre de la versión modificada, eliminar el logotipo o a identificar sus

modificaciones como suyas. Son aceptables siempre y cuando esas obligaciones no sean tan agobiantes que le dificulten la publicación de sus modificaciones. Como ya está aplicando otras modificaciones al programa, no le supondrá un problema hacer algunas más.

Las normas del estilo "si pone a disposición su versión de este modo, también debe hacerlo de este otro modo" también pueden ser, bajo la misma condición, admisibles. Un ejemplo de una norma admisible, sería una que planteara que si ha distribuido una versión modificada, y uno de los programadores de versiones anteriores le pide una copia, deberá mandarle una (tenga en cuenta que esta norma le sigue permitiendo elegir si distribuye, o no, su versión.). Las normas que obligan a liberar el código fuente a los usuarios de las versiones que publica también son admisibles.

2.4.1 FIABILIDAD DEL SOFTWARE LIBRE

A los defensores del software privativo les gusta decir "*El software libre es un bonito sueño, pero todos sabemos que sólo el sistema privativo puede producir productos fiables. Un puñado de "hackers" simplemente no lo puede hacer*".

Sin embargo, la evidencia empírica disiente; pruebas científicas, descritas más adelante, han comprobado que el software libre es más fiable que el software privativo comparable.

Esto no debería ser una sorpresa; existen buenas razones para la alta fiabilidad del software libre, buenas razones para esperar que el software libre tendrá a menudo (aunque no siempre) una alta fiabilidad.

Barton P. Miller y sus colegas probaron la fiabilidad de programas de utilidades de Unix en 1990 y 1995. En ambas ocasiones, las utilidades de

GNU destacaron considerablemente. Probaron siete sistemas Unix comerciales así como GNU. Sometiéndolos a un flujo de entrada aleatorio, pudieron "abortar (con volcado de memoria) o colgarse (bucle infinito) más del 40% (en el peor caso) de las utilidades básicas...".

Estos investigadores comprobaron que los sistemas Unix comerciales tenían una tasa de fallos que iba desde el 15% al 43%. En contraste, la tasa de fallos de GNU fue sólo del 7%.

Miller también dijo que "los tres sistemas comerciales que comparamos tanto en 1990 como en 1995 mejoraron considerablemente en fiabilidad, pero aún tenían tasas de fallo significativas (las utilidades básicas de GNU/Linux todavía eran considerablemente mejores que las de los sistemas comerciales)".

No es casualidad que las utilidades GNU sean más fiables. Hay buenas razones por las cuales el software libre tiende a ser de alta calidad.

Una razón es que el software libre consigue involucrar a toda la comunidad para que trabaje unida para arreglar problemas. Los usuarios no sólo informan de errores, incluso los arreglan. Los usuarios trabajan juntos, conversando por correo electrónico, para llegar al fondo del problema y hacer que el software funcione sin problemas.

Otra es que los desarrolladores se preocupan realmente de la fiabilidad. Los paquetes de software libre no siempre compiten comercialmente, pero sí compiten por una buena reputación y un programa que sea insatisfactorio no alcanzará la popularidad que los desarrolladores esperan. Lo que es más, un autor que pone el código fuente al alcance de la vista de todos arriesga su reputación, y le conviene hacer el software limpio y claro, bajo pena de la desaprobación de la comunidad.

2.4.3 SOFTWARE LIBRE CONTRA PROPIETARIO

A continuación se enumeran las ventajas y desventajas de usar software libre y propietario.

2.4.3.1 VENTAJAS DEL SOFTWARE LIBRE

- Bajo costo de adquisición y libre uso.
- Escrutinio público.
- Independencia del proveedor.
- Adaptación del software.

2.4.3.2 DESVENTAJAS DEL SOFTWARE LIBRE

- La curva de aprendizaje es mayor.
- Carece de garantía proveniente del autor.
- Se necesita dedicar recursos a la reparación de errores.
- Únicamente los proyectos importantes y de trayectoria tienen buen soporte, tanto de los desarrolladores como de los usuarios.
- La diversidad de distribuciones, métodos de empaquetamiento, licencias de uso, herramientas con un mismo fin, etc., pueden crear confusión en cierto número de personas.

2.4.3.3 VENTAJAS DEL SOFTWARE PROPIETARIO

- Control de calidad.
- Recursos a la investigación.
- Uso común por los usuarios.
- Software para aplicaciones muy específicas.

- Amplio campo de expansión de uso en universidades.
- Difusión de publicaciones acerca del uso y aplicación del software.

2.4.3.4 DESVENTAJAS DEL SOFTWARE PROPIETARIO

- Cursos de aprendizaje costosos.
- Secreto del código fuente.
- Soporte técnico ineficiente.
- Ilegal o costosa la adaptación de un módulo del software a necesidades particulares.
- Derecho exclusivo de innovación.
- Ilegalidad de copias sin licencia para el efecto.
- Imposibilidad de compartir.
- Quedar sin soporte técnico.
- Descontinuación de una línea de software.
- Dependencia de proveedores.

2.4.3.5 RESUMEN

A modo de resumen, a continuación se presenta un cuadro comparativo entre software libre y software propietario.

Software	Libre	Propietario
Costo	Ninguno. Uso sin restricción de cantidad de usuarios.	Costo de licencia por usuario.
Uso	Libre	Sujeto a cláusulas.
Adaptación de software	Permitida.	Ilegal o costosa.
Curva de Aprendizaje	Mayor.	Menor.
Código fuente	De libre acceso y modificación.	Acceso restringido. Secreto.
Soporte	Sólo proyectos importantes cuentan con mantenimiento.	Soporte por parte del proveedor, sin embargo, eventualmente este termina.
Mantenimiento	Responsabilidad del usuario. Debe dedicar recursos al mantenimiento.	Brindado por el proveedor.
Calidad	Tiende a ser mayor debido al apoyo de la comunidad.	Debería ser alto, pero termina dependiendo de su servicio de soporte.
Usabilidad	Más difícil de usar.	Su uso tiende a ser más sencillo e intuitivo. En caso no lo sea, los cursos de aprendizaje no son gratuitos.
Distribución	Permitida.	Restringida.
Difusión	No muy conocida por el común de los usuarios.	Ampliamente difundida.

Cuadro 2.1 – Comparación Software Libre y Propietario

CAPÍTULO III

PROCESO DE TOMA DE DECISIONES

3.1 IDENTIFICACIÓN DEL PROBLEMA

Los principales factores de éxito para un proyecto son:

- Cumplir con los requerimientos del usuario: Relacionado directamente a la calidad del software.
- Cumplimiento de los plazos: Un factor importante ya que de este depende la imagen que la empresa obtenga del cliente.
- Eficiencia en los costos: Importante para la empresa, y es dependiente del esfuerzo empleado en el proyecto.

Actualmente la empresa se experimenta sobrecostos, lo cuales se asumen con el objetivo de mantener su compromiso con el cumplimiento de las fechas de entrega.

Este sobrecosto se encuentra relacionado directamente con las siguientes causas:

- Uso de más recursos que los planificados
- Retrasos
- Incremento en el trabajo de cada recurso.
- Proceso de desarrollo no optimizado.

El presente trabajo se centrará en este problema.

3.2 ANÁLISIS DEL PROBLEMA

En función del diagnóstico realizado en el capítulo 1, y tal como se indicó en el apartado 3.1 el problema identificado es el sobrecosto en el que incurre la empresa en la ejecución de sus proyectos de desarrollo.

Para determinar las causas del problema se empleara una herramienta denominada Diagrama de Causa-Efecto. Para este caso las categorías se seleccionaron las distintas fases de un proyecto de desarrollo:

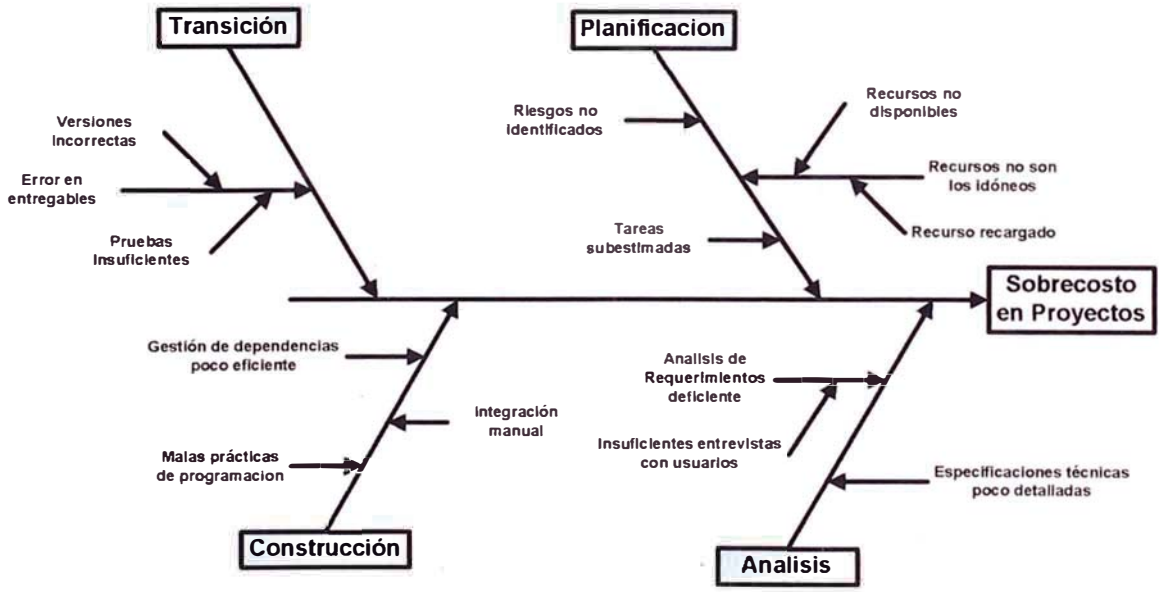


Figura 3.1 Diagrama Causa Efecto para el Sobrecosto en Proyectos

A continuación se detallará cada una de las causas identificadas.

- **Planificación**
    - Riesgos no Identificados: Se trata de riesgos no identificados en el plan de riesgos. Terminan por impactar bastante al proyecto, y este impacto lo asume la empresa.

    - Tareas subestimadas: Tareas que al momento de la ejecución requieren de más tiempo para su culminación. Esto impacta en esfuerzo, lo que conlleva a un incremento en costo.

    - Recursos no son los idóneos: Significa que los recursos asignados no fueron los más adecuados para el proyecto, ya sea falta de capacitación en las tecnologías y/o herramientas a utilizar en el proyecto.
        - Recursos no disponibles: Lo anterior puede ocurrir debido a que los recursos necesarios no se encontraban disponibles por estar asignados a otros proyectos.
        - Recursos sobrecargados: Significa que el recurso asignado está a la vez asignados a otros proyecto, y dependiendo de la fase en la que se encuentre el proyecto el recurso podría dedicar más tiempo a uno que a otro.

- **Análisis**
    - Análisis de Requerimientos incompleto: Los requerimientos identificados y analizados no resultaron ser completos o faltaron requerimientos por identificar. Esta es una de los factores críticos de éxito de un proyecto. El costo del cambio en fases posteriores del proyecto debido a una falla en el análisis de requerimientos es muy alto.
        - Insuficientes entrevistas con los usuarios: No se llevaron a cabo la cantidad necesaria de entrevistas para que se pudieran capturar todos los requerimientos.

- Especificaciones técnicas poco detalladas: Esto provoca que durante la fase de construcción se empleen esfuerzos en obtener estos detalles que pueden ser determinantes para la implementación.

- **Construcción**
  - Malas prácticas de programación: Cada programador tiene su propio estilo para programar, lo que causa problemas al momento de realizar correcciones debido a que no siempre lo termina asumiendo el autor original del código, lo que conlleva a un incremento en el tiempo.

  - Gestión de dependencias poco eficientes: Las soluciones se construyen con componentes o dependencias. Se consume tiempo en escoger y validar las versiones de las dependencias y la distribución de las mismas.

  - Integración manual: La integración del código resulta ser una tarea complicada en función a la cantidad de programadores. El tiempo empleado en esta tarea se incremente de forma exponencial y esta se suele hacer casi al final de la fase.

- **Transición**
  - Error en entregables: Errores detectados en la solución entregada. A más errores, mayor el esfuerzo empleado para corregirlos.

    - Pruebas insuficientes: Haber desarrollado pruebas que no satisfacen todos los casos.
    - Versiones incorrectas: Muchas veces los errores son causados por un error en la versión de uno o más

componentes de la solución final, que muchas veces no se presentan durante la construcción sino hasta que se encuentran en los ambientes de certificación o producción del cliente.

Viendo las causas que provocan el problema se puede observar que aquellas que pertenecen a las categorías de Planificación y Análisis son procedimientos más manuales ya que en su mayoría son de documentación. Otra característica es que para la empresa en estudio, estos errores ocurren con muy poca frecuencia.

Sin embargo por el lado de la Construcción y Transición, se pueden observar oportunidades de mejora de las actividades del proceso de desarrollo de software para poder mitigar o eliminar las causas del problema ya que estas resultan ser más frecuentes, por lo que la solución será orientada a esas fases en particular.

Para detallar el problema, se presenta a continuación el esquema inicial de trabajo.

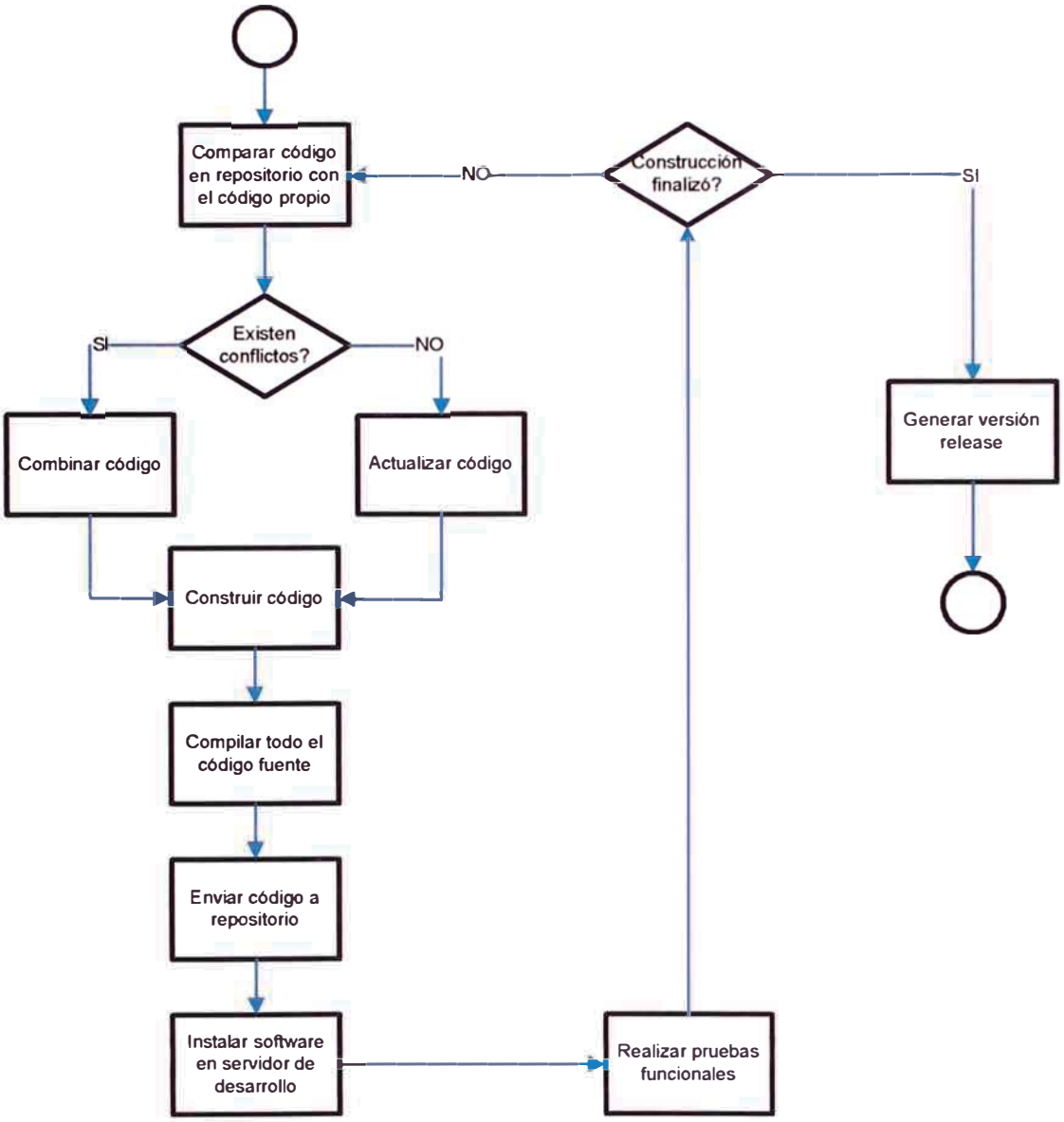


Fig. 3.1 Situación inicial del proceso de desarrollo de software, construcción del código.

Del presente esquema podemos ver que este proceso se repite constantemente. El mismo es realizado por cada programador, quienes suben el código fuente al repositorio de control de versiones cada vez que cada uno crea conveniente. Esto ocasiona que la tarea de Combinar código se convierta en algo completamente dificultoso ya que cuanto mayor sean la cantidad de cambios existentes, mayor es el tiempo que tomará.

Del mismo modo, desde que no se realizan pruebas unitarias, se espera encontrar los errores en código recién cuando se hacen las pruebas

funcionales, lo cual genera sobretiempos en esta ya que primero se debe esperar a que se subsanen los errores de código para recién poder realizar las pruebas funcionales.

Finalmente, para generar la versión release (versión que se entrega al cliente), se requiere de un tiempo adicional para asegurarse que las versiones de código sean las correctas, que las versiones de los componentes de la cual se conforma el software sean las correctas y que todo funcione tal cual validó en las pruebas funcionales. Lo prolongado que puede ser esta tarea termina provocando cierta inseguridad por parte del paquete final.

Y todo esto se encuentra bajo la restricción de tiempo que se tiene en el cronograma. Cuando las fechas de entrega se acercan la presión aumenta y los errores humanos pueden incrementarse.

3.3 PLANTEAMIENTO DE ALTERNATIVAS DE SOLUCIÓN

Con el escenario planteado se pueden proponer las siguientes alternativas de solución.

3.3.1 ADOPTAR LA METODOLOGÍA ÁGIL PROGRAMACIÓN EXTREMA

La metodología mencionada, también conocida como XP, abarca en su totalidad al proceso de desarrollo. Su adopción requiere de un tiempo considerable de pruebas para determinar su adecuación, afecta directamente a la forma de trabajo de todos los miembros del equipo e implica una reducción significativa de la documentación generada. Su ciclo de vida es el iterativo incremental, con iteraciones de corta duración.

Sus características hacen de esta una metodología relativamente apropiada para la empresa.

3.3.2 IMPLEMENTAR UN PROCESO DE INTEGRACIÓN CONTROLADA

Esta propuesta sugiere que para eliminar los problemas incurridos en errores del entregable por temas de integración sean subsanados empleando un proceso de integración controlada, el cual implica el empleo del rol de Integrador quien será el responsable de la integración del producto.

No tiene muchos requerimientos y su implementación es la más simple.

3.3.3 APLICAR UN PROCESO DE INTEGRACIÓN CONTINUA

El proceso de integración continua apoya a las fases de construcción y transición de forma directa.

Las prácticas implicadas agilizan en gran medida a la construcción y da la opción de poder automatizar muchas de las tareas que a un recurso le tomaría un tiempo considerable, al mismo tiempo que reduce la posibilidad de cometer errores humanos. Esto gracias a la adopción de herramientas que permitan gestionar este proceso automáticamente.

Los requisitos para aplicar este proceso resultan beneficiosos e incrementan la calidad del proceso y del producto final.

3.4 CRITERIOS DE DECISIÓN

Se tomarán en consideración los siguientes criterios de decisión:

- Esfuerzo de implementación: Se refiere al esfuerzo necesario para implementar la alternativa seleccionada.

- Complejidad: Implica el grado de complejidad de implementar la alternativa de solución.

- Alcance: Indica el alcance de la solución, en que medida esta consigue eliminar las causas del problema principal.

- Compatibilidad: Indica si la alternativa de solución se alinea a la estrategia principal de la empresa.

3.5 EVALUACIÓN Y SELECCIÓN DE ALTERNATIVA

La evaluación de alternativas se resumirá en el Cuadro 3.1.

	Peso	XP	Integración Controlada	Integración Continua
Tiempo de implementación	**2**	3	6	9
Complejidad	**1**	4	10	7
Alcance	**3**	8	4	4
Compatibilidad	**4**	3	6	10
		4.6	**5.8**	**7.7**

Tabla 3.1 Evaluación de alternativas de solución

Para la primera alternativa, la metodología de Programación Extrema, el costo de implementación es alto debido a que abarca todo el proceso de desarrollo, y la capacitación y cambio en el estilo de trabajo implica un alto costo. Dentro de los beneficios se encuentra en medios debido a que la cantidad de documentación no es el suficiente y aplicaría para muy pocos proyectos. Pero esta alternativa queda descartada debido a que esta no es

compatible, no se encuentra alineada con los propósitos de la empresa, ya que uno de los objetivos estratégicos es el de obtener la certificación CMMi nivel 3, y el nivel de documentación propuesto por esta metodología resulta insuficiente.

La segunda alternativa, la integración controlada, se le considera un costo medio debido a que se necesita emplear recursos adicionales en los proyectos dedicados a la integración del producto. La complejidad de este método resulta ser la más baja debido a que es relativamente simple de implementar. Los beneficios son bajos ya que ayuda a reducir los errores por temas de integración, pero no implica necesariamente un proceso automatizado lo que deja aun el empleo de tiempo para estas tareas. Con respecto a la compatibilidad resulta media debido a que dado que los proyectos tienen un promedio de 5 a 6 personas, resultaría poco probable la asignación del rol Integrador a un recurso ya que esto implicaría sobrecarga de ese recurso o asignar un recurso que se dedique exclusivamente a esa tarea.

La tercera alternativa, la integración continua, si bien el costo se considera medio debido a que poder implementarla correctamente requiere de la adopción de buenas prácticas de desarrollo como lo son la adopción de un estándar de codificación, desarrollo de pruebas unitarias, un servidor de integración continua y un servidor de gestión de dependencias, lo cual agrega complejidad para configurar el ambiente, aunque esto solo debe hacerse una vez por equipo. Los beneficios se consideran medios debido a que solo abarca las fases de construcción y transición del proyecto y la compatibilidad es alta debido a que se ajusta perfectamente a las características de los equipos de proyecto y además el uso de herramientas permite generar información útil que aporta a la documentación requerida por el área de calidad.

Como conclusión la alternativa a elegir es la tercera, la Integración Continua.

3.6 DECISIÓN SOBRE LAS HERRAMIENTAS DE SOFTWARE LIBRE

Dado que se adoptará la Integración Continua, a continuación se hará el proceso de toma de decisiones para las herramientas que se seleccionarán.

3.6.1 CRITERIOS DE DECISIÓN

Estas herramientas serán evaluadas con los siguientes criterios:

- Facilidad de Instalación: Representa que tan sencillo resulta la instalación y configuración inicial de la herramienta.

- Facilidad de Uso: Indica que tan sencillo resulta el aprendizaje del uso de la herramienta.

- Funcionalidad Útil: Indica que tan útiles resultan las funciones ofrecidas por la herramienta.

- Facilidad de Mantenimiento: Indica que tan sencillo es el mantenimiento de la herramienta, como lo es ajuste en las configuraciones o actualización de la versión.

3.6.2 HERRAMIENTA DE GENERACIÓN DE SOFTWARE

3.6.2.1: ALTERNATIVAS DE SOLUCIÓN

Apache Ant: Herramienta que permite la generación de software a través de la preparación de scripts. Sencillo de utilizar, aunque requiere un poco de tiempo para utilizarlo bien.

Apache Maven: Herramienta de generación de software, permite la gestión de dependencias mediante un archivo de configuración dentro del proyecto de desarrollo. Provee de comandos para realizar la compilación, pruebas, empaquetamiento y despliegue del software.

3.6.2.2 Evaluación y Selección de Alternativa

	Peso	Maven	Ant
Facilidad de Instalación	**1**	7	8
Facilidad de Uso	**3**	8	4
Funcionalidad Útil	**4**	10	5
Facilidad de Mantenimiento	**2**	7	10
		8.5	**6**

Tabla 3.2 Evaluación de alternativas de solución – Herramienta de Generación de Software

Se selecciona Apache Maven por la funcionalidad útil que ofrece tanto por la generación de software como para la gestión de dependencias.

3.6.3 Herramienta para la Integración Continua

3.6.3.1: ALTERNATIVAS DE SOLUCIÓN

Hudson CI: Servidor de integración continua. Permite la creación y ejecución de Jobs por cada proyecto. Se conecta al repositorio de control de versiones y puede integrarse con Maven. Cuenta con una interfaz de usuario amigable y es sencillo de configurar.

Continuum: Similar a Hudson CI pero cuenta con muchas otras opciones. Su integración con el repositorio de control de versiones es mejor y permite la generación de versiones mediante la creación de tags en cvs. Requiere de tiempo poder aprender a utilizarlo correctamente y su configuración es más complicada que la alternativa anterior.

3.6.3.2 EVALUACIÓN Y SELECCIÓN DE ALTERNATIVA

	Peso	Hudson CI	Continuum
Facilidad de Instalación	**1**	9	9
Facilidad de Uso	**3**	8	6
Funcionalidad Útil	**4**	8	9
Facilidad de Mantenimiento	**2**	9	6
		8.3	7.5

Tabla 3.3 Evaluación de alternativas de solución – Herramienta para la Integración Continua

Se selecciona Hudson CI por su simplicidad en general.

3.6.4 HERRAMIENTA DE REPOSITORIO DE COMPONENTES

3.6.4.1: ALTERNATIVAS DE SOLUCIÓN

Sonatype Nexus: Tiene la funcionalidad de ser intermediario de repositorios Maven, crear repositorios locales y agruparlos. Utiliza los metadatos generados por Maven para indexar los componentes.

Artifactory: Parecido al anterior, pero ofrece mayor compatibilidad con otros sistemas de integración continua como TeamCity. Accede a los componentes mediante base de datos.

3.6.4.2 EVALUACIÓN Y SELECCIÓN DE ALTERNATIVA

	Peso	Nexus	Artifactory
Facilidad de Instalación	**1**	9	9
Facilidad de Uso	**3**	10	9
Funcionalidad Útil	**4**	8	5
Facilidad de Mantenimiento	**2**	10	10
		9.1	7.6

Tabla 3.4 Evaluación de alternativas de solución – Repositorio de Componentes

Se selecciona Nexus porque su integración con Maven es exclusiva, además que el acceso a los componentes se hace directo al sistema de archivos y no mediante una base de datos.

CAPÍTULO IV

SOLUCIÓN PROPUESTA

En el presente capítulo se describirá a detalle la solución a implementar.
Como se indicó en el capítulo 3, la integración continua se enfoca en el reforzamiento de la fase de construcción del proceso de desarrollo de software.

4.1 DESCRIPCIÓN DE LA SOLUCIÓN

La solución propuesta consiste en la implementación de la Integración continua, para lo cual es necesario lo que se enumera en la presente sección.

4.1.1 NUEVO ESQUEMA DE TRABAJO

La situación propuesta sugiere que la manera en que se trabaja durante la construcción cambie, adoptando la siguiente forma:

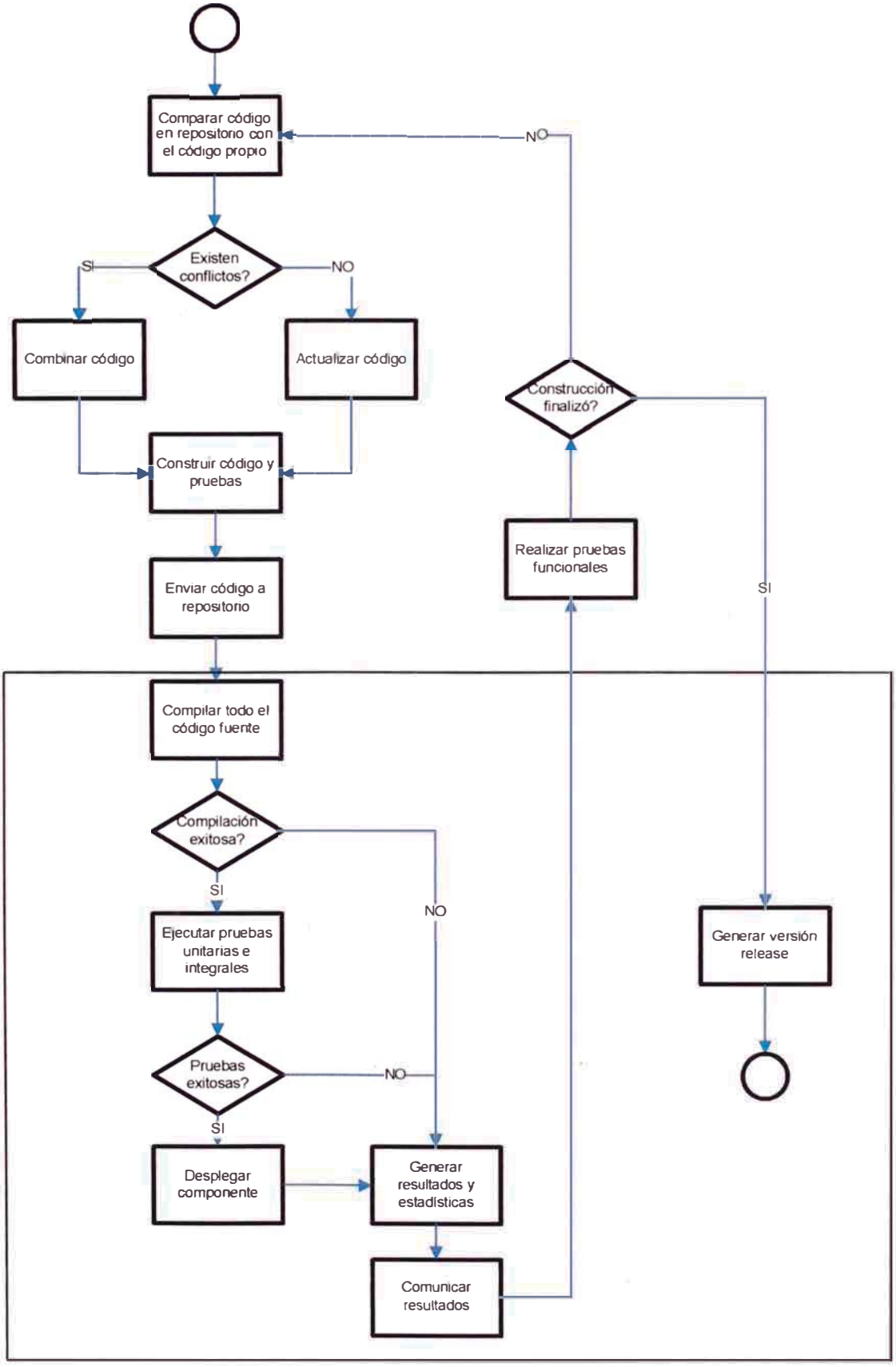


Figura 4.1 Esquema de trabajo para la fase de construcción del desarrollo de software

En la Figura 4.1 se presenta el proceso a seguir para poder seguir un trabajo que permita la integración continua.

Como se puede apreciar, la integración del software da inicio cuando el programador envía su código al repositorio.

Si bien es cierto, para aplicar la integración continua no es realmente necesario contar con herramientas adicionales a un repositorio de control de versiones de código, sin embargo esto haría del proceso algo más prolongado y requeriría de un mayor esfuerzo para obtener los beneficios de la integración continua. Para ello es necesario automatizar este proceso.

Lo enmarcado en el rectángulo representa las actividades que deben ser automatizadas para agilizar el proceso.

4.1.2 INFRAESTRUCTURA DEL ENTORNO DE DESARROLLO

Para implementar un entorno automatizado de integración continua serán necesarios los siguientes nodos:

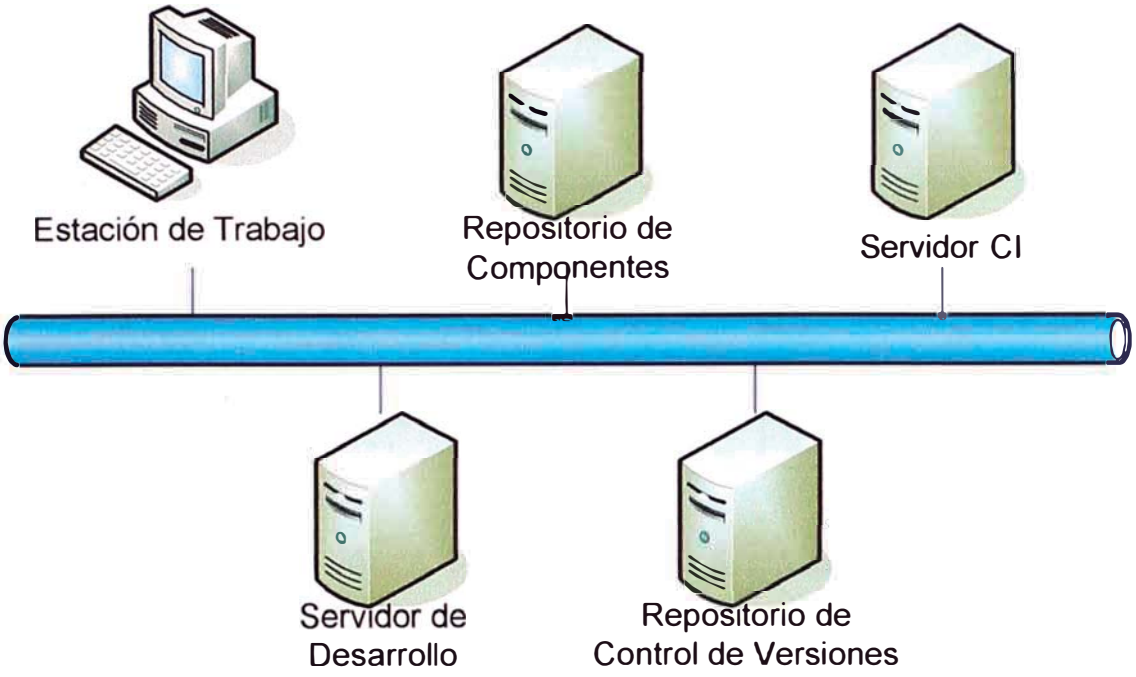


Figura 4.2 Entorno de Desarrollo - Infraestructura

- Repositorio de código fuente: Este es un repositorio que contiene el código fuente de los proyectos. Controla la versión de cada unidad software teniendo un historial por cada una de ellas; también permite agrupar un conjunto de unidades de software con etiquetas (tags) que indican versión de un componente.

- Repositorio de Componentes: Este repositorio almacena los builds generados por el servidor de integración continua, tanto versiones preliminares (conocidas como snapshot) y finales.

- Servidor de integración continua: Es este el servidor que se dedicará a la integración, que consiste en compilar, ejecutar pruebas y desplegar el software en los servidores y/o repositorios que se hayan configurado.

- Servidor de desarrollo: Este servidor es el lugar donde se pueden desplegar los componentes construidos con la finalidad de verificar y/o validar el software.

- Estación de trabajo: Representa el equipo donde el desarrollador realiza la construcción del software basado en la especificación funcional y técnica.

4.1.3 HERRAMIENTAS DE SOFTWARE LIBRE PARA LA AUTOMATIZACIÓN DEL PROCESO

Para implementar la integración continua en el entorno de desarrollo se seleccionaron las siguientes herramientas de software libre:

4.1.3.2 HUDSON CI

Hudson CI, ahora conocido como Jenkins CI, es una aplicación que cumple la función de realizar la integración continua. Esta se encarga de realizar los builds de forma automática y de comunicar al equipo de proyecto del estado de los builds mediante correos electrónicos.

4.1.3.2 APACHE MAVEN

Herramienta que automatiza la generación de software.

Esta herramienta, que se configura en la estación de trabajo del desarrollador, se encarga de gestionar las dependencias de un proyecto,

permite la ejecución de pruebas a nivel local, generar paquetes y hasta realizar el despliegue.
Cabe mencionar que la herramienta Hudson CI se apoya en Maven para la ejecución de estas rutinas de ejecución.

4.1.3.3 SONATYPE NEXUS

Nexus es una aplicación web que sirve de repositorio de componentes el cual ayuda a modo de puerta de enlace con los repositorios locales, para componentes internos generados durante el desarrollo de los proyectos de la empresa, y repositorios externos, aquellos que contienen los componentes (librerías) comunes como aquellos pertenecientes a Apache, Spring, Hibernate, y otros similares.

4.1.3.4 CVS NT

Sistema de control de versiones para el código fuente. En la actualidad, todo entorno de desarrollo cuenta con un sistema de control de versiones ya sea basado en SVN o CVS.
En este caso la empresa ya trabaja con este sistema desde hace dos años.

4.1.3.5 ECLIPSE

Es considerada la IDE para Java más útil por el equipo de trabajo. Cuenta con un cliente de CVS/SVN integrado, así como con un conjunto de plug-ins que ofrecen funcionalidades útiles para la construcción.

4.2 DESARROLLO DE LA SOLUCIÓN

Para la implementación de la solución se eligió la estrategia de implementación por piloto, para el cual se eligió un proyecto inicial donde se hizo uso de las prácticas y herramientas que ha propuesto la solución.

4.2.1 CRONOGRAMA DE PROYECTO

Se consideró los siguientes aspectos:

- La planificación, que tomó en cuenta la selección de herramientas, servidores, buenas prácticas y el proyecto en el cual se aplicaría esta nueva técnica.

- La investigación de las herramientas y prácticas que se pondrían en uso durante el proyecto piloto.

- El acondicionamiento del nuevo entorno de desarrollo, poniendo disponible los servidores y estaciones de trabajo con la instalación de las herramientas.

- Las capacitaciones necesarias a los integrantes del equipo responsables de la construcción del software en el proyecto piloto.

- La ejecución del proyecto piloto.

Cada una de estas acciones se detallan en el siguiente cronograma:

Nombre	Duración
Proyecto Piloto para la Implementación de la Integración Continua	**23.28 días**
Planificación	1.08 días
Lista de herramientas a usar	2 horas
Lista de buenas prácticas de programación aplicables a los proyectos	2 horas
Especificación de Servidores y Requerimientos de Hardware y Red	2 horas
Selección de proyecto piloto	2.4 horas
Reunión con el equipo	2.4 horas
Investigación	1.6 días
Investigar información sobre las herramientas a emplear	1.6 días
Maven	1.6 días
Investigar instalación	4 horas
Investigar uso de herramienta	12 horas
Manual de Instalación	2 horas
Hudson CI	1.6 días
Investigar instalación	4 horas
Investigar uso de herramienta	12 horas
Manual de Instalación	2 horas
Nexus	1.6 días
Investigar instalación	4 horas
Investigar uso de herramienta	12 horas
Manual de Instalación	2 horas
Estándares de codificación y buenas prácticas	1.6 días
Investigar estándares de codificación existentes	16 horas
Investigar sobre buenas prácticas de programación	16 horas
Implementación del Entorno de Desarrollo	1.2 días
Instalación de herramientas	0.6 días
Hudson CI	3 horas
Nexus	3 horas
Maven	6 horas
Configuración	0.9 días
Integrar Hudson CI con CVS NT	4 horas

Integrar Hudson CI con Maven	4 horas
Integrar Maven para que acceda a Nexus	6 horas
Capacitación	5.6 días
Capacitación sobre el estándar de codificación	3.2 horas
Capacitación sobre buenas prácticas	14.4 horas
Capacitación sobre el proceso de construcción a seguir	1.92 días
Capacitación en el uso de herramienta de IC	1.92 días
Ejecución del proyecto piloto – Construcción	60 días

Tabla 4.1 Cronograma de Proyecto

Se estimó un tiempo de 24 días laborales, lo que fue aproximadamente un mes de trabajo. Para más detalles sobre el cronograma referirse al anexo 1.

4.2.2 RECURSOS

Se requirió de 5 personas, las cuales necesitaban las siguientes competencias:

- Conocimiento del inglés a nivel intermedio en lectura.
- Conocimientos programación en Java con experiencia mínima de 1 año en el lenguaje.
- Conocimiento de metodologías ágiles.
- Conocimiento de patrones de diseño.
- Capacidad de investigación.
- Ganas de aprender temas nuevos.

El equipo se formó con 1 arquitecto, 2 analistas técnicos y 2 analistas programadores interesados en el tema.

Para los equipos necesarios, se utilizaron las estaciones de trabajo existentes, donde el equipo del arquitecto sirvió como servidor principal para el sistema de integración continua.

4.2.3 ANÁLISIS DE RIESGOS

Se presenta el análisis de riesgos para este proyecto.

Riesgo	Descripción	Objetivo Impactado	Probabilidad	Estrategia	Acciones
Reasignación de personal	Recursos del proyecto son reasignados por la empresa.	Tiempo	Media	Aceptar	Se reasignaran las actividades y se incrementa el tiempo.
Cambio en políticas de seguridad del cliente	Las políticas de seguridad del cliente establecen restricciones que no permiten la correcta configuración del sistema IC.	Tiempo	Baja	Evitar	Solicitar los permisos al cliente para la comunicación en la red.
Los equipos no alcanzan los requerimientos mínimos	Equipos que actualmente están disponibles no alcancen los requisitos mínimos de hardware.	Costo	Media	Aceptar	Solicitar repotenciación de los equipos que se tienen disponibles.
Cese de soporte de herramientas	Las herramientas seleccionadas dejan de tener soporte de la comunidad.	Costo	Baja	Evitar	Investigar otras herramientas alternativas.
Cancelación de proyecto piloto	El proyecto de desarrollo sobre el cual se está aplicando la IC es cancelado.	Tiempo	Baja	Aceptar	Elegir otro proyecto.

Tabla 4.2 Análisis Cualitativo de Riesgos

4.2.4 EJECUCIÓN DEL PROYECTO

Finalmente se ejecutó el proyecto. Una vez que se tuvo listo el sistema de Integración Continua, se esperó al inicio de la fase de construcción del proyecto piloto para, a su término, evaluar los resultados de aplicar esta práctica.

CAPÍTULO V

EVALUACIÓN DE RESULTADOS

En está sección mencionan los resultados obtenidos en al aplicar la solución planteada en el capítulo anterior.

5.1 CRITERIOS

Los resultados que se evaluarán son los mencionados a continuación.

- Calidad: Indica la mejora de calidad del producto software. En este caso la calidad se representa como cantidad de defectos encontrados durante la validación interna del software. Será medido con la variación de la cantidad de defectos en el producto software, comparando la cantidad de defectos promedio (de proyectos similares), y la cantidad de errores obtenidos en el proyecto piloto.

- Eficiencia: Se evaluará la eficiencia del proceso de desarrollo para generar el producto. Esto se medirá según un índice de variación de esfuerzo, el cual representa la variación del esfuerzo comparando el esfuerzo real y el esfuerzo estimado para el proyecto piloto seleccionado.

- Costo: Se evaluarán los costos de implementación de solución, así como el impacto monetario de la solución implementada, el cual puede ser positivo, negativo o neutral.

5.2 CÁLCULOS Y RESULTADOS

5.2.1 CALIDAD

La calidad se evalúa con un índice de comparación. Este comparará el número de errores promedio en proyectos similares y numero de errores obtenidos en el presente proyecto.

$$C = \frac{DP - D}{DP} \times 100\%$$

Donde:

C	Índice de mejora de la calidad.
DP	Cantidad de defectos promedio de proyectos similares.
D	Cantidad de defectos obtenidos en el proyecto piloto.

Para obtener el valor del DP se consideraron proyectos similares al proyecto piloto, es decir, proyectos con un alcance similar al proyecto en cuestión, que hayan sido trabajados en un tiempo similar y con la misma cantidad de personal asignado.

En este caso la cantidad de errores promedio era de 13 defectos obtenidos durante las pruebas de validación interna.

El proyecto piloto obtuvo 5 defectos durante las pruebas de validación interna.

Por lo que:

$$C' = \frac{13-5}{13} \times 100\% = 61.54\%$$

Se obtuvo un mejora en la calidad del 61.54%. Se redujeron la cantidad de defectos que se suelen obtener en proyectos similares. Esto debido a que las pruebas unitarias e integrales se ejecutan constantemente durante la construcción del software y la capacidad de enterarse tempranamente de los errores, para no esperar a que estos se descubran en fases posteriores como validación con el cliente.

5.2.2 EFICIENCIA

La eficiencia se evaluará en función al esfuerzo empleado en el proyecto, calculado en horas hombre. El indicador será la variación del esfuerzo con respecto a lo planificado.

$$Ef = \frac{EP - ER}{EP} \times 100\%$$

Donde:

- *Ef* Índice de reducción de esfuerzo
- *EP* Esfuerzo planificado para la fase de construcción, en horas hombre.
- *ER* Esfuerzo real empleado en el proyecto durante la fase de construcción.

El esfuerzo planificado se obtiene del cronograma inicial del proyecto. Este esfuerzo se obtiene basándose en el juicio de expertos y el histórico de proyectos para obtener una estimación relativamente precisa. En este caso, el valor de la estimación para la fase de construcción fue de 280 horas programador.

El esfuerzo real se obtuvo del cronograma final del proyecto, el cual indica las horas reales de trabajo por cada programador en el proyecto. En este caso se obtuvo un total de 216 horas hombre.

Con estos datos obtenemos que:

$$Ef = \frac{280 - 216}{280} \times 100\% = 22.86\%$$

Se obtuvo una reducción del esfuerzo en 22.86%, esto debido a la automatización de las tareas repetitivas que comprenden la compilación, ejecución de pruebas, empaquetamiento y despliegue del software generado.

5.2.3 COSTO

El costo se calcula en función al esfuerzo y al costo hora programador para el proyecto. Este costo estimado es calculado para la fase de construcción del software.

Se considera un costo de S/. 15 por hora programador, con lo cual obtenemos lo siguiente

$$CostoPlanificado = 280 \times 15 = S/.4200$$
$$Costo = 216 \times 15 = S/.3240$$
$$Ahorro = 4200 - 3240 = S/.960$$

Obtenemos un ahorro en el proyecto de S/. 960 en la fase de construcción, esto para un proyecto. Este ahorro puede incrementar en función de la cantidad de programadores que participan en el proyecto.

5.2.4 OTROS RESULTADOS

La automatización del proceso para las fases de construcción y transición se puede estimar en un 60%, debido a que las tareas de ejecución de pruebas, inspección, integración y generación de software ahora son de forma automática. A comparación de antes, que todas las tareas se hacían de forma manual.

Los riesgos a que surjan errores inesperados al final de la construcción disminuyeron considerablemente, debido a que la ejecución de la integración continua asegura que el producto software compila, ejecuta las pruebas, y está generado con los componentes de las versiones correctas.

La confiabilidad en el producto software final mucho mayor ahora, aunque difícil de cuantificarlo, es apreciable a opinión del equipo que el producto final es el correcto y no existen dudas de su integridad.

5.2.5 RESUMEN

Se presenta un cuadro con los resultados antes y después de aplicar la solución. El escenario de antes se encuentra basado en valores estimados para el proyecto piloto (estimación de esfuerzo y costo) y a otros datos que

maneja la empresa (cantidad promedio de errores) y directamente el equipo de proyecto.

Criterio	Antes	Después
Cantidad de errores (#)	13	5
Esfuerzo (Horas programador)	280	216
Costo (S/.)	4200	3240
Automatización del proceso	0%	60%
Riesgos de incidencias de errores	Mayor	Menor
Confiabilidad de producto final	Menor	Mayor

Tabla 5.1 Resumen de Resultados

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES:

Se llegaron a las siguientes conclusiones:

- Frente al método tradicional utilizado para la construcción, la integración continua permite un desarrollo más rápido y con menor riesgo de fallos al momento de realizar la integración y entrega del paquete al cliente.

- La elección del uso del software libre nos permite elegir entre varias opciones, permitiéndonos elegir aquella que ofrezca la funcionalidad que más se adapte a nuestra necesidad del proceso, así como aquella que tenga mayor soporte.

- El uso de estas herramientas mejoró en gran medida el desempeño del equipo de desarrollo, dado que no debía preocuparse en ejecutar tareas como ejecución de pruebas e integración de componentes.

- Para una implementación eficaz de la integración continua no basta con solo instalar herramientas y hacerlo andar, aun mas importante lo es entender el proceso y capacitar adecuadamente al equipo en buenas prácticas de proceso y de programación.

- El desarrollo de pruebas unitarias se convierte en piedra angular, gran parte de los beneficios se obtienen de la implementación de casos de prueba que cubran las funcionalidades más importantes como lo son las reglas de negocio.

- Uno de los beneficios más útiles que trae esta práctica es la detección temprana de errores, que permite corregirlos oportunamente, además que permite conocer al responsable del error, y como este conoce los últimos cambios realizados, puede corregirlos mas rápido que si lo hiciera cualquier otro integrante del equipo.

- Con la integración continua se obtiene una mejora en la calidad del producto software debido a la ejecución de las pruebas unitarias e integrales cada vez que se realiza una generación de software.

- Con la implementación de este proceso los desarrolladores tienen más confianza en el producto que entregan al cliente, no hay dudas que el paquete entregado es la versión correcta del software.

- Dentro del proceso de desarrollo existen muchas áreas que requieren atención, la fase de construcción es importante, pero también lo es la etapa de captura y análisis de requerimientos, ya que de estos depende que el producto software final se ajuste a las necesidades del cliente.

RECOMENDACIONES:

Se pueden señalar las siguientes recomendaciones:

- No se debe descuidar la capacitación inicial para nuevos integrantes del equipo de desarrolladores en lo que son las buenas prácticas de programación, de lo contrario esto podría dificultar un poco el mantenimiento del código por parte de los demás integrantes del equipo.

- Realizar mantenimientos periódicos al sistema de integración continua. Estar al tanto cuando cambien las condiciones en el ambiente como direcciones IP de servidores o servidores proxy para la conexión a redes. Se deben actualizar configuraciones cada vez que esto suceda.

- Investigar sobre las innovaciones en el software libre, siempre existe la posibilidad que aparezca en la red una herramienta que resulte mejor que aquellas mencionadas en este informe. Se debe tener presente que primero debe elegirse el software con el cual el equipo se sienta más cómodo.

- Resultaría de especial utilidad aplicar el conocido desarrollo conducido por pruebas (test driven development), debido a su enfoque en las pruebas unitarias.

GLOSARIO DE TÉRMINOS

Generación de software: Se refiere a la acción de compilar, ejecutar pruebas unitarias, empaquetar y desplegar software. Se hace uso de esta expresión en lugar de "software build" en inglés.

Subir código: Se refiere a la acción de resguardar el código fuente realizado por los programadores a un repositorio de control de versiones. Se hace uso de esta expresión en lugar de "commit" en inglés.

Repositorio de Control de Versiones: Se refiere a un equipo que contiene archivos los cuales están controlados por versiones. Esta gestión es realizada por un software de control de versiones, los modelos conocidos actualmente son CVS, SVN y GIT.

Prueba Unitaria: Es una unidad de software especial, cuya función es la de probar la funcionalidad contenida en otra unidad software.

Prueba Integral: Es una prueba funcional que implica todos los componentes del elemento software.

BIBLIOGRAFÍA

BIBLIOGRAFÍA GENERAL

PAUL M. DUVALL 2007	Continuous Integration, Estados Unidos Pearson Education
CULEBRO JUÁREZ, MONZERRAT 2006	Software libre vs Software propietario, México

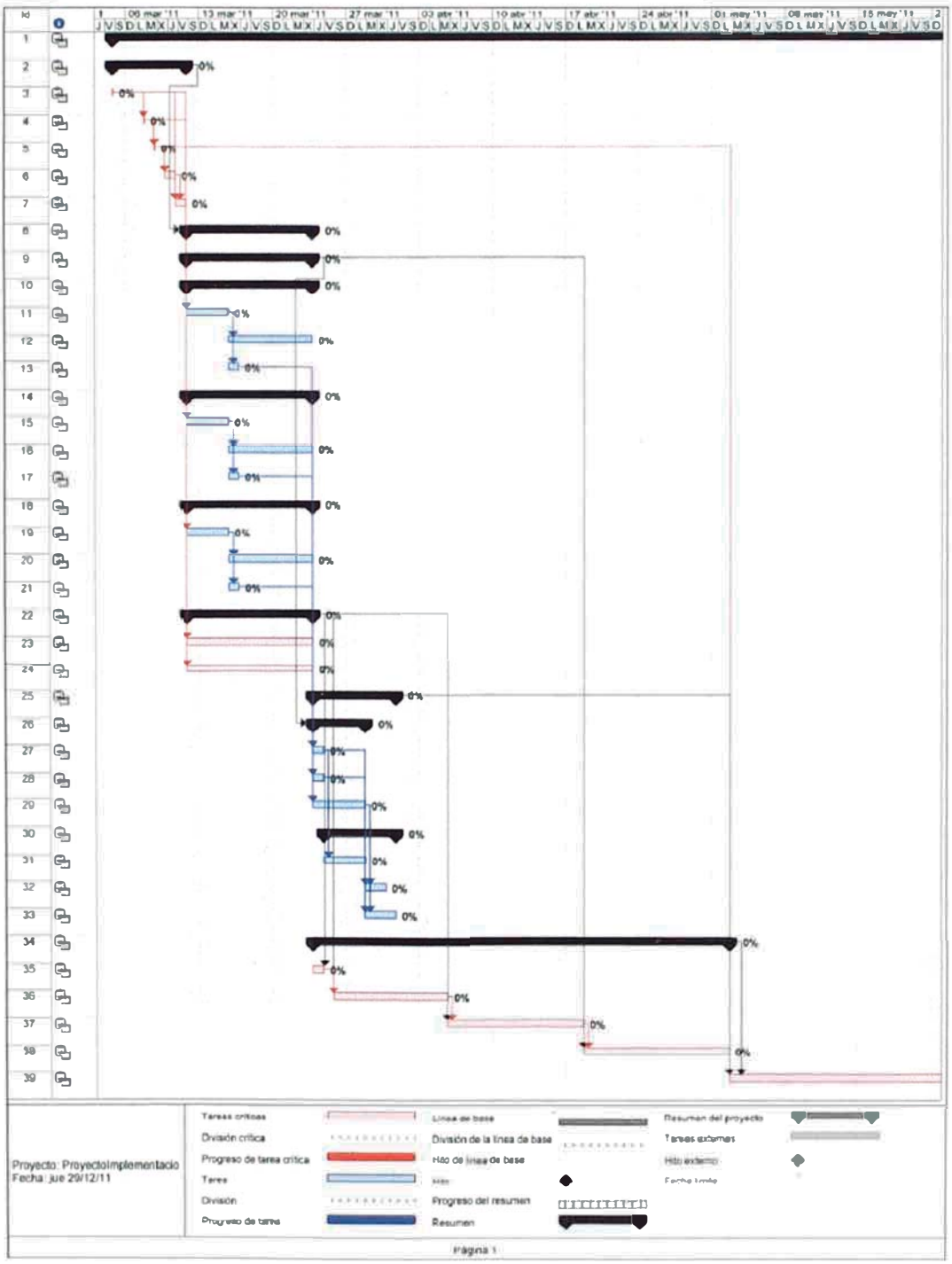
BIBLIOGRAFÍA ELECTRÓNICA

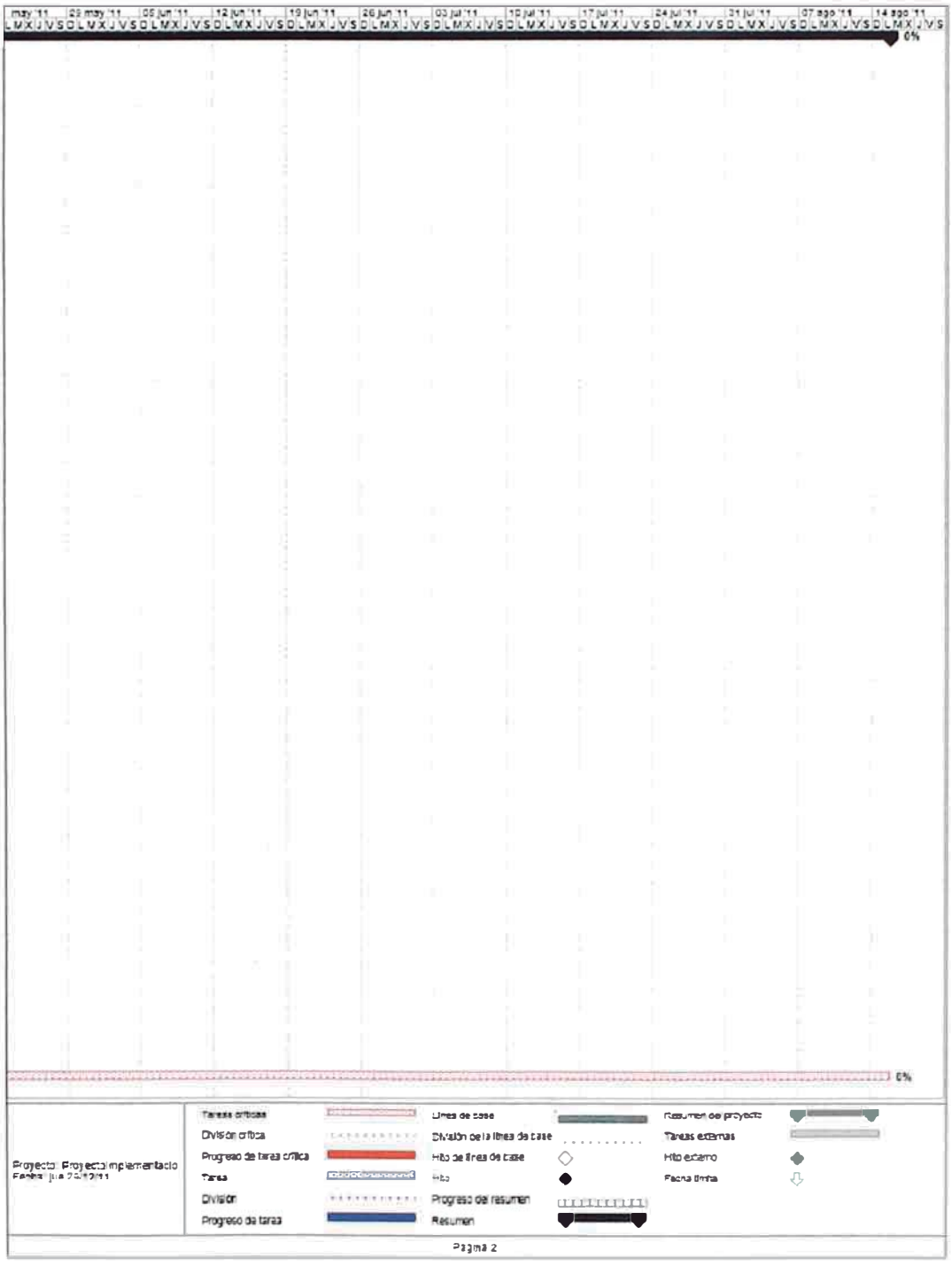
Continuous Integration	http://www.martinfowler.com/articles/continuousIntegration.html
Definición de Software Libre	http://www.gnu.org

ANEXOS

ANEXO 1: Diagrama de Gantt del proyecto de implementación de Integración Continua

Id	Nombre de tarea	Duración	Comienzo
1	**Proyecto Piloto para la Implementación de la Integración Continua**	**23.28 días**	**vie 04/03/11**
2	**Planificacion**	**1.08 días**	**vie 04/03/11**
3	Lista de herramientas a usar	2 horas	vie 04/03/11
4	Lista de buenas prácticas de programación aplicables a los proyectos	2 horas	lun 07/03/11
5	Especificación de Servidores y Requerimientos de Hardware y Red	2 horas	mar 08/03/11
6	Selección de proyecto piloto	2.4 horas	mié 09/03/11
7	Reunion con el equipo	2.4 horas	jue 10/03/11
8	**Investigación**	**1.6 días**	**vie 11/03/11**
9	Investigar información sobre las herramientas a emplear	**1.6 días**	**vie 11/03/11**
10	**Maven**	**1.6 días**	**vie 11/03/11**
11	Investigar instalación	4 horas	vie 11/03/11
12	Investigar uso de herramienta	12 horas	mar 15/03/11
13	Manual de Instalacion	2 horas	mar 15/03/11
14	**Hudson CI**	**1.6 días**	**vie 11/03/11**
15	Investigar instalación	4 horas	vie 11/03/11
16	Investigar uso de herramienta	12 horas	mar 15/03/11
17	Manual de Instalacion	2 horas	mar 15/03/11
18	**Nexus**	**1.6 días**	**vie 11/03/11**
19	Investigar instalación	4 horas	vie 11/03/11
20	Investigar uso de herramienta	12 horas	mar 15/03/11
21	Manual de Instalacion	2 horas	mar 15/03/11
22	**Estándares de codificación y buenas prácticas**	**1.6 días**	**vie 11/03/11**
23	Investigar estándares de codificación existentes	16 horas	vie 11/03/11
24	Investigar sobre buenas practicas de programación	16 horas	vie 11/03/11
25	**Implementacion del Entorno de Desarrollo**	**1.2 días**	**mié 23/03/11**
26	**Instalacion de herramientas**	**0.6 días**	**mié 23/03/11**
27	Hudson CI	3 horas	mié 23/03/11
28	Nexus	3 horas	mié 23/03/11
29	Maven	6 horas	mié 23/03/11
30	**Configuración**	**0.9 días**	**jue 24/03/11**
31	Integrar hudson CI con CVS NT	4 horas	jue 24/03/11
32	Integrar Hudson CI con Maven	4 horas	lun 28/03/11
33	Integrar Maven para que acceda a Nexus	6 horas	lun 28/03/11
34	**Capacitacion**	**5.6 días**	**mié 23/03/11**
35	Capacitación sobre el estándar de codificación	3.2 horas	mié 23/03/11
36	Capacitación sobre buenas prácticas	14.4 horas	vie 25/03/11
37	Capacitación sobre el proceso de construcción a seguir	1.92 días	mar 05/04/11
38	Capacitación en el uso de herramienta de IC	1.92 días	lun 18/04/11
39	Ejecución del proyecto piloto - Construcción	60 días	lun 02/05/11


Proyecto: ProyectoImplementacio
Fecha: jue 29/12/11
Página 1


Tareas críticas
División crítica
Progreso de tarea crítica
Tarea
División
Progreso de tarea
Línea de base
División de la línea de base
Hito de línea de base
Hito
Progreso del resumen
Resumen
Resumen de proyecto
Tareas externas
Hito externo
Fecha límite
Página 2