Universidad de las Ciencias Informáticas
Facultad 2

Sistema para la integración continua de proyectos y el control de *builds* en la empresa Procyon Soluciones.

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

Autor(es): José Antonio Plá Rodríguez.
Damián Ilizastegui Arriba.

Tutor: Lic. Javier Ramón García.
Co-tutor: Lic. Delly lien González Hernández.

Ciudad de la Habana, julio de 2007

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año_____.

_____
Damián Ilizastegui Arriba
(Autor)

_____
José Antonio Plá Rodríguez
(Autor)

_____
Javier Ramón García
(Tutor)

"No vamos a sentarnos y hacer un alto en el camino para pensar cuales son nuestros próximos pasos. Vamos a pensar caminando, vamos a aprender creando y también, por que no decirlo, equivocándonos"

Agradecimientos

A mis padres por el apoyo, el ejemplo y por saberme guiar siempre por el camino correcto. A mis tíos Pura y Mano por ayudarme en todo momento durante estos cinco años. A toda mi familia por confiar en mí y brindarme todo el cariño del mundo.

A mi novia Yanis por su paciencia y su amor.

A mis amigos Mailer y Noslen que los considero mis hermanos y a mis compañeros de estudio por toda la ayuda que me han prestado.

A Héctor y Delly, por el apoyo que me brindaron durante estos años.

A todo el que de una forma u otra ha hecho posible la realización de este sueño.

José Antonio

A mi familia por su apoyo y su amor
A mis compañeros de estudios
A todos los que me han ayudado durante tanto tiempo

Damián

A nuestro Comandante Fidel y a la Revolución Cubana por darnos la oportunidad de convertirnos en profesionales.

Muchas Gracias.

Dedicatoria

A nuestros padres…

Resumen

En este trabajo se presenta la implementación de un sistema de *builds* con el objetivo de automatizar los procesos de compilación, construcción y prueba de los proyectos que se desarrollan por parte de una empresa o entidad. Este sistema es independiente del sistema operativo donde se ejecute y del lenguaje de programación utilizado por los proyectos. Brinda soporte para la notificación a los miembros de los equipos de trabajo a través del envío de correos electrónicos y la extracción del código fuente de los proyectos desde repositorios *Subversion*, facilitando futuras ampliaciones. El sistema cuenta con una interfaz *Web* que brinda información a los usuarios del mismo acerca de los *builds* realizados, ofreciendo también la posibilidad de administrarlo.

Tabla de Contenidos

Introducción

Durante el proceso de desarrollo de un producto de *software* es una práctica común su división en varios módulos para su posterior asignación a uno o más equipos. En estos entornos de trabajo en paralelo se hace necesaria, en algún punto, la integración de todos estos módulos en un solo sistema, que sería el producto terminado. Según Fowler (2006), dejar la integración para el final del proyecto es una práctica riesgosa que puede retardar la fecha de entrega; esto se debe principalmente a que durante la integración se pueden detectar errores de difícil corrección debido a que la mayor parte del código ya está escrita.

La integración continua es una práctica de desarrollo de *software*, originaria de la metodología *Extreme Programming* (XP), donde los miembros de los equipos integran su trabajo frecuentemente (al menos una vez al día). Esta práctica pretende la reducción de los problemas de integración y permite a los equipos desarrollar *software* cohesivo de forma rápida y segura (Fowler, 2006; Kitts, 2004).

Con el objetivo de automatizar el proceso de compilación, construcción y prueba del producto en desarrollo, es común el uso de herramientas, programas, *scripts* que al ser agrupados reciben el nombre de sistema de *builds*. Este tipo de sistema se encarga de la verificación automática de cada integración, con el objetivo de detectar los errores tempranamente e informar de éstos a los desarrolladores, manteniendo al equipo de trabajo al tanto del estado actual del proyecto (Fowler, 2006).

Cuba, en los últimos lustros, ha comenzado a introducirse en el mercado de la industria del *software* a nivel mundial teniendo en cuenta experiencias precedentes en países como la India, Irlanda e Israel (Delgado, Hernán y colaboradores, 2005; TODoS@CICESE, 2001) y la necesidad de ser un país que pueda vivir, cada vez más, de sus producciones intelectuales (Castro, 2003). Con este propósito se han llevado a cabo varias acciones para la formación del capital humano requerido, para la creación de la infraestructura tecnológica necesaria (Valdés, 2007) y han proliferado empresas dedicadas a la producción y comercialización de *software* entre las que se encuentra Procyon Soluciones.

La empresa Procyon Soluciones cuenta, desde sus inicios, con un sistema de *builds* desarrollado en la propia entidad y que continúa en funcionamiento, el cual es conocido como PBS (del inglés: *Procyon*

Builds System). Este sistema está programado en su mayor parte utilizando *scripts* en Perl y Bash y se apoya en servicios *cron*, propios del sistema operativo en el que funciona, imposibilitando que sea multiplataforma y dificultando su escalabilidad. PBS presenta limitaciones en cuanto a su adaptabilidad a otros lenguajes debido a su acoplamiento con los proyectos existentes en este momento. El mantenimiento resulta difícil y propenso a errores, además de necesitar mejoras en la forma en que se almacenan los *builds*.

En este sentido surge la necesidad de reimplementar parte del sistema en un lenguaje de alto nivel y desacoplarlo del entorno de ejecución: tanto del sistema operativo como de la plataforma de desarrollo que usen los proyectos que serán registrados en el mismo, con el objetivo de eliminar sus limitaciones y ampliar sus capacidades.

En tal contexto se plantea la realización de la presente investigación a partir del siguiente problema científico: ¿Cómo resolver las limitantes del sistema de *builds* de la empresa Procyon Soluciones para facilitar el proceso de integración continua?

Para ello se plantea como objetivo general, implementar un sistema de *builds* que permita automatizar y controlar el ciclo de configuración, compilación, empaquetado, pruebas de cada proyecto/producto, conformando los *builds* específicos a cada revisión del código fuente.

Los objetivos específicos son:

- Proveer al sistema de suficiente flexibilidad para lograr su independencia del lenguaje de programación de los proyectos o el sistema operativo en que se ejecute.
- Dotar al mismo de mecanismos de notificación con información acerca del proceso de construcción del *build* (fallos de compilación, pruebas, etc.), utilizando para esto notificaciones por correo electrónico.
- Proveer una interfaz *Web* para brindar toda la información necesaria acerca del proceso de construcción del *build* así como de los componentes generados, las estadísticas almacenadas, los proyectos y la administración del mismo.
- Implementar la lógica del proceso de construcción del *build* en componentes reutilizables para facilitar una futura ampliación del sistema.

Para la realización del trabajo se tiene como objeto de estudio los **Sistemas automatizados para la Integración Continua de proyectos de software** y la **Integración Continua, como práctica en el proceso de desarrollo de software,** como campo de acción.

Dentro de las **tareas** propuestas para la realización de este trabajo se encuentran:

- Estudiar los principales problemas que presenta el sistema actual de la empresa Procyon Soluciones, así como la identificación de posibles mejoras.
- Estudiar y evaluar herramientas existentes para la integración continua de proyectos, con el fin de identificar prácticas aplicables al nuevo sistema.
- Evaluar las diferentes plataformas de desarrollo de *software* y selección de las tecnologías más adecuadas para la implementación del sistema.

El presente trabajo está conformado por cinco capítulos, en los cuales se abordará todo el estudio realizado, así como la descripción de la solución propuesta.

Capítulo 1: Se realiza un análisis del estado actual de la integración continua así como los sistemas de *builds*. Se exponen los requerimientos que debe cumplir el sistema que se adopte en la empresa Procyon Soluciones y se hace un estudio de cuatro sistemas ampliamente utilizados internacionalmente. Se presentan la metodología de desarrollo, plataforma y herramientas utilizadas para la construcción de la solución propuesta.

Capítulo 2: Se enfatiza en las características del sistema a desarrollar, haciendo un análisis de los flujos de trabajo involucrados en el campo de acción y el objeto de automatización de la propuesta.

Capítulo 3: Se detallan los artefactos generados durante las fases de exploración y planificación del proyecto.

Capítulo 4: Se expone todo lo relacionado a los procesos de implementación y pruebas del sistema.

Capítulo 5: Se lleva a cabo un estudio de factibilidad del sistema.

Capítulo I. Fundamentación Teórica

En el presente capítulo se realiza un análisis de la práctica de Integración Continua como parte de la metodología de desarrollo *Extreme Programming* (XP)*,* su aplicación a nivel mundial, en Cuba y en la Universidad de las Ciencias Informáticas (UCI), así como un estudio de los sistemas de *builds* más utilizados. Se abordarán también las tecnologías, herramientas y metodología utilizadas para el desarrollo de la propuesta.

1.1. Integración continua y sistemas de *builds*. Estado del arte

La Integración Continua (CI, del inglés: *Continuous Integration*) surge como una de las principales prácticas de la metodología XP. En esta se plantea que el trabajo de todos los miembros del equipo de desarrollo debe ser integrado frecuentemente, con el fin de detectar errores de integración tan rápido como sea posible y mantener al equipo de trabajo consciente del estado actual del proyecto. Dejar la integración para el final es una práctica riesgosa que puede retardar considerablemente la fecha de entrega del producto, puesto que "(…) la integración es un proceso extenso e impredecible" (Fowler, 2006).

En la actualidad la CI es aplicada por parte de la mayoría de las empresas productoras de *software* a nivel mundial. La introducción de esta práctica generalmente supone un cambio sustancial en cuanto a la forma de trabajo de una organización. En muchas ocasiones requiere de la adquisición de un nivel de disciplina dentro del grupo de desarrollo que se compensa a medidas que el proyecto cobra en amplitud y complejidad (Pérez, 2004).

La aplicación de esta práctica trae consigo una serie de ventajas que propician una mayor calidad en el producto final (Pérez, 2004; Subramaniam, 2004). Estas ventajas son:

- **Minimizar las sesiones de búsqueda de fallos a la hora de integrar el código**. Fallos que eran complicados de encontrar dado que suelen ser efectos colaterales de código que ha sido desarrollado de manera independiente son detectados con mayor facilidad.

- **Permite identificar errores en etapas tempranas**. Esto permite evitar los periodos de integración al final del ciclo de desarrollo del *software*.
- **Aumenta la confianza en el código** escrito por parte de los desarrolladores debido a que ante cada cambio realizado por los mismos, se pueden detectar rápidamente errores de incompatibilidad con lo desarrollado por otros.
- **Se realizan pruebas de unidad inmediatamente después de todos los cambios**. Una vez realizado un cambio en el código del proyecto se procede a la construcción de un nuevo *build* que puede/debe incluir la realización de pruebas de unidad, lo cual permite en todo momento conocer el estado actual del proyecto.
- **Constante disponibilidad de los *builds* realizados**. Esto permite el uso de dichos *builds* con propósitos de prueba, demostración o *release* del producto.

En base a lo anteriormente mencionado y según Pérez (2004) y Subramaniam (2004) se puede afirmar que la CI **permite desarrollar software rápidamente y de manera cohesiva**.

Esta práctica no sería tan popular de ser aplicada manualmente, puesto que conllevaría a un incremento sustancial en el tiempo de desarrollo del proyecto. Sin embargo esto no resulta una preocupación en la actualidad debido a la existencia de sistemas que automatizan esta tarea y que se conocen con el nombre de sistemas de integración continua o sistemas de *builds* (BS, del inglés: *Builds System*).

Hoy existen no menos de 20 BS tanto comerciales como de código abierto y de extenso uso en el desarrollo de proyectos (Codehaus, 2006; Cruise Control, 2007; Luntbuild, 2006; Continuum, 2007; BuildBot, 2007; Bamboo, 2007). Algunos de ellos son multiplataforma, otros utilizan tecnología cliente servidor, pero a pesar de estas diferencias todos tienen como objetivo primordial la automatización del proceso de compilación, construcción y pruebas de los proyectos.

En los últimos años, Cuba se ha introducido paulatinamente en el mercado de la industria del *software* a nivel mundial. Varias acciones así lo demuestran, entre ellas, la creación de programas para facilitar el acceso y formación de recursos humanos en las Tecnologías de la Información y las Telecomunicaciones (TIC) e incrementar la cultura informática de la población; como los Joven Club de Computación y Electrónica en todos los rincones del país, los Institutos Politécnicos de Informática (IPI) en todas las

provincias y el surgimiento de la UCI (Universidad de Ciencias Informáticas) hace unos cinco años con sede central en La Habana y tres facultades regionales recientemente inauguradas.

En este sentido han surgido también empresas y entidades dedicadas a la producción y comercialización de software tanto para clientes extranjeros como para la informatización de entidades y organismos estatales nacionales.

Para la elaboración de la presente investigación, se indagó sobre la utilización de la CI por parte de algunas empresas dedicadas a la producción de software en Cuba mediante la realización de una encuesta enviada por correo electrónico a estos organismos. Ninguno de las entidades en las que se realizo el estudio hacia uso de esta práctica. En algunos casos solo se tiene un control de versiones, pero el proceso de integración se realiza de forma manual. Dentro de los casos estudiados se incluyeron también algunos de los proyectos productivos de la UCI, obteniendo similares resultados.

En el año 2005 surge la empresa Procyon Soluciones con la misión de proveer soluciones y servicios en el mundo de las tecnologías de la información. Desde sus inicios, esta ha contado con un sistema de *builds* desarrollado en la propia entidad y que continúa en funcionamiento, el cual es conocido como PBS (del inglés: *Procyon Builds System*). Este sistema está programado en su mayor parte utilizando *scripts* de Perl y Bash y se apoya en servicios (*cron*) propios del sistema operativo en el que funciona, lo cual imposibilita que sea multiplataforma.

PBS presenta limitaciones en cuanto a su adaptabilidad a otros lenguajes debido a su acoplamiento con los proyectos existentes en este momento. Su mantenimiento resulta difícil y propenso a errores, además de necesitar mejoras en la forma en que se almacenan los *builds* con el fin de economizar espacio en disco.

Para la realización de este trabajo se tomó la decisión por parte de la empresa de implementar un nuevo sistema de *builds* que fuera propio de la entidad y que eliminara las limitantes antes expuestas, con el fin de conocer a plenitud su funcionamiento sin necesidad de realizar un extenso estudio sobre alguno ya existente. El nuevo sistema debe cumplir una serie de requerimientos que son enumerados a continuación:

- Ser **multiplataforma**, para brindar la posibilidad de ejecutarse independientemente del sistema operativo del servidor.
- Lograr una **independencia de la plataforma o lenguaje de programación utilizados para el desarrollo de los proyectos** con el fin de brindarle un mayor nivel de usabilidad al sistema.
- Ser **flexible y configurable** con el objetivo de que pueda adaptarse ante la variedad, en cuanto a características, de los proyectos existentes hoy en la empresa así como los que puedan ser incorporados.
- Contar con un **mecanismo de registro en caliente** que le permita adicionar, editar o eliminar un proyecto mediante la manipulación directa de los archivos de configuración así como el reconocimiento de estos cambios por parte del sistema.
- El sistema debe ser de **fácil instalación, configuración y administración**. El sistema debe ser fácil de utilizar con el objetivo de disminuir el tiempo empleado por los usuarios del mismo en tareas de administración.
- Debe tener una **interfaz *Web* para la muestra de información y reportes** a los usuarios.
- **Manejar estadísticas acerca de los proyectos y los *builds* realizados** para cada uno de estos, que son de interés para la empresa.

1.2. Casos de estudio

Para la realización de este trabajo se llevó a cabo un estudio sobre los sistemas de *build*s más populares a nivel mundial con el propósito de identificar las técnicas y prácticas de las que hacen uso para considerar su implementación en el sistema a desarrollar. A continuación se presentan los casos que fueron seleccionados.

1.2.1. Cruise Control

El *Cruise Control* (Cruise Control, 2007) es el sistema de *builds* más utilizado a nivel mundial, esto se debe a que es un proyecto maduro con más de cinco años de desarrollo y respaldado por una amplia comunidad de usuarios de todas partes del mundo. Es un proyecto de código abierto que se distribuye bajo una licencia al estilo BSD (Watt, 2006).

Este sistema basa su funcionamiento en *plugins* que se le añaden a un núcleo básico, permitiendo que usuarios avanzados puedan crear y registrar sus propios *plugins*. Esta característica lo hace flexible y extremadamente extensible.

La construcción de los proyectos se hace de forma automática y puede ser planificada por el usuario. El sistema no define la lógica de construcción del *build*, sino que esta descansa sobre alguna herramienta de construcción y estará definida en archivos de configuración específicos de acuerdo a la misma. Por defecto el *Cruise Control* brinda soporte para el uso de algunas de las más populares como son Apache Ant (Ant, 2007), Maven (Maven, 2007) y NAnt (NAnt, 2006).

El *Cruise Control* brinda soporte para una gran variedad de sistemas de control de versiones, incluyendo algunos muy populares como son CVS (CVS, 2006), SVN (SVN, 2006) y *Visual Source Safe* (VSS) (VSS, 2007). Soporta además otros medios de almacenamiento como por ejemplo servidores FTP o *Web*.

En cuanto a mecanismos de notificación, el sistema presenta una gran cantidad de *plugins* que le permiten interactuar con una amplia variedad de estos, entre los cuales se encuentran el correo electrónico y algunos protocolos de mensajería instantánea como el Jabber.

La instalación de *Cruise Control* es un proceso trivial y que no requiere de un alto nivel de conocimientos por parte del usuario. Su configuración se basa en un único archivo 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. Esto hace que el nivel de complejidad del archivo crezca a medida que aumenta el número de proyectos y en algunos casos se torne ininteligible y confuso.

Provee una interfaz *Web* que muestra un reporte con los detalles de los *builds* realizados. Este reporte obtiene la información a mostrar de los ficheros de *logs* y archivos generados durante el ciclo de construcción. Esta interfaz brinda escasa información, puesto que solo muestra datos acerca de los últimos *builds* realizados, ignorando cualquier dato acerca del historial de estos para cada uno de los proyectos y del sistema en sí. Como aspecto a destacar se encuentra el que no necesita un servidor *Web* para la interfaz sino que utiliza su propio servidor *Web* Jetty (Jetty, 2003) embebido.

De este sistema resulta interesante el mecanismo de *plugins* con que cuenta, el cual le brinda un nivel de flexibilidad considerable y que resulta una característica a incluir en la solución propuesta. Otro rasgo de interés en este sistema es el uso de un servidor *Web* embebido, lo cual le brinda una mayor portabilidad.

1.2.2. Apache Continuum

Apache Continuum (Continuum, 2007) es un sistema de *builds* dedicado a la construcción de proyectos de software. Es un proyecto relativamente joven con menos de dos años de desarrollo. Es un producto de código abierto y distribuido bajo licencia Apache 2 (Apache, 2004).

Este servidor no se encarga por si mismo de la lógica de construcción de los proyectos sino que brinda soporte para el uso de Ant, Maven 1 y 2 y *Shell Scripts* como herramientas de automatización.

El Continuum brinda soporte solo para cinco sistemas de control de versiones entre los que se destacan SVN y CVS. Para lo cual se auxilia de herramientas y programas que deben ser previamente instalados en el sistema operativo en que funciona. Por defecto el sistema realiza encuestas a los repositorios de código de cada proyecto cada una hora, en busca de modificaciones en el código fuente.

En su versión 1.0.3 brinda soporte para cuatro mecanismos de notificación, tres de ellos son protocolos de mensajería instantánea además del envío de correos electrónicos.

Apache Continuum trae consigo un módulo de administración *Web*, el cual no necesita de ningún servidor para su ejecución sino que posee un servidor *Web* (Jetty) embebido. La instalación, configuración y

administración de proyectos resulta fácil puesto que todas estas acciones son llevadas a cabo mediante esta interfaz.

Del *Continuum* resultó de interés el hecho de que su interfaz *Web* la primera vez que es utilizada realiza la configuración de la cuenta de administración, requiriendo por parte del usuario el nombre de usuario y la contraseña de la misma.

1.2.3. BuildBot

BuildBot (BuildBot, 2007) es un sistema de *builds* que se distribuye bajo licencia GPL (GPL, 2006) y que está programado totalmente en Python lo que le da la posibilidad de ejecutarse en varios sistemas operativos con cambios simples en su configuración.

El *BuildBot* utiliza una arquitectura cliente-servidor, existiendo una sola máquina servidor que controla el proceso y envía comandos a las máquinas clientes mediante conexiones TCP. Son los clientes los que realizan el ciclo de construcción de forma local al recibir las órdenes adecuadas del servidor. Este tipo de arquitectura, le proporciona la posibilidad de construir *builds* simultáneamente en varias plataformas.

Cuenta con un sistema de *plugins* que brinda la posibilidad a los usuarios de adicionar funcionalidades, extendiendo las capacidades iniciales del sistema y adecuándolo a sus necesidades.

Es capaz de interactuar con varios de los sistemas de control de versiones más populares entre los que cabe señalar CVS y SVN.
Cuenta con diversos mecanismos para notificar los resultados del proceso de construcción de los *builds,* entre los que se destacan el envío de correos electrónicos, y el soporte para varios protocolos de mensajería instantánea.

Los archivos de configuración del *BuildBot* son basados en un pseudolenguaje definido por los desarrolladores del sistema en los que se debe especificar mediante una serie de pasos como se va a realizar la construcción del *build*. No existe soporte para ninguna herramienta de construcción específica.

La ejecución de esta, o de cualquier otra herramienta de automatización, debe realizarse mediante comandos ejecutados por el sistema y definidos en los archivos de configuración. Esta característica le brinda gran flexibilidad en cuanto al lenguaje de programación y la forma de compilación de los proyectos pero dificulta en gran medida la configuración y administración del sistema por usuarios noveles.

De este sistema resulta interesante el no brindar soporte para ninguna herramienta de construcción en sí, sino que realiza el proceso de construcción del *build* partiendo de un comando especificado por el usuario, lo cual brinda gran independencia de la plataforma y lenguaje utilizados para la codificación de los proyectos.

1.2.4. Luntbuild

Luntbuild (Luntbuild, 2006) es una herramienta de código abierto bajo licencia de dominio público basada en Apache Ant para la automatización y manejo del proceso de construcción de un proyecto. Es ampliamente conocida debido, entre otras características, a su fácil instalación. Existe también una versión comercial de este proyecto llamada *QuickBuild* (Quickbuild, 2006).

La iniciación del proceso de construcción de los *builds* puede ser planificada por el usuario, permitiéndose también forzarla a través de la interfaz *Web*. El planificador puede ser configurado de dos formas diferentes. Una forma consiste en iniciar el proceso de construcción después de un intervalo de tiempo fijo, la otra forma es planificar el inicio de la construcción utilizando el estilo de las tareas *cron*. Además de esto, el sistema puede efectuar encuestas al repositorio de código cada un tiempo definido para decidir si es necesaria o no la construcción de un nuevo *build*.

Luntbuild no se encarga por sí mismo de la lógica de construcción del proyecto, sino que brinda soporte para Ant, Maven 1 y 2, *Shell Scripts* y Rake (Rake, 2006) como herramientas de construcción.

Brinda soporte para proyectos alojados en el sistema de archivos local además de ocho sistemas de control de versiones, entre los que se incluyen: CVS, VSS y SVN.

Posee varios métodos para notificar los resultados de la construcción de los *builds*, entre los que se destacan: el envío de correos electrónicos además del uso del protocolo de mensajería de Jabber.

Toda la configuración, manejo y administración del sistema se hace mediante una interfaz *Web*, que es ejecutada por el contenedor *Web* (Jetty) que viene embebido en el sistema, aunque brinda la posibilidad de utilizar algún otro servidor.

Este sistema solamente puede ser configurado a través de su interfaz *Web* ya que no posee ningún tipo de archivo de configuración. Toda la información concerniente a los proyectos y al sistema en sí se guarda en una base de datos definida por el usuario o en una embebida dentro del mismo. La configuración y administración del sistema se dificultan en ocasiones debido a la gran flexibilidad con que cuenta el mismo, lo cual muchas veces conlleva a un proceso extenso y tedioso para adicionar un nuevo proyecto.

En el *Luntbuild* no se identificó ninguna práctica nueva que resultara de interés, solo la utilización de algunas de las mencionadas por los anteriores sistemas.

Los autores de este trabajo, luego del proceso de evaluación de estas herramientas, consideran oportuno tomar un conjunto de técnicas comunes en muchas de ellas para su posterior implementación en el sistema que aquí se propone. Estas son:

- Uso de un **sistema de *plugins*** para incrementar la escalabilidad del sistema.
- Uso del **contenedor *Web* Jetty** para independizar al sistema de cualquier servidor, brindando mayor portabilidad.
- La **configuración de la cuenta de administración de la interfaz *Web* la primera vez que se inicia el sistema** para evitar la necesidad de manipular directamente este archivo de configuración.
- **No brindar soporte para ninguna herramienta de automatización**. En lugar de esto, el proceso de construcción del *build* será iniciado por un comando previamente definido con el objetivo de incrementar el nivel de usabilidad del sistema, independizándolo en gran medida de la plataforma y el lenguaje de programación utilizados para la codificación de los proyectos.

1.3. Metodología utilizada para el desarrollo del proyecto

Para la realización del sistema propuesto en este trabajo se decidió utilizar la metodología de desarrollo *Extreme Programming*. XP es una metodología basada en valores de simplicidad, comunicación, retroalimentación y coraje. Funciona mediante la unión de todo el equipo y la aplicación de prácticas simples, con suficiente retroalimentación para permitirle al equipo ver donde están y adaptar sus prácticas a cada situación única (Beck y Fowler, 2000).

1.3.1. ¿Por que XP?

Se decidió utilizar XP debido a que se adapta en gran medida tanto al tipo de proyecto a desarrollar como a las condiciones de trabajo. A continuación se exponen varias de las razones que llevaron al uso de esta metodología.

- **El proyecto es pequeño**. XP está concebida para ser utilizada dentro de proyectos pequeños.
- **No existe un contrato previo especificando tiempo, recursos y alcance**. Para el desarrollo del sistema no se dispone de un contrato con un presupuesto ni un alcance previamente definidos, puesto que es un proyecto para el uso interno de la empresa y será llevado a cabo por programadores pertenecientes a la misma.
- **Los requisitos del sistema cambian frecuentemente**. Con la aceptación de nuevos tipos de proyectos, con estructura y requerimientos disímiles, el sistema debe cambiar y ampliar sus funcionalidades de forma que sea capaz de adaptarse a cada nueva situación. Uno de los principios básicos de XP es que el cambio frecuente de los requerimientos es algo normal en el proceso de desarrollo. Esta metodología se adapta perfectamente a los proyectos cuyos requerimientos cambian a menudo.
- **El cliente forma parte del equipo de desarrollo**. Mediante la aplicación de XP se puede lograr una retroalimentación mayor y lograr un producto que satisfaga sus necesidades.
- El **riesgo de desarrollo** es elevado debido al corto tiempo de entrega planteado y a los continuos cambios de requerimientos. XP está diseñada a mitigar los riesgos en proyectos con estas características.

- **Poca disponibilidad de personal**. El sistema debe ser realizado por dos personas solamente, no siendo posible la existencia de muchos roles ni la especialización en un rol especifico por parte de los miembros. Uno de los principios básicos de XP es la programación en equipos pequeños (2 a 12 personas) con pocos roles, pudiendo los miembros del equipo intercambiar responsabilidades en un momento determinado.
- Algunas **prácticas de XP** como la Integración Continua y el Desarrollo Guiado por Pruebas (TDD, del inglés: *Test Driven Development*) son parte de la política de desarrollo de software de la empresa Procyon Soluciones.
- **Propiedad colectiva del código.** XP plantea que todos los programadores pueden realizar cambios en cualquier parte del código en cualquier momento. En el proceso de desarrollo con que cuenta la empresa esta es una práctica común.
- XP enfatiza la **comunicación de los programadores a través del código**, utilizando líneas directivas para la codificación que están bien establecidas. Desde sus comienzos la empresa cuenta con una línea directiva para la codificación.

1.4. Plataforma de desarrollo

Para el desarrollo de este trabajo se escogió Java como lenguaje de programación y JEE (del inglés: *Java Enterprise Edition*) como plataforma de desarrollo. Esta decisión fue basada en que la empresa Procyon Soluciones tiene como política la utilización de la misma para el desarrollo de sus proyectos, así como la existencia de numerosas APIs (del inglés: *Application Programming Interface*) que facilitan el trabajo de programación, la mayoría de ellas bajo alguna licencia de código abierto.

La plataforma JEE es una tecnología para el diseño de componentes de transacción escalables y basados en servidor: Comprende un conjunto de especificaciones y funcionalidades orientadas al desarrollo de aplicaciones distribuidas (JEE, 2007).

1.5. Herramientas de desarrollo utilizadas

Durante la etapa de implementación del sistema propuesto en este trabajo se utilizó un gran número de herramientas para facilitar esta actividad.

Los Entornos de Desarrollo Integrados (IDE, del inglés: *Integrated Development Environment*) son herramientas que facilitan en gran media el trabajo de los desarrolladores de un proyecto de software. Para la implementación del sistema que se propone se utilizó Eclipse debido a la gran cantidad de prestaciones que brinda así como el grado de familiarización de los programadores con el mismo.

Dentro del grupo de herramientas que se manejaron durante la etapa de construcción del proyecto se pueden destacar el Apache Ant como herramienta de construcción, y SVN para el control de versiones. Otras que resultaron de gran ayuda fueron las APIs de Quartz, como planificador y SVNKit para la interacción con los repositorios de SVN, además del *framework* Struts 2 para el desarrollo de la interfaz Web.

En este capítulo se realizó un análisis de la situación problémica que da origen al presente trabajo así como un estudio de los sistemas de *builds* más utilizados a nivel mundial, con el fin de identificar técnicas utilizadas por estos y que fueran de interés para los autores del presente trabajo.

Capítulo II. Características del Sistema

En el presente capítulo se realiza un análisis de las características del sistema a desarrollar, haciendo hincapié en la situación problémica que da origen al mismo. Se detallan los flujos de trabajo de la empresa Procyon Soluciones comprendidos dentro del campo de acción de la presente investigación y se establece una comparación entre la solución propuesta y otros sistemas existentes en la actualidad.

Procyon Soluciones surge en el año 2005 planteándose como objetivo proveer soluciones y servicios en el mundo de las tecnologías de la información, generando un alto valor y satisfacción a sus clientes. Con el fin de garantizar que sus productos cuenten con el nivel de calidad requerido, mantiene bajo una constante actualización su infraestructura tecnológica. De aquí que surja la implantación de la práctica de integración continua, como parte de la filosofía de trabajo de la entidad.

2.1. Flujo de los procesos involucrados en el campo de acción

Actualmente el proceso de desarrollo de la empresa incluye una serie de flujos y acciones que rigen el comportamiento del mismo. Estos flujos permiten la estandarización del entorno de trabajo para todos los proyectos que se llevan a cabo, y brindan un marco altamente configurable para todos los integrantes de los diferentes equipos de desarrollo. A continuación se muestra un gráfico que representa el entorno de desarrollo de la empresa y se hace una breve descripción de los flujos que intervienen durante la etapa de implementación de un proyecto.

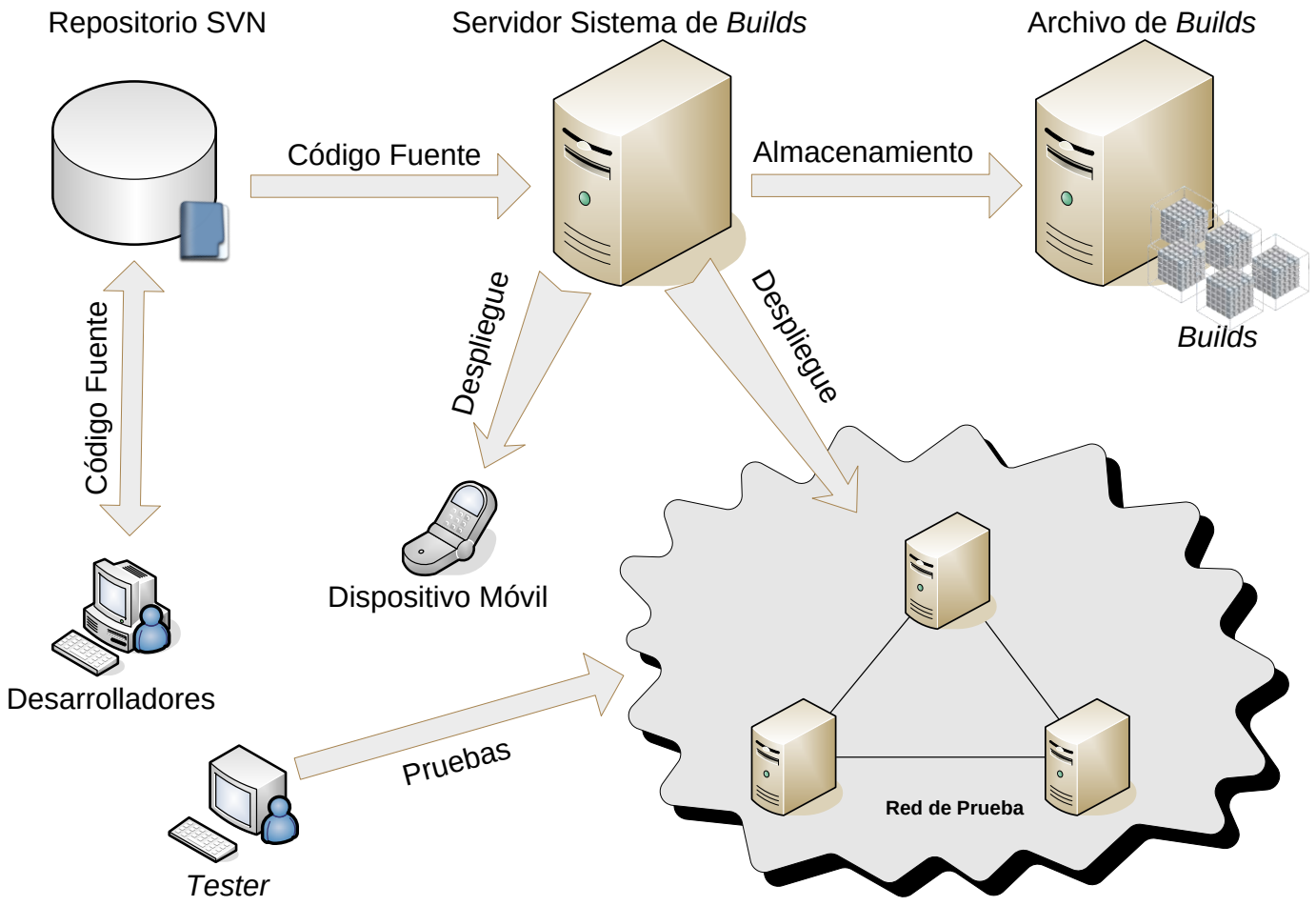

Figura 2.1. Entorno de desarrollo de la empresa Procyon Soluciones.

Durante la etapa de implementación de un proyecto dentro de la empresa, los desarrolladores almacenan todo el código en un repositorio central. Este es monitoreado por el sistema de *builds*, el cual de forma automática, ante la detección de un cambio en las fuentes, o en un momento planificado con anterioridad, procede a la construcción del proyecto.

Una vez concluido este proceso, se procede al almacenamiento del *build*, el cual, en dependencia de las características del proyecto es desplegado en la red de pruebas o los dispositivos móviles. Este proceso se realiza con el objetivo de que los *testers* puedan realizar pruebas sobre el producto en desarrollo para evaluar su desempeño y detectar errores.

2.2. Análisis crítico del funcionamiento del sistema actual

En la actualidad existen numerosos sistemas de *builds* tanto comerciales como de código abierto. Cada uno de ellos tiene sus propias características, pero todos persiguen el objetivo de automatizar el proceso de compilación, empaquetado y pruebas de un proyecto en desarrollo. En el capitulo anterior se realizó un estudio de los BS más utilizados a nivel mundial, haciendo un análisis funcional de cada uno de ellos, con el fin de tomar experiencia de estos para el desarrollo de la propuesta.

Desde su surgimiento, la empresa Procyon Soluciones ha contado con un sistema de *builds* desarrollado por parte de los trabajadores de la entidad, el cual con el transcurso de los años y el incremento del número de proyectos y la complejidad de estos, ha comenzado a presentar dificultades. Este sistema se diseñó para responder a las necesidades inmediatas de la institución. Esta programado en su mayor parte utilizando *scripts* de Perl y Bash y se apoya en servicios *cron*, propios del sistema (Unix) en que funciona, para la planificación del proceso de construcción de los *builds*, característica que imposibilita que sea multiplataforma.

Dicho sistema necesita mejoras en el mecanismo de almacenamiento de los *builds*, pues no hace un uso racional del espacio en disco; para solucionar este problema se brindará la posibilidad de compactar estos últimos. Otra limitante es en cuanto a su adaptabilidad a otros lenguajes, debido en su mayor parte al nivel de acoplamiento que presenta con los actuales proyectos y la estructura de estos dentro del repositorio de código. Todo lo anterior propicia que el mantenimiento resulte difícil y propenso a errores y conllevó a la reimplementación de parte del sistema para mejorar su funcionamiento y facilitar su administración.

2.3. Objeto de automatización

Durante el ciclo de desarrollo de un software existen varios procesos que deben ser automatizados, puesto que su ejecución de forma manual resulta tediosa y propensa a errores, además de consumir una valiosa porción del tiempo de los desarrolladores. Muchos de estos procesos son de vital importancia para lograr un nivel de calidad óptimo en los productos finales. Los BS en particular se encargan de automatizar los procesos que están vinculados con la práctica de integración continua. A continuación se realiza una descripción de los procesos que fueron automatizados con el sistema propuesto, detallando las acciones llevadas a cabo en cada uno.

- **Configuración de ambiente**. Proceso mediante el cual se crea en el servidor de *builds* el ambiente necesario para realizar la construcción del proyecto, este proceso puede incluir la definición de variables de entorno y la instalación automática de programas y librerías.
- **Compilación**. Consiste en la compilación de todo el código fuente del proyecto.
- **Empaquetado**. Durante este proceso se unen varios archivos en un paquete de *software* que puede representar un componente de sistema o el producto completo.
- **Pruebas de unidad *(Unit Test*)**. En caso de que el proyecto tenga pruebas de unidad estas son ejecutadas de forma automática como parte del proceso de construcción del *build*.
- **Almacenamiento**. Una vez concluida la construcción del *build* se archivan todos los artefactos generados y que forman parte de éste conjuntamente con archivos de *logs* e información concerniente al entorno en que se construyó el proyecto.
- **Manejo de estadísticas**. Con la información almacenada referente a cada *build* se generan estadísticas y reportes de interés para la empresa.

2.4. Características de la propuesta de sistema

En el presente trabajo se propone la implementación de un sistema de *builds* que provea las mismas funcionalidades que el actual además de otras que serán añadidas. El mismo estará programado en Java y será independiente tanto del sistema operativo donde se ejecute como del lenguaje y plataforma utilizados para desarrollar los proyectos. Procyon Soluciones es una empresa donde se desarrolla utilizando varias plataformas y lenguajes de programación, es por esto que el sistema debe poseer un alto nivel de configuración, con el objetivo de permitir la fácil extensión del mismo y su adaptabilidad a los diferentes entornos de trabajo.

El sistema propuesto por esta investigación está programado al estilo de un demonio (*daemon*) de los sistemas operativos *Unix* y constará de cinco módulos esenciales que son: planificación, publicación, notificación, interacción con los CVS y la interfaz Web de administración. A continuación se muestra un gráfico sobre la arquitectura del sistema, conjuntamente a con una explicación detallada de cada uno de sus correspondientes módulos.

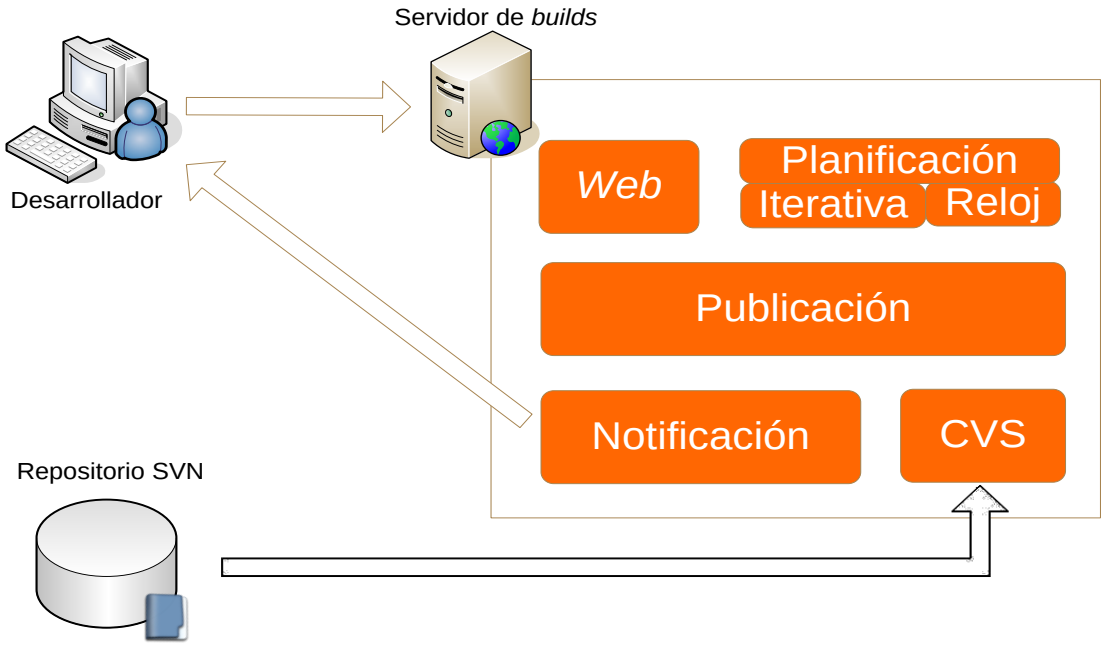

Figura 2.2. Arquitectura del sistema propuesto.

El módulo de planificación: Este módulo es el encargado de planificar la construcción de los *builds.* En su versión inicial el sistema constará con dos planificadores; uno iterativo y otro por reloj. El primero realiza la construcción de los proyectos de forma iterativa, es decir, solo un proyecto a la vez, al llegar al final del listado espera un tiempo (que puede ser configurado) y comienza desde el inicio. Por otro lado, el planificador por reloj permite establecer momentos para la construcción de cada proyecto mediante disparadores. Estos últimos se agrupan dentro de cinco grupos:

- **Disparador por intervalo.** Es utilizado para indicar que la construcción del *build* se realizará cada un intervalo de tiempo y a su vez está formado por tres tipos que son: intervalo en segundos, minutos y horas.
- **Disparador por fecha.** Este tipo de disparador es utilizado para lanzar la construcción de un proyecto en una fecha específica.
- **Disparador diario.** Se utiliza para planificar la construcción de un proyecto diariamente.
- **Disparador semanal.** Se utiliza para planificar la construcción de un proyecto semanalmente.
- **Disparador mensual.** Se utiliza para planificar la construcción de un proyecto mensualmente.

El sistema una vez ejecutado inicia ambos planificadores, permitiendo registrar proyectos en uno u otro y en caso de ser necesario en ambos de ellos. Estos comparten una misma cola, a la que llegan las peticiones de construcción de los proyectos, que pueden ser marcados como exclusivos, lo cual significa que no se puede construir ningún otro de forma paralela al mismo. Esta característica impide que dos proyectos que realicen operaciones mutuamente excluyentes o necesiten gran cantidad de recursos para la creación del *build* sean construidos al mismo tiempo.

El módulo de publicación: Una vez concluida la construcción del *build* se procede al almacenamiento del mismo. Este proceso se realiza mediante el uso de las siguientes políticas: archivar el *build* siempre, en caso de ser satisfactorio o en caso de fallar. Inicialmente el sistema propuesto constará con un *publisher* para manejar el almacenamiento de los *builds* en el sistema de archivos del servidor donde se ejecute.

El módulo de notificación: Con el objetivo de mantener a los miembros de los equipos de desarrollo informados sobre el estado actual del proyecto, el sistema brindará soporte para la notificación a los mismos mediante el envío de correos electrónicos. En el momento en que se va a realizar la notificación se crean los *notifiers* concernientes al proyecto; esta construcción se realiza mediante la utilización del patrón se diseño *Abstract Factory* que, a través de una clase, busca dentro de un archivo de

configuración del sistema el tipo de *notifier* correspondiente al identificador especificado en la configuración del proyecto. Es por esto que en caso de necesitar otro canal de aviso, el usuario puede implementar su propio *notifier* y registrarlo en la configuración del sistema para su posterior uso.

El modulo de interacción con los CVS: Este modulo permite la interacción con los CVS para obtener el código fuente del proyecto a construir. En la primera versión del sistema se brindará soporte para SVN. Mediante el uso del patrón *Abstract Factory* se posibilita la extensibilidad del sistema, brindando la posibilidad al usuario de implementar su propio *Source Extractor*.

La interfaz *Web*: El sistema contará con una interfaz *Web* que tendrá como objetivos principales:

- **Administrar y configurar el sistema.** Desde esta interfaz se pueden manejar los proyectos registrados, permitiendo editar su configuración, eliminarlos o adicionar uno nuevo. El sistema cuenta además de un mecanismo de registro en caliente que le permite adicionar o eliminar un proyecto mediante la manipulación directa de los archivos de configuración de los mismos (eliminando o adicionando una nueva carpeta para el caso de un proyecto nuevo). Otra tarea que puede efectuarse mediante esta interfaz es la configuración de variables del sistema como el tiempo de espera del planificador iterativo.
- **Mostrar información acerca de los *builds* realizados.** El sistema, mediante esta interfaz, muestra a los usuarios un conjunto de informaciones que incluye; un listado de los últimos *builds* realizados para cada proyecto, listado de *builds* por proyecto, información estadística para un proyecto específico (*builds* realizados, *builds* satisfactorios), detalles de un *build* y datos estadísticos acerca del sistema (tiempo de trabajo, iteraciones realizadas por el planificador iterativo) entre otras. Esta interfaz fue creada de forma tal que permita su adaptabilidad ante la adición de nuevas funcionalidades al sistema (nuevos *notifiers* o *source extractors*).

2.5. Análisis comparativo de otras soluciones existentes con la propuesta

Durante la etapa inicial del desarrollo del sistema, se analizaron algunas de las herramientas más utilizadas en el campo de la integración continua. Luego de un estudio de las mismas se tomaron características e ideas que resultaron interesantes con el objetivo de incorporarlas al nuevo PBS. Se

definieron también un conjunto de funcionalidades que no fueron halladas en ninguno de los sistemas estudiados y que resultaban deseables para la propuesta.

Una de las funcionalidades más importantes que debe tener el sistema propuesto es la configuración del ambiente donde se va a construir el *build* debido a que existen grandes diferencias entre los proyectos que desarrolla la empresa y el BS debe ser capaz de adaptar su comportamiento en dependencia del las necesidades de cada uno, por ejemplo, un proyecto de un juego para un dispositivo móvil necesita un ambiente de compilación, ejecución y pruebas muy diferente al que necesita un proyecto desarrollado bajo la plataforma JEE. Otra funcionalidad importante que debe presentar el sistema es la generación de reportes estadísticos que sirven a la empresa para la toma de decisiones. Los sistemas de *build* estudiados se podrían catalogar como “de propósito general”, debido a que ninguno de ellos brinda las funcionalidades antes expuestas.

En el presente capítulo se realizó un análisis sobre los procesos llevados a cabo dentro del entorno de desarrollo de la empresa y que se encuentran vinculados al campo de acción del trabajo, enfatizando en cuales de ellos fueron objeto de automatización, además de la exposición de una descripción detallada sobre las características de la solución propuesta.

Capítulo III. Exploración y Planificación

En el presente capítulo se hace alusión a las fases de exploración y planificación propias de la metodología de desarrollo utilizada para la implementación del sistema que se propone. Se exponen además los artefactos generados durante el transcurso de las mismas.

3.1. Fase de exploración

La metodología de desarrollo *Extreme Programming* comienza con la fase de exploración. Durante esta se realiza el proceso de identificación de las historias de usuario (UH, del inglés: *User Histories*), así como la familiarización de los equipos de trabajo con las tecnologías y herramientas seleccionadas para la construcción del proyecto (Beck, 2000).

3.1.1. Historias de Usuario

Las historias de usuario son la forma en que se especifican en XP los requisitos del sistema. Estas se escriben desde la perspectiva del cliente aunque los desarrolladores pueden brindar también su ayuda en la identificación de las mismas (Beck, 2000). El contenido de estas debe ser concreto y sencillo (XP, 2006). Durante la fase de exploración se identificaron seis UH, las cuales se detallan a continuación.

Tabla 3.1.1. Historia de usuario planificación de *builds* por reloj.

Historia de Usuario	
Número: 1	**Nombre:** Planificación de *builds* por reloj.
Usuario: Administrador	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Medio
Puntos Estimados: 1	**Iteración Asignada:** 1
Descripción: El planificador por reloj posibilitará la iniciación del proceso de construcción de dos formas: cada un intervalo de tiempo o en un momento en especifico. Cada proyecto especifica en su configuración cual de estas formas usar. Se gestiona además el alta y baja de los proyectos en el planificador.	
Observaciones:	

Tabla 3.1.2. Historia de usuario construcción del *build*.

Historia de Usuario	
Número: 2	**Nombre:** Construcción del *build.*
Usuario: Administrador	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alto
Puntos Estimados: 3	**Iteración Asignada:** 1
Descripción: Se efectúa el proceso de construcción del *build* que incluye la creación del ambiente necesario, la obtención del código fuente y la ejecución de la lógica de construcción. Se almacenan los artefactos generados durante este proceso y se notifica a los desarrolladores sobre el resultado del mismo.	
Observaciones: La notificación de los resultados del proceso de construcción del *build* es configurable para cada proyecto y definida por el administrador del sistema.	

Tabla 3.1.3. Historia de usuario planificación iterativa de *builds*.

Historia de Usuario	
Número: 3	**Nombre:** Planificación iterativa de *builds.*
Usuario: Administrador	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Medio
Puntos Estimados: 1	**Iteración Asignada:** 2
Descripción: El planificador iterativo iniciará la construcción de los proyectos uno tras otro, esperando un tiempo (que puede ser configurado) entre el fin de una iteración y el comienzo de la otra. Se gestiona además el alta y baja de los proyectos en el planificador.	
Observaciones:	

Tabla 3.1.4. Historia de usuario gestión de proyectos a construir.

Historia de Usuario	
Número: 4	**Nombre:** Gestión de proyectos a construir.
Usuario: Administrador	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Bajo
Puntos Estimados: 2	**Iteración Asignada:** 2
Descripción: Se gestiona la adición y eliminación de proyectos al sistema, así como la modificación sus datos.	
Observaciones:	

Tabla 3.1.5. Historia de usuario gestión de estadísticas.

Historia de Usuario	
Número: 5	**Nombre:** Gestión de estadísticas
Usuario: Todos	
Prioridad en Negocio: Baja	**Riesgo en Desarrollo:** Bajo
Puntos Estimados: 1	**Iteración Asignada:** 3
Descripción: Generación y administración de los datos estadísticos acerca de los proyectos, del BS y de los *builds* construidos.	
Observaciones:	

Tabla 3.1.6. Historias de usuario creación de la interfaz *Web*.

Historia de Usuario	
Número: 6	**Nombre:** Creación de la interfaz *Web.*
Usuario: Todos	
Prioridad en Negocio: Baja	**Riesgo en Desarrollo:** Bajo
Puntos Estimados: 3	**Iteración Asignada:** 3
Descripción: La interfaz *Web* mostrará los reportes estadísticos generados por el sistema además de brindar la posibilidad de administrarlo y gestionar los proyectos registrados.	
Observaciones: Esta interfaz se ejecutará sobre un servidor *Web* (Jetty) que se encuentra embebido dentro del sistema, por lo que será diseñada para ser ligera, evitando el uso de grandes imágenes o contenido multimedia innecesario.	

3.2. Planificación

Durante la fase de planificación se realiza una estimación del esfuerzo que costará implementar cada historia de usuario. Este se expresa utilizado como medida el punto. Un punto se considera como una semana ideal de trabajo donde los miembros de los equipos de desarrollo trabajan el tiempo planeado sin ningún tipo de interrupción (Beck y Fowler, 2000). Esta estimación incluye todo el esfuerzo asociado a la implementación de la historia de usuario, por ejemplo: las pruebas unitarias, la integración y refactorización del código y la preparación y ejecución de las pruebas de aceptación (Beck, 2000).

3.2.1. Estimación de esfuerzo por Historias de Usuario

Para el desarrollo del sistema propuesto en este trabajo se realizó una estimación del esfuerzo para cada una de las historias de usuario identificadas, llegándose a los resultados que se muestran en la siguiente tabla.

Tabla 3.2. Estimación de esfuerzo por historia de usuario.

Historia de Usuario	Puntos estimados
Planificación de *builds* por reloj	1
Planificación iterativa de *builds*	1
Construcción del *build*	3
Gestión de proyectos a construir	2
Gestión de estadísticas	1
Construcción de la Interfaz *Web*	3

3.2.2. Plan de Iteraciones

Una se vez identificadas las historias de usuario del sistema y estimado el esfuerzo dedicado a la realización de cada una de estas se procede a la planificación de la etapa de implementación del proyecto

(Beck y Fowler, 2000). En base a lo antes mencionado de decide realizar esta en tres iteraciones, las cuales se detallan a continuación.

3.2.3. Iteración 1

Esta iteración tiene como objetivo la implementación de las historias de usuario de mayor prioridad. Durante el transcurso de la misma se creará la base de la arquitectura del sistema con una funcionalidad mínima haciendo mayor énfasis en la planificación y construcción de los *builds*. Al final de esta se contará con una primera versión de prueba, la cual será mostrada al cliente con el objetivo de obtener una retroalimentación para el grupo de trabajo.

3.2.4. Iteración 2

El objetivo de esta es la implementación de las historias de usuario de prioridad media. Al finalizar se contará con una versión de prueba con las funcionalidades concernientes a la gestión de los proyectos y la planificación de los *builds* además de las implementadas en la iteración anterior. Esta será mostrada al cliente con el objetivo de realizar cambios necesarios en base a la opinión del mismo.

3.2.5. Iteración 3

Durante el transcurso de esta se implementaron las historias de usuario de baja prioridad. Al finalizar la misma se constará de la versión 1.0 del producto final, adicionando todo lo concerniente al manejo de estadísticas y la creación de la interfaz *Web* a las funcionalidades anteriores. Como resultado de esta, el sistema será puesto en funcionamiento durante un periodo de tiempo para evaluar su desempeño.

Plan de duración de las iteraciones

Como parte del ciclo de vida de un proyecto utilizando XP se crea el plan de duración de cada una de las iteraciones, en este caso se hace para el único equipo de desarrollo con que se cuenta. Este plan se encarga de mostrar las historias de usuario que serán abordadas en cada una de las iteraciones, así como la duración estimada de estas últimas y el orden en que se implementaran las UH.

Tabla 3.3. Equipo de desarrollo #1

Iteración	Orden de la historias de usuario a implementar	Duración total de la iteración
Iteración 1	1- Planificación de *builds* por reloj. 2- Construcción del *build.*	4 semanas.
Iteración 2	1- Planificación iterativa de *builds.* 2- Gestión de proyectos a construir.	3 semanas.
Iteración 3	1- Gestión de estadísticas. 2- Creación de la Interfaz *Web.*	4 semanas.

3.2.6 Plan de entregas

A continuación se presenta el plan de entregas ideado para la fase de implementación. Como producto del mismo se harán *releases* del sistema al finalizar cada iteración en la fecha aproximada que se indica en la siguiente tabla.

Tabla 3.4. Plan de entregas

Módulo	Final 1ra Iteración 1ra semana de febrero	Final 2da Iteración 1ra semana de marzo	Final 3ra Iteración 1ra semana de abril
pbs-core	**0.1**	**0.2**	**1.0**
pbs-web			**1.0**

Como parte del presente capítulo se abordó todo lo referente a las fases de exploración y planificación del proyecto, haciendo una descripción de cada uno de los artefactos generados durante el transcurso de las mismas.

Capítulo IV. Implementación y Pruebas

XP plantea que la implementación de un *software* debe realizarse de forma iterativa, obteniendo al culminar cada iteración un producto funcional que debe ser probado y mostrado al cliente para retroalimentar a los desarrolladores con la opinión de este. En el presente capítulo se detallan las tres iteraciones llevadas a cabo durante la etapa de construcción del sistema, exponiendo las tareas generadas por cada historia de usuario, así como las pruebas de aceptación efectuadas sobre el proyecto.

Durante el transcurso de las iteraciones se realiza la implementación de las historias de usuario seleccionadas para cada una de estas. Al inicio de las mismas, se lleva a cabo una revisión del plan de iteraciones y se modifica de ser necesario (Beck y Fowler, 2000). Como parte de este plan, se descomponen las UH en tareas de desarrollo, asignando posteriormente cada una de estas a un equipo (o una persona) responsable de su implementación (Beck, 2000). Estas tareas son para el uso de los programadores, pueden escribirse utilizando un lenguaje técnico y no necesariamente deben ser entendibles para el cliente (Beck, 2000).

Ajustándose a la planificación realizada, se llevaron a cabo tres iteraciones de desarrollo sobre el sistema, obteniéndose al finalizar, un producto listo para su puesta en producción. A continuación se detalla cada una de las iteraciones.

4.1. Iteración 1

Durante esta iteración se abordaron las historias de usuario de mayor prioridad y se construyó la base de la arquitectura del sistema con el fin de obtener un producto con las funcionalidades críticas para ser mostrado al cliente y obtener una rápida retroalimentación de este.

Tabla 4.1. Historias abordadas en la primera iteración.

Historia de Usuario	Estimación	Real
Planificación de *builds* por reloj	3	3
Construcción del *build*	1	1
Total	**4**	**4**

4.1.1. Tareas de las historias de usuario abordadas en la iteración

Planificación de *builds* por reloj

Tabla 4.1.1. Tarea #1 de la historia de usuario Planificación de *builds* por reloj.

Tarea	
Número tarea: 1	**Número historia:** 1
Nombre tarea: Creación del planificador por reloj.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.4
Fecha inicio: 10 enero 2007	**Fecha fin:** 12 enero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Se creará un planificador por reloj capaz de iniciar la construcción de los proyectos cada un intervalo de tiempo o en un momento en especifico. El planificador leerá de la configuración del proyecto cual de estas formas utilizar.	

Tabla 4.1.2. Tarea #2 de la historia de usuario Planificación de *builds* por reloj.

Tarea	
Número tarea: 2	**Número historia:** 1
Nombre tarea: Manejo de persistencia del planificador por reloj.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.3
Fecha inicio: 15 enero 2007	**Fecha fin:** 16 enero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Los datos de la planificación del proyecto se almacenarán en archivos de configuración. La configuración del planificador también será almacenada de forma persistente.	

Tabla 4.1.3. Tarea #3 de la historia de usuario Planificación de *builds* por reloj.

Tarea	
Número tarea: 3	**Número historia:** 1
Nombre tarea: Registro de los proyectos en el planificador por reloj.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.3
Fecha inicio: 16 enero 2007	**Fecha fin:** 17 enero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Los proyectos se darán de alta y baja en el planificador, iniciando solamente la construcción de los proyectos que estén registrados en él. Un proyecto puede estar registrado en varios planificadores.	

Construcción del *build*

Tabla 4.1.4. Tarea #1 de la historia de usuario construcción del *build*.

Tarea	
Número tarea: 1	**Número historia:** 2
Nombre tarea: Configuración de ambiente	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.6
Fecha inicio: 18 enero 2007	**Fecha fin:** 22 enero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Se establecerá una configuración de ambiente determinada para construir el proyecto, la cual está formada por variables de ambiente y programas que deben ser instalados en el servidor donde se ejecuta el sistema de *builds*. Las variables de ambiente necesarias se especificaran en archivos tipo *Properties* con estructura llave=valor. Existirán al menos 3 juegos de variables para cada *build*: variables del sistema operativo, variables globales a todo el BS y las variables especificas para proyecto. Se brinda la posibilidad de asignar a una nueva variable el valor de una definida previamente, o la salida de la ejecución de un comando de consola.	

Tabla 4.1.5. Tarea #2 de la historia de usuario construcción del *build*.

Tarea	
Número tarea: 2	**Número historia:** 2
Nombre tarea: Obtención del código fuente.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 23 enero 2007	**Fecha fin:** 30 enero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Se extraerá el código fuente del medio de almacenamiento en que se encuentre. Se crearán mecanismos para extender la cantidad de medios de almacenamientos soportados por el sistema. El sistema contará, por defecto, con soporte para la extracción del código de repositorios de tipo SVN.	

Tabla 4.1.6. Tarea #3 de la historia de usuario construcción del *build*.

Tarea	
Número tarea: 3	**Número historia:** 2
Nombre tarea: Construcción del *build*	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.4
Fecha inicio: 31 enero 2007	**Fecha fin:** 1 febrero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Ejecución del comando para construir el *build* con el ambiente determinado, así como la captura de la salida del mismo para la creación de archivos de *logs*.	

Tabla 4.1.7. Tarea #4 de la historia de usuario construcción del *build*.

Tarea	
Número tarea: 4	**Número historia:** 2
Nombre tarea: Archivar resultados del *build*	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 2 febrero 2007	**Fecha fin:** 6 febrero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Los artefactos (paquetes, archivos, ficheros de *log*, diagramas, librerías, etc.) que se quieran archivar son especificados en un fichero tipo *Properties* que será almacenado conjuntamente con el código fuente del proyecto, todos los artefactos expuestos en este archivo serán archivados en un directorio público dentro del servidor donde se ejecuta el sistema de *builds*.	

Tabla 4.1.8. Tarea #5 de la historia de usuario construcción del *build*.

Tarea	
Número tarea: 5	**Número historia:** 2
Nombre tarea: Notificación de los resultados de un *build*	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 6 febrero 2007	**Fecha fin:** 8 febrero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: El resultado del proceso de construcción de cada *build* podrá será notificado a los desarrolladores según sea configurado. Se crearán mecanismos para ampliar los métodos de notificación soportados. En su versión inicial el sistema utilizara el envío de correos electrónicos.	

4.2. Iteración 2

Durante el transcurso de la presente iteración se concluyó la implementación de las funcionalidades concernientes a la planificación de los *builds*, así como la administración de los proyectos registrados en el sistema.

Tabla 4.2. Historias de la segunda iteración.

Historia de Usuario	Estimación	Real
Planificación iterativa de *builds*	1	1
Gestión de proyectos a construir	2	2
Total	**3**	**3**

4.2.1. Tareas de las historias de usuario abordadas en la iteración

Planificación iterativa de *builds*

Tabla 4.2.1. Tarea #1 de la historia de usuario planificación iterativa de *builds*.

Tarea	
Número tarea: 1	**Número historia:** 3
Nombre tarea: Creación del planificador iterativo.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.4
Fecha inicio: 8 febrero 2007	**Fecha fin:** 12 febrero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Se creará un planificador iterativo que dispare la construcción de los *builds* de los proyectos uno tras otro, esperando un tiempo configurable entre cada iteración.	

Tabla 4.2.2. Tarea #2 de la historia de usuario planificación iterativa de *builds*.

Tarea	
Número tarea: 2	**Número historia:** 3
Nombre tarea: Manejo de persistencia del planificador iterativo.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.3
Fecha inicio: 13 febrero 2007	**Fecha fin:** 14 febrero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Los datos de la planificación del proyecto se almacenarán en archivos de configuración. La información concerniente al planificador también será almacenada de forma persistente.	

Tabla 4.2.3. Tarea #3 de la historia de usuario planificación iterativa de *builds*.

Tarea	
Número tarea: 3	**Número historia:** 3
Nombre tarea: Registro de los proyectos en el planificador iterativo.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.3
Fecha inicio: 14 febrero 2007	**Fecha fin:** 15 febrero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Los proyectos se darán de alta y baja en el planificador, el cual solamente iniciará la construcción de aquellos que estén registrados en él. Un proyecto puede estar registrado en varios planificadores.	

Gestión de proyectos a construir

Tabla 4.2.4. Tarea #1 de la historia de usuario gestión de proyectos a construir.

Tarea	
Número tarea: 1	**Número historia:** 4
Nombre tarea: Creación, modificación y eliminación de proyectos a construir	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 16 febrero 2007	**Fecha fin:** 23 febrero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Administración básica de proyectos que permite la creación, modificación y eliminación de los mismos.	

Tabla 4.2.5. Tarea #2 de la historia de usuario gestión de proyectos a construir.

Tarea	
Número tarea: 2	**Número historia:** 4
Nombre tarea: Manejo de la persistencia de los proyectos	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 24 febrero 2007	**Fecha fin:** 28 febrero 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Se implementará un módulo de persistencia utilizando el sistema de archivos del servidor donde se ejecute el sistema de *builds*. Se utilizarán archivos de tipo *Properties*, con estructura llave=valor, para facilitar la manipulación manual de los mismos.	

Tabla 4.2.6. Tarea #3 de la historia de usuario gestión de proyectos a construir.

Tarea	
Número tarea: 3	**Número historia:** 4
Nombre tarea: Gestión automática de proyectos	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 28 febrero 2007	**Fecha fin:** 2 marzo 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Se implementará un mecanismo para lograr la adición y eliminación automática de los proyectos, mediante la manipulación directa de los archivos de configuración dentro del servidor de donde se ejecute el sistema de *builds.*	

4.3. Iteración 3

En el transcurso de esta iteración se implementaron las historias de usuario restantes que involucraban las funcionalidades concernientes a las estadísticas y a la creación de la interfaz *Web* del sistema. Al culminar esta, se consta de un producto listo para su puesta en funcionamiento.

Tabla 4.3. Historias de la tercera iteración.

Historia de Usuario	Estimación	Real
Gestión de estadísticas	1	1
Creación de la Interfaz *Web*	3	3
Total	**4**	**4**

4.3.1. Tareas de las historias de usuario abordadas en la iteración

Gestión de estadísticas

Tabla 4.3.1. Tarea #1 de la historia de usuario gestión de estadísticas.

Tarea	
Número tarea: 1	**Número historia:** 5
Nombre tarea: Generación de estadísticas	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 5 marzo 2007	**Fecha fin:** 8 marzo 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: El sistema será capaz de generar estadísticas basadas en la información almacenada de todos los *build* construidos.	

Tabla 4.3.2. Tarea #2 de la historia de usuario gestión de estadísticas.

Tarea	
Número tarea: 2	**Número historia:** 5
Nombre tarea: Almacenamiento de estadísticas.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 8 marzo 2007	**Fecha fin:** 12 marzo 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Se almacenarán de forma persistente en el sistema de archivos las estadísticas del sistema.	

Creación de la Interfaz *Web*

Tabla 4.3.3. Tarea #1 de la historia de usuario creación de la interfaz *Web*.

Tarea	
Número tarea: 1	**Número historia:** 6
Nombre tarea: Interfaz de estadísticas.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 13 marzo 2007	**Fecha fin:** 20 marzo 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Creación de una interfaz *Web* simple y amigable para mostrar las estadísticas generadas por el sistema.	

Tabla 4.3.4. Tarea #2 de la historia de usuario creación de la interfaz *Web*.

Tarea	
Número tarea: 2	**Número historia:** 6
Nombre tarea: Interfaz de gestión de proyectos.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 21 marzo 2007	**Fecha fin:** 28 marzo 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Creación de una interfaz *Web* sencilla que permita la creación y eliminación de proyectos así como la modificación de sus datos.	

Tabla 4.3.5. Tarea #3 de la historia de usuario creación de la interfaz *Web*.

Tarea	
Número tarea: 3	**Número historia:** 6
Nombre tarea: Interfaz de administración del sistema	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 29 marzo 2007	**Fecha fin:** 5 abril 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Creación de una interfaz *Web* sencilla que permita realizar operaciones de administración sobre el sistema como: cambios en la configuración de los planificadores, iniciar, detener y reiniciar el sistema.	

Tabla 4.3.6. Tarea #4 de la historia de usuario creación de la interfaz *Web*.

Tarea	
Número tarea: 4	**Número historia:** 6
Nombre tarea: Mecanismo de autenticación.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 29 marzo 2007	**Fecha fin:** 5 abril 2007
Programador responsable: José Antonio Plá Rodríguez – Damián Ilizástegui Arriba	
Descripción: Se creará un mecanismo de autenticación sencillo. Solamente existirá un usuario cuyo nombre y contraseña se establecerá la primera vez que se inicie el sistema. Estos datos se almacenarán en un archivo XML, garantizando la seguridad mediante el uso de md5 para ocultar el texto de la contraseña.	

4.4. Diagramas de clases

Para el diseño de las aplicaciones, la metodología XP no requiere la representación del sistema mediante diagramas de clases utilizando notación UML, en su lugar se usan otras técnicas como las tarjetas CRC (XP, 2006). No obstante el uso de estos puede aplicarse siempre y cuando mejore la comunicación, no sea un peso su mantenimiento, no sean extensos y se enfoquen en la información importante (Fowler, 2004). Los diagramas de clases creados durante el desarrollo del sistema propuesto pueden verse en el Anexo1.

4.5. Pruebas

Uno de los pilares fundamentales de XP es el proceso de pruebas (Beck, 2000), el cual anima a los desarrolladores a probar constantemente y tanto como sea posible. Mediante esta filosofía se reduce el número de errores no detectados así como el tiempo entre la introducción de este en el sistema y su detección (Crispin y House, 2002). Todo esto contribuye a elevar la calidad de los productos desarrollados y la seguridad de los programadores a la hora de introducir cambios y modificaciones.

XP divide las pruebas en dos grupos: pruebas unitarias, desarrolladas por los programadores y encargadas de verificar el código de forma automática y las pruebas de aceptación, destinadas a evaluar

si al final de una iteración se consiguió la funcionalidad requerida además de comprobar que dicha funcionalidad sea la esperada por el cliente (Crispin y House, 2002).

4.6. Pruebas de Aceptación

Las pruebas de aceptación son pruebas de caja negra que se crean a partir de las historias de usuario (Crispin y House, 2002). Durante las iteraciones las UH seleccionadas serán traducidas a pruebas de aceptación. En ellas se especifican, desde la perspectiva del cliente, los escenarios para probar que una UH ha sido implementada correctamente. Una UH puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento. El objetivo final de estas es garantizar que los requerimientos han sido cumplidos y que el sistema es aceptable (XP, 2006). Las pruebas de aceptación realizadas al sistema pueden verse en el Anexo 2.

En el presente capítulo se hace alusión a las etapas de implementación y pruebas del software en desarrollo. Para ello se exponen todos los artefactos generados, realizando una descripción de cada uno de ellos.

Capítulo V. Estudio de Factibilidad

Una de las tareas de mayor importancia en la planificación de proyectos de software es la estimación, la cual consiste en determinar, con cierto grado de certeza, los recursos de hardware y software, costo, tiempo y esfuerzo necesarios para el desarrollo de los mismos. En el presente capítulo se realiza un estudio de factibilidad para la realización del sistema propuesto, haciendo una estimación del esfuerzo necesario para llevar a cabo el mismo.

5.1. Características del proyecto

Tabla 5.1. Entradas externas.

Nombre de la entrada externa	Cantidad de ficheros	Cantidad de elementos de datos	Clasificación (simple, media y compleja)
Adicionar nuevo proyecto	1	7	Simple
Cambiar tiempo de espera del planificador iterativo	1	1	simple
Total		**8**	

Tabla 5.2. Salidas externas.

Nombre de la salida externa	Cantidad de ficheros	Cantidad de elementos de datos	Clasificación (simple, media y compleja)
Últimos *builds* realizados.	1	2	Simple
Detalles de un *build* especifico.	1	2	Simple
Todos los proyectos.	1	4	Simple
Tiempo de espera del planificador iterativo.	1	2	Simple
Estadísticas del sistema.	1	1	Simple
Total		**11**	

Tabla 5.3. Ficheros internos.

Nombre del fichero interno	Cantidad de ficheros	Cantidad de elementos de datos	Clasificación (simple, media y compleja)
Notificadores	1	2	Simple
Planificadores	1	4	Simple
Interacción con CVS	1	2	Simple
Total		**8**	

Tabla 5.4. Peticiones.

Nombre de la petición	Cantidad de ficheros	Cantidad de elementos de datos	Clasificación (simple, media y compleja)
Total		**0**	

Tabla 5.5. Interfaces externas.

Nombre de la interfaz externa	Cantidad de ficheros	Cantidad de elementos de datos	Clasificación (simple, media y compleja)
Total		**0**	

Tabla 5.5. Puntos de función desajustados.

Elementos	Simple		Medio		Complejo		Subtotal
	No.	Peso	No.	Peso	No.	Peso	
Entradas externas	8	3	0	4	0	6	24
Salidas externas	11	4	0	5	0	7	44
Ficheros internos	8	7	0	10	0	15	56
Peticiones	0	3	0	4	0	6	0
Interfaces externas	0	5	0	7	0	10	0
Total							**124**

5.2. Cálculo de instrucciones fuentes, esfuerzo, tiempo de desarrollo, cantidad de hombres y costo

El modelo COCOMO II usa Puntos Función y/o Líneas de Código Fuente (SLOC) como base para medir tamaño en los modelos de estimación de Diseño Temprano y Post-Arquitectura. Los Puntos de Función procuran cuantificar la funcionalidad de un sistema de software. La meta es obtener un número que

caracterice completamente al sistema. Son útiles estimadores ya que están basados en información que está disponible en las etapas tempranas del ciclo de vida del desarrollo de *software*. COCOMO II considera solamente UFP (del inglés: Puntos Función Desajustados).

Tabla 5.6. Características.

Características	Valor
Puntos de función desajustados	124
Lenguaje (Java)	53
Instrucciones fuentes por puntos de función	6572
Instrucciones fuentes	6.57 ksloc

Tabla 5.7. Factores de escala.

Nombre	Valor	Justificación
PREC	1.24	Existen varios proyectos similares a nivel mundial, pero no a nivel nacional.
FLEX	2.03	Cuenta con una alta flexibilidad.
TEAM	2.19	El equipo de desarrollo presenta una alta cohesión.
RESL	4.24	Se identificaron solo dos riesgos críticos.
PMAT	1.56	Existe una experiencia previa en el desarrollo de aplicaciones de este tipo.
Total(SF)	**11.26**	

Tabla 5.8. Multiplicadores de esfuerzo.

Nombre	Valor	Justificación
RCPX	1.10	El sistema presenta un nivel alto de complejidad
RUSE	1.07	El nivel de reutilización es alto.
PDIF	1.00	Uso de memoria y almacenamiento normal, plataforma estable.
PREX	0.90	Alto grado de experiencia en el desarrollo sobre la plataforma y el lenguaje.
PERS	0.88	Alta capacidad del personal.
FCIL	0.82	Se utilizan entornos de desarrollo integrados y herramientas de modelación que facilitan el trabajo.
SCED	1.00	Se empleó el tiempo planificado para el desarrollo del sistema.
Total(EM)	**0.76**	

Cálculos

$A = 2.94 \qquad B = 0.91 \qquad C = 3.67 \qquad D = 0.24$

$E = B + 0.01 * SF = 0.91 + 0.01 * 11.26 = 1.0226$

$PM = A * Size\text{^} E * \prod EM = 2.94 * 6.57\text{^}1.0226 * 0.76 = 15.32$

$F = D + 0.2 * 0.01 * \sum SF = 0.24 + 0.2 * 0.01 * 11.26 = 0.26$

$TDEV = C * PM\text{^} F = 3.67 * 15.32\text{^}0.26 = 7.46$

$CH = \frac{PM}{TDEV} = \frac{15.32}{7.46} \approx 2$

$C = CH * Sal * PM = 2 * 150 * 15.32 = \$4596$

Tabla 5.9. Resultados.

Cálculo de:	Valor
Esfuerzo	15.32 hombres/mes
Tiempo de desarrollo	7.5 meses
Cantidad de hombres	2 hombres
Salario medio	$150
Costo	$4596

5.3. Beneficios tangibles e intangibles

La incorporación de la práctica de integración continua dentro del proceso de desarrollo de una empresa supone una serie de beneficios, entre los que se destaca la reducción a casi nulo del tiempo de integración de los diferentes módulos que conforman un proyecto. Esto se debe a que los errores de integración son detectados rápidamente, posibilitando su corrección de forma inmediata. La utilización de la CI de forma manual no es algo aconsejable, de ahí la importancia que adquieren los sistemas de *builds*.

El desarrollo del producto propuesto en este trabajo brinda como beneficio principal a la empresa Procyon Soluciones la resolución de las limitantes presentadas por el sistema anterior, permitiendo economizar gran parte del tiempo que se dedicaba a las labores de mantenimiento y administración del mismo. PBS

no está concebido para ser un producto comercial, aunque en un futuro pueda reportar beneficios monetarios a la entidad adquiriendo este carácter.

5.4. Análisis de costo

El desarrollo de un producto siempre tiene un costo de producción, el cual debe ser justificado en base a los beneficios reportados por el mismo. El sistema que se propone en este trabajo no conlleva a grandes gastos, puesto que solo es influyente el salario de los desarrolladores, por lo cual se concluye que su implementación es factible. Esto se debe en gran medida a la utilización de plataformas, APIs y herramientas libres que no requieren el pago de alguna licencia.

En el presente capítulo se realizó un análisis de factibilidad de la solución propuesta, arribando a la conclusión de que es viable su desarrollo comparando los costos de producción con los beneficios reportados por su puesta en funcionamiento.

Conclusiones

- La implementación del sistema de *builds* PBS resuelve las limitantes presentadas por el anterior y pone en manos de empresas, entidades y organizaciones dedicadas a la producción de *software* una herramienta que permite automatizar el proceso de compilación, construcción y prueba de los proyectos que se desarrollan.

- En la actualidad es muy bajo el nivel de utilización de sistemas de *builds* por parte de las entidades dedicadas a la producción de *software* en Cuba. En los centros estudiados el proceso de integración se realiza de forma manual y al final de la etapa de implementación del proyecto.

- El sistema desarrollado es independiente del lenguaje de programación utilizado para la implementación de los proyectos registrados en el mismo, así como del sistema operativo del servidor en que se ejecute.

- El sistema propuesto cuenta con un mecanismo de notificación mediante el envío de correos electrónicos a los integrantes de los diferentes grupos de desarrollo, permitiendo su fácil ampliación para brindar soporte a otros canales de comunicación.

- El sistema cuenta con una interfaz *Web* que muestra información referente a los *builds* realizados, además de permitir la administración del mismo.

- El sistema no define la lógica de construcción de los proyectos, permitiendo a los usuarios del mismo apoyarse en herramientas como Apache Ant.

- PBS adiciona funcionalidades que no incorporan otras herramientas estudiadas como es el caso de la creación del entorno necesario para la construcción del proyecto y el manejo de información estadística relacionada a los *builds* de cada proyecto y al sistema en sí.

Recomendaciones

- Adicionar métodos para la notificación a los miembros de los equipos de desarrollo mediante la utilización de canales RSS.

- Desarrollar *plugins* para incrementar las capacidades del sistema.

- Implantar el nuevo sistema en la empresa Procyon Soluciones, los proyectos productivos de la universidad y las entidades dedicadas a la producción de *software* en Cuba.

Bibliografía

Apache (2004). Apache License, Version 2.0. Disponible en [http://www.apache.org/licenses/LICENSE-2.0.html]. [Consultado: 10/enero/2007].

Bamboo (2007). Bamboo — Continuous Integration Server. Disponible en [http://www.atlassian.com/software/bamboo/]. [Consultado: 10/enero/2007].

Beck, K. (2000). Programación Extrema Explicada. Addison Wesley. Título original: Extreme Programming Explained.

Beck K. y Fowler M. (2000). Planeando en Programación Extrema. Addison Wesley. Título original: Planning Extreme Programming.

Belloch, L. (2005). Integración Continua. Disponible en [http://luisbelloch.blogspot.com/2005/01/integracin-contnua.html]. [Consultado: 7/enero/2007].

BuildBot (2007). The BuildBot. Disponible en [http://buildbot.sourceforge.net/]. [Consultado: 10/enero/2007].

Castro, F. (2003). Discurso pronunciado por el Presidente de la República de Cuba, Fidel Castro Ruz, en la clausura del VI Congreso de los CDR, en el teatro "Karl Marx", el 28 de septiembre del 2003. Disponible en [http://www.cuba.cu/gobierno/discursos/2003/esp/f280903e.html]. [Consultado: 8/enero/2007].

Clark, M. (2004). Automatización pragmática de proyectos, cómo construir, desplegar, y supervisar aplicaciones Java. Pragmatic Bookshelf. Título Original: Pragmatic Project Automation, How to Build, Deploy, and Monitor Java Applications.

Codehaus (2006). Continuous Integration Server Feature Matrix. Disponible en [http://damagecontrol.codehaus.org/Continuous+Integration+Server+Feature+Matrix]. [Consultado: 10/enero/2007].

Continuum (2007). Apache Maven Continuum. Disponible en [http://maven.apache.org/continuum/]. [Consultado: 10/enero/2007].

Crispin, L. y House, T. (2002). Probando la Programación Extrema. Addison Wesley. Título original: Testing Extreme Programming.

Cruise Control (2007). CruiseControl Home. Disponible en [http://cruisecontrol.sourceforge.net/]. [Consultado: 10/enero/2007].

CVS (2006). CVS - Concurrent Versions System. Disponible en [http://www.nongnu.org/cvs/]. [Consultado: 10/enero/2007].

Delgado, J., Hernán, G. y colaboradores (2005). Políticas de Promoción de la Industria del Software. Análisis Comparativo de Argentina con las 3I (India, Irlanda, e Israel). Disponible en [http://www.cema.edu.ar/postgrado/download/tesinas2005/MADE_Delgado.pdf]. [Consultado: 8/enero/2007].

Duvall, P. (2006). Automation for the people: Choosing a Continuous Integration server. Disponible en [http://www-128.ibm.com/developerworks/java/library/j-ap09056/]. [Consultado: 3/febrero/2007].

Ferguson, J. (2006). Which open source CI tool is best suited for your application's environment? Disponible en [http://www.javaworld.com/javaworld/jw-11-2006/jw-1101-ci.html?fsrc=rss-index]. [Consultado: 3/febrero/2006].

Fowler, M. (2004). Is Design Dead? Disponible en [http://www.martinfowler.com/articles/designDead.html]. [Consultado: 17/febrero/2007].

Fowler, M. (2005). The New Methodology. Disponible en [http://martinfowler.com/articles/newMethodology.html]. [Consultado: 13/febrero/2007].

Fowler, M. (2006). Continuous Integration. Disponible en [http://martinfowler.com/articles/continuousIntegration.html]. [Consultado: 5/enero/2007].

GPL (2006). GNU General Public License. Disponible en [http://www.gnu.org/copyleft/gpl.html]. [Consultado: 10/enero/2007].

JEE (2007). Java EE at a Glance. Disponible en [http://java.sun.com/javaee/]. [Consultado: 12/febrero/2007].

Jetty (2003). Jetty Java HTTP Server. Disponible en [http://www.mortbay.com/software/Jetty.html]. [Consultado: 10/enero/2007].

Kitts, R. (2004). Continuous Integration Hell. Disponible en [http://www.artima.com/weblogs/viewpost.jsp?thread=30031]. [Consultado: 5/enero/2007].

Luntbuild (2006). Luntbuild - automate and manage your builds. Disponible en [http://luntbuild.javaforge.com/]. [Consultado: 10/enero/2007].

Maven (2007). Apache Maven Project. Disponible en [http://maven.apache.org/]. [Consultado: 10/enero/2007].

NAnt (2006). NAnt: A .NET Build Tool. Disponible en [http://nant.sourceforge.net/]. [Consultado: 10/enero/2007].

Pérez, J. (2004). Integración Continua utilizando herramientas Open Source. Disponible en [http://javahispano.net/frs/download.php/125/CasoDeUsoEmpleoDeTecnologiasJ2EE.pdf]. [Consultado: 8/enero/2007].

Quickbuild (2006). Quickbuild - automate and manage your builds. Disponible en [http://www.pmease.com/app.do]. [Consultado: 10/enero/2007].

Rake (2006). RAKE — Ruby Make. Disponible en [http://rake.rubyforge.org/]. [Consultado: 10/enero/2007].

Spolsky, J. (2000). The Joel Test: 12 Steps to Better Code. Disponible en [http://joelonsoftware.com/articles/fog0000000043.html]. [Consultado: 12/febrero/2007].

Spolsky, J. (2001). Daily Builds Are Your Friend. Disponible en [http://joelonsoftware.com/articles/fog0000000023.html]. [Consultado: 10/febrero/2007].

Subramaniam, V. (2004). Test Driven Development – Part III: Continuous Integration. Disponible en [http://www.agiledeveloper.com/articles/TDDPartIII.pdf]. [Consultado: 3/marzo/2007].

SVN (2006). Tigris.org. Open Source Software Engineering Tools. Disponible en [http://subversion.tigris.org]. [Consultado: 10/enero/2007].

TODoS@CICESE (2001). Crónica de un viaje a India, país líder en TI. Disponible en [http://gaceta.cicese.mx/ver.php?topico=breviario&ejemplar=114&id=126]. [Consultado: 8/enero/2007].

Valdés, R. (2007). Discurso pronunciado por el Comandante de la Revolución, Ramiro Valdés Menéndez, Ministro de la Informática y las Comunicaciones en el Acto Inaugural de la XII Convención y Expo Internacional, Informática 2007. Disponible en [http://www.cubaminrex.cu/Sociedad_Informacion/2007/DiscursoRamiro.htm]. [Consultado: 8/enero/2007].

VSS (2007). Visual SourceSafe 2005. Disponible en [http://msdn2.microsoft.com/en-us/vstudio/aa718670.aspx]. [Consultado: 10/enero/2007].

Watt, A. (2006). BSD License. Disponible en [http://www.istumbler.net/license.txt]. [Consultado: 10/enero/2007].

XP (2006). Extreme Programming: A gentle introduction. Disponible en [http://www.extremeprogramming.org/]. [Consultado: 17/febrero/2007].

Anexo I. Diagramas de Clases del Sistema

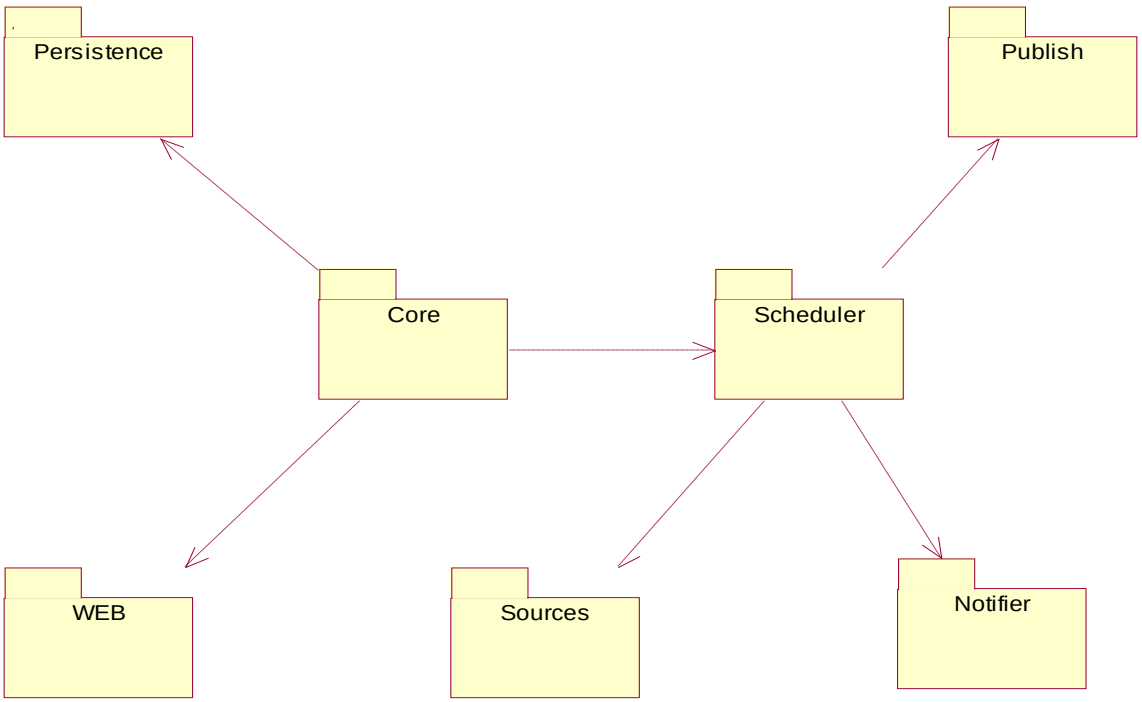

Figura A.1. Paquetes del sistema

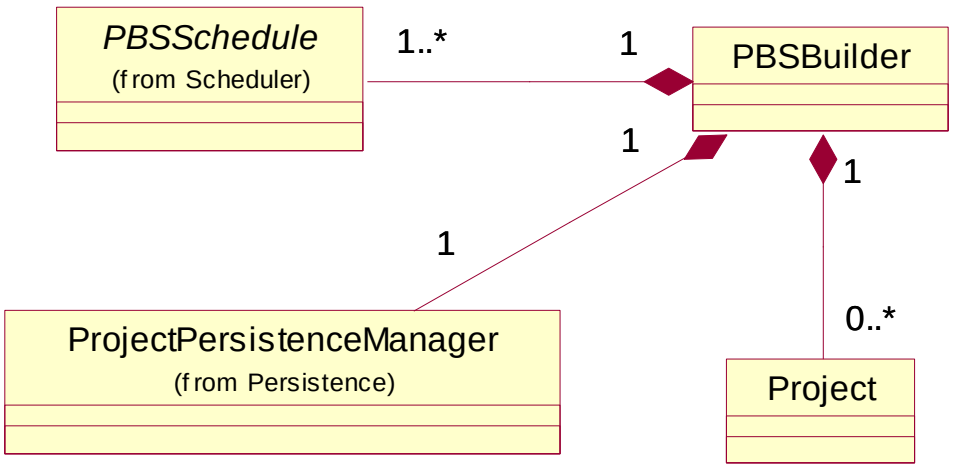

Figura A.2. Diagrama de clases del paquete *Core*.

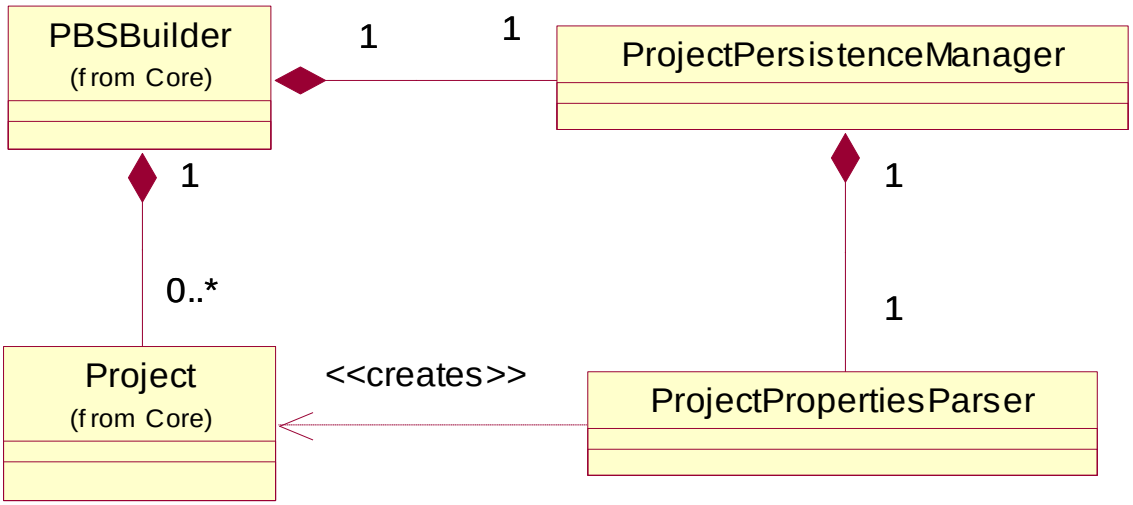

Figura A.3. Diagrama de clases del paquete *Persistence*.

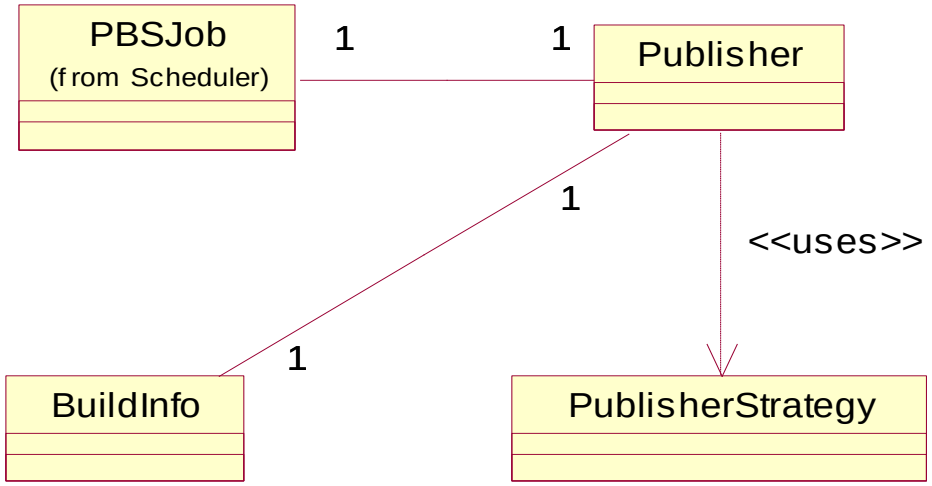

Figura A.4. Diagrama de clases del paquete *Publish*.

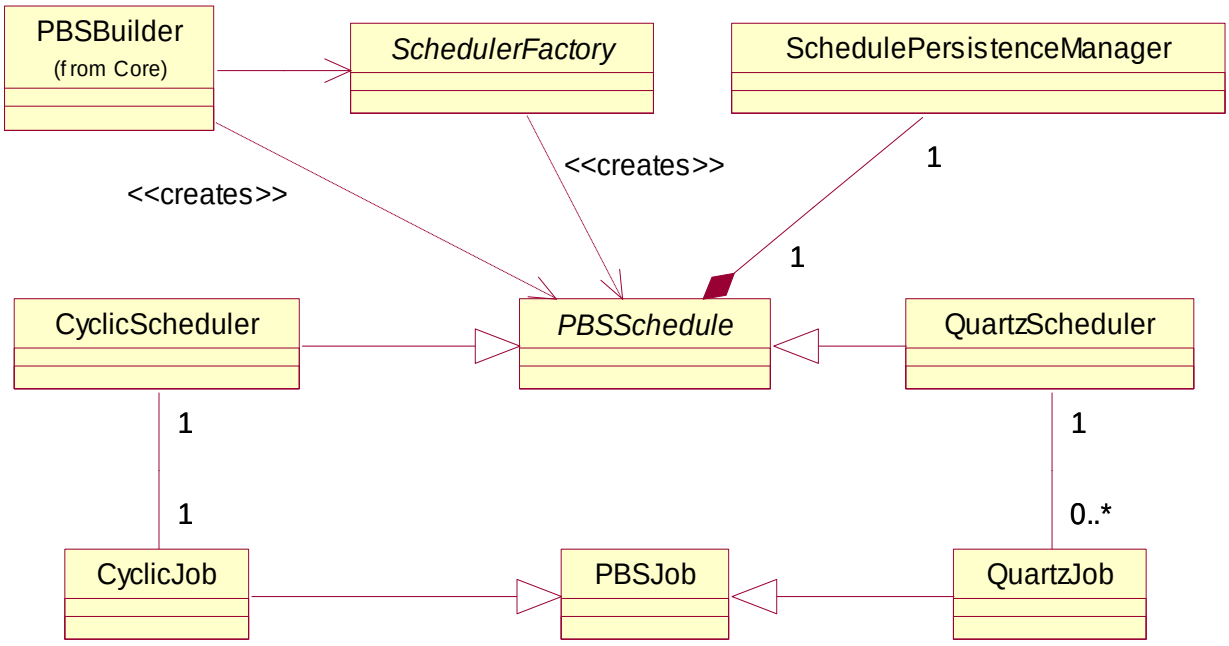

Figura A.5. Diagrama de clases del paquete *Schedule*.

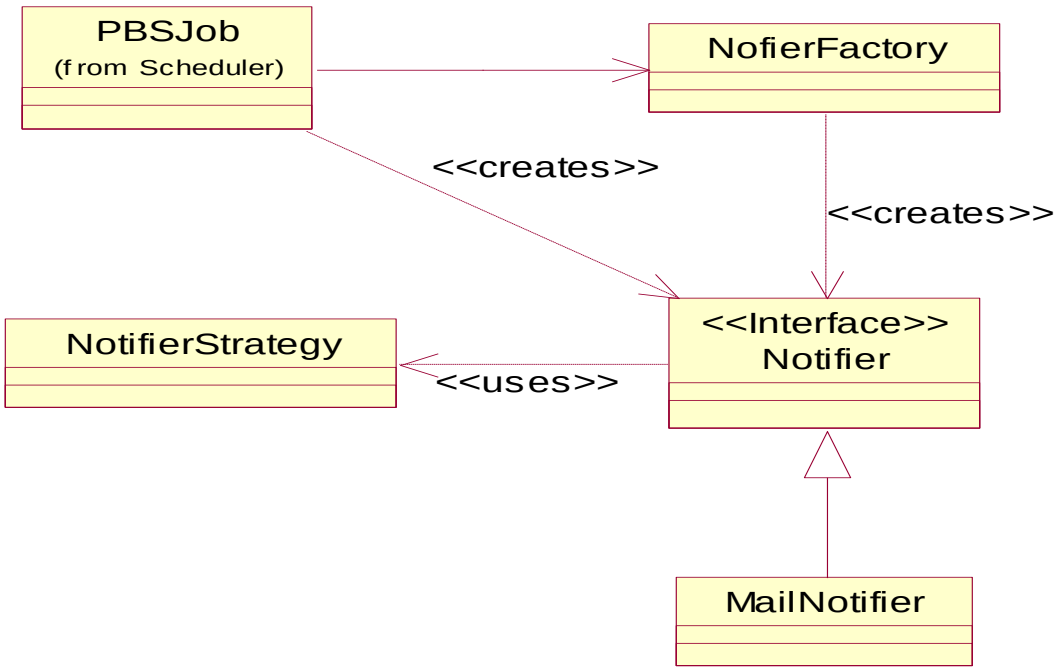

Figura A.6. Diagrama de clases del paquete *Notifier*.

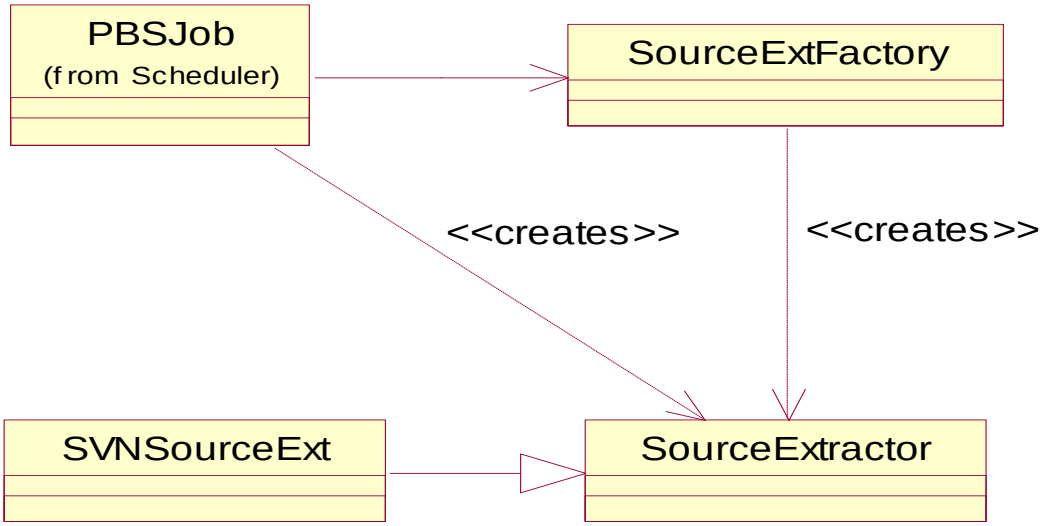

Figura A.7. Diagrama de clases del paquete *Sources*.

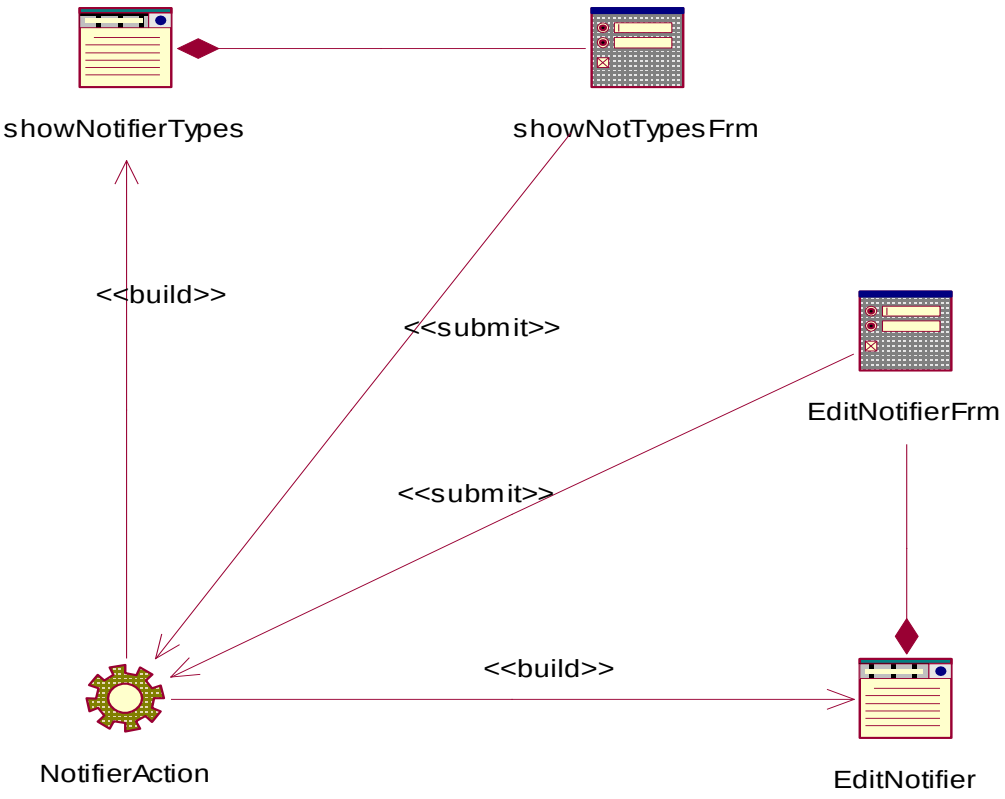

Figura A.8. Diagrama de clases del paquete *Web. Notifiers.*

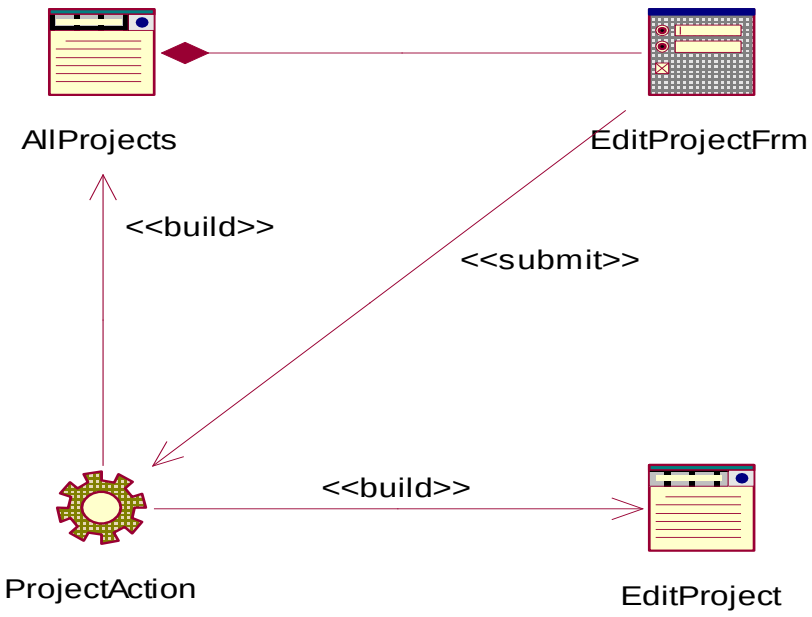

Figura A.9. Diagrama de clases del paquete *Web. Proyectos.*

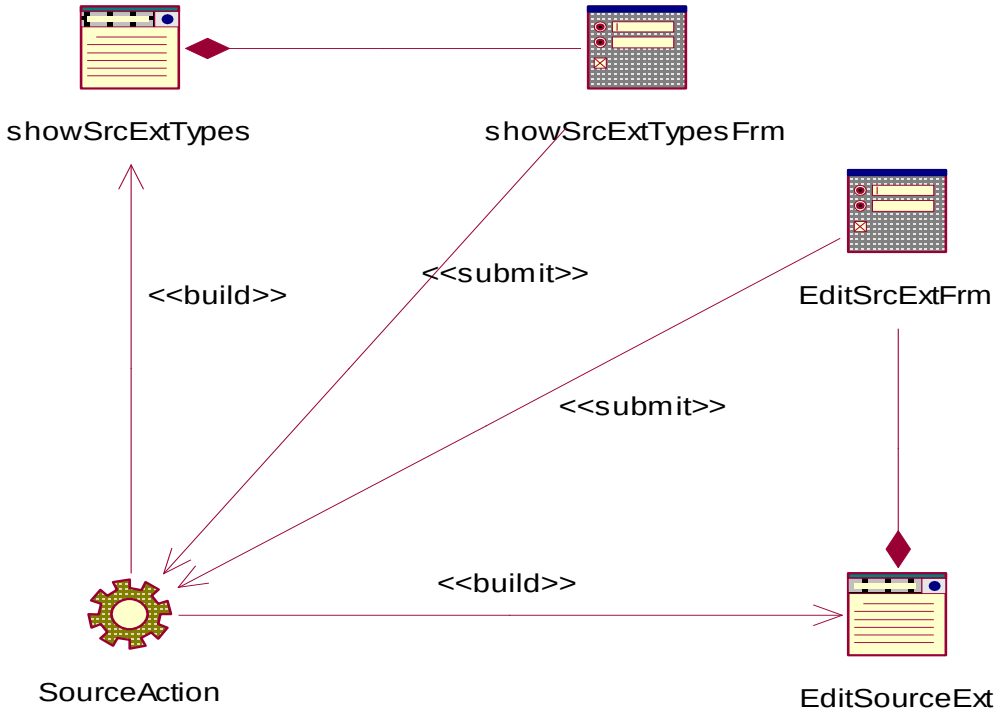

Figura A.10. Diagrama de clases del paquete *Web. Source Extractors.*

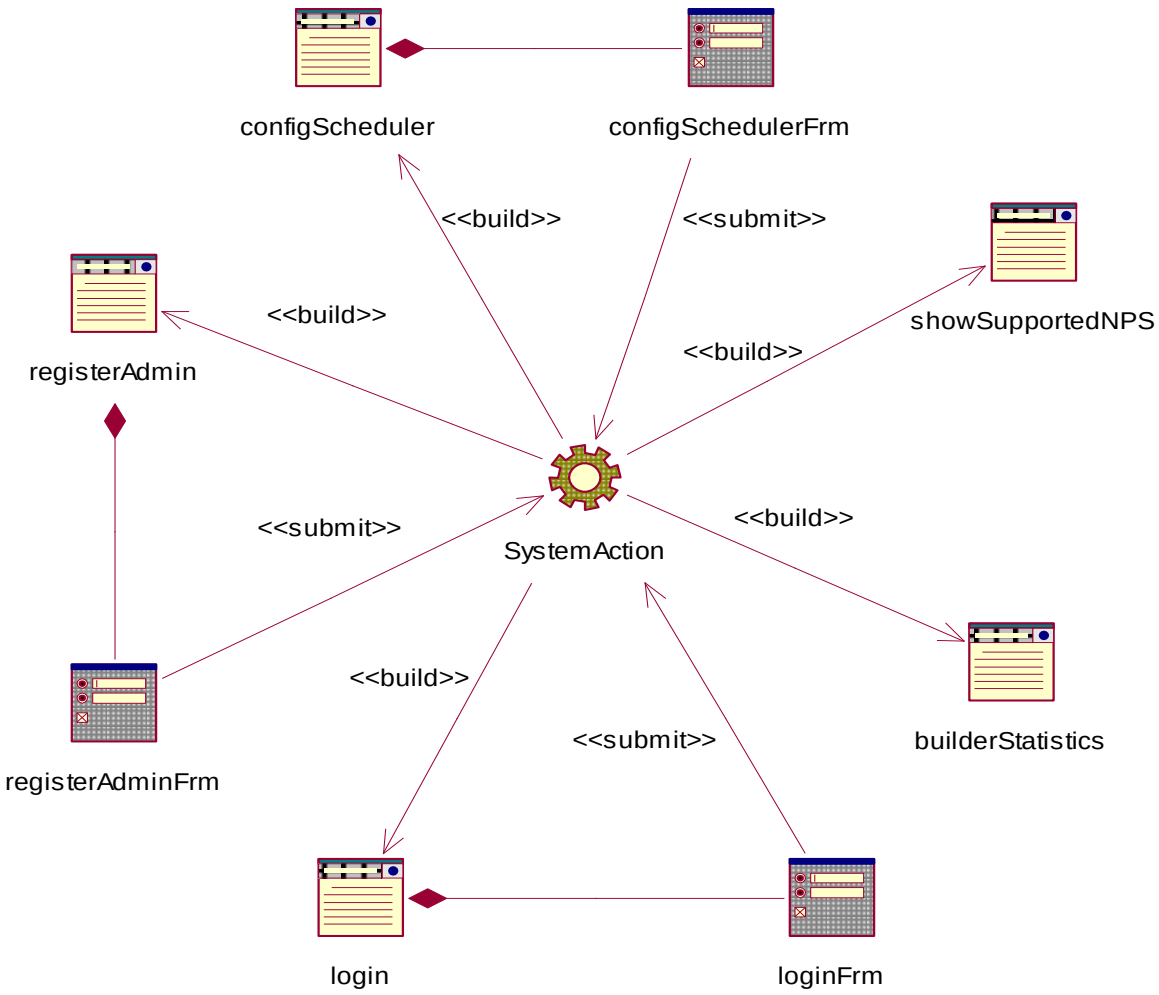

Figura A.11. Diagrama de clases del paquete *Web. System.*

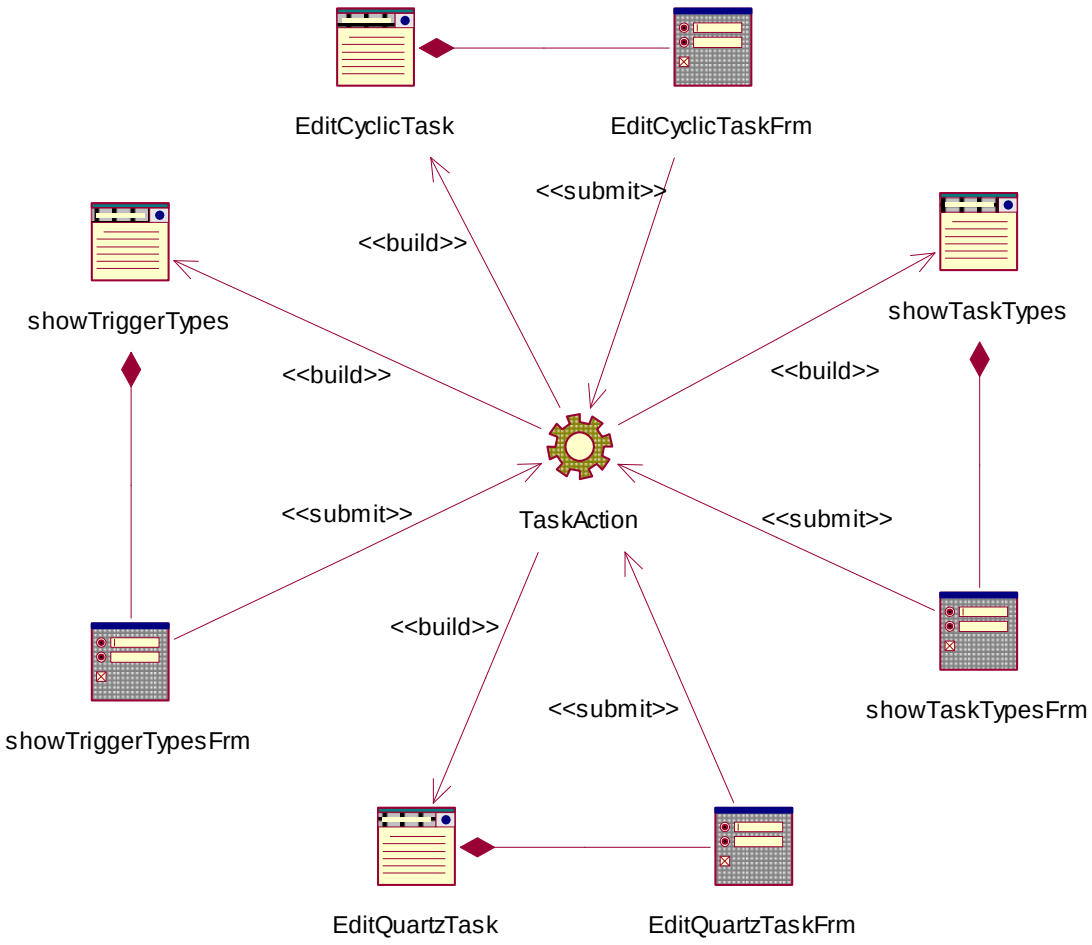

Figura A.12. Diagrama de clases del paquete *Web. Tasks.*

Anexo II. Pruebas de Aceptación

Pruebas de aceptación para la historia de usuario **planificación de *builds* por reloj**

Caso de Prueba de Aceptación	
Código: HU1_P1	**Historia de Usuario:** 1 **-** Planificación de *builds* por reloj.
Nombre: Alta de un proyecto en el planificador por reloj.	
Descripción: Prueba para la funcionalidad de dar de alta a un proyecto en el planificador.	
Condiciones de Ejecución: El planificador debe estar iniciado y ejecutándose. Se utilizará un proyecto con datos válidos.	
Entrada / Pasos de ejecución: Se intenta dar de alta a un proyecto con datos válidos	
Resultado Esperado: El proyecto es registrado sin errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU1_P2	**Historia de Usuario:** 1 **-** Planificación de *builds* por reloj.
Nombre: Baja de un proyecto en el planificador por reloj.	
Descripción: Prueba para la funcionalidad de dar de baja a un proyecto en el planificador.	
Condiciones de Ejecución: El planificador debe estar iniciado y ejecutándose. El proyecto debe estar registrado en el planificador.	
Entrada / Pasos de ejecución: Se intenta dar de baja al proyecto registrado.	
Resultado Esperado: El proyecto es desregistrado sin errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU1_P3	**Historia de Usuario:** 1 **-** Planificación de *builds* por reloj.
Nombre: Planificación inválida para iniciación de construcción.	
Descripción: Se planificará un proyecto para que inicie su construcción en una fecha inválida.	
Condiciones de Ejecución: El planificador debe estar iniciado y ejecutándose. El proyecto ya ha sido registrado en el planificador	
Entrada / Pasos de ejecución: Se planificará un proyecto para que inicie su construcción en la fecha inválida 31/febrero/1887.	
Resultado Esperado: El planificador genera un error que es capturado por el sistema y registrado en el fichero de *logs.*	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU1_P4	**Historia de Usuario:** 1 **-** Planificación de *builds* por reloj.
Nombre: Planificación para iniciación de construcción cada un intervalo de tiempo.	
Descripción: Se planificará un proyecto para que inicie su construcción cada un intervalo de tiempo.	
Condiciones de Ejecución: El planificador debe estar iniciado y ejecutándose. El proyecto ya ha sido registrado en el planificador	
Entrada / Pasos de ejecución: Se planificará un proyecto para que inicie su ejecución cada 5 minutos.	
Resultado Esperado: No se generan errores, los datos de la planificación son aceptados y cada 5 minutos se inicia la construcción del proyecto.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU1_P5	**Historia de Usuario:** 1 **-** Planificación de *builds* por reloj.
Nombre: Planificación para iniciación de construcción en una fecha fija.	
Descripción: Se planificará un proyecto para que inicie su construcción en una fecha válida.	
Condiciones de Ejecución: El planificador debe estar iniciado y ejecutándose. El proyecto ya ha sido registrado en el planificador	
Entrada / Pasos de ejecución: Se planificará un proyecto para que inicie su ejecución el 9/febrero/2007 a las 8:15 AM.	
Resultado Esperado: No se generan errores, los datos de la planificación son aceptados y se inicia la construcción del proyecto en la fecha fijada.	
Evaluación de la Prueba: Prueba satisfactoria	

Pruebas de aceptación para la historia de usuario **construcción del *build***

Caso de Prueba de Aceptación	
Código: HU2_P1	**Historia de Usuario:** 2 **–** Construcción del *build*.
Nombre: Construcción satisfactoria de un proyecto.	
Descripción: Se iniciará la construcción de un proyecto simple de ejemplo del cual se sabe con anterioridad que se construye satisfactoriamente.	
Condiciones de Ejecución: Los datos del proyecto son válidos. Se conoce con anterioridad que el proyecto se construye satisfactoriamente. Las fuentes del proyecto son accesibles. Los correos electrónicos de los desarrolladores son válidos.	
Entrada / Pasos de ejecución: Se inicia la construcción del proyecto de ejemplo. Se espera 15 minutos después que finalice la construcción (para recibir los correos de notificación)	
Resultado Esperado: La construcción del *build* se efectúa satisfactoriamente y el resultado es satisfactorio. No se generan errores. Se crean los artefactos deseados. Se notifica mediante el envío de correos electrónicos a los desarrolladores sobre del resultado del *build.*	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU2_P2	**Historia de Usuario:** 2 **–** Construcción del *build*.
Nombre: Construcción fallida de un proyecto.	
Descripción: Se iniciará la construcción de un proyecto simple de ejemplo del cual se sabe con anterioridad que su construcción fallará.	
Condiciones de Ejecución: Los datos del proyecto son válidos. Se conoce con anterioridad que el proyecto falla debido a un error de compilación. Las fuentes del proyecto son accesibles. Los correos electrónicos de los desarrolladores son válidos.	
Entrada / Pasos de ejecución: Se inicia la construcción del proyecto de ejemplo. Se espera 15 minutos después que finalice la construcción (para recibir los correos de notificación)	
Resultado Esperado: La construcción del *build* se efectúa satisfactoriamente y el resultado es fallido. No se generan errores. Se crean los artefactos deseados hasta que se detecte el error de compilación. Se notifica mediante el envío de correos electrónicos a los desarrolladores sobre del resultado del *build.*	
Evaluación de la Prueba: Prueba satisfactoria	

Pruebas de aceptación para la historia de usuario **planificación Iterativa de *builds***

Caso de Prueba de Aceptación	
Código: HU3_P1	**Historia de Usuario:** 3 **-** Planificación iterativa de *builds*.
Nombre: Alta de un proyecto en el planificador iterativo.	
Descripción: Se pondrá a prueba la funcionalidad para dar de alta a un proyecto en el planificador. Se utilizará un proyecto con datos válidos. Se utilizará una interfaz rudimentaria para la prueba.	
Condiciones de Ejecución: El planificador debe estar iniciado y ejecutándose.	
Entrada / Pasos de ejecución: Se intenta dar de alta a un proyecto con datos válidos	
Resultado Esperado: El proyecto es registrado sin errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU3_P2	**Historia de Usuario:** 3 **-** Planificación iterativa de *builds*.
Nombre: Baja de un proyecto en el planificador iterativo.	
Descripción: Se pondrá a prueba la funcionalidad para dar de baja a un proyecto en el planificador. Se utilizará una interfaz rudimentaria para la prueba.	
Condiciones de Ejecución: El planificador debe estar iniciado y ejecutándose. El proyecto debe estar registrado en el planificador.	
Entrada / Pasos de ejecución: Se intenta dar de baja a un proyecto registrado	
Resultado Esperado: El proyecto es desregistrado sin errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU3_P3	**Historia de Usuario:** 3 **-** Planificación iterativa de *builds*.
Nombre: Iniciación de la construcción de los *builds* de forma iterativa.	
Descripción: Se iniciara el planificador con dos proyectos registrados para que inicie su construcción de forma iterativa.	
Condiciones de Ejecución: El planificador esta inactivo. Los datos de los proyectos son válidos. Los proyectos están registrados en el planificador.	
Entrada / Pasos de ejecución: Se inicia el planificador. Se configura el planificador para que espere 20 segundos entre iteraciones.	
Resultado Esperado: El planificador se inicia normalmente. El planificador inicia la construcción del proyecto de los proyectos de forma iterativa. El planificador espera 20 segundos entre el fin de una iteración y el comienzo de la otra.	
Evaluación de la Prueba: Prueba satisfactoria	

Pruebas de aceptación para la historia de usuario **gestión de proyectos a construir**

Caso de Prueba de Aceptación	
Código: HU4_P1	**Historia de Usuario:** 4 **–** Gestión de proyectos a construir.
Nombre: Adición de proyecto válido.	
Descripción: Se adicionara al sistema un proyecto con datos válidos.	
Condiciones de Ejecución: Los datos del proyecto son válidos.	
Entrada / Pasos de ejecución: Se ejecutan las funciones necesarias para la adición de un proyecto.	
Resultado Esperado: Se adiciona el proyecto en el sistema. Se registra el proyecto en los planificadores, según su configuración. Se comienzan a construir *builds* del proyecto. No se producen errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU4_P2	**Historia de Usuario:** 4 **–** Gestión de proyectos a construir.
Nombre: Adición de proyecto inválido.	
Descripción: Se intentará adicionar al sistema un proyecto con datos inválidos.	
Condiciones de Ejecución: Los datos del proyecto son inválidos.	
Entrada / Pasos de ejecución: Se ejecutan las funciones necesarias para la adición de un proyecto.	
Resultado Esperado: Se produce un error que es capturado por el sistema y registrado en los *logs*.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU4_P3	**Historia de Usuario:** 4 **–** Gestión de proyectos a construir.
Nombre: Eliminación de proyecto.	
Descripción: Se eliminará un proyecto existente en el sistema.	
Condiciones de Ejecución: El proyecto debe estar registrado en el sistema.	
Entrada / Pasos de ejecución: Se ejecutaran las funciones necesarias para la eliminación del proyecto.	
Resultado Esperado: Se desregistra el proyecto de todos los planificadores. Se elimina el proyecto del sistema. No se construyen más *builds* del proyecto. No se producen errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU4_P4	**Historia de Usuario:** 4 **–** Gestión de proyectos a construir.
Nombre: Modificación de proyecto.	
Descripción: Se modificarán los datos de un proyecto existente en el sistema.	
Condiciones de Ejecución: El proyecto debe estar registrado en el sistema.	
Entrada / Pasos de ejecución: Se ejecutarán las funciones necesarias para la modificación de los datos del proyecto.	
Resultado Esperado: Se modifican los datos del proyecto. Se construyen los nuevos *builds* con los datos modificados. No se producen errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Pruebas de aceptación para la historia de usuario **gestión de estadísticas**

Caso de Prueba de Aceptación	
Código: HU5_P1	**Historia de Usuario:** 5 **–** Gestión de estadísticas.
Nombre: Generación de estadísticas.	
Descripción: Se pondrá en funcionamiento el sistema durante un tiempo determinado y se verificarán las estadísticas generadas.	
Condiciones de Ejecución: Se sabe con anticipación cuales son las estadísticas correctas que el sistema debe generar.	
Entrada / Pasos de ejecución: Se iniciará el sistema con varios proyectos registrados en los planificadores. Se mantendrá en ejecución durante 15 minutos. Se evaluarán las estadísticas generadas con respecto a las que se conocen son correctas.	
Resultado Esperado: Se generan las estadísticas. Se almacenan las estadísticas en el sistema de archivos. Las estadísticas generadas por el sistema son idénticas a las que se conocen con anticipación que son correctas.	
Evaluación de la Prueba: Prueba satisfactoria	

Pruebas de aceptación para la historia de usuario **creación de la interfaz *Web***

Caso de Prueba de Aceptación	
Código: HU6_P1	**Historia de Usuario:** 6 – Creación de la interfaz *Web*.
Nombre: Autenticación por primera vez.	
Descripción: Se comprobará el mecanismo de autenticación para la primera vez que un usuario accede a la interfaz *Web.*	
Condiciones de Ejecución: No se ha accedido previamente a la interfaz *Web*, por tanto no se ha establecido nombre de usuario ni contraseña para el administrador del sistema.	
Entrada / Pasos de ejecución: Se accede mediante un explorador a la interfaz *Web*. El sistema solicita el nombre de usuario y la contraseña del administrador. El usuario introduce un nombre de usuario y contraseña.	
Resultado Esperado: Se detecta primeramente la falta de credenciales para el administrador. Se solicitan las credenciales al usuario. Se almacenan los datos introducidos por el usuario como las credenciales del administrador. Se autentica al usuario como administrador. Se deja al usuario acceder al sitio. No se producen errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU6_P2	**Historia de Usuario:** 6 – Creación de la interfaz *Web*.
Nombre: Autenticación correcta.	
Descripción: Se probará el mecanismo de autenticación para el administrador con credenciales válidas.	
Condiciones de Ejecución: Las credenciales del administrador son válidas.	
Entrada / Pasos de ejecución: Se accede mediante un explorador a la interfaz *Web.* Se solicitan las credenciales del usuario. El usuario introduce las credenciales del administrador.	
Resultado Esperado: Se detecta primeramente la existencia de credenciales para el administrador. Se solicitan las credenciales al usuario. Se autentica al usuario como administrador. Se deja al usuario acceder al sitio. No se producen errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU6_P3	**Historia de Usuario:** 6 – Creación de la interfaz *Web*.
Nombre: Autenticación incorrecta.	
Descripción: Se probará el mecanismo de autenticación para el administrador con credenciales inválidas.	
Condiciones de Ejecución: Las credenciales del administrador son inválidas.	
Entrada / Pasos de ejecución: Se accede mediante un explorador a la interfaz *Web.* Se solicitan las credenciales del usuario. El usuario introduce las credenciales del administrador.	
Resultado Esperado: Se detecta primeramente la existencia de credenciales para el administrador. Se solicitan las credenciales al usuario. Se muestra el mensaje de error: "*Login failed, please try again*" Se vuelve a la página de *login*. No se producen errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU6_P4	**Historia de Usuario:** 6 – Creación de la interfaz *Web*.
Nombre: Reiniciar planificadores.	
Descripción: Se probará la funcionalidad de reiniciar los planificadores del sistema mediante la interfaz *Web*.	
Condiciones de Ejecución: Los planificadores deben estar iniciados y en ejecución. El usuario debe estar autentificado como administrador.	
Entrada / Pasos de ejecución: Se accede a la página de administración del sistema en la interfaz *Web.* Se presiona el botón "*Restart*".	
Resultado Esperado: Se reinician los planificadores. No se producen errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU6_P5	**Historia de Usuario:** 6 – Creación de la interfaz *Web*.
Nombre: Detener sistema	
Descripción: Se probará la funcionalidad de detener el sistema mediante la interfaz *Web.*	
Condiciones de Ejecución: El usuario debe estar autentificado como administrador.	
Entrada / Pasos de ejecución: Se accede a la página de administración del sistema en la interfaz *Web.* Se presiona el botón "*Shutdown*".	
Resultado Esperado: Se detiene la ejecución del sistema. No se producen errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU6_P6	**Historia de Usuario:** 6 – Creación de la interfaz *Web*.
Nombre: Mostrar estadísticas.	
Descripción: Prueba para la funcionalidad de mostrar estadísticas.	
Condiciones de Ejecución: El usuario debe estar autenticado como administrador.	
Entrada / Pasos de ejecución: Se accede a la página de estadísticas del sistema en la interfaz *Web.*	
Resultado Esperado: Se muestran las estadísticas correctamente. No se producen errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU6_P7	**Historia de Usuario:** 6 – Creación de la interfaz *Web*.
Nombre: Adición y eliminación de un proyecto.	
Descripción: Prueba para las funcionalidades de adición y eliminación de un proyecto mediante la interfaz *Web*	
Condiciones de Ejecución: El usuario debe estar autenticado como administrador. Los datos del proyecto son válidos.	
Entrada / Pasos de ejecución: Se accede a la página de administración de proyectos en la interfaz *Web.* Se presiona el botón "*New*" Se llenan los datos del proyecto. Se adiciona el proyecto presionando el botón "*Save*". Se accede a la lista de proyectos del sistema. Se elimina el proyecto presionando en el enlace "*Delete*" correspondiente.	
Resultado Esperado: Se adiciona el proyecto satisfactoriamente. Se muestra el proyecto en la lista de proyectos del sistema. Se comienzan a construir *builds* del proyecto. Se elimina el proyecto satisfactoriamente. No se producen errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU6_P8	**Historia de Usuario:** 6 – Creación de la interfaz *Web*.
Nombre: Modificación de un proyecto	
Descripción: Prueba de la funcionalidad de modificación de un proyecto mediante la interfaz *Web*	
Condiciones de Ejecución: El usuario debe estar autenticado como administrador. El proyecto esta registrado en el sistema. Los nuevos datos del proyecto son válidos.	
Entrada / Pasos de ejecución: Se accede a la lista de proyectos del sistema. Se presiona sobre el enlace "*Edit*" correspondiente al proyecto de prueba. Se modifican los datos del proyecto. Se presiona el botón "*Save*".	
Resultado Esperado: Se muestra el proyecto en la lista de proyectos del sistema. Se modifican los datos del proyecto satisfactoriamente. Se empiezan a construir *builds* del proyecto con sus nuevos datos. No se producen errores.	
Evaluación de la Prueba: Prueba satisfactoria	

Glosario de términos

Bash. Interprete de comandos Unix.

Build. Resultado del proceso de compilación y empaquetado de un producto de software.

Jabber. Protocolo de mensajería instantánea.

Notifier. Mecanismo mediante el cual se notifica por un canal específico a una o varias personas acerca del resultado del proceso de construcción del *build* de un proyecto.

Plugin. "Parche" para un programa que le añade características nuevas.

Publisher. Mecanismo mediante el cual se publican en un sitio determinado los artefactos relacionados a un *build*.

Release. Versión de un producto informático.

Shell Script. Se entiende por "*Shell Script*" una serie de comandos escritos en un archivo de texto plano. Es exactamente como un archivo "*batch*" de MS-DOS pero con mayor poder.

Servicios Cron. Servicios de planificación basada en tiempo propios de sistemas operativos tipo Unix.

Source Extractor. Mecanismo mediante el cual se crea una copia de trabajo del código perteneciente al proyecto que se desea construir y que se encuentra almacenado bajo un sistema de control de versiones.

Tester. Rol de la persona que se dedica a hacer pruebas a un producto de *software*.