UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 4

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Cliente para la gestión de tareas asociado al gestor de proyectos Redmine.

Autor: Remberto Ramos Rodríguez.

Tutor: Ing. Jorge Martínez Padrón.

La Habana, Cuba

Junio del 2014

Cliente para la gestión de tareas asociado al gestor de proyectos Redmine

DECLARACIÓN DE AUTORÍA

Declaro ser único autor de la presente tesis y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Remberto Ramos Rodríguez

_____
Firma del Autor

Ing. Jorge Martínez Padrón

_____
Firma del Tutor

Agradecimientos

A mi MADRE, por su dedicación y amor, sobre todo por guiarme en este largo camino.

A mi hermano por ser mi ejemplo a seguir.

A la madre de mi niña por darme un motivo más por el cual vivir.

A mi familia en general.

A mi tutor, por ser paciente y entenderme en los momentos cuando más lo necesitaba.

A mis mejores amigos: Maday Cristina, Yuselis Iglesias, Yunier Broche, Yasiel Tejeda, Oniel González y Raúl Infante, con quienes compartí los mejores años de universidad y me ayudaron aun cuando sabían que no tenía la razón.

A mis otros amigos: Orelvis Lago (el infantil), Eduardo Alejandro (el crack), Yordanis Morejón (la mula), Ernesto Parsons (Pastosin), Yaneisi Hernández y Annelys Mamposo (Pulgarcita).

A ***Esther García Naranjo*** *por acordarse de mí estando tan lejos y por ayudarme cuando más lo necesitaba. A ella, muchas gracias.*

A todos los profes que me formaron durante los 5 años de estudio.

Dedicatoria

A mi madre por ser el motivo y sentido de este sueño, por su amor incondicional.

A mi pequeña Angelyn, por ser lo más importante en mi vida.

A mi familia por obligarme a ser mejor persona.

A mi hermano por creer en mí y darme unas bellas sobrinitas.

A todos mis amigos y compañeros de estudio por su compañía y los momentos vividos.

A todos mis amigos, los que están y los que no están, los que una vez fueron y ya no son.

Resumen

En el desarrollo de un proyecto es necesario la planificación y orientación de las tareas a realizar. Para la planificación, orientación y monitoreo de las mismas se han desarrollado aplicaciones que facilitan este trabajo; por ejemplo, el gestor de proyectos Redmine. Redmine es una aplicación web que se encarga de gestionar proyectos y todo lo relacionado a ellos: tareas, hitos, diagramas, usuarios, etc. El mismo cuenta con un sistema de notificación por defecto que no es lo suficientemente efectivo para el correcto funcionamiento y monitoreo de las tareas. Teniendo en cuenta lo antes expuesto, se propone el desarrollo de un sistema que permita apoyar el proceso de gestión de las tareas en el gestor de proyectos Redmine.

Índice

Introducción

En las últimas décadas el desarrollo de *software* ha sido un eslabón fundamental para satisfacer las necesidades en las distintas áreas de la ciencia. Los sistemas informáticos, frutos del desarrollo de *software*, deben responder a las necesidades del cliente para lograr el objetivo para el cuál fue desarrollado. Un sistema informático es un sistema que permite almacenar y procesar información, es el conjunto de partes interrelacionadas: en este caso, *hardware*, *software* y recursos humanos (1). La planificación, orientación, monitoreo y coordinación de las tareas en el desarrollo de *software* son aspectos fundamentales para la obtención de un sistema informático funcional y acorde a lo previsto.

Actualmente existen herramientas que se encargan de gestionar todo lo relacionado al desarrollo de *software*, tal es el caso del gestor de proyectos Redmine. La gestión de proyectos es el proceso mediante el cual se planifica y se controla el desarrollo de un sistema aceptable con un coste mínimo y dentro de un período de tiempo específico (2). Redmine es una aplicación web de fácil uso que permite entre otras funcionalidades la gestión de proyectos como función principal, incluyendo calendario, diagrama de Gantt y gestión de tareas. Por su calidad y su condición de *software* libre ha sido muy bien acogida nacional e internacionalmente. Utilizando el Redmine como base, en la Universidad de las Ciencias Informáticas (UCI) se desarrolló la solución GESPRO, que se encuentra disponible para todos los proyectos de la universidad. Los proyectos que hacen uso de esta herramienta, planifican sus tareas o peticiones y definen un responsable para cada una de ellas.

El uso de esta herramienta para la planificación y gestión de proyectos es una política de la universidad, por lo que a cada integrante de proyecto se le hace necesario interactuar con el GESPRO, y se convierte en una actividad fundamental el control y seguimiento tanto de las tareas asignadas a él como las asignadas por él. Aunque la aplicación cuenta con un sistema de notificaciones por defecto basado en el envío de correo electrónico a cada uno de sus integrantes, por lo general no resulta suficiente; pues no permite actualizar los atributos de las tareas como son: porciento de cumplimiento o el estado una vez concluida. Además, no permite al usuario tener una visión global de sus tareas y la prioridad de las mismas, pues es notificado de forma independiente para cada una de ellas. En muchos casos es necesario identificarse en el GESPRO y navegar hasta las tareas para realizar alguna otra operación sobre ellas. Además, los integrantes del proyecto se demoran en conocer que tienen tareas asignadas porque en muchos casos el sistema de notificación por defecto no funciona correctamente.

Teniendo en cuenta los elementos planteados se identifica el siguiente **problema a resolver**: ¿Cómo apoyar la gestión de tareas en el gestor de proyectos Redmine?
Se define como **objeto de estudio:** los sistemas de gestión de tareas. El trabajo se encuentra enmarcado dentro del **campo de acción:** sistemas de gestión de tareas asociados al gestor de proyectos Redmine. Se plantea como **objetivo general:** desarrollar una aplicación que apoye la gestión de tareas en el gestor de proyectos Redmine.

A partir de un análisis del objetivo general se derivan los siguientes **objetivos específicos:**

- Analizar el estado del arte sobre los sistemas de gestión de tareas.
- Definir los requisitos de la aplicación.
- Realizar el análisis y diseño de la aplicación.
- Implementar y probar la aplicación.

Con el fin de dar cumplimiento a los objetivos específicos se definieron como **tareas de investigación:**

- Análisis del estado del arte sobre los sistemas de gestión de tareas que existen.
- Descripción de los requisitos del sistema para caracterizar las funcionalidades de la aplicación.
- Caracterización y selección de la tecnología a utilizar.
- Análisis del modelo de datos a utilizar.
- Elaboración de una propuesta de diseño como base para la implementación de la aplicación.
- Implementación de la propuesta de solución elaborada.
- Realización de pruebas para identificar y corregir los errores de la aplicación.

Luego de un análisis de la problemática se plantea la siguiente **idea a defender**: si se implementa un cliente para la gestión de tareas asociado al gestor de proyectos Redmine, se logrará apoyar la gestión de las tareas en el mismo.

Los **métodos científicos** que se tomarán en cuenta en el proceso de investigación y elaboración de este trabajo son los siguientes:
Dentro de los métodos teóricos:

- Analítico-Sintético: Permite analizar la información relacionada con los sistemas de gestión de tareas para llegar a conclusiones sobre ellos que aporten a la solución de esta investigación.

Dentro de los métodos empíricos:

- Entrevista: Se utiliza para definir a través del cliente si existe algún sistema previo en la universidad y las funcionalidades necesarias del sistema. Para más información ver anexo 1.

Estructura capitular:

Capítulo 1: Fundamentación teórica.

En este capítulo se abordan temas relacionados con el gestor de proyectos Redmine y su relación con el estilo arquitectónico REST (Transferencia de Estado Representativo). También se aborda sobre la gestión de tareas haciendo énfasis en los clientes asociados al gestor de proyectos Redmine, presentando los elementos más significativos. Por último se hace un estudio sobre las herramientas, tecnologías y metodología a utilizar para el desarrollo de la aplicación.

Capítulo 2: Características y diseño del sistema.

Se presenta la propuesta de solución y se muestran los artefactos propios de la metodología de desarrollo seleccionada durante la investigación.

Capítulo 3: Implementación y prueba.

Se expone lo relacionado al proceso de implementación y prueba de la aplicación. Se realizan los casos de pruebas necesarios para asegurar el correcto funcionamiento de la aplicación y se generan otros artefactos propios de la metodología de desarrollo de *software*.

Capítulo 1.Fundamentación teórica

1.1 Introducción

Desde los inicios de la informática, el hombre ha buscado la manera más amena de resolver sus necesidades en un ordenador, ya sea utilizando *software* de terceros o desarrollándolo el mismo. La producción de *software* requiere de personal calificado para el desarrollo de un sistema informático que satisfaga las necesidades del cliente, donde la planificación, organización y continuidad de las tareas es un factor imprescindible en la industria del *software*. Actualmente existen herramientas que facilitan este trabajo y que brindan funcionalidades adicionales que ayudan al correcto desarrollo y al trabajo en equipo en un proyecto. Un ejemplo de estas herramientas es el Redmine, una herramienta de código abierto desarrollado con un único fin, la gestión de proyectos. En el presente capítulo se realiza un estudio sobre el gestor de proyectos Redmine, su relación con el estilo arquitectónico Transferencia de Estado Representativo (REST, por sus siglas en inglés) y las principales deficiencias del sistema de notificación por defecto que posee. También se abordan temas relacionados con la gestión de tareas y los clientes asociados al gestor de proyectos Redmine. Seguidamente se detallan las tecnologías y herramientas a utilizar para el desarrollo de la aplicación y por último se presenta un estudio sobre las metodologías de desarrollo de *software*.

1.2 Gestión de proyectos con el Redmine

La gestión de proyectos es una disciplina de planeamiento, organización, motivación y control de los recursos con el propósito de alcanzar uno o varios objetivos. Se estima que en un 70% de los fracasos de los proyectos de *software* influye la no estimación de costos reales asociados a los mismos y la no incorporación de prácticas esenciales, además de la no estandarización en los procesos de *software* (3).

Redmine es una de las web más populares basadas en la aplicación de gestión de proyectos. Está desarrollado utilizando el *framework* [1]de desarrollo Ruby on Rails. Una de las fortalezas a destacar es su cualidad de ser multiplataforma y soportar múltiples motores de bases de datos relacionales como por ejemplo: MySQL, PostgreSQL y SQLite (4).

[1] Estructura conceptual y tecnológica que puede servir de base para la organización y desarrollo de software.

Algunas de sus funcionalidades básicas son: fábrica de wiki por proyecto, calendario, diagrama de Gantt, gestión de tareas y creación de sub-proyectos. Otra característica peculiar es por ejemplo: cada usuario puede tener un papel diferente en cada proyecto, la posibilidad de manipular la visibilidad de los proyectos, o sea, cada proyecto puede ser declarado como público (visible por cualquier persona) o privado (solo visible por los integrantes del equipo de trabajo del proyecto). La sección administrativa del Redmine permite crear proyectos, usuarios y vincularlos entre sí, además de poder gestionar campos personalizados.

1.2.1 Deficiencias del sistema de notificación por defecto del Redmine

El sistema de notificación por defecto del gestor de proyecto Redmine es el correo, el mismo está definido de manera global, de tal forma que existe para todos los proyectos o no existe para ninguno. Previamente se identifican los eventos que activan el sistema de notificación y después cada usuario en su configuración personal elige recibir notificaciones de cualquier evento o solo los relacionados con él. Puede configurarse el servidor de correo entrante, permitiendo así actualizar peticiones simplemente por email e incluso crear nuevas peticiones.

Este sistema de notificación no permite al usuario tener una visión global de las tareas por prioridad, pues en cada correo muestra los datos de los cambios de una tarea pero no muestra las demás en conjunto, por lo que no le ayuda a decidir cuál realizar primero. Además, el sistema de notificación por defecto no permite actualizar los atributos de las tareas como son: porciento de cumplimiento o el estado una vez concluida. Por otro lado, los usuarios se demoran en conocer que tienen tareas asignadas porque en muchos casos el sistema de notificación por defecto no funciona correctamente.

1.3 Transferencia de Estado Representativo (REST)

Según Roy T Fielding[2], REST es un estilo de arquitectura de *software* para sistemas hipermedias distribuidos como la web, derivado de varios de los estilos arquitectónicos basados en la red y en combinación con las restricciones adicionales que definen a un conector de interfaz uniforme (5).

Principios básicos:

- Utiliza los métodos HTTP[3] de manera explícita (GET, POST, DELETE, PUT).

[2] Creador del estilo arquitectónico REST en el año 2000.
[3] Protocolo de transferencia de hipertexto.

- No mantiene estado (cada petición es independiente).
- URL[4] en forma de directorios.
- Transferencia de información en formato XML[5] y JSON[6] principalmente.

El estilo define un conjunto de restricciones o características que deben cumplir las arquitecturas de servicios web:

- cliente – servidor
- sin estado
- caché
- interfaz uniforme
- código bajo demanda
- sistema de capas

A continuación se muestra una breve descripción de las mismas (5):

Cliente-servidor:

Esta restricción separa lo que es competencia del cliente y el servidor, distinguiendo lo que concierne a la interfaz de usuario del almacenamiento de datos, así se puede acceder a esta desde múltiples plataformas. Mejora el rendimiento, pues simplifica los componentes del servidor al no tener que implementar las funcionalidades que van asociadas a la interfaz del usuario, permitiendo componentes independientes.

Sin estado:

Esta restricción define que cada petición del cliente debe contener toda la información necesaria para que el servidor la comprenda. Esto permite eliminar el almacenamiento de datos sobre el contexto de la comunicación.

Caché:

Las respuestas a una petición deben poder ser etiquetadas como *cacheable* o no cacheable, si una respuesta es cacheable, entonces al cliente caché (dispositivo intermedio) se le da permiso para reutilizar la respuesta más tarde si se hace una petición equivalente.

Interfaz uniforme:

[4] Localizador de recursos uniforme, usado para nombrar recursos en internet.
[5] Lenguaje de marcas extensible.
[6] JavaScript Object Notation, formato para el intercambio de datos.

La principal característica que distingue a REST del resto de estilos de arquitecturas es el énfasis de usar una interfaz uniforme entre los componentes, a los cuales, aplicándoles los principios de generalidad de la ingeniería del *software* simplifican la arquitectura del sistema global, mejorando así, la visibilidad de interacciones, separando la programación de los servicios que proporciona y animando al desarrollo independiente. La interfaz está diseñada para ser eficiente con transferencias de datos de hipermedias, resultando optimizada para la mayor parte de la web pero no siendo así para otras formas de arquitectura de interacción.

Código bajo demanda:

Esta restricción es opcional, consiste en permitir a los clientes tener la funcionalidad de descargar y ejecutar código. Simplifica el lado del cliente al reducir el número de funcionalidades que tiene que tener implementadas al crearse, las cuales pueden ser descargadas posteriormente, aumentando así el nivel de extensión del sistema. La principal desventaja es que reduce la visibilidad y puede influir en la seguridad del sistema.

Sistema de capas:

Para poder mejorar el comportamiento del rendimiento en Internet, se añade la restricción del sistema de capas, garantizando una arquitectura compuesta por jerarquías. Este sistema de capas permite limitar el comportamiento de los componentes al no acceder más allá de la capa con la que se está comunicando. Se simplifican los componentes, moviendo las funcionalidades de uso infrecuente hacia sistemas intermedios compartidos que pueden usarse para mejorar el rendimiento, permitiendo un balance de la carga de los servicios a través de múltiples redes y procesadores. La principal desventaja del sistema de capas es que añade cabeceras y retrasos al procesamiento de datos.

1.3.1 REST API y Redmine

Redmine ofrece una API[7] para gestionar información en formato XML o JSON a través de REST y usando el protocolo de comunicación HTTP. Dicha información se gestiona a través de los métodos de acceso HTTP que se muestran a continuación (6):

- Http put: Pemite crear nuevos objetos
- Http get : Leer objetos
- Http post : Actualizar objetos
- Http delete : Eliminar un objeto

[7] Interfaz de programación de aplicaciones, conjunto de funciones y procedimientos que ofrece cierta biblioteca.

```
http://localhost:8080/redmine/users/3.xml?include=memberships
<user>
    <id>3</id>
    <login>ale</login>
    <firstname>Alexander</firstname>
    <lastname>Ortiz</lastname>
    <mail>ale@uci.cu</mail>
    <created_on>2014-05-05T02:50:16Z</created_on>
    <last_login_on>2014-05-07T04:39:38Z</last_login_on>
    <api_key>f1ebccf29766691f758cb3977564ef43da287fcd</api_key>
    <memberships type="array">
    <membership>
        <id>1</id>
        <project id="1" name="alfaomega_version2"/>
        <roles type="array">
            <role id="3" name="Jefe de proyecto"/>
        </roles>
    </membership>
</user>

```

Figura 1: Información asociada a un usuario con id=3 en formato XML

```
http://localhost:8080/redmine/issues.xml?assigned_to_id=3&status=*
<issues total_count="5" offset="0" limit="25" type="array">
    <issue>
        <id>9</id>
        <project id="2" name="Juegos educativos"/>
        <tracker id="1" name="Errores"/>
        <status id="1" name="Nueva"/>
        <priority id="2" name="Normal"/>
        <author id="4" name="Maday Crsitina Larduet Pedroso"/>
        <assigned_to id="3" name="Alexander Ortiz"/>
        <subject>Capitulo 3</subject>
        <description></description>
        <start_date>2014-05-05</start_date>
        <due_date/>
        <done_ratio>0</done_ratio>
        <estimated_hours/>
        <created_on>2014-05-05T04:21:32Z</created_on>
        <updated_on>2014-05-05T04:21:32Z</updated_on>
        <closed_on/>
    </issue>
    <issue></issue>
    <issue></issue>
    <issue></issue>
    <issue></issue>
</issues>

```

Figura 2: Tareas asociadas a un usuario con id = 3

1.4 Cliente para la gestión de tareas asociado al gestor de proyectos Redmine

Un cliente informático es cualquier elemento de un sistema de información que requiere un servicio mediante el envío de solicitudes al servidor (7). Generalmente estos clientes solicitan la información mediante servicios y se basan en la consulta de información externa. A continuación, una tabla que muestra la clasificación de los clientes informáticos teniendo en cuenta dos aspectos fundamentales.

Tipo de Cliente	Almacenamiento local	Procesamiento local
Cliente liviano	NO	NO
Cliente pesado	SI	SI
Cliente híbrido	NO	SI

Tabla 1: Clasificación de los clientes informáticos. (7)

1.4.1 Gestión de tareas

La gestión de tareas es el proceso mediante el cual se define un conjunto de actividades de planeación, control y ejecución de tareas (8). Teniendo en cuenta lo dicho anteriormente se puede decir que un *software* de gestión de tareas es una aplicación informática que permite planificar y monitorear todo lo relacionado en cuanto a las mismas. De ahí que, un cliente para la gestión de tareas asociado al gestor de proyectos Redmine, es una aplicación que gestiona las tareas de los proyectos registrados en el mismo. Existen varios clientes informáticos, a continuación se detallan algunos.

1.4.2 RedmineClient

La versión del RedmineClient 0.4.0 fue lanzada el 22 de diciembre del 2009, esta versión introduce caché interna de la información recuperada desde el servidor de Redmine, por lo que mejora el rendimiento especialmente para los usuarios con gran cúmulo de información (9).

Principales funcionalidades:

- Autenticación utilizando la API del Redmine.
- Lista de todos los proyectos registrados.
- Lista de las tareas por proyectos.
- Lista de todas las actividades.
- Lista de todos los usuarios en el sistema.
- Creación de nuevas tareas o peticiones.

- Integración con la barra de íconos del escritorio.

1.4.3 RedMinerDroid

RedMinerDroid es un sencillo pero avanzado cliente del gestor de proyectos Redmine, desarrollado en Android[8] para dispositivos móviles. El mismo brinda las funcionalidades básicas que permiten gestionar todo lo relacionado a las tareas en el Redmine. RedMinerDroid es compatible con las versiones desde 1.1 hasta la 2.2.x y hace uso del servicio REST del Redmine para su funcionamiento (10).

Principales funcionalidades:

- Permite obtener todos los proyectos registrados en el sistema.
- Visualiza los detalles relacionados a las tareas o peticiones.
- Soporta SSL[9] (https).
- Permite añadir peticiones y hacer cambios offline y luego sincronizar los datos con el servidor.
- Obtiene las tareas ordenadas por id, nombre y prioridad.
- Permite compartir y filtrar las tareas o peticiones.
- Permite abrir una tarea en el navegador desde el cliente.

1.4.4 JRedmine

JRedmine es un cliente libre, desarrollado completamente en java y compatible con las siguientes versiones del gestor de proyectos Redmine, v1.3.2, 1.4.4, 2.0.4 y 2.1.2. Esta herramienta usa dos *plugins* del *framework* Ruby on Rails para su funcionamiento con las versiones Redmine v1.x y v2.x. El mismo brinda las funcionalidades básicas relacionadas con las peticiones o tareas registradas y añade documentación en inglés para su uso. Además, permite trabajar con campos personalizados, añadirlos y hacer búsquedas teniendo en cuenta los mismos (11).

Principales funcionalidades:

- Permite obtener todos los proyectos registrados en el sistema.
- Visualiza los detalles relacionados a las tareas o peticiones.
- Lista de todos los usuarios en el sistema.

[8] Sistema operativo basado en Linux diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas

[9] Secure Sockets Layer, protocolo de comunicación segura.

- Creación de nuevas tareas o peticiones.

1.5 Tecnologías y herramientas para el desarrollo

Durante el ciclo de vida de desarrollo de un *software* los desarrolladores pueden hacer uso de diferentes tecnologías y herramientas para darle cumplimiento al mismo. La buena selección de las mismas es un elemento considerable, pues un cambio a mediados de desarrollo podría atrasar significativamente el producto final.

Cuando se utilizan tecnologías y herramientas eficaces para el desarrollo de *software* se tiene una gran probabilidad de que el producto final presente alta calidad, que se desarrolle en el tiempo planificado y con los requerimientos establecidos.

1.5.1 Java

Java, un lenguaje desarrollado por James Gosling[10] de Sun MicroSystem y publicado en 1995. Es un lenguaje de programación de propósito general, concurrente y orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias como fuera posible. Su sintaxis deriva mucho de C y C++ pero tiene menos facilidades de bajo nivel como estos lenguajes y permite ejecutarse en cualquier máquina virtual sin importar la arquitectura de la computadora subyacente (12).

Características:

- Orientado a objetos: en este aspecto java fue diseñado desde cero, en el mismo el concepto de objeto resulta sencillo y fácil de ampliar.
- Robusto*:* java consigue un alto margen de codificación sin errores, pues a través de las herramientas de la máquina virtual y el IDE en cuestión, verifica su código a la misma vez que se escribe.
- Arquitectura Neutral*:* java está diseñado para correr en cualquier plataforma independientemente del sistema operativo, para esto hace una compilación en una representación intermedia que recibe el nombre de *códigos* de *byte*, que pueden interpretarse en cualquier sistema con un intérprete de java.

[10] Científico de la computación conocido como el padre del lenguaje de programación Java.

- Distintos tipos de aplicaciones*:* en java se pueden crear diferentes tipos de aplicaciones, desde aplicaciones desktop hasta componentes del propio lenguaje que se pueden integrar entre sí.

Ventajas:

- No se permiten los accesos ilegales a memoria, lo que elimina la propagación de virus y otros programas dañinos.
- Código seguro: el código java pasa muchas pruebas antes de ejecutarse en una máquina.
- Distribuido: java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir *sockets* y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

Desventajas:

- Velocidad: debido a que los programas son interpretados nunca alcanzan la velocidad de un verdadero ejecutable (13).
- Clases de librerías: al poseer clases de librerías el manejo del lenguaje se hace más difícil y consume mucho tiempo.

En Java el problema fugas de memoria se evita en gran medida gracias a la recolección de basura *(o automatic garbage collector)*. El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (Java runtime) es el responsable de gestionar el ciclo de vida de los mismos. Cuando no quedan referencias a un objeto, el recolector de basura de java borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas.

1.5.2 C sharp *(C#)*

C# es el lenguaje que Microsoft[11] desarrolló principalmente para la plataforma .Net. Para su creación se usaron conceptos de C, C++ y Java, entre otros (14).

Características:

[11] Empresa multinacional de origen estadounidense fundada el 4 de abril de 1975 por Bill Gates y Paul Allen dedicada al sector del software.

- Moderno: mejora la productividad en el desarrollo de *software*, incorpora características del estado del arte de los lenguajes actuales.
- Simple: permite una sintaxis sencilla y elegante, evita la utilización de punteros, la gestión de memoria y la validación de límites de arreglos.
- Poderoso: permite el desarrollo de código “seguro” y “no seguro”.
- De propósito general: puede ser utilizado para la construcción de aplicaciones web, aplicaciones de escritorio, servicios web, aplicaciones para celulares y componentes.
- Totalmente orientado a objetos.

Ventajas:

- Incorpora las características de un lenguaje de última generación y está en continuo desarrollo.
- Concepto formalizado de los métodos *get* y *set*, con lo que se consigue código mucho más legible.

Desventaja:

- No es multiplataforma *(al menos de forma nativa*) (15).

1.5.3 C++

En la actualidad, C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar uno de los primeros puestos como herramienta de desarrollo de aplicaciones. C++ es considerado como un lenguaje de nivel intermedio, pues posee características tanto de alto nivel como de bajo nivel (16).

Características:

- Incluye el concepto de puntero[12].
- Programación de bajo nivel.
- Soporte para hilos.

Ventajas:

- Programación estructurada y orientada a objetos.
- Rápido y eficiente.

[12] Variable que hace referencia a una región de memoria.

Desventajas:

- Complicado para aplicaciones web.
- Es más complicado de aprender que java.
- No posee interfaces definidas para la interacción con el API del Redmine.

A continuación una tabla comparativa de los lenguajes de programación mencionados anteriormente teniendo en cuenta los siguientes aspectos: curva de aprendizaje (baja, media, alta), sistema operativo (si es multiplataforma o no), interfaces para la interacción con el API del Redmine y experiencia del desarrollador(ED) sobre este lenguaje.

Lenguaje	Curva aprendizaje	Multiplataforma	API	ED
Java	Media	X	X	X
C#	Media	X (Project Mono)	X	
C++	Larga	X		

Tabla 2: Tabla comparativa de los lenguajes de programación.

1.5.4 NetBeans 7.4

NetBeans es un IDE[13] de código abierto, con su uso se obtienen todas las herramientas necesarias para crear aplicaciones profesionales de escritorio, de empresa, web y móviles con el lenguaje Java, C/C++, e incluso lenguajes dinámicos como PHP, JavaScript, Groovy, y Ruby. NetBeans IDE es fácil de instalar y utilizar, además, se ejecuta en muchas plataformas, incluyendo Windows, Linux, Mac OS X y Solaris. Mantener una visión clara de las aplicaciones grandes, con miles de carpetas y archivos, y millones de líneas de código, es una tarea de enormes proporciones. NetBeans IDE proporciona diferentes vistas de los datos, de múltiples ventanas del proyecto y herramientas útiles para la creación de sus aplicaciones y la gestión de manera eficiente, lo que le permite profundizar en sus datos de forma rápida y sencilla (17).

Características:

- Soporte e integración con sistemas de control de versiones.
- Herramientas para depurado de errores.
- Selector de proyectos intuitivo.
- Disponible en muchos idiomas.

[13] Entorno de desarrollo integrado.

- Ficheros de ayuda en programación.
- Resaltado de errores, palabras reservadas, códigos de otros usuarios, etc.
- Gran documentación.

Ventajas:

- Admite distintos tipos de lenguaje.
- Multiplataforma.

Desventaja:

- Gran consumo de recursos (CPU y RAM).

1.5.5 Eclipse

Eclipse es un IDE que en un principio fue diseñado y desarrollado por IBM[14] y que luego fue lanzada a la comunidad de *software* libre. Eclipse es una plataforma universal que integra herramientas de desarrollo, con una arquitectura abierta y basada en *plugins* (18).

Esta arquitectura de *plugins* permite integrar diversos lenguajes sobre un mismo IDE e introducir otras aplicaciones. Eclipse brinda soporte a todo tipo de proyectos que abarcan el ciclo de vida del desarrollo de aplicaciones incluyendo soporte para modelado. Además, conserva el registro de las versiones, genera y mantiene la documentación de cada etapa del proyecto.

Características:

- Editor visual con sintaxis coloreada.
- Compilación incremental de código.
- Modifica e inspecciona valores de variables.

Ventajas:

- Arquitectura basada en *plugins*.
- Compilación en tiempo real.
- Asistentes para la creación de proyectos.
- Instalación sencilla.

[14] Empresa multinacional estadounidense de tecnología y consultoría fundada en 1911.

Desventaja:

- Se necesita cierto nivel de aprendizaje para interactuar con él.

1.5.6 Jbuilder

JBuilder es un IDE Java de Borland, la versión 2007 ya es multiplataforma, está disponible para Windows, Linux y MacOS X. Soporta plenamente la última versión del *Kit* de desarrollo de Java (JDK) proporcionado por Sun[15]. Otra característica importante de JBuilder es su soporte sin compromisos de componentes, escritos sola y exclusivamente en Java, lo que maximiza la portabilidad real del código, frente a otros entornos que por apoyarse en código no Java, ven muy mermada su portabilidad (19).

Sus programas se pueden ejecutar en PC´s, Macintosh, Unix, redes de computadoras o en cualquier plataforma que soporte Java. Jbuilder ha sido desarrollado íntegramente para desarrollos Java, de manera que no se encuentran restos de otro lenguaje. El entorno visual ofrece un entorno seguro para crear aplicaciones multiplataforma, y también se puede alternar entre las herramientas visuales y el código, manteniendo siempre la coherencia entre las mismas.

Ventajas:

- Crea rápidamente aplicaciones basadas en Swing[16].
- Acelera la creación de servicios web.

Desventajas:

- Consume muchos recursos.
- Lento a la hora de compilar los archivos.
- Es privativo.

1.5.7 UML (Unified Modeling Language)

El Lenguaje Unificado de Modelado (UML), es una técnica para la especificación de sistemas en todas sus fases. Este ha sido desarrollado por los más importantes autores en materia de Análisis y Diseño de Sistemas y ha sido usada con éxito en sistemas hechos para toda clase de industrias

[15] Empresa informática fundada en 1982 en California, Estados Unidos.
[16] Biblioteca gráfica para java, incluye cajas de texto, botones y tablas.

alrededor del mundo. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de *software* (20).

UML permite realizar modelos para:

- Visualizar cómo es un sistema o cómo se quiere que sea.
- Especificar la estructura y/o comportamiento de un sistema.
- Hacer una plantilla que guíe la construcción de los sistemas.
- Documentar las decisiones que se han tomado.

Es necesario aclarar que UML es un "lenguaje" para especificar y no un método o un proceso. UML se usa para definir un sistema de *software*; para detallar los artefactos en el sistema; para documentar y construir, es el lenguaje en el que está descrito el modelo. UML se puede usar en una gran variedad de formas para soportar una metodología de desarrollo de *software* pero no especifica en sí mismo qué metodología o proceso usar.

1.5.8 Visual Paradigm 8.0

Es una herramienta CASE de modelado visual capaz de construir todos los tipos de diagramas de UML. Es compatible con una amplia gestión de casos de uso. El *software* de modelado UML ayuda a una más rápida construcción de aplicaciones de buena calidad y menor coste. Permite construir todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (21).

Esta herramienta es ideal para ingenieros de *software*, analistas y arquitectos de sistemas que están interesados en construcción de sistemas a gran escala y necesitan confiabilidad y estabilidad en el desarrollo orientado a objetos.

Algunas de sus **características** son:

- Navegación intuitiva entre la escritura del código y su visualización.
- Potente generador de informes en formato PDF/HTML.
- Sincronización de código fuente en tiempo real.

1.5.9 Persistencia de datos: SQLite

SQLite es una librería escrita en C que implementa un motor de base de datos embebido. Se puede entender como base de datos embebidas aquellas que no inician un servicio independiente de la aplicación (22).

La principal característica de SQLite es su completo soporte para tablas e índices en un único archivo de base de datos, soporte transaccional, rapidez, escaso tamaño y su portabilidad.

Es usado tanto en dispositivos móviles como en aplicaciones web y de escritorio, funciona en ordenadores con poca memoria manteniendo un buen rendimiento.

Características:

- Transacciones ACID *(atómicas, coherentes, aisladas, durables).*
- Autónomo, sin dependencias.
- Compatibilidad con múltiples plataformas: Linux, Windows, entre otros.

1.5.10 Pencil Project

Es una aplicación multiplataforma y se utiliza directamente en el navegador (23). Pencil Project permite realizar bocetos de forma sencilla añadiéndolo a Mozilla Firefox como un *plugin*.

Características:

- Diseño de plantillas y creación de prototipos.
- Documento de múltiples páginas, todo el sitio web en un solo archivo.
- Edición de texto enriquecido.
- Operaciones estándares de dibujo: alineación, orden, escalado, rotación, dimensiones, etc.
- La exportación a HTML, PNG, documento de Openoffice.org, documento de Word y PDF (24).

1.6 Metodología

Un proceso de *software* detallado y completo suele denominarse “metodología” (25). Las metodologías de desarrollo de *software* se basan en una combinación de los modelos de proceso genéricos (cascada, evolutivo, incremental, espiral entre otros). Adicionalmente una metodología debería definir con precisión los artefactos, roles y actividades de los involucrados, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología al proyecto y guías para uso de herramientas de apoyo, entre otras cosas.

Por una parte se tienen aquellas metodologías tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Por otro lado están las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del *software* con iteraciones muy cortas. Este enfoque ha mostrado su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir los tiempos de desarrollo pero manteniendo una alta calidad. A continuación se muestra una tabla comparativa entre estos dos grupos.

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.
Menos énfasis en la arquitectura del *software.*	La arquitectura del *software* es esencial y se expresa mediante modelos.

Tabla 3: Metodologías ágiles vs metodologías tradicionales. (25)

Teniendo en cuenta las posibilidades que brindan las metodologías ágiles con respecto a las tradicionales y a las características que posee el sistema a desarrollar (requisitos cambiantes y

el cliente forma parte del equipo de trabajo) se decide usar como guía para el proceso se construcción del *software* una metodología ágil por lo que se descarta a RUP como alternativa a utilizar.

1.6.1 XP

XP se basa en darle prioridad a los trabajos que dan un resultado directo y que reducen la burocracia que hay alrededor de la programación. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios (26).

XP define un conjunto de buenas prácticas que facilitan el desarrollo de *software* y que se deben tener en cuenta durante todo el proceso de implementación. A continuación se describen algunas (27).

Prácticas básicas de XP:

- Programación en parejas: Requiere que todos los programadores XP escriban su código en parejas, compartiendo una sola máquina.
- Refactorización: Permite a los equipos de programadores XP mejorar el diseño del sistema a través de todo el proceso de desarrollo.
- Diseño simple: Se basa en la filosofía de que el mayor valor de negocio es entregado por el programa más sencillo que cumpla los requerimientos.
- Propiedad colectiva del código: Nadie es el propietario de nada, todos son el propietario de todo. Cualquier integrante puede realizar cambios o mejoras a cualquier parte del código en cualquier momento.
- Pequeñas entregas: Las entregas deben ser lo más pequeñas posibles con una duración no mayor a dos meses. Esto permite que los clientes puedan ver el avance del proyecto y los programadores puedan retroalimentarse de las opiniones de los mismos.
- Estándares de codificación: El equipo de desarrollo define un estándar para el código, dejando de lado estilos de programación particulares. Todos deben conocer y seguir el estándar de manera tal que al final el sistema parezca ser programado por una sola persona.

1.6.2 Scrum

Es una metodología para la gestión y desarrollo de *software* basada en un proceso iterativo e incremental utilizado comúnmente en entornos basados en el desarrollo ágil de *software*. Es sencillo, requiere de poco esfuerzo para comenzar a trabajar, permite el desarrollo, prueba y correcciones de manera rápida, posibilita la entrega de productos en tiempo y con buena calidad.

Aunque solo funciona bien en equipos pequeños y ágiles, se necesita que los miembros del equipo de desarrollo sean experimentados y si un miembro del equipo abandona su puesto puede conllevar grandes problemas (28).

Esta metodología establece que las iteraciones de entrega se realicen de 2 a 4 semanas y se conocen como *sprint*. Al finalizar cada iteración y el cliente habiendo presentado su conformidad con el producto, las tareas correspondientes a dicha iteración no se vuelven a tocar en ningún momento. Scrum es una metodología ágil basada en la administración del proyecto (29).

1.7 Conclusiones parciales

A través del estudio realizado sobre los clientes asociados al gestor de proyectos Redmine, se identificaron requisitos funcionales básicos que servirán como base para el desarrollo de la aplicación, por ejemplo: listado de las tareas ordenadas por prioridad, listado de los proyectos del sistema, listado de los usuarios del sistema y creación de nuevas tareas o peticiones, entre otros.

El estudio realizado sobre las herramientas, tecnologías y metodologías existentes para el desarrollo de la aplicación arrojó los siguientes resultados:

Como lenguaje de programación se seleccionó Java porque la curva de aprendizaje es más corta que la de C++ y similar a la de C#, además de la experiencia que posee el autor con dicho lenguaje. Como IDE de programación se seleccionó Netbeans por ser multiplataforma, libre y gratis, aspecto que lo pone por encima de Jbuilder al ser privativo. Con respecto a Eclipse, este posee una arquitectura basada en *plugins*, lo que lo hace dependiente de los mismos y puede retrasar el proceso de desarrollo. Para el almacenamiento de los datos se usará SQLite ya que facilita la portabilidad de la aplicación y es multiplataforma.

Como metodología de desarrollo de *software* se escogió XP porque el tiempo previsto para el desarrollo de la aplicación es relativamente pequeño. Con respecto a SCRUM, esta define que una vez realizada una iteración o sprint no se vuelven a tocar las funcionalidades realizadas en

dicha iteración, siendo vulnerable en cuanto a cambios en todo el proceso de desarrollo, aspecto en el cual XP es susceptible.

Capítulo 2. Características y diseño del sistema

En el presente capítulo se hace referencia a las fases de exploración y planificación de la metodología XP, siendo la seleccionada para el desarrollo del sistema propuesto. Se describe la propuesta de solución, se muestran las Historias de Usuario (HU) relacionadas con la aplicación y se muestran las tarjetas CRC (Clase o Cargo, Responsabilidad, Colaboración). Además, se generan otros artefactos propios de la metodología de desarrollo de *software* seleccionada.

2.1 Propuesta de solución

La propuesta de solución tiene como objetivo principal apoyar la gestión de las tareas en el gestor de proyectos Redmine. El usuario podrá autenticarse con su usuario y contraseña que lo identifica, alternativamente podrá hacerlo a través de la clave de acceso a la API. Una vez en el sistema, el usuario visualizará todas las tareas o peticiones que fueron asignadas por él y las que le fueron asignadas también. La herramienta permitirá realizar las operaciones básicas CRUD[17] sobre las tareas relacionadas con el usuario. Además, podrá modificar algunos atributos como son: estado, % de realizado, fecha de inicio, fecha de fin y asignado a. El usuario podrá visualizar los proyectos a los que él se encuentra vinculado, pudiendo alternar entre uno y otro en caso de estar relacionado con más de uno.

El sistema notificará al usuario si el servidor del Redmine en el cual se encuentra registrado deja de estar en línea, y permitirá realizar cambios localmente para luego ser enviados al mismo una vez disponible. El chequeo del estado del servidor se realizará mediante un demonio[18](Linux) o servicio (Windows) en segundo plano que verificará cada 20 segundos (tiempo mínimo y personalizable) si se encuentra en línea o no. El sistema contará con un mecanismo de notificación que posibilitará al usuario estar actualizado, pues se realizarán peticiones al servidor en segundo plano con el objetivo de verificar si la información asociada al mismo ha sido modificada o se ha añadido una nueva petición o tarea. Además, permitirá abrir una tarea o petición en el navegador si el usuario lo desea.

[17] Del original en inglés: **C**reate, **R**ead, **U**pdate and **D**elete. Se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software(Crear, Obtener, Actualizar y Borrar).
[18] Tipo especial de proceso informático no interactivo, es decir, que se ejecuta en segundo plano. Nomenclatura usada en sistemas UNIX.

2.2 Lista de reserva

Una vez conocidos todos los conceptos que rodean al objeto de estudio, se puede analizar qué debe hacer el sistema para que se cumplan los objetivos planteados al inicio de este trabajo. Para ello se enumeran a través de la Lista de Reserva las funcionalidades, necesidades y restricciones que el sistema debe cumplir y ser capaz de brindar al usuario. De acuerdo a los objetivos planteados, el sistema debe ser capaz de:

- Autenticar mediante usuario y contraseña.
- Autenticar mediante clave de acceso a la API.
- Mostrar proyectos a los que está vinculado el usuario.
- Mostrar peticiones asignadas al usuario.
- Mostrar peticiones asignadas por el usuario.
- Editar peticiones asignadas al usuario.
- Editar peticiones asignadas por el usuario.
- Mostrar detalles de una petición.
- Abrir una petición en el navegador.
- Guardar configuración.
- Cambiar de proyecto.
- Ordenar peticiones por prioridad.
- Crear una petición.
- Notificar al usuario ante cambios en el servidor.
- Editar la configuración.
- Actualizar los cambios locales.
- Eliminar una petición.

2.3 Aspectos no funcionales del sistema

Los aspectos no funcionales son propiedades o cualidades que el producto debe tener. Son características del mismo que lo hacen atractivo, usable, rápido y confiable.

1. **Software**
    - La aplicación deberá funcionar en los sistemas operativos Windows (XP o superior) y las distribuciones GNU/Linux.
    - Se requiere de la máquina virtual de Java versión 6.25 o superior.

2. **Hardware**
    - Memoria RAM: 512 MB como mínimo para sistemas Windows y 128 MB para sistemas GNU/Linux.
    - Disco duro: 80 GB mínimo para sistemas Windows y 40 para sistemas GNU/Linux.
    - Microprocesador: Pentium 4 como mínimo.
3. **Diseño e implementación**
    - Para el desarrollo de la herramienta se empleará la versión 7.4 o superior del IDE NetBeans.
    - Estandarizar el código de la aplicación basándose en la notación camelCase que propone Sun MicroSystems.
4. **Apariencia o interfaz externa**
    - El diseño de la interfaz debe ser sencillo e intuitivo de manera que facilite su uso al usuario.
5. **Ayudas contextuales**
    - El sistema garantizará que todos los campos que se deben llenar tengan al menos un nombre que describa los datos de entrada.
6. **Portabilidad**
    - El cliente debe permitir ser portable, moverlo de un ordenador a otro sin causar pérdida de información o daños colaterales al usuario.
7. **Aspectos legales**
    - La herramienta una vez desarrollada será de código abierto y licencia GPL.
8. **Seguridad**
    - Los datos sensibles como contraseña y clave de acceso a la API deben ser codificados antes de guardarse en base de datos.

2.4 Exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase

de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología (30).

Los programadores estiman los tiempos de desarrollo en base a esta información, debe quedar claro que las estimaciones realizadas en esta fase son primarias y podrían variar cuando se analicen más en detalle en cada iteración. El resultado es una visión general del sistema y un plazo total estimado.

2.4.1 Historias de usuario (HU)

Las historias de usuario son redacciones "sencillas" mediante la cual el cliente define lo que necesita, aunque tienen el mismo propósito que los casos de uso, éstas son escritas sin terminología técnica (30).

Los desarrolladores deberán hacer una estimación de cuánto tiempo, idealmente, les llevará implementar cada historia de usuario. Las condiciones ideales son aquellas en las que se codifica la historia de usuario sin otras distracciones y sabiendo exactamente qué es lo que hay que implementar. Como resultado deberíamos obtener un periodo ideal de 1, 2 ó 3 semanas.

Respecto a los atributos contenidos en las historias de usuario, no existe un modelo general que defina los mismos, por lo que quedan a libre selección. A continuación se describen los seleccionados por el autor de la presente investigación y se muestran las historias de usuarios principales.

Nombre: Se refiere al nombre de la HU.

Número: Número consecutivo de la HU.

Prioridad: Alta, media o baja.

Riesgo: Alta, media o baja.

Estimación: Tiempo en semanas que durará la implementación de la HU.

Iteración: Iteración a la que corresponde la realización de la HU.

Descripción: Breve descripción de lo que realizará la HU.

Historia de usuario	
Nombre: Autenticar mediante usuario y contraseña.	**Número:** 1
Estimación: 1 semana	**Iteración:** 1
Prioridad: Alta	
Riesgo en Desarrollo: Medio	

Descripción: El sistema debe permitir autenticar los usuarios usando las credenciales básicas de autenticación: usuario y contraseña.

Tabla 4: Autenticar mediante usuario y contraseña.

Historia de usuario	
Nombre: Autenticar mediante clave de acceso a la API.	**Número:** 2
Estimación: ½ semana	**Iteración:** 1
Prioridad: Media	
Riesgo en Desarrollo: Bajo	
Descripción: El sistema debe permitir autenticar los usuarios usando la clave de acceso a la API.	

Tabla 5: Autenticar mediante clave de acceso a la API.

Historia de usuario	
Nombre: Mostrar proyectos a los que está vinculado el usuario.	**Número:** 3
Estimación: ½ semana	**Iteración:** 1
Prioridad: Alta	
Riesgo en Desarrollo: Media	
Descripción: El sistema debe mostrar los proyectos a los que se encuentra vinculado el usuario.	

Tabla 6: Mostrar proyectos a los que está vinculado el usuario.

Historia de usuario	
Nombre: Mostrar peticiones asignadas al usuario.	**Número:** 4
Estimación: ½ semana	**Iteración:** 1
Prioridad: Alta	
Riesgo en Desarrollo: Media	
Descripción: El sistema debe mostrar las peticiones asignadas al usuario ordenadas por prioridad, se mostraran las 25 primeras correspondiente al primer proyecto al que se encuentra vinculado en caso de tener más de uno en secciones de 10.	

Tabla 7: Mostrar peticiones asignadas al usuario.

Historia de usuario	
Nombre: Mostrar peticiones asignadas por el usuario.	**Número:** 12
Estimación: ½ semana	**Iteración:** 3
Prioridad: Alta	
Riesgo en Desarrollo: Media	
Descripción: El sistema debe mostrar las peticiones asignadas por el usuario	

Tabla 8: Mostrar peticiones asignadas por el usuario.

Historia de usuario	
Nombre: Editar peticiones asignadas al usuario.	**Número:** 9
Estimación: 1 semana	**Iteración:** 2
Prioridad: Alta	
Riesgo en Desarrollo: Alta	
Descripción: El sistema debe permitir editar las peticiones asignadas por los usuarios, exactamente los campos: prioridad, estado y % realizado.	

Tabla 9: Editar peticiones asignadas al usuario.

Historia de usuario	
Nombre: Editar peticiones asignadas por el usuario.	**Número:** 11
Estimación: 1 semana	**Iteración:** 3
Prioridad: Alta	
Riesgo en Desarrollo: Alta	
Descripción: El sistema debe permitir editar las peticiones asignadas por los usuarios, exactamente los campos: fecha de inicio, fecha de fin, asignado a, prioridad y estado.	

Tabla 10: Editar peticiones asignadas por el usuario.

Historia de usuario	
Nombre: Mostrar detalles de una petición.	**Número:** 5
Estimación: ½ semana	**Iteración:** 1
Prioridad: Media	
Riesgo en Desarrollo: Media	

Descripción: El sistema debe permitir al usuario visualizar los detalles de una petición mientras presiona sobre ella.

Tabla 11: Mostrar detalles de una petición.

Historia de usuario	
Nombre: Abrir una petición en el navegador.	**Número:** 8
Estimación: ½ semana	**Iteración:** 2
Prioridad: Media	
Riesgo en Desarrollo: Media	
Descripción: El sistema debe permitir al usuario abrir cualquier petición en el navegador desde el cliente.	

Tabla 12: Abrir una petición en el navegador.

Historia de usuario	
Nombre: Guardar configuración.	**Número:** 6
Estimación: 1 semana	**Iteración:** 2
Prioridad: Media	
Riesgo en Desarrollo: Media	
Descripción: El sistema debe guardar los datos del usuario en el momento de autenticarse (usuario, contraseña, clave de acceso API, método de entrada).	

Tabla 13: Guardar configuración.

Historia de usuario	
Nombre: Cambiar de proyecto.	**Número:** 7
Estimación: ½ semana	**Iteración:** 2
Prioridad: Alta	
Riesgo en Desarrollo: Media	
Descripción: El sistema debe permitir al usuario cambiar de un proyecto a otro.	

Tabla 14: Cambiar de proyecto.

Historia de usuario	
Nombre: Ordenar peticiones por prioridad.	**Número:** 10

Estimación: ½ semana	**Iteración:** 3
Prioridad: Alta	
Riesgo en Desarrollo: Media	
Descripción: El sistema debe mostrar las peticiones ordenadas por prioridad.	

Tabla 15: Ordenar peticiones por prioridad.

Historia de usuario	
Nombre: Crear una petición.	**Número:** 16
Estimación: 1 semana	**Iteración:** 4
Prioridad: Alta	
Riesgo en Desarrollo: Media	
Descripción: El sistema debe permitir crear una petición.	

Tabla 16: Crear una petición.

Historia de usuario	
Nombre: Notificar al usuario ante cambios en el servidor.	**Número:** 13
Estimación: 1 semana	**Iteración:** 3
Prioridad: Alta	
Riesgo en Desarrollo: Media	
Descripción: El sistema debe notificarle al usuario ante cambios realizados en el servidor.	

Tabla 17: Notificar al usuario ante cambios en el servidor.

Historia de usuario	
Nombre: Editar la configuración.	**Número:** 14
Estimación: ½ semana	**Iteración:** 4
Prioridad: Media	
Riesgo en Desarrollo: Media	
Descripción: El sistema debe permitir editar los datos iniciales del usuario.	

Tabla 18: Editar la configuración.

Historia de usuario

Nombre: Actualizar los cambios locales.	**Número:** 15
Estimación: 1 semana	**Iteración:** 4
Prioridad: Alta	
Riesgo en Desarrollo: Alta	
Descripción: El sistema debe subir los cambios realizados localmente hacia el servidor.	

Tabla 19: Actualizar los cambios locales.

Historia de usuario	
Nombre: Eliminar una petición.	**Número:** 17
Estimación: ½ semana	**Iteración:** 4
Prioridad: Media	
Riesgo en Desarrollo: Bajo	
Descripción: El sistema debe permitir eliminar una petición.	

Tabla 20: Eliminar una petición.

2.5 Planificación

En esta fase, XP propone un diálogo continuo entre las partes involucradas en el desarrollo del producto, incluyendo al cliente como factor principal. Las entradas a esta fase lo constituyen las historias de usuario (HU) realizadas anteriormente, a partir de las mismas los programadores evalúan el tiempo de desarrollo de cada una de ellas y elabora un plan de entregas que puede ser cambiado cuando las situaciones lo requieran (31).

El objetivo de este plan de entregas es definir las HU que son más importantes desde el punto de vista del cliente para conformar una entrega, se ordenan por prioridad teniendo en cuenta este aspecto y dejando para el final las de menor importancia.

Historia de usuario	Iteración 1	Iteración 2	Iteración 3	Iteración 4
Autenticar mediante usuario y contraseña.	cv_1.0	Realizado	Realizado	Realizado

Autenticar mediante clave de acceso a la API.	cv_1.0	Realizado	Realizado	Realizado
Mostrar proyectos a los que está vinculado el usuario.	cv_1.0	Realizado	Realizado	Realizado
Mostrar peticiones asignadas al usuario.	cv_1.0	Realizado	Realizado	Realizado
Mostrar detalles de una petición.	cv_1.0	Realizado	Realizado	Realizado
Guardar configuración.	----	cv_2.0	Realizado	Realizado
Cambiar de proyecto.	---	cv_2.0	Realizado	Realizado
Abrir una petición en el navegador.		cv_2.0	Realizado	Realizado
Editar peticiones asignadas al usuario (estado, % realizado).	---	cv_2.0	Realizado	Realizado
Ordenar peticiones por prioridad.	---	---	cv_3.0	Realizado
Mostrar peticiones asignadas por el usuario.	---	---	cv_3.0	Realizado
Editar peticiones asignadas por el usuario.	---	---	cv_3.0	Realizado
Notificar al usuario ante cambios en el servidor.	---	---	cv_3.0	Realizado
Editar la configuración.	---	---	---	cv_4.0
Actualizar los cambios locales en el servidor.	---	---	---	cv_4.0
Crear una petición.	---	---	---	cv_4.0
Eliminar una petición.	---	---	---	cv_4.0

Tabla 21: Plan de entrega.

2.5.1 Iteraciones

Una vez realizado el plan de entrega es posible realizar el plan de iteraciones, donde a cada iteración le corresponde un período de tiempo de desarrollo de 1 a 3 semanas. De esta forma, un proyecto se divide en una docena de iteraciones aproximadamente sin violar el orden de la realización de las HU en cada iteración (31).

Iteraciones	Orden de las HU a implementar	Tiempo de trabajo
	Autenticar mediante usuario y contraseña Autenticar mediante clave de acceso a la API	

Iteración 1	Mostrar proyectos a los que está vinculados el usuario Mostrar peticiones asignadas al usuario Mostrar detalles de una petición	3 semanas
Iteración 2	Guardar la configuración Cambiar de proyecto Abrir una petición en el navegador Editar peticiones asignadas al usuario	3 semanas
Iteración 3	Ordenar peticiones por prioridad Mostrar peticiones asignadas por el usuario Editar peticiones asignadas por el usuario Notificar al usuario ante cambios en el servidor	3 semanas
Iteración 4	Editar la configuración Actualizar los cambios locales en el servidor Crear una petición Eliminar una petición	3 semanas

Tabla 22: Plan de iteraciones.

2.6 Tarjetas CRC

Para poder diseñar el sistema como un equipo se debe cumplir con tres principios: Cargo o Clase, Responsabilidad y Colaboración (CRC). Las tarjetas CRC permitirán desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. Las tarjetas CRC permiten que el equipo completo contribuya en la tarea del diseño. Una tarjeta CRC representa un objeto. El nombre de la clase se coloca a modo de título en la tarjeta, las responsabilidades se colocan a la izquierda, y las clases que se implican en cada responsabilidad a la derecha, en la misma línea que su requerimiento correspondiente (32). A continuación, se muestran las tarjetas CRC correspondientes a la propuesta de solución.

Nombre de la clase: redmineManager	
Responsabilidades	**Colaboraciones**
update	Transport, Identifiable
validate	TimeEntry, Identifiable

getUsers	Transport, User, BasicNameValuePair
getUserById	
getCurrentUser	
getIssues	
getIssuesByAuthor	
getStatuses	Transport, IssueStatus
shutdown	Transport

Tabla 23: Tarjeta CRC: redmineManager

Nombre de la clase: localConexion	
Responsabilidades	**Colaboraciones**
guardarCredencial	localUsuario
cargarCredencial	
existeUsuario	
actualizarUsuario	
guardarProyecto	localProyecto
cargarProyectos	
existeUsuarioProyecto	
actualizarProyecto	
guardarPeticion	localPeticion
cargarPeticion	

Tabla 24: Tarjeta CRC: localConexion

Nombre de la clase: redmineTransport	
Responsabilidades	**Colaboraciones**
getCurrentUser	User, NameValuePair
updateObject	NameValuePair, Identifiable
deleteObject	Identifiable

Tabla 25: Tarjeta CRC: redmineTransport

Nombre de la clase: localFuncion

Responsabilidades	**Colaboraciones**
LoadCredential	localUsuario, localConexion
OpenBrowser	localUsuario

Tabla 26: Tarjeta CRC: localFuncion

Nombre de la clase: localLogin	
Responsabilidades	**Colaboraciones**
getUsuario	redmineManager, localUsuario
getRedmine	redmineManager
isAutenticado	redmineManager, localUsuario,User

Tabla 27: Tarjeta CRC: localLogin

2.7 Conclusiones parciales

En el presente capítulo se documentaron las HU, estimando para cada una el tiempo de implementación aproximado por los desarrolladores, generando un plan de entrega que permite obtener en versiones pequeñas de 4 iteraciones un producto funcional durante el transcurso de su desarrollo. Por otro lado, se realizó un plan de iteraciones donde se define para cada iteración el orden de las HU a implementar en el proceso de desarrollo de la aplicación y el tiempo por cada iteración, arrojando como resultado 4 iteraciones de 3 semanas cada una, para un total de 12 semanas, aproximadamente 3 meses. Por último se muestran las tarjetas CRC correspondientes a cada clase del sistema, especificando para cada una de ellas sus responsabilidades y las clases que colaboran con la misma.

Capítulo 3. Implementación y prueba

3.1 Arquitectura del sistema propuesto

Arquitectura de 2 capas (Cliente-Servidor):

Esta arquitectura consiste básicamente en un cliente que realiza peticiones a otro programa (el servidor) que le da respuesta. Aunque esta idea se puede aplicar a programas que se ejecutan sobre una sola computadora es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

La siguiente imagen muestra como está estructurada la arquitectura cliente-servidor.

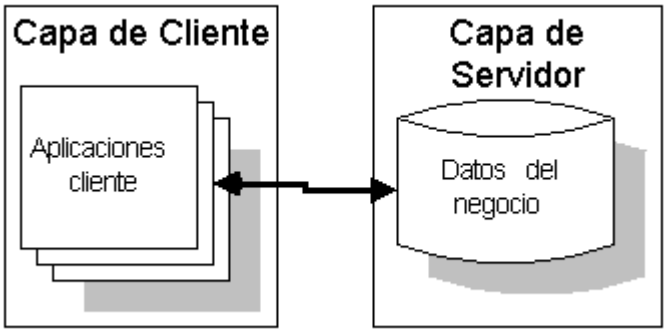

Figura 3: Representación arquitectura cliente – servidor

La lógica de la aplicación se encuentra repartida en el cliente y el servidor, pues a través de una única conexión se realiza el envío y recepción de varios datos permitiendo ganar en rendimiento (33).

Ventajas:

- Distribución de aplicaciones.
- Acceso independiente del lugar físico.
- Escalabilidad.

Desventajas:

- Aumenta el tráfico de red.
- Falta de robustez.

3.2 Diagrama Entidad Relación (DER)

El diagrama entidad relación es la descripción de la organización de una base de datos, siendo una representación gráfica orientada a la obtención de la estructura de datos mediante métodos. A continuación se muestra el DER obtenido a través de la herramienta Visual Paradigm para UML:

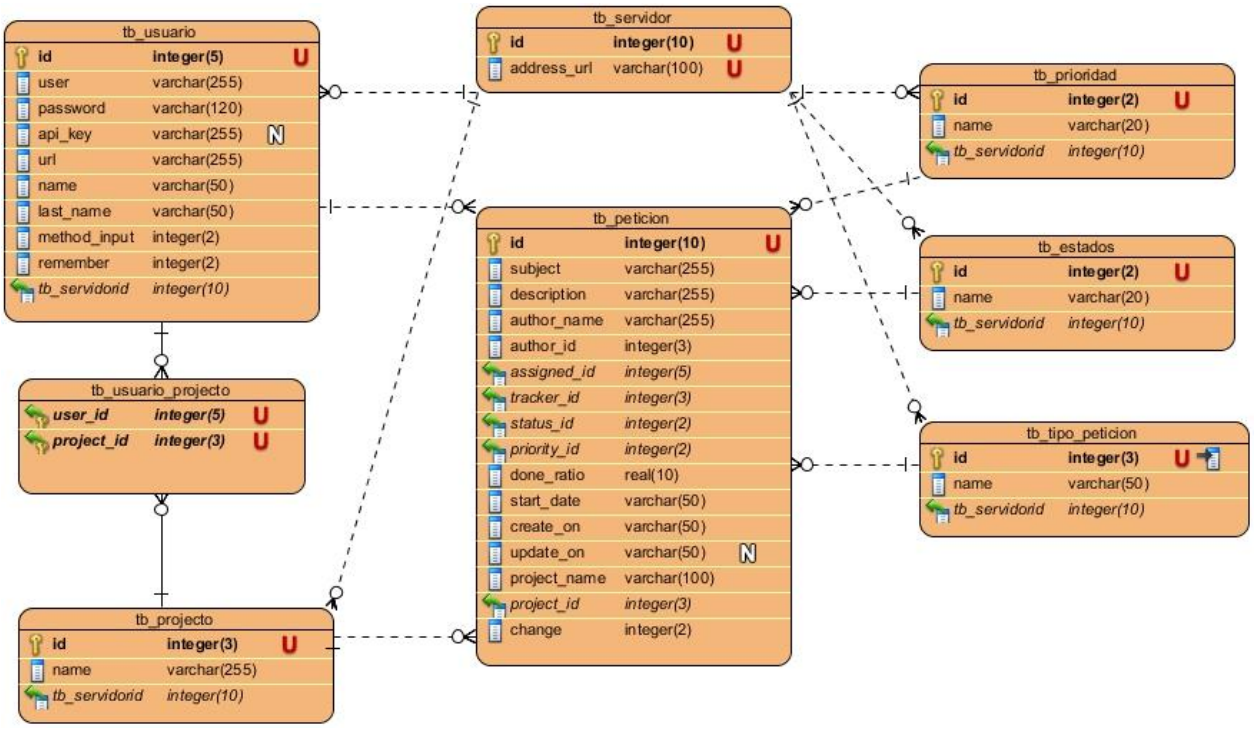

Figura 4: Diagrama entidad relación

A continuación se muestra una descripción de las tablas que conforman la base de datos:

tb_usuario: Es la encargada de almacenar los datos relacionados con el usuario, posee una relación con la tabla petición y con la tabla proyecto. Con la tabla proyecto presenta una relación de mucho a mucho dando origen a una tercera tabla, pues un usuario puede estar en varios proyectos y en un proyecto pueden haber varios usuarios. Respecto a la tabla petición, esta presenta una relación de 1 a mucho, ya que un usuario puede tener asignadas muchas peticiones.

tb_proyecto: Es la encargada de almacenar los datos relacionados con los proyectos en el sistema, posee una relación de mucho a mucho con la tabla usuario y una relación de 1 a mucho con la tabla petición, pues en un proyecto pueden haber muchas peticiones.

tb_peticion: Es la encargada de almacenar los datos relacionados con las peticiones, posee cuatro relaciones de 1 a mucho con las siguientes tablas: usuario, proyecto, estados y prioridad.

tb_prioridad: Es la encargada de almacenar los datos relacionados con las prioridades de las peticiones, es por esto que posee una relación de 1 a mucho con dicha tabla.

tb_estados: Es la encargada de almacenar los datos relacionados con los estados de las peticiones, pues una petición posee un solo estado, mientras que un estado puede estar en muchas peticiones.

tb_usuario_proyecto: Es la encargada de establecer la relación entre los usuarios y los proyectos a los que pertenece, es fruto de la relación de mucho a mucho entre las tablas usuario y proyecto, pues en un proyecto pueden haber muchos usuarios y un usuario puede estar en muchos proyectos.

tb_tipo_peticion: Esta tabla es la encargada de almacenar los tipos de peticiones, por ejemplo: tarea, errores, etc.

tb_servidor: Es la encargada de almacenar las distintas direcciones de los servidores de Redmine que proporcionará el usuario. Está relacionada con casi todas las tablas en la base de datos excepto con la tabla peticiones.

3.3 Estilos y estándares de codificación

En el proceso de implementación de un *software* es beneficioso el uso de estándares de codificación, pues permiten que en cualquier momento puedan hacerse modificaciones sin conocimiento previo de lo que se modifica. Además, los estándares de programación mantienen el código legible para los miembros del equipo. Seguidamente se muestran los estándares de codificación usados en la implementación de la aplicación.

- Sufija las interfaces con Interface.
- El código se encuentra organizado de la siguiente forma: 4 espacios para indentación, 8 espacios para tabuladores y ajuste de línea.
- Las llaves en las clases y los métodos van después del nombre de la clase o del método respectivamente.
- Los nombres de los paquetes y clases locales comienzan por local<nombre_paquete> *(no incluye las clases de las librerías)*.
- Los comentarios multilíneas se escriben comenzando por /* y terminando */, los de una sola línea comienzan con los caracteres //.
- Los métodos serán comentariados para luego generar la documentación asociada a cada uno a través del IDE Netbeans.

- Se utiliza la notación lowerCamelCase de CamelCase[19] para las variables compuestas.
- Los atributos de las clases son privados, el acceso y modificación a ellos se realiza mediante los métodos get<atributo> y set<atributo> respectivamente.

3.4 Tareas de ingeniería

XP plantea que las HU definidas en las primeras fases de desarrollo sean implementadas en el orden que fueron planificadas. Para facilitar la implementación de las HU, XP propone que sean agrupadas en tareas de ingeniería y asignadas a sus correspondientes desarrolladores. Además plantea que las mismas sean descritas en lenguaje técnico. A continuación se muestran las tareas de ingeniería relacionadas con la implementación de la aplicación.

Tarea	
No. de tarea: 1	**No. de HU:** 1, 2, 6
Nombre tarea: Interfaz login.	
Tipo de tarea: Desarrollo	**Estimación:** 1 día
Programador responsable: Remberto Ramos Rodríguez.	
Descripción: En esta tarea se llevará a cabo el diseño correspondiente a la interfaz de login de la aplicación.	

Tabla 28: Tarea de ingeniería: Interfaz login

Tarea	
No. de tarea: 2	**No. de HU:** 1
Nombre tarea: autenticar usuario/contraseña.	
Tipo de tarea: Desarrollo	**Estimación:** 2 días
Programador responsable: Remberto Ramos Rodríguez.	
Descripción: Implementar la funcionalidad autenticar mediante usuario/contraseña en la clase localConexion.	

Tabla 29: Tarea de ingeniería: autenticar usuario/contraseña

Tarea	
No. de tarea: 3	**No. de HU:** 2
Nombre tarea: autenticar api/key.	
Tipo de tarea: Desarrollo	**Estimación:** 2 días
Programador responsable: Remberto Ramos Rodríguez.	

[19] Estilo de escritura que se aplica a frases o palabras compuestas.

Descripción: Implementar la funcionalidad autenticar mediante la llave de acceso a la api (api key).

Tabla 30: Tarea de ingeniería: autenticar api/key

Tarea	
No. de tarea: 4	**No. de HU:** 3, 4, 5, 6, 7, 12, 17
Nombre tarea: interfaz principal.	
Tipo de tarea: Desarrollo	**Estimación:** 3 días
Programador responsable: Remberto Ramos Rodríguez.	
Descripción: Diseñar las interfaz principal del sistema.	

Tabla 31: Tarea de ingeniería: interfaz principal

Tarea	
No. de tarea: 5	**No. de HU:** 3, 7
Nombre tarea: cargar proyecto.	
Tipo de tarea: Desarrollo	**Estimación:** 3 días
Programador responsable: Remberto Ramos Rodríguez.	
Descripción: Implementar la funcionalidad cargar proyecto.	

Tabla 32: Tarea de ingeniería: cargar proyecto

Tarea	
No. de tarea: 6	**No. de HU:** 4,12
Nombre tarea: cargar peticiones.	
Tipo de tarea: Desarrollo	**Estimación:** 3 días
Programador responsable: Remberto Ramos Rodríguez.	
Descripción: Implementar la funcionalidad cargar peticiones.	

Tabla 33: Tarea de ingeniería: cargar peticiones

Tarea	
No. de tarea: 7	**No. de HU:** 5
Nombre tarea: detalles de petición	
Tipo de tarea: Desarrollo	**Estimación:** 2 días
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Mostrar detalles de una petición.	

Tabla 34: Tarea de ingeniería: detalles de petición

Tarea	
No. de tarea: 8	**No. de HU:** 6
Nombre tarea: guardar configuración	
Tipo de tarea: Desarrollo	**Estimación:** 3 días
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Implementar la funcionalidad guardar configuración del usuario.	

Tabla 35: Tarea de ingeniería: guardar configuración

Tarea	
No. de tarea: 9	**No. de HU:** 9, 11
Nombre tarea: interfaz editar petición	
Tipo de tarea: Desarrollo	**Estimación:** 3 días
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Realizar las interfaces para la edición de las peticiones.	

Tabla 36: Tarea de ingeniería: interfaz editar petición

Tarea	
No. de tarea: 10	**No. de HU:** 9
Nombre tarea: editar petición	
Tipo de tarea: Desarrollo	**Estimación:** 3 días
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Implementar la funcionalidad editar petición asignada al usuario.	

Tabla 37: Tarea de ingeniería: editar petición

Tarea	
No. de tarea: 11	**No. de HU:** 8
Nombre tarea: Navegador	
Tipo de tarea: Desarrollo	**Estimación:** 1 ½ días
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Implementar la funcionalidad para abrir una petición en el navegador.	

Tabla 38: Tarea de ingeniería: navegador

Tarea	
No. de tarea: 12	**No. de HU:** 10
Nombre tarea: ordenar peticiones	

Tipo de tarea: Desarrollo	**Estimación:** 1 días
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Implementar la funcionalidad ordenar peticiones relacionadas con los usuarios.	

Tabla 39: Tarea de ingeniería: ordenar peticiones

Tarea	
No. de tarea: 13	**No. de HU:** 11
Nombre tarea: editar petición	
Tipo de tarea: Desarrollo	**Estimación:** 3 días
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Implementar la funcionalidad editar petición asignada por el usuario	

Tabla 40: Tarea de ingeniería: editar petición

Tarea	
No. de tarea: 14	**No. de HU:** 13
Nombre tarea: notificaciones	
Tipo de tarea: Desarrollo	**Estimación:** 3 días
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Implementar las clases necesarias para establecer el sistema de notificación de la aplicación.	

Tabla 41: Tarea de ingeniería: notificaciones

Tarea	
No. de tarea: 15	**No. de HU:** 14
Nombre tarea: interfaz configuración	
Tipo de tarea: Desarrollo	**Estimación:** 2 día
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Realizar el diseño de la interfaz para editar la configuración del usuario y la funcionalidad asociada a la misma.	

Tabla 42: Tarea de ingeniería: interfaz configuración

Tarea	
No. de tarea: 16	**No. de HU:** 16
Nombre tarea: interfaz nueva petición	
Tipo de tarea: Desarrollo	**Estimación:** 1 día
Programador responsable: Remberto Ramos Rodríguez	

Descripción: Realizar el diseño de la interfaz para la creación de una nueva petición

Tabla 43: Tarea de ingeniería: interfaz nueva petición

Tarea	
No. de tarea: 17	**No. de HU:** 3, 7
Nombre tarea: cargar usuarios	
Tipo de tarea: Desarrollo	**Estimación:** 1 día
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Implementar la funcionalidad cargar usuarios del sistema	

Tabla 44: Tarea de ingeniería: cargar usuarios

Tarea	
No. de tarea: 18	**No. de HU:** 15
Nombre tarea: Demonio	
Tipo de tarea: Desarrollo	**Estimación:** 3 días
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Implementar un demonio para el chequeo de las distintas acciones al servidor.	

Tabla 45: Tarea de ingeniería: demonio

Tarea	
No. de tarea: 19	**No. de HU:** 15
Nombre tarea: subir cambios	
Tipo de tarea: Desarrollo	**Estimación:** 2 días
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Implementar la funcionalidad subir cambios para el servidor.	

Tabla 46: Tarea de ingeniería: subir cambios

Tarea	
No. de tarea: 20	**No. de HU:** 17
Nombre tarea: eliminar petición	
Tipo de tarea: Desarrollo	**Estimación:** 2 días
Programador responsable: Remberto Ramos Rodríguez	
Descripción: Implementar la funcionalidad para la eliminar una petición.	

Tabla 47: Tarea de ingeniería: eliminar petición

3.5 Pruebas

En esta fase el cliente decide si acepta o no el producto final. Se realizan las pruebas al producto para determinar y corregir errores que pueden formar parte de una nueva versión o pasar a ser corregidos en la versión actual.

3.5.1 Pruebas de aceptación

Las pruebas de aceptación son elaboradas teniendo en cuenta las HU, las mismas no se consideran terminadas hasta que no pasen correctamente todas las pruebas de aceptación propuestas. Para cada HU es necesario realizar una o varias pruebas para asegurarse de ha sido implementada correctamente y así poder documentar sin problema alguno. Destacar que es el cliente el encargado de realizar las pruebas y verificar el resultado de las mismas. A continuación se muestran las pruebas realizadas.

Caso de prueba de aceptación	
Código: *HU1_CP1*	**Historia de usuario:** *1*
Nombre: *Autenticar mediante usuario y contraseña.*	
Condiciones de ejecución: *El usuario debe estar registrado en el servidor de gestión de proyectos Redmine.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: Seleccionar el botón usuario/contraseña, llenar los campos necesarios y presionar el botón de aceptar.*	
Resultado esperado: *El usuario es autenticado correctamente.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 48: Caso de prueba de aceptación #1.

Caso de prueba de aceptación	
Código: *HU2_CP1*	**Historia de usuario:** *2*
Nombre: *Autenticar mediante llave de acceso a la API*	
Condiciones de ejecución: *El usuario debe estar registrado en el servidor de gestión de proyectos Redmine.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: Seleccionar el botón API key, llenar los campos necesarios y presionar el botón de aceptar.*	
Resultado esperado: *El usuario es autenticado correctamente.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 49: Caso de prueba de aceptación #2.

Caso de prueba de aceptación	
Código: *HU3_CP1*	**Historia de usuario:** *3*
Nombre: *Visualizar proyectos*	
Condiciones de ejecución: *El usuario debe haberse autenticado previamente.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: Autenticarse y visualizar los proyectos vinculados en el lado superior derecho de la aplicación.*	
Resultado esperado: *El usuario visualice los proyectos vinculados a él.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 50: Caso de prueba de aceptación #3.

Caso de prueba de aceptación	
Código: *HU4_CP1*	**Historia de usuario:** *4*
Nombre: *Ver peticiones asignadas al usuario*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: Seleccionar el botón asignadas a mí y observar en la tabla las peticiones correspondientes.*	
Resultado esperado: *Se muestren las peticiones asignadas al usuario.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 51: Caso de prueba de aceptación #4.

Caso de prueba de aceptación	
Código: *HU5_CP1*	**Historia de usuario:** *5*
Nombre: *Mostrar detalles de una petición.*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: Seleccionar una petición en el listado mostrado, observar los detalles de la misma en la parte derecha.*	
Resultado esperado: *Se muestran los detalles de la petición seleccionada.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 52: Caso de prueba de aceptación #5.

Caso de prueba de aceptación

Código: *HU6_CP1*	**Historia de usuario:** *6*
Nombre: *Guardar la configuración del usuario.*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: Llenar los campos necesarios en el proceso de autenticación y presionar aceptar.*	
Resultado esperado: *Los datos del usuario son insertados en la base de datos.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 53: Caso de prueba de aceptación #6.

Caso de prueba de aceptación	
Código: *HU7_CP1*	**Historia de usuario:** 7
Nombre: *Cambiar de proyecto.*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: Para cambiar de proyecto el usuario debe presionar el componente en el lado superior derecho y seleccionar otro proyecto.*	
Resultado esperado: *El proyecto es cambiado correctamente.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 54: Caso de prueba de aceptación #7.

Caso de prueba de aceptación	
Código: *HU8_CP1*	**Historia de usuario:** *8*
Nombre: *Abrir petición en el navegador*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: El usuario debe seleccionar una petición en la lista mostrada y presionar el botón con el icono de Mozilla Firefox.*	
Resultado esperado: *La petición es abierta en el navegador correctamente.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 55: Caso de prueba de aceptación #8.

Caso de prueba de aceptación	
Código: *HU9_CP1*	**Historia de usuario:** *9*
Nombre: *Editar peticiones asignadas al usuario.*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	

Entrada/ Pasos de ejecución: *Pasos de ejecución: Seleccionar una petición y presionar editar, llenar los campos correspondientes y presionar el botón editar.*
Resultado esperado: *Los cambios hechos son guardados correctamente.*
Evaluación de la prueba: *Prueba satisfactoria*

Tabla 56: Caso de prueba de aceptación #9.

Caso de prueba de aceptación	
Código: *HU10_CP1*	**Historia de usuario:** *10*
Nombre: *Ordenar peticiones por prioridad*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: El usuario debe visualizar las peticiones ordenadas según la prioridad.*	
Resultado esperado: *Las peticiones ordenadas por prioridad.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 57: Caso de prueba de aceptación #10.

Caso de prueba de aceptación	
Código: *HU11_CP1*	**Historia de usuario:** *11*
Nombre: *Editar peticiones asignadas por el usuario.*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: Seleccionar una petición y presionar el botón editar, llenar los campos correspondientes y presionar aceptar.*	
Resultado esperado: *Los cambios son guardados correctamente.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 58: Caso de prueba de aceptación #11.

Caso de prueba de aceptación	
Código: *HU12_CP1*	**Historia de usuario:** *12*
Nombre: *Mostrar las peticiones asignadas por el usuario.*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: El usuario debe seleccionar el botón de asignadas por mí y el sistema debe mostrar las peticiones asignadas por el mismo.*	

Resultado esperado: *El sistema muestra las peticiones asignadas por el usuario.*
Evaluación de la prueba: *Prueba satisfactoria*

Tabla 59: Caso de prueba de aceptación #12.

Caso de prueba de aceptación	
Código: *HU13_CP1*	**Historia de usuario:** *13*
Nombre: *Notificar al usuario ante cambios.*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: Cambios en el servidor.*	
Resultado esperado: *Se muestra un mensaje al usuario informándole que ha habido cambios en el servidor.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 60: Caso de prueba de aceptación #13.

Caso de prueba de aceptación	
Código: *HU14_CP1*	**Historia de usuario:** *14*
Nombre: *Editar la configuración*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: El usuario debe presionar el botón de editar usuario y realizar los cambios necesarios.*	
Resultado esperado: *Los datos del usuario son cambiados correctamente.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 61: Caso de prueba de aceptación #14.

Caso de prueba de aceptación	
Código: *HU15_CP1*	**Historia de usuario:** *15*
Nombre: *Actualizar los cambios locales.*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: El usuario realiza cambios locales y el sistema se encarga de enviarlos al servidor.*	
Resultado esperado: *Los cambios locales son enviados correctamente al servidor.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 62: Caso de prueba de aceptación #15.

Caso de prueba de aceptación	
Código: *HU16_CP1*	**Historia de usuario:** 16
Nombre: *Crear una petición.*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: El usuario debe seleccionar crear una petición, llenar los campos necesarios y presionar aceptar.*	
Resultado esperado: *La petición es creada correctamente.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 63: Caso de prueba de aceptación #16.

Caso de prueba de aceptación	
Código: *HU17_CP1*	**Historia de usuario:** *17*
Nombre: *Eliminar una petición.*	
Condiciones de ejecución: *El usuario debe estar autenticado.*	
Entrada/ Pasos de ejecución: *Pasos de ejecución: Debe seleccionar una petición y presionar el botón de eliminar.*	
Resultado esperado: *La petición es eliminada correctamente.*	
Evaluación de la prueba: *Prueba satisfactoria*	

Tabla 64: Caso de prueba de aceptación #17.

3.5.2 Resultados de las pruebas

Una vez terminada cada iteración, se realizaron las pruebas planificadas para asegurar el estado del producto durante la implementación. La siguiente tabla muestra el resultado de las mismas.

	Iteración 1	***Iteración 2***	***Iteración 3***	***Iteración 4***
No conformidades	***2***	***1***	***2***	***3***

Tabla 65: No conformidades por iteración.

Las no conformidades identificadas fueron corregidas satisfactoriamente, garantizando la aceptación del producto por parte del cliente.

3.6 Conclusiones parciales

En este capítulo se abordaron temas referentes al proceso de implementación y prueba de la aplicación. Se describió la arquitectura del sistema y se generó el DER asociado a la aplicación. Se definieron los estilos y estándares de codificación usados durante el desarrollo del sistema, permitiendo la limpieza y uniformidad en el código. Además, se formularon las pruebas de aceptación concluyendo con resultados satisfactorios dándole solución a las no conformidades identificada y se generaron las tareas de ingeniería para dar solución a las HU. Con el fin de este capítulo se da por terminada la propuesta que trae este trabajo.

Conclusiones generales

Luego de concluida la investigación, teniendo en cuenta los resultados obtenidos, se presentan las siguientes conclusiones:

- Se realizó un estudio de las soluciones similares existentes; identificando las ventajas, desventajas y puntos en común para enriquecer el proceso de investigación.
- Se realizó un estudio de las herramientas disponibles para el desarrollo, seleccionando las más adecuadas según algunos aspectos como: curva de aprendizaje, licencia y dominio del equipo de desarrollo, entre otros.
- Se obtuvo como resultado una aplicación desarrollada completamente con tecnologías y herramientas libres, que cumple con todos los requerimientos especificados y buenas prácticas de desarrollo de software.
- Se realizaron pruebas de aceptación al sistema obteniendo salidas positivas. Los resultados arrojados afirman que la propuesta de solución cumple con las expectativas del cliente, pues resuelve el problema planteado por el mismo.
- Con la elaboración de una aplicación que facilite el proceso de gestión de las tareas en el gestor de proyectos Redmine, se ha dado solución al problema planteado y por consiguiente se ha cumplido el objetivo general.

Recomendaciones

- Continuar con el estudio de los sistemas de gestión de tareas con vistas a lograr una mayor experiencia para las futuras mejoras de la propuesta.
- Optimizar el software con la implementación de otras funcionalidades como pueden ser:

    1. Gestión de usuarios.
    2. Gestión de proyectos.

- Crear el manual de usuario para un mejor entendimiento de las funcionalidades que posee la aplicación.

Referencias Bibliográficas.

1. Definición de Sistema informático. [En línea] [Citado el: 3 de mayo de 2014.] http://www.alegsa.com.ar/Dic/sistema%20informatico.php.

2. Redmine. [En línea] [Citado el: 18 de Enero de 2014.] http://www.tecnologiapyme.com/software/redmine-gestor-de-proyectos-de-codigo-libre-para-nuestras-.

3. Romero Cortina, Delquis. Gestión De Proyectos. [En línea] [Citado el: 3 de mayo de 2014.] http://www.slideshare.net/DELQUIS/gestion-de-proyectos-2390454.

4. Redmine. *Redmine-gestor-de-proyectos.pdf.* [En línea] [Citado el: 18 de Enero de 2014.] http://rooteando.com/wp-content/uploads/kalins-pdf/singles/redmine-gestor-de-proyectos.pdf.

5. Fielding, Roy Tomas. *Architectural Styles and the Design of Network -based Software Architectures.* 2000.

6. Rest api - Redmine. [En línea] [Citado el: 11 de Mayo de 2014.] http://www.redmine.org/projects/redmine/wiki/Rest_api.

7. Definición ABC. [En línea] [Citado el: 10 de Enero de 2014.] http://www.definicionabc.com/general/cliente.php.

8. Sistema de Gestión de Tareas. [En línea] [Citado el: 3 de mayo de 2014.] https://extranet.mcu.es/gestdomus/queEsElGestor.asp.

9. Redmine Client. [En línea] [Citado el: 15 de diciembre de 2013.] http://redmineclient.sourceforge.net/.

10. RedMinerDroid. [En línea] [Citado el: 15 de diciembre de 2013.] https://play.google.com/store/apps/details?id=com.vhqlabs.redminerdroid&hl=es_419.

11. JRedmine 1.4. [En línea] [Citado el: 18 de diciembre de 2013.] http://www.nuiton.org/news/334.

12. Garcia, Frank. Historia de Java. [En línea] [Citado el: 15 de Enero de 2014.] http://zarza.usal.es/~fgarcia/doc/tuto2/I_2.html.

13. MEETING JAVA. [En línea] [Citado el: 3 de mayo de 2014.] http://meetingjava.blogspot.com/2006/08/ventajas-y-desventajas-del-java.html.

14. Rodriguez Guerra, Marcia Leidis. Análisis, Diseño e Implementación del Sistema Estudio de Contrarios para el Voleibol. [En línea] 2009. [Citado el: 3 de mayo de 2014.] http://repositorio_institucional.uci.cu//jspui/handle/ident/TD_2282_09.

15. Palanca, Moisés. C# (C Sharp). [En línea] 5 de octubre de 2012. [Citado el: 3 de mayo de 2014.] http://csharpyfoxpro.blogspot.com/2012/10/noticia-futuro-de-c.html.

16. Bastard. Mi lenguaje de programación de preferencia c++. [En línea] [Citado el: 3 de mayo de 2014.] http://www.slideshare.net/bastard1/mi-lenguaje-de-programacion-de-preferencia-c#,Basic C++ presentation.

17. NetBeans IDE. [En línea] [Citado el: 18 de diciembre de 2013.] http://www.genbetadev.com/herramientas/netbeans-1.

18. Ganma, Erich . *Contributing to Eclipse.* 2003.

19. Definición de Jbuilder. [En línea] [Citado el: 24 de enero de 2014.] http://www.alegsa.com.ar/Dic/jbuilder.php.

20. Introducción a UML. [En línea] [Citado el: 10 de febrero de 2014.] http://docs.kde.org/stable/es/kdesdk/umbrello/uml-basics.html.

21. Visual Paradigm. [En línea] [Citado el: 18 de diciembre de 2013.] http://www.visualparadigm.com/.

22. SQLite | Usemos Software Libre. [En línea] [Citado el: 28 de marzo de 2014.] http://usemossoftwarelibre.wordpress.com/cc/tutorial-sqlite-en-espanol/.

23. Figueroa, Roberth G y Solís, Camilo. *Metodologías Tradicionales Vs. Metodologías Ágiles.* 2009.

24. Beck, Kent. *Extreme Programming Explained Embrace Change, Pearson Education.* 1999.

25. —. *Extreme Programming explined. 1ra edición.* 2000.

26. Scrum Methodology & Agile Scrum Methodologies. [En línea] [Citado el: 24 de febrero de 2014.] http://scrummethodology.com/.

27. Hgal, Debora. Diferencias entre scrum y xp. [En línea] [Citado el: 4 de mayo de 2014.] http://www.slideshare.net/deborahgal/diferencias-entre-scrum-y-xp-2219336.

28. CALABRIA, Luis y PÍRIZ, Pablo. *Metodología XP.Cátedra de Ingeniería de Software.* 2003.

29. Letelier, Patricio y Penadés, Carmen. *Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* Valencia : Valencia.

30. Beck, Kent y Fowler, Martin . *Planning Extreme Pro -gramming. AddisonWesley.*

31. Quinodóz, Prof. Carolina. [En línea] [Citado el: 12 de mayo de 2014.] http://profecarolinaquinodoz.com/principal.

Anexo 1

Entrevista:

Nombre y apellidos: Jorge Martínez Padrón

Objetivo # 1: Saber si existen aplicaciones para la gestión de tareas a nivel de universidad.

Objetivo # 2: Definir las funcionalidades básicas necesarias que permitan darle solución al problema actual.

Aspectos a encuestar

1. ¿Conoce usted de aplicaciones para la gestión de tareas en la universidad? ¿Mencione algunas? ¿Has hecho uso de alguna en específico, cuál?
2. ¿Cuáles son las funcionalidades imprescindibles a implementar?