Universidad de las Ciencias Informáticas.
Facultad #-3

Título: Plugin para la gestión de pruebas en la herramienta REDMINE

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

Autores: Alejandro Alvarez Hernández
Humberto Rodríguez Peláez

Tutores: Ing. Giselle Almeida González
Ing. Iliannis Pupo Leyva

Ciudad de la Habana, Junio de 2011
"Año 53 de la Revolución"

2011

" ... Si los jóvenes fallan, todo fallará. Es mi más profunda convicción que la juventud cubana luchará por impedirlo. Creo en ustedes."

23 junio 2007

Declaración de autoría

Declaramos ser los únicos autores de la presente tesis y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso de ella para lo que necesite, cediéndole de esta forma los derechos patrimoniales de la misma, con carácter exclusivo.

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

Autores:

Alejandro Alvarez Hernández

_____

Humberto Rodríguez Peláez

_____

Tutores:

Ing. Giselle Almeida González

_____

Ing. Iliannis Pupo Leyva

_____

Datos de contacto

Síntesis de los tutores:

- Ing. Giselle Almeida González

Graduada de Ingeniero en Ciencias Informáticas en la UCI en el 2008. A partir de ese momento se incorpora como Analista en la línea de Costos y Procesos en el proyecto ERP Cuba en el Centro para la Informatización de Gestión de Entidades. Actualmente se desempeña como Especialista de Calidad en el grupo de Gestión de Calidad del mismo centro. En su trabajo de diploma identificó, modeló y realizó una propuesta de mejoras de los procesos de negocio que se desarrollaban en el laboratorio de Calisoft. Ha sido tribunal de tesis y ha impartido clases de Contabilidad y Finanzas, Ingeniería de Software I e Ingeniería de Software II.
Categoría docente: Instructor recién graduado.
Especialista de la Dirección de Calidad de Software.
Correo electrónico: galmeida@uci.cu

- Ing. Iliannis Pupo Leyva.

Graduada de Ingeniero en Ciencias Informáticas en la Universidad de las Ciencias Informáticas desde Julio del 2007. Se ha desempeñado en el campo de la Calidad de Software y la Gestión de Proyectos y actualmente se desempeña como Administradora de Calidad del Centro de Informatización de la Gestión de Entidades (CEIGE).
Categoría docente: Instructor recién graduado.
Especialista de la Dirección de Calidad de Software.
Correo electrónico: ipupo@uci.cu

Agradecimientos

A Fidel y la Revolución por darnos la oportunidad de formarnos como ingenieros en ciencias informáticas en esta Universidad de excelencia.

Al gran equipo que han conformado tutores, tribunal, oponente, amigos, y en general a todos los que de una forma u otra han contribuido en el desarrollo de esta tesis con sus opiniones, criterios y ayuda.

Alejandro

A mis padres, mi hermano y mi familia por la confianza depositada.

A mi compañero de tesis pues sin él no hubiera sido posible la realización de este trabajo.

A mi novia Rogsany por regalarme todo su amor, cariño y comprensión. Por haber sido más que mi pareja. Por todo lo que hemos compartido juntos, por su ayuda en todo momento, y sobre todo por haber estado a mi lado durante estos cinco años sin importar las circunstancias.

A todos los amigos que he hecho en esta universidad que se que son muchos y no podría mencionarlos a todos, en especial a Tamara, Roberto y Humberto por haberse convertido en parte de mi familia.

A todos los profes que han contribuido a mi formación, transmitiéndome sus conocimientos.

Humberto

A mi compañero de tesis Alejandro hermano más que nunca en este año de desarrollo de la tesis.

A la familia que se creó desde primer año y ha durado hasta ahora compartiendo alegrías y aventuras que tomo el nombre de piquete.

A todos los compañeros de estudio que he conocido desde mi primer año en esta universidad y a todos los profesores de la misma que me han ayudado.

A todos gracias por aportar su granito de arena.

Dedicatoria

Humberto

A mis padres Isabel y Humberto por ser los primeros maestros en enseñarme el aprendizaje de la vida.

A mi abuela Migdalia por indicarme siempre el camino a seguir.

A mis otros dos padres con los que llevo compartiendo ya más de 4 años Lusia y Gil que siempre me han apoyado.

A mi hermana que con su ejemplo siempre me ha guiado.

En general a toda la familia donde se incluye Mercedes que también es de la familia por todos esos momentos felices y por creer siempre en mí.

Alejandro

A mis padres Anita y Nilo por todo su sacrificio y por toda la confianza que siempre han depositado en mí. A ellos que han sabido guiarme, educarme y formarme. Por ser un ejemplo de dedicación, responsabilidad, entrega, y sobre todo por ser mi mayor orgullo.

A mi hermano Abel por haberme dado mucho más que su apoyo. Por ser mi amigo incondicional, mi brazo derecho, la persona con la que siempre voy a poder contar sin importar las circunstancias.

A mi novia simplemente por estar.

Resumen

En la Universidad de la Ciencias Informáticas debido a la gran cantidad de proyectos existentes y para lograr un manejo de los mismos de forma eficiente se comenzó a utilizar la herramienta web para la gestión de proyectos REDMINE. Apoyándose en la misma el Grupo de Gestión de la Calidad de Software del Centro de Informatización Integral de Gestión de Entidades maneja un grupo de artefactos que se generan durante el proceso de pruebas de los diferentes proyectos que forman parte de este centro.

Aunque esta herramienta es bastante completa actualmente posee algunas limitaciones debido a que se hace difícil la creación de reportes relacionados con las no conformidades de liberación, siendo estas las más utilizadas en el proceso de pruebas de los componentes del centro por lo que su correcta gestión reviste una gran importancia.

Este trabajo tiene como objetivo el desarrollo de un plugin para el REDMINE que se integre de forma fácil al mismo y permita la creación de reportes de forma rápida y sencilla permitiendo una mejor gestión de las pruebas. A partir de estas consideraciones se muestran los resultados, logrando el objetivo propuesto.

Palabras claves:
Plugin, Pruebas, REDMINE, Reporte

Tabla de contenido

Índice de figuras

Introducción

La informática sin duda es uno de los más grandes logros del hombre y desde su surgimiento la sociedad en si ha cambiado. Gracias a los grandes avances tecnológicos y el auge de la misma hoy existen empresas y compañías dedicadas enteramente a la producción de software, las cuales han utilizado estos conocimientos para el perfeccionamiento de la producción y su desarrollo, convirtiéndose la producción de software en uno de los principales renglones de la economía de muchos países.

Dadas las condiciones existentes y la necesidad de que el software creado cumpla con la calidad requerida y las expectativas cada vez más altas de los clientes, se lleva a cabo el proceso de control de la calidad, el cual está presente en todo el ciclo de vida del desarrollo de un producto informático. El eslabón fundamental del proceso antes mencionado es la gestión y realización de pruebas ya que estas permiten medir la excelencia y desempeño óptimo del mismo durante todo el proceso de desarrollo e inclusive durante su posterior despliegue.

Cuba aprovechando el gran potencial profesional con el que cuenta ha creado organismos y empresas para solventar la necesidad creciente que existe de desarrollar productos informáticos ya sea para la comercialización y demanda nacional o internacional. Entre las principales entidades productoras de software se encuentra la Universidad de Ciencias Informáticas (UCI) en la cual existen diferentes infraestructuras productivas para el desarrollo de proyectos; uno de ellos es el Centro de Informatización Integral de Gestión de Entidades (CEIGE) que entre sus principales objetivos se encuentra el desarrollo de software de gestión o ERP. Como parte de la estructura de este centro se encuentra el Grupo de Gestión de la Calidad de Software que se encarga de evaluar y verificar la calidad de los productos informáticos producidos por el centro.

Dentro de las funciones de este grupo está la realización de pruebas al software las cuales juegan un papel fundamental dentro de todo el proceso para evaluar la calidad del producto. El proceso de las pruebas consiste en demostrar que los artefactos de un producto estén libres de errores tales como faltas de ortografía, funcionalidades incorrectas, no correspondencia entre la aplicación y la documentación, por solo mencionar algunos. Estas pruebas involucran las operaciones del sistema bajo condiciones controladas y evaluando los resultados obtenidos para corregir posteriormente las deficiencias detectadas. Además se verifica si los productos de una fase determinada satisfacen la condición impuesta y valida sus componentes para evaluar si satisface los requisitos impuestos por CALISOFT (Dirección de Calidad de

Software de la UCI), entidad rectora en la universidad de los temas relacionados con la calidad de todos los productos informáticos, la cual garantiza el crecimiento continuo de una producción de software con calidad en la organización.

En estos momentos el Grupo de Gestión de la Calidad de Software cuenta con un grupo de probadores, especialistas y técnicos que son los encargados de llevar diariamente el control y desarrollo de las pruebas en las diferentes fases de los proyectos del centro. Estas pruebas se gestionan actualmente mediante la herramienta web para la gestión de proyectos REDMINE donde se registran los resultados de las pruebas, no conformidades, inconsistencias del software, peticiones, tareas correctivas, entre otras acciones que se generan durante el proceso de revisiones.

El REDMINE aunque es una herramienta bastante completa, actualmente no cuenta con todas las funcionalidades y requisitos que se necesitan para realizar una adecuada gestión de las pruebas realizadas. Esto trae como consecuencia que un gran cúmulo de información sea manejada de forma manual, al igual que gran parte de los reportes que son necesarios generar para llevar a cabo un seguimiento del trabajo realizado, por lo que se ve afectada la rapidez de las revisiones, la toma de decisiones y los partes diarios dentro del centro.

Como parte del mejoramiento de esta herramienta y para dar un mejor soporte al proceso de registro, análisis y seguimiento de los datos relacionados con las pruebas en el REDMINE, se decide la implementación de un plugin o complemento, que cubra las demandas del Grupo de Gestión de la Calidad de Software.

Por lo antes planteado surge el siguiente **problema** a resolver: ¿Cómo realizar un análisis de los procesos que se desarrollan en el laboratorio de calidad del Centro para la Informatización de Gestión de Entidades (CEIGE) durante la fase de pruebas?

Por tanto el **objeto de estudio de la investigación** se centra en la gestión de las pruebas, definiendo como **campo de acción** la gestión de pruebas en la herramienta REDMINE.

Se define además como **objetivo general** desarrollar un Plugin para la gestión de las pruebas en la herramienta REDMINE.

Para dar cumplimiento al objetivo general fueron trazados los siguientes **objetivos específicos:**

1. Fundamentar la investigación, mediante la elaboración del marco teórico.
2. Analizar las características que debe cumplir el sistema.
3. Realizar el análisis y diseño de la solución.
4. Implementar la propuesta de solución.

5. Aplicar pruebas al producto para validar su correcto funcionamiento.

Con el cumplimiento de los elementos antes planteados se esperan obtener como **posibles resultados** el análisis, diseño e implementación del plugin para la gestión de las pruebas en la herramienta REDMINE.

Estructuración del contenido.

Para la mejor comprensión del trabajo, el mismo se estructura en cuatro capítulos de la siguiente manera.

- **Capítulo 1: Fundamentación teórica.**

  Se realiza un estudio sobre las pruebas de software, su definición e importancia, se investiga acerca del estado del arte del tema a desarrollar, su necesidad dentro de la herramienta REDMINE, y por último un análisis de las tecnologías, y metodologías seleccionadas.
- **Capítulo 2: Características, análisis y diseño del sistema**.

  Se presenta la propuesta de solución que se quiere implementar, se muestra el entorno donde se desarrolla el plugin mediante el modelo de dominio. Además se hacen las historias de los usuarios y se lleva a cabo la representación y descripción de caso de uso del sistema.

  Se realiza el diseño del plugin, se describe y muestra el modelo conceptual. Además, se hace un análisis de la arquitectura Modelo-Vista-Controlador y se realiza el diagrama secuencia y el de actividades.
- **Capítulo 3: Implementación.**

  Se explica todo lo relacionado con la implementación, se elaboran los diagramas de despliegue y de componente y se muestran distintos elementos del desarrollo del plugin.
- **Capítulo 4: Pruebas, validación e instalación.**

  Se realiza todo lo relacionado con las pruebas, se tratan temas de la validación y se expone como debe ser instalado el plugin. Además se elaboran los casos de pruebas para garantizar un software con calidad.

Capítulo I:

Fundamentación teórica

1.1 Introducción

En el presente capítulo se desarrolla una investigación sobre el proceso de pruebas de software y su estado del arte. Se realiza además un estudio sobre las herramientas, lenguajes y metodologías a utilizar a fin de seleccionar las más óptimas para darle solución al problema planteado teniendo en cuenta las particularidades y requerimientos del mismo.

1.2 Pruebas de software

1.2.1 Definiciones

Según la definición clásica de Myers *"Pruebas de software es el proceso de ejecución de un programa o sistema con la intención de encontrar errores."* (1)
De acuerdo con la definición dada por Hetzel: *" Testing es cualquier actividad enfocada hacia la evaluación de un atributo o capacidad de un programa o sistema para determinar que cumple con los resultados esperados."* (2)
La prueba de un sistema se define como el proceso de ejercitar o evaluar el sistema, por medios manuales o automáticos, para verificar que satisface los requerimientos o, para identificar diferencias entre los resultados esperados y los que producen el sistema (IEEE, 1990).(3)

Otro concepto muy sencillo es el expuesto por Pressman donde se refiere a las pruebas como: "*Las pruebas del software son el proceso de ejecución de un programa con la intención de descubrir un error*." (Pressman, 2005). (4)
Luis Vinicio León en cuanto a las pruebas de software afirma: "*La prueba de software es una actividad ingenieril que exige un perfil y una formación muy particular, en la que se utilizan técnicas y herramientas para detectar niveles inadecuados de calidad, aplicando una cantidad de recursos limitados (en especial tiempo y dinero) de forma tal que genere un valor agregado en el proceso de desarrollo de software."* (5)

Podemos afirmar entonces que las pruebas de software no son más que un proceso orientado a evaluar y pesquisar un determinado sistema o programa en busca de errores de distintas índoles, constituyendo uno de los eslabones fundamentales en el desarrollo de un producto.

1.2.2 Objetivos

Teniendo en cuenta lo expresado anteriormente y basándose en la investigación realizada, se puede resumir que las pruebas de software es el proceso de encontrar la desviación de los requisitos y expectativas de los usuarios en el producto con los objetivos de:

- ✓ Chequear un producto contra las especificaciones.
- ✓ Encontrar errores en el programa.
- ✓ Asegurarse de que un sistema está en óptimas condiciones para su uso.
- ✓ Ganar confidencia de que el programa funciona.
- ✓ Mostrar que un programa funciona correctamente.
- ✓ Demostrar que los errores no están presentes.
- ✓ Entender los límites del rendimiento.
- ✓ Aprender lo que el sistema no puede hacer.
- ✓ Evaluar las capacidades de un sistema.
- ✓ Verificar la documentación.
- ✓ Convencer de que el trabajo ya está terminado.

1.2.3 Importancia

Las pruebas de software, sin lugar a duda, desde su surgimiento han jugado un papel fundamental en el desarrollo de todo producto informático. Las mismas aseguran un conjunto de parámetros, como calidad, estabilidad, seguridad y eficiencia, que debe tener el sistema permitiendo detectar y corregir errores oportunamente. Su correcta realización nos brinda la posibilidad de lograr un producto con la mayor calidad posible y que cumpla con las expectativas del cliente. Debido a esto actualmente se le presta especial atención dedicando gran cantidad de recursos a las mismas y preparando personal especializado dada su importancia para la culminación y liberación de un software.

1.3 Estado del Arte

1.3.1 Ámbito internacional

Las pruebas de software tienen un notable crecimiento en el ámbito internacional adyacente a la exigencia por la calidad, enfatizada en buena medida por la globalización y el incremento en la cultura informática de en mundo actual.

Acerca de las pruebas de software en el ámbito internacional Luis Vinicio León expresó: *"En países desarrollados encontramos toda una industria de prueba de software, constituida entre otras cosas por una buena cantidad de profesionales especializados, proveedores de herramientas, congresos, publicaciones periódicas, y*

múltiples alternativas de capacitación y certificación. La industria de software de esos países consume esa especialización, así como la objetividad e independencia con que ésta puede venir acompañada." (5)

En la revista STaaS - Software Testing as a Service publicada por www.es.sogeti.com se define la situación actual de las pruebas de software a nivel mundial como: *"En el mercado de la industria del software los procesos no son una opción comercial, sobre todo cuando la actividad implica trato con el cliente o es crítica para el negocio. En consecuencia, las pruebas siguen siendo un elemento esencial de la eficacia operativa y de la gestión de riesgos. No obstante, para algunas empresas, la implantación de un proceso de pruebas efectivo y rentable supone un reto considerable, especialmente si los presupuestos destinados a inversiones o recursos son limitados*." (6)

Hoy en día en la urbe informatizada sigue predominando la ignorancia y la enajenación hacia la calidad de software. Los grandes consumidores de software, son los habitantes del mundo desarrollado. Estos en gran medida no se fijan en la calidad del producto que adquieren. Muchos de estos los consumen de forma indebida sin tener una conducta ética informática ya que obtienen estos productos de fuentes no confiables descargándolos o comprándolos en sitios o a personas que no cuentan con patentes, licencias de producción o derechos de autores del producto por lo que no se garantiza la calidad del software que se adquiere.
No obstante las grandes empresas de producción de software y las personas con amplios conocimientos de informática muestran hoy en día, cada vez más, que el proceso de desarrollo y gestión de las pruebas de software, dedicados a garantizar la calidad y las exigencias del cliente siguen un proceso de progresión y perfección.

Estas compañías, empresas, entidades y personas que utilizan la computación como forma de desarrollo tanto económico como social reconocen que la eficacia del producto se convierte en ahorro de costos y en avance general. Cuando se deja a un lado el proceso de calidad de software durante el ciclo de vida de un proyecto puede arruinar la imagen y la valía de la empresa. Las negligencias y la mala manipulación de las pruebas de software durante el proceso de calidad pueden resultar significantes en la pérdida de la cantidad total de dinero que se ha gastado en el proyecto. Contar con un plan para efectuar dichas pruebas es uno de los elementos fundamentales a tener presente en una entidad de desarrollo de aplicaciones informáticas.

1.3.2 Ámbito Nacional

Cuba a pesar de ser una nación pequeña, perteneciente al tercer mundo y bloqueada por el mayor imperio que ha existido, cuenta con grandes avances en el campo de la informática pues ya hace 41 años que viene trabajando en estos temas, dedicándose cada día más al cambio progresivo para su desarrollo y sostenibilidad. En abril de 1969 fue organizado en la mayor de las Antillas el primer grupo de investigación sobre esta ciencia, que dio origen al actual Instituto Central de Investigaciones Digitales (ICID), cuyos resultados iniciales no se hicieron esperar. El surgimiento de este centro demostró las grandes potencialidades y ventajas que brindaría el estudio y desarrollo de este campo. Con la creación por el Comandante en Jefe Fidel Castro de la UCI, la producción de software se ha convertido, sin lugar a duda, en uno de los renglones fundamentales de la economía cubana. En esta universidad de excelencia donde se produce software basándose en la eficiencia y el rendimiento, prestando especial atención a la superación de sus trabajadores y estudiantes, se mantiene un estrecho control y seguimiento sobre la calidad de los productos.

Cuba cuenta además con entidades y empresas dedicadas a la terminología de las producciones y seguimiento del software, entre las principales se encuentra: la Empresa de Telecomunicaciones de Cuba (ETECSA), el Centro de Calidad del Software de la UCI (CALISOFT), la empresa que ofrece soluciones informáticas para el Sistema de Salud (SOFTEL), la Empresa Nacional de Software DESOFT, la Empresa de Tecnologías de la Información y Servicios Telemáticos Avanzados CITMATEL, La Empresa de Consultoría y Seguridad Informática Segurmática, por solo mencionar algunas. Todas estas empresas trabajan en función de asegurar la calidad de sus productos, con la utilización de métodos y sistemas, que a través de estos años ha venido fortaleciéndose de forma constante. Basándose en estos principios se toma la decisión de trabajar por la certificación de la calidad de todo el sistema de gestión de calidad de la empresa bajo las normas ISO 9000 del 2000, para poder lograr un producto que cumpla con las normas internacionales y en base a normas desarrolladas para la elaboración de los productos de software y a las métricas definidas por la norma cubana ISO/IEC 9126-1:2004.

1.4 Metodologías de desarrollo de software

Dado el enorme desarrollo que ha adquirido la producción de software en el mundo se ha creado una gran cantidad de documentación formal referida a los procesos, políticas y procedimientos para garantizar la eficiencia del producto. Cumplir con los requisitos iniciales y minimizar las pérdidas de tiempo en el proceso de desarrollo, son

los principales objetivos de las metodologías de desarrollo a las cuales se va a hacer referencia.

Es de suma importancia establecer una metodología de desarrollo de software adecuada para así minimizar riesgos e incrementar las posibilidades de éxito en el desarrollo de los productos.

1.4.1 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. Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Este primeramente surgió como metodología para el desarrollo de productos tecnológicos, pero en la actualidad se emplea también en ambientes que trabajan con requisitos inestables y que requieren rapidez y flexibilidad; situaciones frecuentes en el desarrollo de determinados sistemas 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.

1.4.2 Rational Unified Process (RUP)

El Proceso Unificado de Rational (RUP) describe cómo aplicar efectivamente el desarrollo de software. La utilización de esta metodología de desarrollo se conoce actualmente como "mejores prácticas" ya que son utilizados en la industria por organizaciones triunfantes y con resultados exitosos de su uso.

RUP provee a cada miembro del equipo de las guías de proceso, plantillas y mentores de herramientas necesarios para que estos tomen ventaja de, entre otras, las siguientes mejores prácticas:

- ✓ Administración de los requisitos.
- ✓ Arquitectura basada en componentes.
- ✓ Modelar visualmente el software.
- ✓ Verificar la calidad.
- ✓ Controlar los cambios de software

RUP constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

1.4.3 XP. Programación Extrema

De esta metodología Kent Beck expresó: *"Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. "*(7)

XP es una metodología ágil basada en la simplicidad, la comunicación e interacción permanente con el cliente (comprobación de requisitos constante) y en el "pair-programming", que es la técnica de programación por parejas donde uno de los programadores escribe código y el otro lo prueba y después se cambian los papeles. De esta forma ya desde el principio se van probando los programas en cuanto a cumplimiento de requisitos como a funcionalidad.

Esta metodología se caracteriza por:

- ✓ Permite introducir nuevos requisitos o cambiar los anteriores ágilmente.
- ✓ Adecuado para proyectos pequeños y medianos.
- ✓ Adecuado para proyectos con alto riesgo.
- ✓ Su ciclo de vida es iterativo e incremental.
- ✓ Cada iteración dura entre una y tres semanas.
- ✓ No produce demasiada documentación acerca del diseño o la planificación.

XP se basa fundamentalmente en ser ligero, cercano al desarrollo, basado en Historias de Usuario, fuerte comunicación con el cliente, el código pertenece a todos, programación por parejas y pruebas como base de la funcionalidad.

Teniendo en cuenta las características y peculiaridades de las metodologías analizadas anteriormente y basándose en las particularidades del desarrollo del plugin a implementar se decidió utilizar XP-Programación Extrema, para el proceso de desarrollo y solución del problema a resolver ya que cuenta con diferentes características que son similares a las condiciones de trabajo y desarrollo. Las demás metodologías, aunque cabe destacar que son muy buenas y ampliamente utilizadas en el mundo, no se ajustan a las necesidades del trabajo y por ende no son las más aconsejables a manejar ya que traerían consecuencias negativas en las posteriores fases de desarrollo del producto.

1.5 Herramientas de desarrollo de software

Las herramientas de desarrollo de software desempeñan un papel fundamental en la creación de aplicaciones. Estas han experimentado en los últimos años grandes cambios y siguen avanzando en su desarrollo, evolucionando constantemente.

Entre las principales corrientes del software se encuentra el software privativo y el software libre. El privativo es aquel que, por su uso, redistribución o modificación está prohibida, o requiere permiso expreso del titular del software. Software libre trata de mejorar sus herramientas y adaptarlas a las necesidades de cada persona que interviene en el proceso, sin costos por su uso, esto no implica que la adquisición del producto sea gratis. Para la elección de las herramientas a utilizar se realizó un estudio de las mismas para así seleccionar las más adecuadas con las características de desarrollo del plugin.

1.5.1 Herramientas CASE

CASE son las siglas correspondientes a Computer Aided Software Engineering, que en su traducción al español significa Ingeniería de Software Asistida por Computadoras.

Perissé describe estas herramientas como: *"*(Ingeniería de Software Asistida por Computadora) a la aplicación de métodos y técnicas a través de las cuales las personas pueden modelar o diseñar sistemas por medio de programas, procedimientos y su respectiva documentación. *"* (8)

Estas herramientas mejoran la productividad de los analistas, la eficiencia en el desarrollo de software, contribuye a la disminución del tiempo empleado para el desarrollo, garantizan la consistencia de los procedimientos, mejora la calidad del sistema de información del producto.

1.5.1.1 Rational Rose

Rational Rose fue desarrollada por (Booch, Rumbaugh y Jacobson) quienes crearon el UML. Cubre todo el ciclo de vida de un proyecto: concepción y elaboración del modelo, construcción de los componentes, transición a los usuarios. Permite especificar, analizar, diseñar el sistema antes de codificarlo. Establece una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable; facilita el desarrollo de un proceso cooperativo en el que todos los agentes tienen sus propias vistas de información (vista de Casos de Uso, vista Lógica, vista de Componentes y vista de Despliegue), pero comparten un mismo modelo a lo largo de todo el ciclo de vida del proyecto.

1.5.1.2 Visual Paradigm 6.4

Es una herramienta de diseño UML y herramienta CASE diseñada para la ayuda en el proceso de desarrollo de software. Soporta estándares claves en la industria del software tales como Lenguaje de Modelado Unificado (UML), SysML, BPMN, XMI, entre otros. Ofrece un completo conjunto de herramientas de los equipos de desarrollo de software necesario para la captura de requisitos, la planificación de programas, la planificación de controles, la clase de modelado, modelado de datos. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Se decide tomar esta herramienta como medio para modelar el proyecto ya que brinda grandes facilidades y cuenta con las características necesarias para el diseño y modelado del plugin a implementar. Además en la universidad se cuenta con la licencia para su uso y tiene la característica de ser multiplataforma.

1.5.2 Herramientas de programación

Las herramientas de programación son básicamente programas informáticos que permiten crear aplicaciones, programas o sistemas así como software en general con los cuales se hace funcionar la parte física del ordenador y permiten al usuario interactuar obteniendo resultados.

Para el desarrollo del plugin se van a analizar tres de estas herramientas de las cuales se seleccionara una teniendo en cuenta sus características y prestaciones, tratando de escoger la que más se adecue a las necesidades de desarrollo.

1.5.2.1 SciTE

CIntilla based Text Editor o SciTE es un editor de textos multiplataforma escrito por Neil Hodgson. Es muy ligero, está diseñado principalmente para edición de código fuente. Soporta el resaltado de sintaxis para muchos lenguajes entre los cuales se encuentra Ruby siendo uno de los editores que llegan en la instalación de este.

1.5.2.2 Aptana

Es un entorno de desarrollo integrado basado en eclipse. Aptana Studio es gratuito, de código abierto y multiplataforma, por lo que puede instalarse en cualquier PC con sistemas operativos Windows, Linux o Mac OSX. Esta permite trabajar con diferentes lenguajes y tecnologías de programación web como HTML, DOM, JavaScript y CSS. Mediante plugin gratuitos se pueden agregar funcionalidades o lenguajes como: PHP, Jaxer, Ruby on Rails, Python, Adobe AIR, entre otros.

1.5.2.3 IDE NetBeans 6.9

Plataforma libre de código abierto, entorno de desarrollo integrado para desarrolladores de software. Brinda todas las herramientas necesarias para crear aplicaciones profesionales de escritorio, empresariales, web y aplicaciones móviles con la plataforma Java, así como C / C + +, PHP, JavaScript, Groovy y Ruby.

Este IDE es el más recomendable ya que contempla el lenguaje a utilizar para la implementación del plugin permitiendo el completamiento de código y la integración con el entorno de desarrollo. Además es muy fácil de usar para la programación web y orientada a objetos que es la que se va a estar utilizando durante el ciclo de desarrollo. Como característica adicional se puede mencionar que presenta una excelente integración con el subversion que es la herramienta colaborativa que se va a estar utilizando. Se ejecuta en varias plataformas, incluyendo Windows, Linux, Mac OS X y Solaris.

1.5.3 Herramientas de desarrollo colaborativo

Las herramientas de desarrollo colaborativo facilitan el trabajo en equipo, permitiendo que varias personas desarrollen sobre un mismo archivo siendo esta su principal funcionalidad y ventaja. Con las mismas se puede llevar a cabo un control de los cambios realizados es decir un control de versiones.

1.5.3.1 Subversion 1.6.12

Subversion es un sistema del tipo cliente-servidor, se conoce como SVN y se utiliza para llevar a cabo un control de versiones. Se le llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas.
Está preparado para funcionar en red y se distribuye bajo licencia libre. Entre sus principales características se pueden mencionar:

- ✓ Mantiene versiones no sólo de archivos, sino también de directorios
- ✓ Además de los cambios en el contenido de los documentos, se mantiene la historia de todas las operaciones de cada elemento, incluyendo la copia, cambio de directorio o de nombre.
- ✓ Atomicidad de las actualizaciones, una lista de cambios constituye una única transacción o actualización del repositorio, esta característica minimiza el riesgo de que aparezcan inconsistencias entre distintas partes del repositorio.
- ✓ Soporte tanto de ficheros de texto como de binarios.

- ✓ Mejor uso del ancho de banda, ya que en las transacciones se transmiten sólo las diferencias y no los archivos completos.

Lo más importante del versionamiento es que permite tener un control eficiente de código fuente; en pocas palabras posibilita que varias personas trabajen en el mismo proyecto de forma simultánea.

1.5.3.2 RapidSVN 0.12.0

RapidSVN es una plataforma visual para el sistema Subversion escrito en C++, con código abierto y software libre bajo la licencia GNU General Public License (GPL) versión 3. Está disponible en varios idiomas diferentes y actúa de cliente gráfico para el acceso al repositorio SVN, tanto si éste es remoto como si es local. Es de fácil manejo para los usuarios principiantes pero lo suficientemente potente y con herramientas interesantes para los usuarios avanzados.
El programa funciona en cualquier plataforma y se puede ejecutar en Linux, Windows, Mac OS / X, Solaris, etc.

Los dos softwares analizados se seleccionaron para su eso debido a las características que presentan las cuales son afines con el proyecto a desarrollar. El SVN es uno de los sistemas de control de versiones más utilizados actualmente y el RapidSVN es uno de los clientes más completos con se cuenta para su uso.

1.5.4 Servidor web

Un servidor web es un programa o software que se ejecuta continuamente en una computadora para garantizar la disponibilidad del servicio, manteniéndose a la espera de peticiones de ejecución que le hará un cliente o un usuario de Internet. El servidor web se encarga de contestar a estas peticiones de forma adecuada, entregando como resultado una página web o información de todo tipo de acuerdo a los comandos solicitados. Durante la fase de desarrollo de una aplicación web juegan un papel fundamental.

1.5.4.1 WEBrick 1.3.1

WEBrick es una librería para servidores HTTP escrita por continuos parches aplicados por la inmensa comunidad de usuarios y desarrolladores de Ruby. Comenzó con un artículo titulado “Internet Programming with Ruby” en una revista de ingeniería de redes japonesa “OpenDesign”'. Y ahora, es parte de las librerías estándar desde Ruby 1.8. Se puede utilizar la clase WEBrick de Ruby para funcionar como servidor de aplicaciones basadas en HTTP. Se puede usar también como base del desarrollo de nuevos frameworks para la creación de aplicaciones Web como por ejemplo IOWA,

Tofu, etc. También puede ser usado para crear servidores no basados en HTTP, como el servidor DaytimeServer de ejemplo en la página de Inicio de WEBrick, aunque en ese caso no se tendrá la posibilidad de usar el soporte para el protocolo HTTP en ningún momento.

En el paradigma de las aplicaciones Web, WEBrick es de un nivel bastante bajo. Sin embargo se usa muy a menudo como servidor inicial para el entorno de desarrollo o de pruebas de las aplicaciones de Ruby. Esto es debido a que su configuración es muy sencilla (un fichero para cada tecnología externa a aplicar y un solo fichero de configuración interna). En general es fiable y la velocidad de servicio es aceptable. Es por esto que normalmente se utiliza este servidor para desarrollo y pruebas de las aplicaciones.

Se selecciona este servidor para el desarrollo del proyecto teniendo en cuenta las siguientes características:

- ✓ Es el más recomendado para las pruebas y el desarrollo de aplicaciones web en Ruby.
- ✓ Es totalmente gratuito y open source.
- ✓ Es sencillo de configurar.
- ✓ La mayoría de los IDEs de desarrollo con Ruby imponen este servidor por defecto al crear proyectos.

1.5.4.2 Mongrel 1.1.5

Mongrel es una librería http open-source HTTP y un servidor Web para las aplicaciones Web de Ruby escrito por Zed A. Shaw. Una característica significativa de Mongrel es que usa html plano para comunicarse con otros servidores que han podido ser desplegados por delante de él. La gran mayoría piensa que Mongrel es más rápido y más estable que WEBrick, y más fácil de configurar que Apache.

WEBrick será el encargado de atender las peticiones HTTP que lleguen del servidor web y ejecutar la aplicación en el intérprete de Ruby durante toda la fase de desarrollo. Para la etapa de pruebas será utilizado el servidor Mongrel debido a que es más estable.

1.5.5 Sistemas de gestión de base de datos

Los servidores de bases de datos se crean debido a la necesidad de almacenar y manejar grandes y complejos volúmenes de datos, además que se requiere compartir la información de una manera segura y fiable.

Los SGBD brindan herramientas de administración completas que simplifican la tarea de la configuración, seguridad, creación y gestión de bases de datos. También proporcionan mecanismos de integración con otros sistemas y políticas de copias de seguridad, así como permitan su programación tanto a nivel de diseño como a nivel de reglas y procedimientos. Estos sistemas deben proporcionar mecanismos de comunicación con otras plataformas que actúen como clientes o servidores de datos.

1.5.5.1 PostgreSQL 9.0

El Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos conocido como PostgreSQL (y brevemente llamado Postgres95) está derivado del paquete Postgres escrito en Berkeley. Con cerca de una década de desarrollo tras él, PostgreSQL es el gestor de bases de datos de código abierto más avanzado hoy en día, ofreciendo control de concurrencia multi-versión, soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones y funciones definidas por el usuario), contando también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, perl, tcl, python y Ruby). Es un sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia BSD.

Presenta grandes ventajas y características como son: alta concurrencia y mediante un sistema denominado MVCC (Acceso concurrente multi-versión, por sus siglas en inglés) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo commit. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

Además provee soporte para:

- ✓ Números de precisión arbitraria.
- ✓ Texto de largo ilimitado.
- ✓ Figuras geométricas (con una variedad de funciones asociadas).
- ✓ Direcciones IP (IPv4 e IPv6).
- ✓ Bloques de direcciones estilo CIDR.
- ✓ Direcciones MAC.
- ✓ Arrays.

1.5.5.2 Pgadmin III

Pgadmin III es una aplicación gráfica para el gestor de bases de datos PostgreSQL, siendo la más completa y popular con licencia Open Source. Está escrita en C++ usando la librería gráfica multiplataforma wxWidgets, lo que permite que se pueda usar

tanto en Linux, FreeBSD, Solaris, Mac OS X y Windows. Es capaz de gestionar versiones a partir de la PostgreSQL 7.3 ejecutándose en cualquier plataforma, así como versiones comerciales de PostgreSQL como Pervasive Postgres, EnterpriseDB, Mammoth Replicator y SRA PowerGres.

Pgadmin III está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas SQL simples hasta desarrollar bases de datos complejas. La interfaz gráfica soporta todas las características de PostgreSQL y facilita enormemente la administración. La aplicación también incluye un editor SQL con resaltado de sintaxis, un editor de código de la parte del servidor, un agente para lanzar scripts programados, soporte para el motor de replicación Slony-I entre otras características. La conexión al servidor puede hacerse mediante conexión TCP/IP o Unix Domain Sockets (en plataformas *nix), y puede encriptarse mediante SSL para mayor seguridad.

Debido a la política que sigue la universidad no se realizó el análisis de otros servidores de base de datos ya que el PostgreSQL es el que está soportado para el desarrollo de aplicaciones y es el establecido por la dirección técnica de la infraestructura productiva para ser utilizado en el REDMINE.

1.5.6 Navegador web

Los navegadores web son aplicaciones capaces de interpretar las órdenes recibidas en forma de código HTML fundamentalmente y convertirlas en las páginas que son el resultado de dicha orden.

El navegador se comunica con el servidor a través del protocolo HTTP y le pide el archivo solicitado en código HTML, después lo interpreta y muestra en pantalla para el usuario.

1.5.6.1 Mozilla Firefox 4.0

Mozilla Firefox es un navegador de software libre y código abierto, creado por la Corporación Mozilla, la Fundación Mozilla y numerosos voluntarios externos. Se sitúa en la segunda posición de navegadores más usados solamente superado por Internet Explorer, con una gran aceptación por parte de los usuarios que lo definen como más seguro, rápido y de mejor rendimiento, destacando también por su sencillez y fácil manejo.

Como características añadidas a las habituales de todos los navegadores, Mozilla Firefox ofrece también múltiples plugins, extensiones add-ons y la posibilidad de personalizar su apariencia, además ofrece herramientas muy útiles para los programadores web como la consola de errores, el inspector DOM o extensiones

como Firebug, por estas razones y el hecho de ser de código abierto es el seleccionado para utilizar durante el desarrollo del producto.

1.6 Lenguajes a utilizar

Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo.

También existen lenguajes para el modelado que son utilizados para modelar, valga la redundancia, especificar y construir de forma gráfica cada una de las partes del software. El modelado es una de las actividades más importantes en cualquier proceso de desarrollo de software.

1.6.1 Lenguajes de programación

1.6.1.1 Ruby

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos. Creado por el programador japonés Yukihiro "Matz" Matsumoto, quien comenzó a trabajar en Ruby en 1993, y lo presentó públicamente en 1995. Combina una sintaxis inspirada en Python y Perl con características de programación orientada a objetos similares a Smalltalk. Su implementación oficial es distribuida bajo una licencia de software libre. Es la base del framework Ruby on Rails.

1.6.1.2 Ruby on Rails

El framework Ruby on Rails también conocido como RoR o Rails es sobre el cual está desarrollado el REDMINE. Ruby on Rails es de código abierto y está escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC), trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros framework y con un mínimo de configuración.

Se selecciona para la creación de la aplicación el lenguaje Ruby on Rails ya que los plugin que se desarrollan para su posterior incorporación en la herramienta REDMINE se implementan en este lenguaje.

1.6.1.3 HTML

Es el lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento y puede incluir un

script, el cual puede afectar el comportamiento de navegadores web y otros procesadores de HTML.
Este lenguaje será utilizado para crear las vistas del plugin conjuntamente con Ruby on Rails.

1.6.2 UML Lenguaje de modelado

UML (Unified Modeling Language) es un lenguaje de modelado unificado el cual proporciona un medio gráfico de modelar varios componentes de un sistema de software. El componente diagrama de clase de UML se basa en diagramas E-R (Estructura general que permite expresar gráficamente el esquema de una empresa.).
Esta es una de las mejores herramientas para analizar y diseñar sistemas de software que ofrece un lenguaje común que todo desarrollador debe conocer. La misma es de suma importancia ya que permite tener una visión más completa del sistema mediante varios tipos de diagramas.

Debido a las características del problema a resolver y el producto a desarrollar se ha definido este lenguaje a utilizar ya que es de suma comodidad para el trabajo con lenguajes orientados a objetos el cual será el utilizado en el proceso de implementación del producto.

1.7 Conclusiones

Mediante la realización de una investigación acerca de las pruebas de software y el estado del arte de las mismas se puede concluir que en la línea calidad del centro CEIGE existe la necesidad de un plugin para la gestión de las mismas en la herramienta web REDMINE. Además se pudo constatar las ventajas y la necesidad que existe de llevar un correcto control y seguimiento de las revisiones realizadas durante las diferentes fases de desarrollo de un software.

Como metodología para el desarrollo del plugin se propone XP, una metodología de las llamadas ágiles que se centra en la interacción con el cliente, el trabajo por roles y el seguimiento de la calidad. Como herramienta para realizar los distintos artefactos ingenieriles se eligió el Visual Paradigm, el cual es multiplataforma y cuenta con múltiples características que los hace una herramienta muy potente. El lenguaje de modelado será el potente y versátil UML, lenguaje gráfico para visualizar, especificar, construir y documentar cada una de las partes que comprende el desarrollo del proyecto. Por último, para desarrollar el plugin se escogió el IDE NetBeans, que soporta e integra completamente el lenguaje Ruby el cual será el utilizado para el desarrollo del plugin.

Capítulo II:

Características, análisis y diseño del sistema

2.1 Introducción

El presente capítulo está dedicado a las características del sistema, en el mismo se realiza una descripción de la situación actual en la universidad donde surge la necesidad de un sistema para la gestión de pruebas en la herramienta RDMINE. Se representan conceptos y términos con los que se podrá facilitar la comprensión de los requerimientos del plugin a desarrollar. Se describe el modelo de dominio para el plugin, las historias de usuario así como el procedimiento a ejecutar por el usuario para el uso de las funcionalidades de la herramienta.

2.2 Breve descripción de la situación actual.

Las pruebas de software son una necesidad para las empresas, entidades y organismos dedicados al desarrollo y producción de software. En el centro CEIGE, luego de realizar un proceso de entrevistas con especialistas de la línea de calidad, se pudo constatar la existencia de varias deficiencias que atentan contra el proceso de pruebas, como son:

- ✓ Utilización de recursos innecesarios.
- ✓ Desviaciones en el cronograma.
- ✓ Exceso de personal que se centra en la realización de las pruebas.
- ✓ Imprecisión en los resultados de las pruebas.

Todo esto está dado por la forma en que se lleva el tratamiento de la administración de las pruebas de software y la necesidad de contar con herramientas que realicen la gestión de las mismas para así realizar un estudio y análisis pertinente del estado de estas. Con la creación de este plugin se espera poder disminuir los costos y planificar tanto recursos humanos como materiales, así como dirigir el esfuerzo hacia donde sea verdaderamente necesario.

2.3 Propuesta de sistema

Se propone como solución un servicio web que se integre a la herramienta REDMINE con el cual se pretende satisfacer la necesidad de contar con un sistema para la gestión de pruebas.

Se trata de un complemento web completamente exportable para la gestión de las pruebas de software que genera reportes por proyectos donde se reflejan los tipos de

errores más frecuentes, estado de las pruebas, duración de estas, cantidad de revisiones entre otras funcionalidades. Esto facilitara la toma de decisiones en los laboratorios de calidad y además se puede lograr un manejo eficiente de recursos dígase tiempo de trabajo así como capital humano dedicado a las pruebas.

La arquitectura que se propone para el desarrollo de este plugin es: Modelo-Vista-Controlador (MVC) (Ver Anexo 1) junto con Cliente-Servidor. Este sistema contará como lenguaje básico el Ruby utilizando el framework Ruby on Rails a implementarse en el IDE de desarrollo NetBeans que cuenta con el soporte necesario para este. Además se utilizará como servidor web el WebRick y Mongrel utilizando como servidor de base de datos PostgreSQL.

2.4 Modelo de dominio

Dada la propuesta de la creación del plugin para la gestión de pruebas en el REDMINE se propone la definición de un modelo de dominio o modelo conceptual el cual mostrará los principales conceptos a utilizar en el desarrollo de este complemento web. Se decide emplear el modelo conceptual ya que se cuenta con pocos datos los cuales no brindan gran cantidad de información para poder crear un modelo de negocio.

"Para lograr una mayor comprensión del dominio del problema, se crea el artefacto modelo conceptual o de dominio, que no es más que una representación visual de los conceptos u objetos del mundo real que son significativos para el problema o el área que se analiza; representando las clases conceptuales, no los componentes de software. Puede verse como un modelo que comunica a los interesados, cuáles son los términos importantes y cómo se relacionan entre sí." (9)

El modelo de dominio contempla la representación de conceptos u objetivos que son importantes dentro de un problema. Estos conceptos representan y simbolizan objetos del mundo real que se encuentran dentro del negocio y ofrecen a su vez un entendimiento del problema en cuestión.

2.4.1 Representación del modelo de dominio

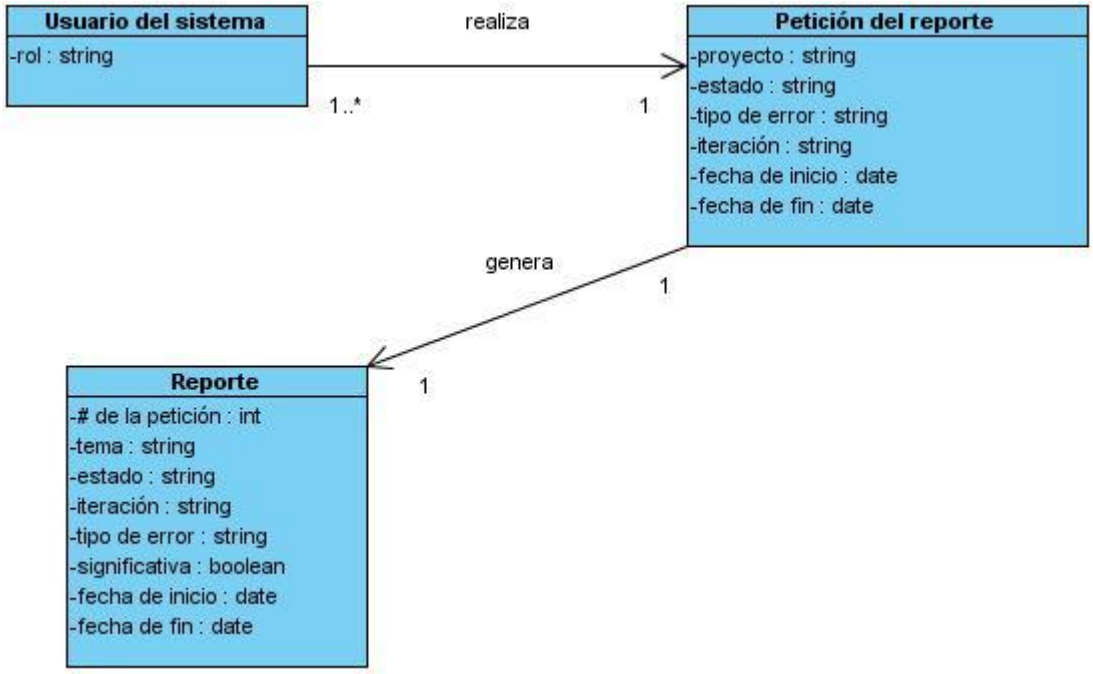


Figura 1. Modelo del dominio

2.4.2 Entidades y conceptos

Usuario del sistema: Es quien va a interactuar con la aplicación y realizar la solicitud de creación del reporte.

Rol: Actividad que realiza el usuario o la responsabilidad que ejerce. Se recomienda que sea el Administrador de la calidad pero se puede definir cualquier rol del sistema.

Línea: Nombre del proyecto.

Iteración: Número de veces que pasa el proyecto por las revisiones y pruebas.

Estado: Estado en que se encuentra la revisión. Puede ser Abierta, Cerrada, Rechazada, etc.

Tipo de error: Tipo de errores que se encuentran en las revisiones y pruebas. Puede ser de documento o de aplicación.

Reporte: Documento que se genera con todos los datos.

2.5 Especificación de los requerimientos

La IEEE define a la especificación de requisitos como: *Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo. Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente. El propósito de la especificación de requerimientos es reunir en un documento escrito los requisitos de todo el sistema de software o parte de él. Esto con la finalidad de plasmar que es el sistema y cuál es su alcance.* (10)

2.5.1 Requisitos funcionales

Son capacidades o condiciones que el sistema debe cumplir.

- ✓ RF 1: Filtrar peticiones por criterios de búsqueda.
  - RF 1.1: Filtrar peticiones por Proyecto.
  - RF 1.2: Filtrar peticiones por Estado.
  - RF 1.3: Filtrar peticiones por Tipo de error.
  - RF 1.5: Filtrar peticiones por Iteración.
  - RF 1.6: Filtrar peticiones por Fechas.
- ✓ RF 2: Generar reporte según el filtro.

2.5.2 Requisitos funcionales

Son capacidades o condiciones que el sistema debe cumplir.

2.5.3 Requisitos no funcionales

Son propiedades o cualidades que el producto debe tener y cumplir.

2.5.3.1 Software

A nivel del cliente o usuario el plugin puede ser accedido desde cualquier plataforma o sistema operativo, solo es necesario contar con una conexión a la red apropiada y un navegador web. Aunque se recomienda el uso del navegador Mozilla Firefox para su funcionamiento óptimo.

2.5.3.2 Hardware

Como requerimiento mínimo el sistema debe contar con procesador Pentium II en adelante con no menos de 256 de RAM.

Se recomienda para su funcionamiento óptimo máquinas con procesadores Pentium IV y 512 de RAM.

2.5.3.3 Apariencia o interfaz

La interfaz deberá ser sencilla de usar, amigable y aunque su ámbito principal es para el desarrollo de software, cualquier usuario podría hacer uso de esta sin dificultad.

2.5.3.4 Seguridad

El plugin debe mantener la integridad y confiabilidad ante la información manejada, ya que la ausencia de estos aspectos puede conllevar a grandes riesgos pues se ve afectado el manejo de los datos de las pruebas en el laboratorio. Para lograr lo antes mencionado se definirán roles y el acceso a la aplicación será según los permisos con que cuente cada rol definido, evitando así acciones que se realicen no autorizadas.

2.5.3.5 Confiabilidad

El plugin deberá tener un 100% de disponibilidad por lo que podrá ser usado siempre y cuando la aplicación principal esté ejecutándose es decir el REDMINE.
El tiempo medio de reparación se estima que sea menor de 1 día.

2.5.3.6 Requerimientos del diseño e implementación

Estos requerimientos determinan los elementos que se utilizarán durante las fases más importantes del proyecto.

- ✓ Se debe implementar en el lenguaje Ruby utilizando su framework Ruby on Rails
- ✓ Para el tratamiento de la base de dato se utilizará PostgreSQL.
- ✓ Para el análisis y el diseño del plugin se utilizará la metodología XP, usando el lenguaje de modelación UML y como herramienta para llevar a cabo el modelado Visual Paradigm.
- ✓ Para la implementación del plugin se utilizará el IDE NetBeans.

2.6 Modelación del sistema

Se muestran los actores del sistema además de la representación de las historias de usuario definidas en la metodología XP y su análisis correspondiente.

2.6.1 Actores del sistema

El actor del sistema es un usuario fuera del sistema que interactúa con él. A continuación se muestra en la siguiente tabla su justificación.

Nombre del Actor	Descripción
Líder de proyecto	Estos usuarios pueden gestionar todos los datos que se pueden mostrar en los reportes de las pruebas.
Administrador de la calidad	
Jefe de línea	Solo tiene permisos a consultar los datos de la línea al cual pertenece.

Tabla 1. Actores del sistema

2.6.2 Lista de reserva del producto

La tabla que a continuación se muestra relaciona las Historias de Usuario con la prioridad que tienen, la estimación del tiempo para efectuarlas en días y los usuarios que se encargan de desarrollarlas.

Prioridad	Ítem	Descripción	Estimación (días)	Estimado por:
Alta	1	Crear la estructura inicial del plugin dentro de la herramienta REDMINE	3	Programador
Alta	2	Creación de la interfaz principal para la aplicación.	5	Programador
Alta	3	Creación de las clases del modelo.	10	Programador
Alta	4	Creación de las clases de control.	10	Programador
Alta	5	Configuración y creación de los permisos para el plugin.	10	Programador

Tabla 2. Lista de reserva del producto

2.6.3 Descripción de las historias de usuarios

En este epígrafe se realiza una descripción de las Historias de Usuarios y las tareas ingenieriles que cada una tiene asignada además del responsable del cumplimiento de dicha tarea.

Historia del usuario 1. Crear la estructura inicial del plugin dentro de la herramienta REDMINE

Historias del usuario	
Número: 1	**Nombre Historia del Usuario:** Crear la estructura inicial del plugin dentro de la herramienta REDMINE.
Modificación de Historia del Usuario Número: Ninguna	
Usuario: Humberto Rodríguez Peláez	**Iteración Asignada:**
Prioridad en Negocio: Alta	**Puntos estimados:** 1 semana
Riesgos en Desarrollo: Bajo	**Puntos Reales:** 3 días
Descripción: Se necesita realizar la estructura del plugin dentro del REDMINE para la creación de la interfaz dentro de la misma e iniciar la programación de esta.	
Observaciones:	
Prototipo de interface:	

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 1
Nombre Tarea: Realizar estudio sobre plugin en la herramienta REDMINE y como crearlos.	
Tipo de Tarea: Investigativa	**Puntos Estimados:** 4 semanas
Fecha Inicio: 02/12/2010	**Fecha Fin:** 20/01/2011
Programador Responsable: Alejandro Álvarez Hernández y Humberto Rodríguez Peláez	
Descripción: Se realiza la investigación dentro del lenguaje Ruby on Rails de cómo crear plugins para las aplicaciones y más específico dentro del REDMINE.	

Tarea de Ingeniería 1

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario:** 1
Nombre Tarea: Creación de la estructura del plugin.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 1 semana
Fecha Inicio: 22/01/2011	**Fecha Fin:** 01/02/2011
Programador Responsable: Humberto Rodríguez Peláez	
Descripción: Se crea la estructura del plugin dentro del REDMINE para alistar las condiciones para el comienzo de la implementación.	

Tarea de Ingeniería 2

Historia del usuario 2. Creación de la interfaz principal para la aplicación

Historias del usuario	
Número: 2	**Nombre Historia del Usuario:** Creación de la interfaz principal para la aplicación.
Modificación de Historia del Usuario Número: Ninguna	
Usuario: Alejandro Álvarez Hernández	**Iteración Asignada:**
Prioridad en Negocio: Alta	**Puntos estimados:** 1 semana

Riesgos en Desarrollo: Bajo	**Puntos Reales:** 5 días
Descripción: A partir de las mismas comienza todo el desarrollo del plugin.	
Observaciones:	
Prototipo de interface:	

Tarea de Ingeniería	
Número Tarea: 3	**Número Historia de Usuario:** 2
Nombre Tarea: Investigación sobre Ruby on Rails.	
Tipo de Tarea: Investigación	**Puntos Estimados:** 1 semana
Fecha Inicio: 5/02/2011	**Fecha Fin:** 15/02/2011
Programador Responsable: Alejandro Álvarez Hernández y Humberto Rodríguez Peláez	
Descripción: Se dispensa de un estudio sobre los diferentes tipos de formularios y HTML que existen dentro de Ruby on Rails para posterior implementación.	

Tarea de Ingeniería 3

Tarea de Ingeniería	
Número Tarea: 4	**Número Historia de Usuario:** 2
Nombre Tarea: Implementación de las interfaces y formularios principales.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 8 semanas
Fecha Inicio: 20/02/2011	**Fecha Fin:** 30/04/2011
Programador Responsable: Alejandro Álvarez Hernández y Humberto Rodríguez Peláez	
Descripción: Se necesita la existencia de las interfaces para crear los filtros y generar los reportes necesarios para que el sistema cumpla con las necesidades del cliente.	

Tarea de Ingeniería 4

Historia del usuario 3. Creación de las clases del modelo

Historias del usuario	
Número: 3	**Nombre Historia del Usuario:** Creación de las clases del modelo.
Modificación de Historia del Usuario Número: Ninguna.	
Usuario: Humberto Rodríguez Peláez	**Iteración Asignada:**
Prioridad en Negocio: Alta	**Puntos estimados:** 2 semanas
Riesgos en Desarrollo: Bajo	**Puntos Reales:** 10 días
Descripción: Se necesita contar con la las clases del modelo para la lógica del negocio y la validación de los campos obligatorios.	
Observaciones:	
Prototipo de interface:	

Tarea de Ingeniería	
Número Tarea: 5	**Número Historia de Usuario:** 3
Nombre Tarea: Implementación de las clases del modelo.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 1 semana
Fecha Inicio: 2/05/2011	**Fecha Fin:** 10/05/2011
Programador Responsable: Humberto Rodríguez Peláez	
Descripción: Se necesita la existencia clases del modelo para controlar los datos que transitan en la aplicación y así velar por la integridad de los datos.	

Tarea de Ingeniería 5

Historia del usuario 4. Creación de las clases de control

Historias del usuario	
Número: 4	**Nombre Historia del Usuario:** Creación de las clases de control.
Modificación de Historia del Usuario Número: Ninguna	
Usuario: Alejandro Álvarez Hernández	**Iteración Asignada:**
Prioridad en Negocio: Alta	**Puntos estimados:** 2 semanas
Riesgos en Desarrollo: Bajo	**Puntos Reales:** 10 días
Descripción: Se necesita implementar las clases de control pues estas se encargan de las principales funcionalidades del sistema.	
Observaciones:	
Prototipo de interface:	

Tarea de Ingeniería	
Número Tarea: 6	**Número Historia de Usuario:** 4
Nombre Tarea: Implementación de las clases de control.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 1 semana
Fecha Inicio: 12/05/2011	**Fecha Fin:** 18/05/2011
Programador Responsable: Alejandro Álvarez Hernández	
Descripción: Se necesita de la controladora para dar funcionalidad a la aplicación.	

Tarea de Ingeniería 6

Historia del usuario 5. Configuración y creación de permisos para el plugin

Historias del usuario	
Número: 5	**Nombre Historia del Usuario:** Configuración y creación de permisos para el plugin.
Modificación de Historia del Usuario Número: Ninguna	
Usuario: Humberto Rodríguez Peláez	**Iteración Asignada:**
Prioridad en Negocio: Alta	**Puntos estimados:** 2 semanas
Riesgos en Desarrollo: Bajo	**Puntos Reales:** 10 días
Descripción: Como elemento clave de todo sistema se necesitan otorgarle los permisos necesarios al plugin así como su accesibilidad.	
Observaciones:	
Prototipo de interface:	

2.6.4 Diagrama de caso de uso del sistema

Para la realización del diseño se deben conocer las características principales del sistema especificando las diferentes acciones que debe permitir y los actores que intervienen en las mismas. Las relaciones que se establecen entre estos dos elementos se reflejan en el siguiente diagrama de casos de uso del sistema. Este representa gráficamente a los procesos y su interacción con los actores.

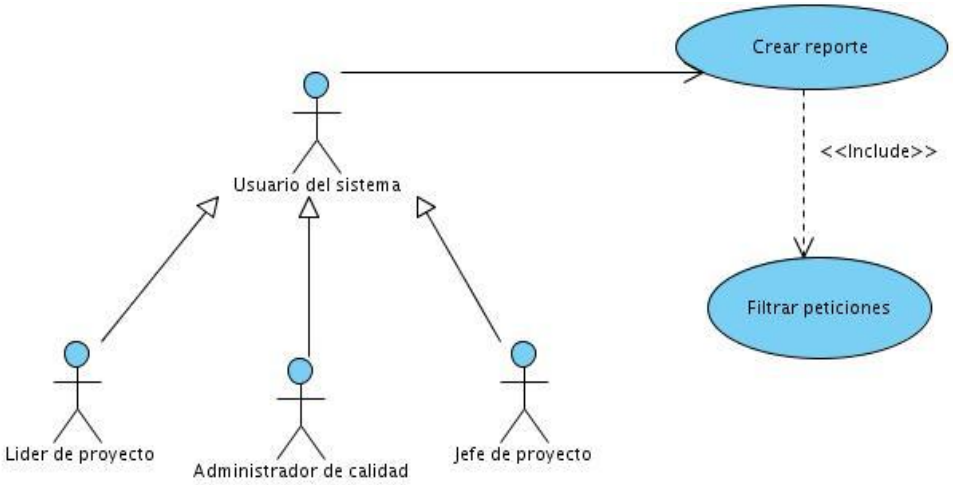


Figura 2. Diagrama de casos de usos del sistema

2.6.5 Descripción del caso de uso del sistema

Descripción del caso de uso	
Nombre del caso de uso	Generar reporte según filtro.
Objetivo	Permite a los usuarios del sistema crear reportes de peticiones de liberación existentes a partir de los filtros seleccionados y visualizarlas en formato html dentro del REDMINE.
Actores	Usuarios del sistema
Resumen	El caso de uso se inicia cuando el usuario ingresa en el REDMINE obteniendo según su rol los permisos para cargar el filtro de peticiones y crear los reportes. El caso de uso finaliza al realizar estas acciones.
Precondiciones	Se debe seleccionar un proyecto. Se crean los reportes después de filtrar por los criterios seleccionados.
Pos condiciones	Quedan filtradas las peticiones de liberación existentes según los criterios de búsqueda mostrándose el reporte generado.
Referencias	RF 1, RF 1.1, RF 1.2, RF 1.3, RF 1.4, RF 1.5, RF 2
Curso normal de los eventos	
Acción del actor	Respuesta del sistema
1. El usuario del sistema selecciona la pestaña del plugin dentro del menú de proyectos.	2. El sistema muestra la interfaz principal para filtrar peticiones.
3. El usuario introduce los criterios de búsqueda para filtrar las peticiones.	4. El sistema valida los datos. 5. El sistema muestra un mensaje de acción satisfactoria con la tabla de peticiones según los filtros seleccionados.
Flujo alternativo	

1. a El usuario del sistema introduce datos erróneos o deja datos en blanco.	2. b El sistema emite un mensaje de error y regresa al paso 2.
Prioridad	Crítico
Prototipo	

Tabla 3. Descripción del caso de uso

2.7 Diseño del sistema

El diseño constituye una parte imprescindible dentro del proceso de desarrollo de todo producto, el mismo transforma el esbozo preliminar y crea un conjunto de elementos del modelo que serán posteriormente implementados utilizando para su modelación el UML.

2.7.1 Estilo de arquitectura

Según la IEEE 1471-2000: *La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.* (11)

La arquitectura proporciona una visión global del sistema a construir. Describe la estructura y la organización de los componentes del software, sus propiedades y las conexiones entre ellos. Los componentes del software incluyen módulos de programas y varias representaciones de datos que son manipulados por el programa. (12)

El estilo arquitectónico seleccionado para el desarrollo de este plugin es el MVC (Ver Anexo 1) que se especificó anteriormente. Este estilo o patrón estructura el desarrollo de la aplicación separándola en tres componentes distintos, lógica, control y presentación. El mismo está involucrado dentro de los de llamada y retornos por el uso que hace de los datos que maneja.

Se estructura el desarrollo de la aplicación de la siguiente forma.

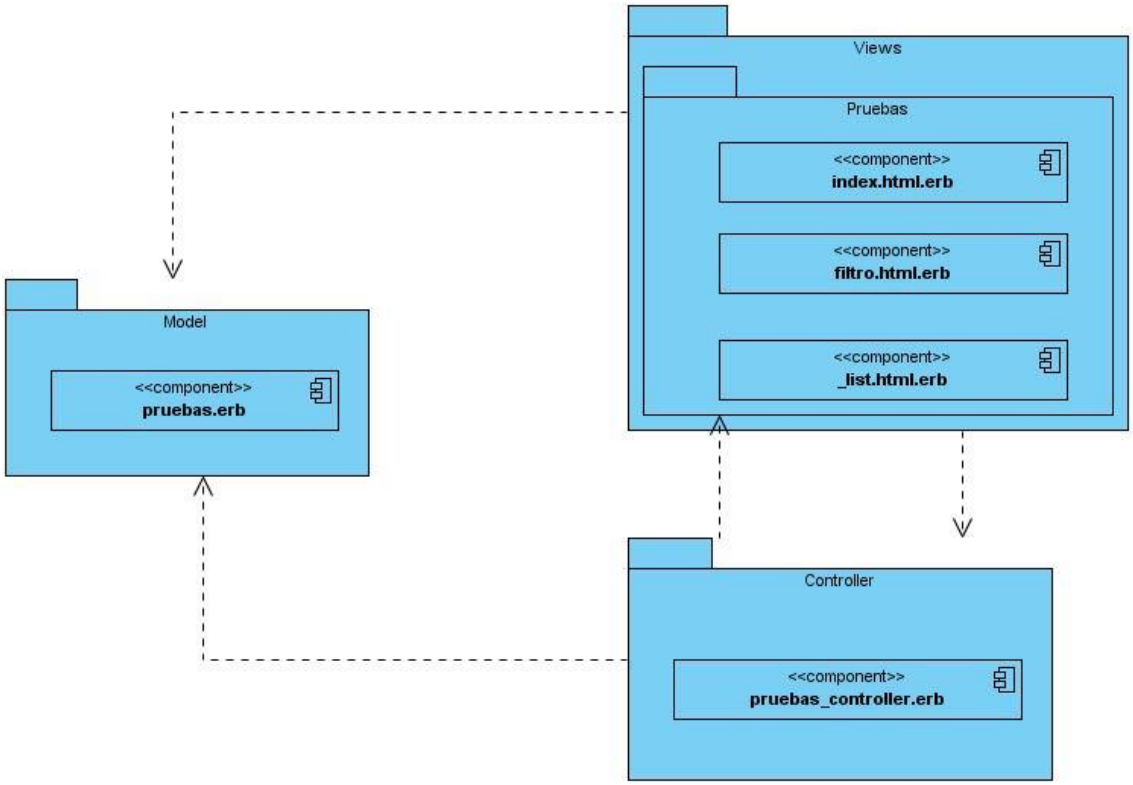


Figura 3. Modelo-Vista-Controlador

Vista (Views): Se presenta usualmente como un medio de interfaz de usuario o para mostrar información contenida en el modelo.

Modelo (Model): Se realizan validaciones permitiendo asegurar la integridad de los datos con los que se trabajan, permite el apoyo a la controladora en la creación de funciones para la muestra de la información enviada a las vistas.

Controladora (Controller): Este se ejecuta mediante eventos tanto en la interfaz de usuario como en la modelo, este mayormente provocado por acciones de los usuarios dentro de las interfaces presentes en las vistas realizando modificaciones posteriores en las vistas y los modelos.

2.7.2 Prototipos de interfaz de usuario

Los prototipos de interfaz de usuario ayudan a identificar, comunicar y mostrar un producto antes de crearlo. Los mismos ilustran como se desea que quede el producto final y pautan la línea a seguir para el desarrollo. A través de estos prototipos se validaron los requisitos funcionales.

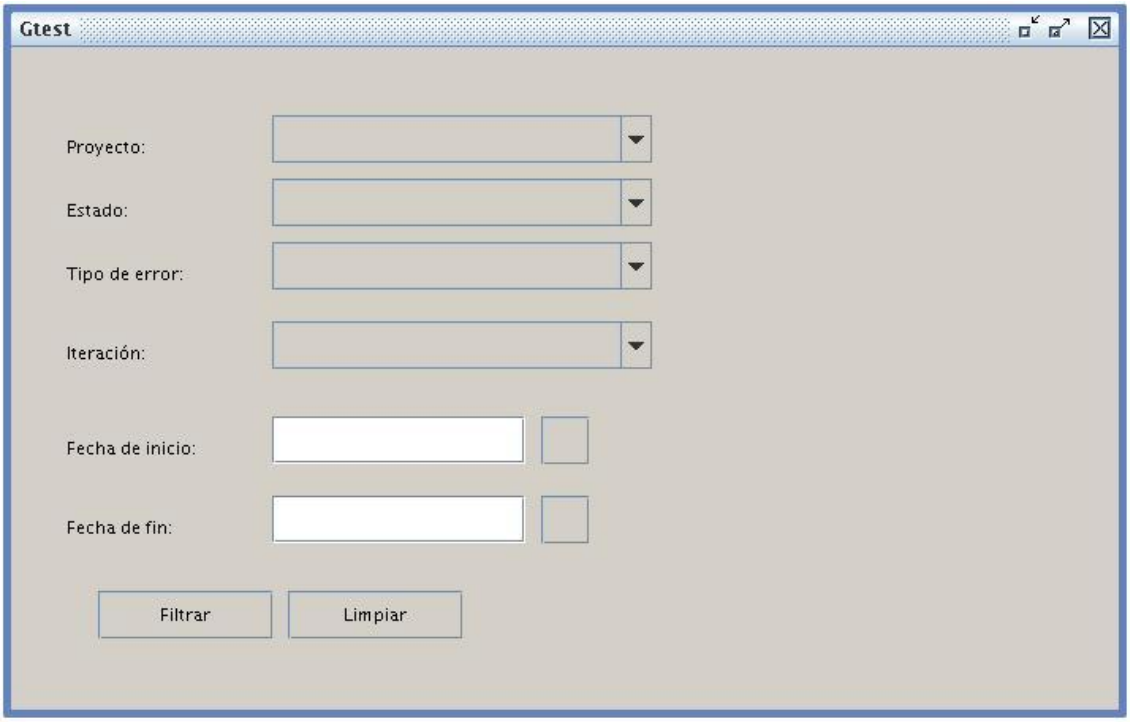


Figura 4. Prototipo de interfaz de usuario filtrar peticiones

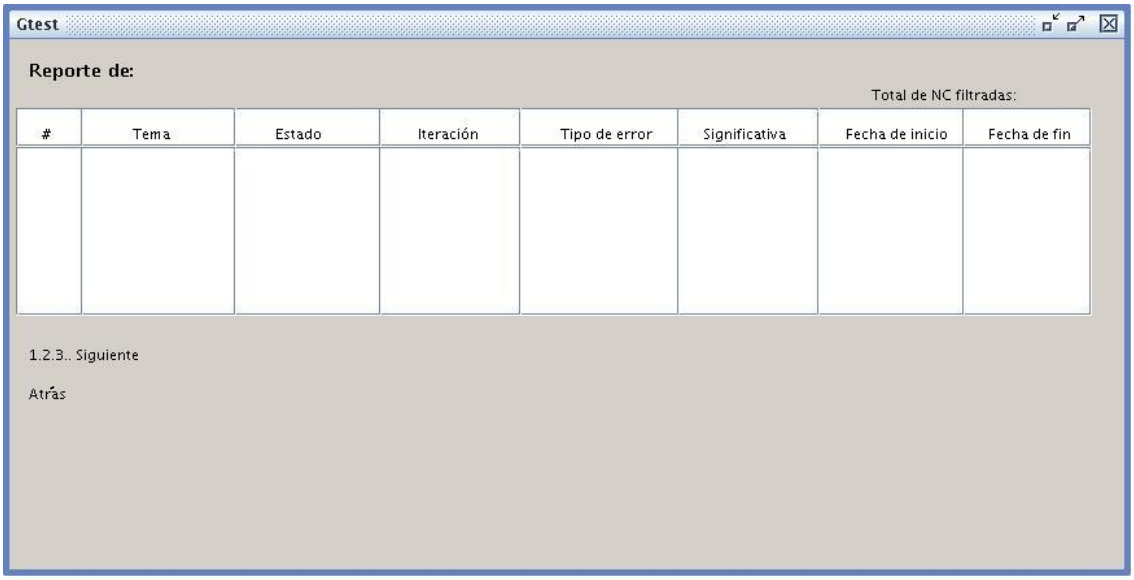


Figura 5. Prototipo de interfaz de usuario reporte de peticiones

2.7.3 Diagrama de secuencia

El diagrama de secuencia es uno de los diagramas más efectivos para modelar interacción entre objetos en un sistema. El mismo muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada método de la clase. Este contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementarlo.

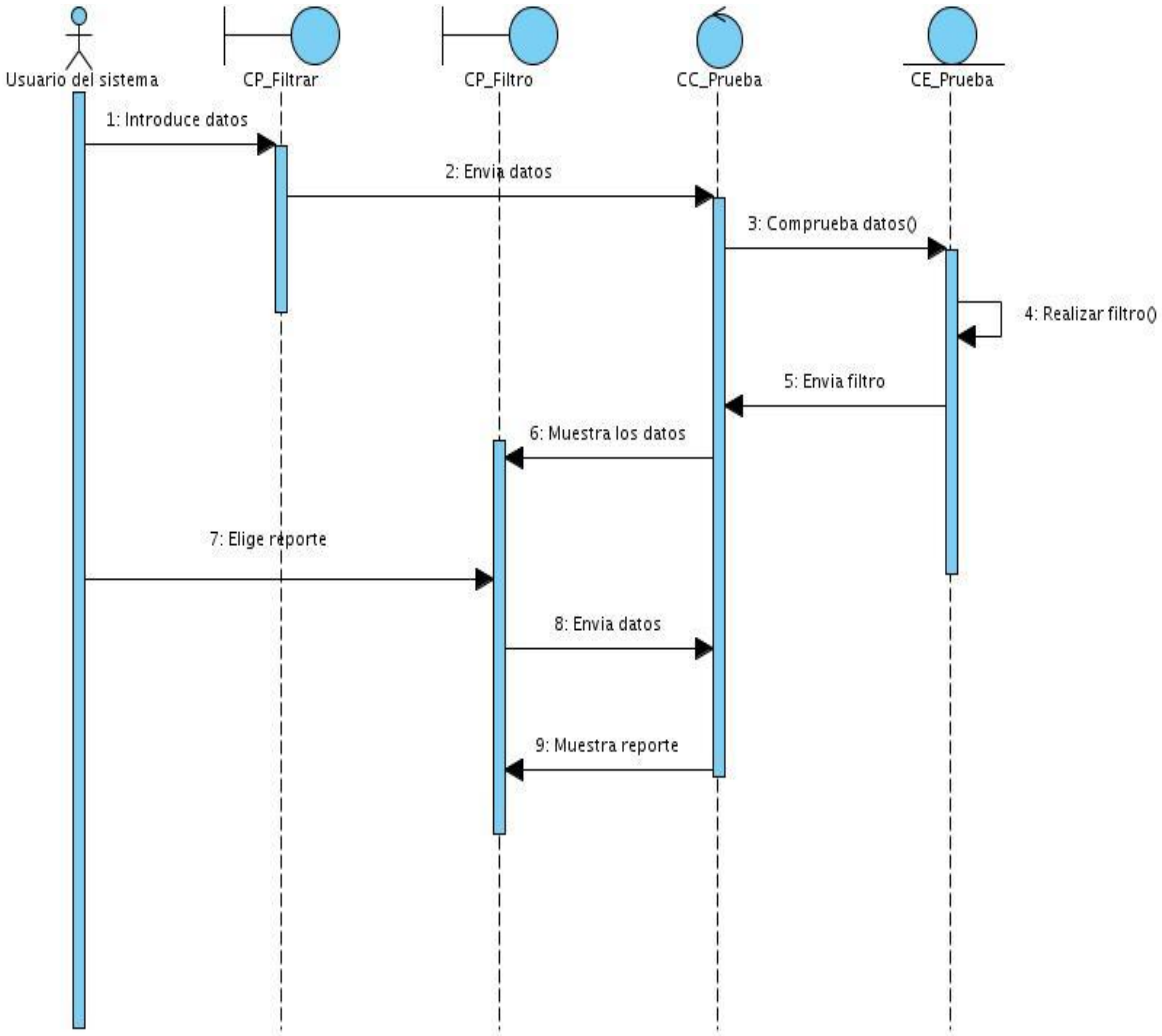


Figura 6. Diagrama de secuencia

2.7.4 Diagrama de actividades

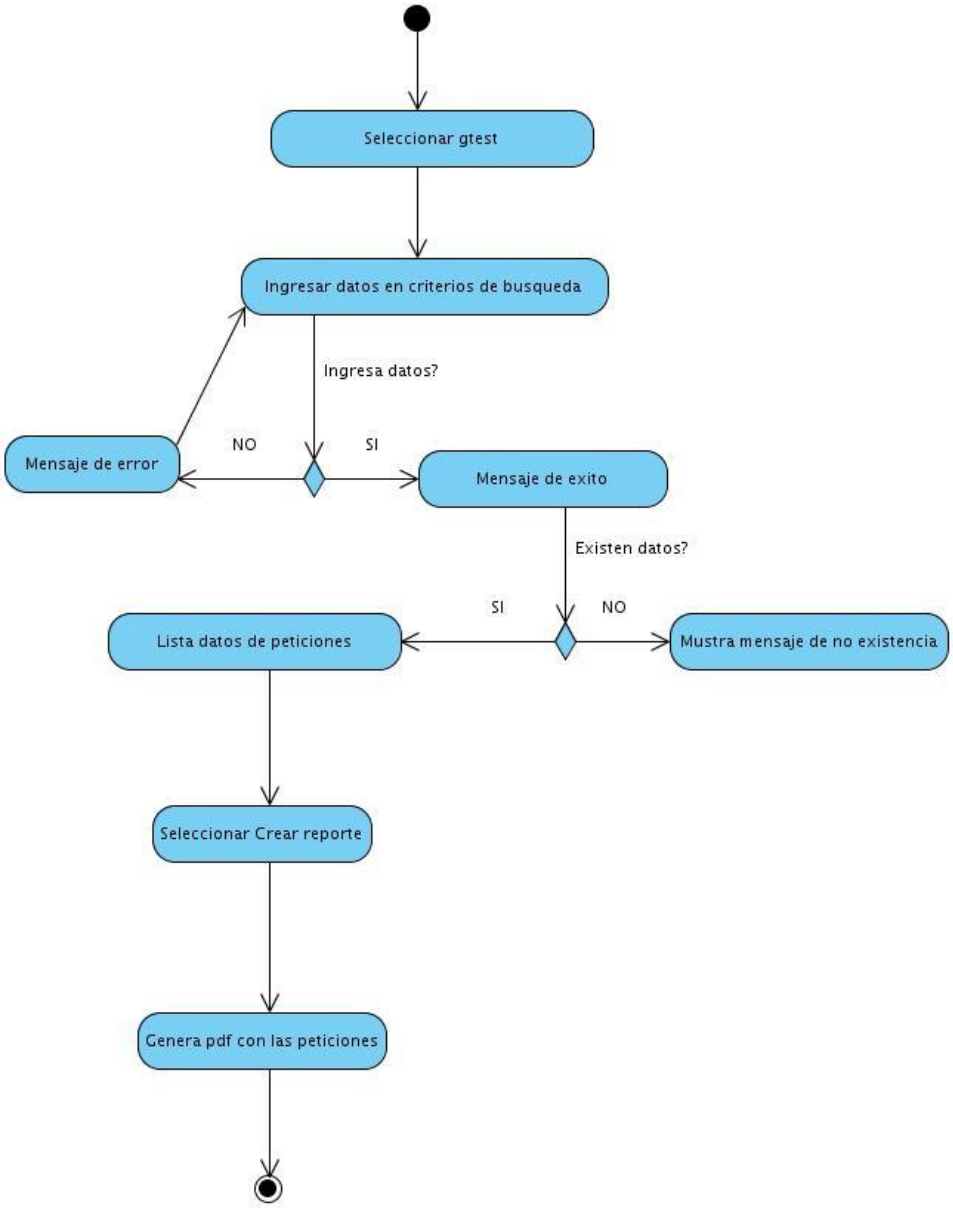


Figura 7. Diagrama de actividades

2.8 Conclusiones

En este capítulo se exponen los elementos fundamentales del análisis y diseño. Se analiza la propuesta de solución basada en el modelo del dominio. Se describen las historias de los usuarios y el caso de uso del sistema. Se definen los requisitos no funcionales que debe cumplir el plugin y se sientan las bases para la implementación.

Capítulo III: Implementación

3.1 Introducción

Como parte del desarrollo del plugin un elemento clave es su implementación por lo que hacer un análisis de la misma se torna imprescindible. En el presente capítulo se exponen los diagramas de despliegue y de componentes y se realiza una evaluación del funcionamiento del framework Rails donde se muestra la estructura del mismo.

3.2 Modelo de despliegue

El Modelo de Despliegue o diagrama de despliegue es un tipo de diagrama que se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes. En muchos casos el modelado de la vista de despliegue implica modelar la topología del hardware sobre el que se ejecuta el sistema.

El modelo de despliegue que se presenta está compuesto por tres nodos que representan la computadora del usuario del sistema, el servidor de la herramienta REDMINE interconectados por el protocolo HTTP y la base de datos con la cual interactúa el sistema.

Figura 8. Modelo de despliegue

3.3 Diagrama de componentes

Modelan la vista estática de un sistema y describen los elementos físicos, los mismos pueden ser archivos, librerías, módulos, ejecutables, o paquetes y pueden ser usados para modelar y documentar cualquier arquitectura de sistema. Además en los diagramas de componentes se muestran los elementos de diseño de un sistema y por otra parte permiten visualizar con más facilidad la estructura general del sistema y el comportamiento del servicio que estos componentes proporcionan y utilizan a través de las interfaces.

En el siguiente diagrama de componentes, compuesto por diferentes archivos de extensión erb o rb, que responden al funcionamiento del sistema del lado del cliente con el uso del framework Ruby on Rails, se representan físicamente cada una de las clases que se necesitan para el funcionamiento del plugin.

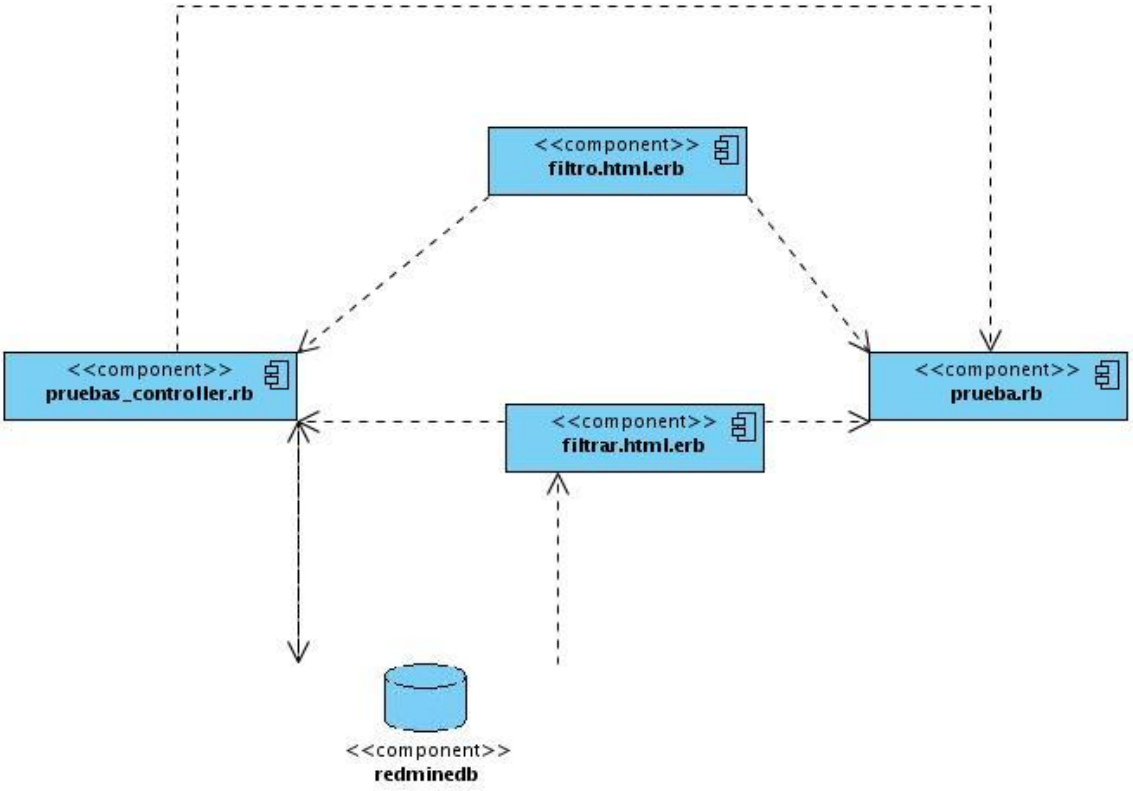


Figura 9. Diagrama de componentes

3.4 Framework de desarrollo Ruby on Rails

3.4.1 Descripción del Framework Ruby on Rails

Rails es una plataforma de aplicaciones Web de código abierto la cual fue desarrollada en el lenguaje de programación Ruby. La simplicidad en el código y en la configuración son las principales ventajas al desarrollar aplicaciones ya que se ahorra código en comparación con otros frameworks. Esto permite al programador ahorrar tiempo en la producción de código y crear una sintaxis que muchos de sus usuarios encuentran muy legible. La plataforma Rails se distribuye a través de RubyGems, que es el formato oficial de paquete y canal de distribución de librerías y aplicaciones Ruby.
Ver anexo 2.

3.4.2 Estructura del Framework Ruby on Rails

Dentro de este framework existe una gran estructura de paquetes y carpetas de las cuales las más importantes se describen a continuación:

Los subdirectorios más importantes dentro de este lenguaje son:

- app: contiene el core de la aplicación, la división entre los subdirectorios model, view, controller y helper.

Subdirectorios de app:

  - controllers: aquí es donde el framework pretende encontrar las clases controladoras. el objetivo del controlador es manejar las peticiones web realizadas por los usuarios desde el navegador.
  - views: sujeta las plantillas de visualización que se llenarán con los datos de la aplicación, las mismas que serán convertidas en HTML y devueltas al navegador del usuario.
  - models: posee las clases de modelación de datos almacenados en la base de datos de la aplicación.
  - helpers: en ella se guardan las clases de ayuda que se usan para asistir a las clases tanto models como views y los controladores. Esto ayuda a que dichas clases se mantengan sin crecer de tamaño, dedicadas y ordenadas a funciones específicas.

- config: contiene el archivo database.yml que proporciona los detalles de configuración y acceso a la base de datos que se usa en la aplicación.
  - locales: se almacenan los archivos dedicados a los idiomas de la aplicación permitiendo la fácil migración a cualquiera de estos.
- db: contiene archivos SQL para la creación de tablas en la base de datos.
- log: contiene un registro de todas las acciones que Rails lleva a cabo, muy útil para rastrear errores.
- public: es el directorio disponible para Apache, que incluye subdirectorios de imágenes, javascripts, y de hojas de estilos.

Ver anexo 3.

3.4.3 Clases fundamentales dentro del lenguaje

ActionController: Son el núcleo de una solicitud web en Rails. Se componen de una o más acciones que se ejecutan y luego, o bien hacer una plantilla o redirigir a otra acción. Una acción se define como un método público en el controlador, lo que automáticamente se pondrá a disposición del servidor a través de las rutas de Rails.

```
class PruebasController < ApplicationController


  def index
    @pruebasf =Prueba.find_all_by_nombre
    @pruebas = Prueba.all

    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @pruebas }
    end
  end
```

Figura 10. ActionController

ActiveRecord: La representación de los datos y las relaciones entre ellos (llamado también lógica de negocios) consiste en las clases que representan a las tablas de la base de datos y las mismas son gestionadas por ActiveRecord. Por lo general, lo único que tiene que hacer el programador es heredar de la clase ActiveRecord::Base, y el programa averiguará automáticamente qué tabla usar y qué columnas tiene.

```
class Prueba < ActiveRecord::Base

  def self.find_all_by_nombre
    find (:all, :conditions => {:nombre =>  ["humbert"]})

  end
end
```

Figura 11. ActiveRecord

Estas son las principales clases que dan funcionamiento al framework Ruby on Rails aunque claramente sin todas las otras que existen dentro de las estructura de app la aplicación no funciona.

3.5 Descripción de las principales clases y funcionalidades del plugin

pruebas_controller.rb: como bien el nombre lo indica esta es la clase que se encuentra dentro del paquete Controllers control donde se manejan las peticiones realizadas por el usuario y se le da funcionalidad al plugin para que posteriormente muestre los resultados en la vista.

Esta clase contiene el método **filtrar** que es el de mayor peso dentro del plugin. En este método se recoge mediante parámetros los datos de las peticiones que se necesitan y en él se realiza una búsqueda dentro de la base de datos de la herramienta REDMINE y se envían los resultados al HTML **filtro.html.erb**.

```ruby
def filtrar
  @proyecto = params[:pruebas][:proyecto]
  @estado = params[:pruebas][:estado]
  @fechai = params[:pruebas][:fecha_inicio]
  @fechaf = params[:pruebas][:fecha_fin]
  @t_error = params[:pruebas][:t_error]
  @iteracion = params[:pruebas][:iteracion]
  limit = 20

  respond_to do |format|
    if @proyecto == ""
      flash[:error] = l(:error_project_empty)
      format.html { redirect_to :action => "index",:project_id => Project.find(params[:project_id]) }
    elsif @fechaf < @fechai
      flash[:error] = l(:error_date)
      format.html { redirect_to :action => "index",:project_id => Project.find(params[:project_id]) }
    else
      @proj = Project.find(:all, :conditions => ["id = :proyecto", params[:pruebas]])
      @count = Issue.find(:all,
        :conditions => [ " project_id = #{@proyecto} and "+
            cestado(@estado)+
            cfecha(@fechai,@fechaf)+
            "tracker_id = '1' and "+
            cvalue(@t_error,@iteracion),params[:pruebas]],
        :joins=>"inner join custom_values E on issues.id = E.customized_id"+
          cjoin(@t_error, @iteracion)).count
      @pruebas_page = Paginator.new self, @count, limit, params['page']
```

Figura 12. Muestra del método filtrar

prueba.rb: ubicada dentro del paquete Models esta se encarga de la lógica de negocio del sistema además de validar la presencia de campos obligatorios dentro de la aplicación.

index.html.erb: presente dentro de las Views su función es ejercer de formulario para que el usuario introduzca los criterios de búsquedas para filtrar las peticiones necesarias.

En esta clase se utiliza tanto código HTML como Ruby para poder crear el formulario de forma dinámica extrayendo los datos de la base de datos y así procurar que ante cualquier cambio que ocurra en un futuro, el plugin sea capaz de adaptarse a los mismos.

```
<h2>Filtrar peticiones por criterios de búsqueda</h2>
<% form_for :pruebas, @pruebas  , :url => {:action => "filtrar", :project_id => @project} do |f| %>
  <%= f.error_messages %>

  <div class="box">
    <table>
      <tr>
        <td>
          <p>
            <%= f.label l(:label_proyecto) %>:<b>  </b><b style="color: red">*</b>
          </p>
          <p></p>
        </td>
        <td>
          <p>
            <%=  select("pruebas", :proyecto, Project.all.collect {|p| [ p.name, p.id ] },{:prompt => 'Seleccione .
          </p>
        </td>
      </tr>
      <tr>
        <td>
          <p>
            <%= f.label l(:label_estado) %>:
          </p>
          <p></p>
        </td>
        <td>
          <p>
            <%=  select("pruebas",:estado,  IssueStatus.all.collect {|p| [ p.name, p.id ] },{:prompt => 'Seleccione
          </p>
        </td>
      </tr>
```

Figura 13. Muestra de la clase index.html.erb

filtro.html.erb: también dentro de las Views, esta clase junto a _list.html.rb se encarga de mostrar los resultados de las peticiones que cumplan con los criterios de búsqueda que el usuario específico en la index. En estas también se combina código de RAILS y HTML.

```
1  <% @proj.each do |a| %>
2    <h2><%=  l(:label_reported) %> <%=h a.name%></h2>
3  <%end%>
4
5  <% if @pruebasf.any? %>
6    <%= render :partial => 'pruebas/list' %>
7    <p class="pagination"><%= pagination_links_full @pruebas_page%></p>
8
9  <%else %>
10   <p class="nodata"><%= l(:label_no_data) %></p>
11 <%end%>
12
13 <%= link_to '&#171; ' + (l(:button_back)), :action => 'index',:project_id => @project%>
14
15
```

Figura 14. Muestra de la clase filtro.html.erb

```
6    <table class="list">
7      <thead>
8        <tr>
9          <th><p> #</p> </th>
10         <th><p><%=  l(:button_tema) %></p> </th>
11         <th><p><%=  l(:button_estado) %> </p> </th>
12         <th><p><%=  l(:button_iteracion) %></p> </th>
13         <th><p><%=  l(:button_t_error) %> </p> </th>
14         <th><p><%=  l(:button_significativa) %>  </p> </th>
15         <th><p><%=  l(:button_fecha_inicio) %> </p> </th>
16         <th><p><%=  l(:button_fecha_fin) %> </p> </th>
17
18       </tr>
19     </thead>
20
21     <tbody  align="center" class="list">
22       <% @pruebasf.each do |prueba| %>
23         <% sign = CustomValue.find_by_sql("SELECT  value  FROM custom_values WHERE cust
24         <tr>
25           <td class="id"><%= link_to prueba.id, :controller => 'issues', :action => 'sh
26           <td><p><%=h prueba.subject %></p></td>
27           <td><p><%=h IssueStatus.find_by_sql("SELECT  name  FROM issue_statuses WHERE
28           <td><p><%=h CustomValue.find_by_sql("SELECT  value  FROM custom_values WHERE
29           <td><p><%=h CustomValue.find_by_sql("SELECT  value  FROM custom_values WHERE
30           <% if sign.empty? %>
31             <td></td>
32           <%else%>
33             <td align="center"><img alt="Toggle_check" src="/images/toggle_check.png?12
```

Figura 15. Muestra de la clase _list.html.erb

init.rb: Esta clase se encarga de la configuración del plugin. La misma permite gestionar una serie de opciones como son: nombre del plugin, autores, descripción así como la versión. Además se manejan elementos claves relacionados con la seguridad. Permite también ubicar la interfaz principal del plugin dentro de la herramienta donde se vaya a agregar determinando su accesibilidad según defina el administrador del sistema.

```ruby
require 'redmine'

Redmine::Plugin.register :redmine_pruebas do
  name 'Redmine Gtest plugin'
  author 'Alejandro Alvarez Hdez y Humberto Rodriguez Pelaez'
  description 'Provide a report for the liberation issues'
  version '0.1.45 beta'

  project_module :Gtest do
    permission :ver_gtest, {:pruebas => [:index, :filtrar] }

    menu :project_menu, :pruebas, { :controller => 'pruebas', :action => 'index' },
      :caption => 'Gtest', :after => :activity, :param => :project_id

  end
end
```

Figura 16. Muestra de la clase init.rb

3.6 Normas de los comentarios

Con el objetivo de contar con un código más legible y entendible para su posterior uso y desarrollo, se lleva a cabo la inserción de comentarios en el código que se genera durante la implementación del sistema logrando así un mejor entendimiento para en un futuro poder hacer cambios mucho más fácil. Estos comentarios deben seguir una normativa para que se considere que fue descrito por un solo desarrollador aunque en realidad se haya elaborado por varias personas quedando así estructurados de una misma forma para que se haga más fácil su entendimiento. Todo esto facilita el proceso de soporte y ampliación del plugin.

Comentarios en Rails

Los comentarios dentro de las clases de lenguaje Ruby se insertan con el signo # para que no se interpreten las líneas de código que son antecedidas por este símbolo.

```ruby
# Filters added to this controller apply to all controllers in the application.
# Likewise, all the methods added will be available for all controllers.

class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time
  protect_from_forgery # See ActionController::RequestForgeryProtection for details
```

Figura 17. Comentarios en Rails

Clases

Antes de la implementación de una clase se debe dar una breve descripción de los resultados y funciones que se quieren obtener con la implementación de la misma y hacer énfasis en posibles datos que puedan resultar de difícil entendimiento. Estas deben estar compuestas por un nombre que identifique la acción de la clase y la descripción de la misma.

La estructura de estos comentarios es la siguiente:

```
##########################################################################################
# Configuración del plugin
#
#En esta clase se configuran todas las funcionalidades del plugin, posicion
#de este dentro de la aplicacion y permisos.
##########################################################################################
require 'redmine'

Redmine::Plugin.register :redmine_pruebas do
  name 'Plugin para la gestión de pruebas en el REDMINE'
```

Figura 18. Comentarios en las clases

Funciones

La nomenclatura descrita anteriormente es completamente utilizable en las funciones, la diferencia radica en que los comentarios se introducen encima de las funciones y no al empezar de la clase.

```
##########################################################################################
#Filtrar
#Con esta función se filtran todas las peticiones por los criterios de busqueda
##########################################################################################

  def filtro
    @prueba = params[:pruebas]
    @pru = Prueba.new(params[:pruebas])
```

Figura 19. Comentarios en las funciones

Comentarios por líneas

En las líneas de código donde sea necesario enfatizar en algún detalle o especificar alguna funcionalidad se pone acto seguido del código haciéndose anteceder por la nomenclatura de la inserción de comentarios.

```
flash[:notice] = 'Reporte creado'    #Mensaje de exito del sistema
format.html { render :action => "filtro" }
format.xml  { render :xml => @pruebas, :status => :created, :location => @pruebas}
```

Figura 20. Comentarios por líneas

3.7 Tratamiento de errores

Se tuvo en cuenta el tratamiento de errores a través de mensajes de fácil comprensión y lo más descriptivos posible, manteniendo al usuario informando de posibles riesgos o errores cometidos.

Los principales errores que puedan existir se tratan en la clase controladora y en la modelo. En la clase controladora se tratan principalmente los errores de funcionalidad del sistema y en la modelo se tratan los errores de falta de datos o datos erróneos.

Tratamiento de errores en la controladora:

Este tratamiento se lleva mediante funciones que dan como resultado que se muestren los mensajes en las vistas.

```
def filtro
  @prueba = params[:pruebas]
  @pru = Prueba.new(params[:pruebas])

  respond_to do |format|
    if @pru.proyecto!=""
      @proj = Project.find(:all, :conditions => ["id = :proyecto", params[:pruebas]])
      @pruebasf= Issue.find(:all,
        :conditions => [" project_id = :proyecto"+
            " and tracker_id = '1' and "+vfecha( @pru.fecha_inicio,@pru.fecha_fin)+
            " value = :t_error",params[:pruebas]],
        :joins=>"INNER JOIN custom_values ON custom_values.customized_id = issues.id")

      flash[:notice] = 'Reporte creado'
      format.html { render :action => "filtro" }
      format.xml  { render :xml => @pruebas, :status => :created, :location => @pruebas}
    else
      flash[:error] = 'Error en los datos'
      format.html { redirect_to :action => "filtrar" }
      format.xml  { render :xml => @pruebas.errors, :status => :unprocessable_entity }

    end
  end
end
end
```

Figura 21. Tratamiento de errores en la controladora

Tratamiento de errores en la modelo:

En la clase modelo se controlan la integridad de los datos mediante el manejo de errores dentro del framework, estos se utilizan de la siguiente forma:

```
class Prueba < ActiveRecord::Base

  validates_presence_of :proyecto, :message => " Debe insertar un proyecto"

end
```

Figura 22. Tratamiento de errores en la modelo

3.8 Las consultas SQL

La utilización de consultas dentro del desarrollo y funcionamiento del plugin es fundamental ya que estas componen la mayor parte de las funcionalidades y operaciones de la aplicación y sin las mismas no se pudiera dar solución a muchas de las operaciones que se desarrollan en el plugin. Estas consultas se tratan de diferentes formas en el software las cuales se explican a continuación:

Consultas SQL.

Se utilizan las consultas SQL para obtener datos que no necesiten variables y parámetros o contengan pocos de ellos en su interior. Estas consultas utilizan la sintaxis común del lenguaje SQL pero tienen que estar convocadas por la función del framework find_by_sql.

```
<tr>
  <td>
    <p>
      <%= f.label :t_error, 'Tipo de error' %>:
    </p>
  </td>
  <td>
    <p>
      <%=  select("pruebas",:t_error,
        CustomValue.find_by_sql("SELECT DISTINCT value
                                 FROM custom_values
                                 WHERE custom_field_id = '5' ORDER BY value asc ")  ,
        {:prompt => 'Seleccione ...'}) %>
    </p>
  </td>
</tr>
```

Figura 23. Consultas sql

Consultas en RoR

Este framework presenta diferentes funcionalidades de búsqueda que en su interior funcionan como consultas SQL pero que se definen mediante código Ruby y son factibles cuando existe la necesidad de utilizar parámetros y variables dentro de ellas. Como se muestra en la figura la sentencia :conditions representa lo que sería un where en sql, así mismo es :joins que es donde se declaran los joins de la consulta. Aquí también se pueden agregar otros parámetros como son: limit, offset, order_by entre otros.

```
@pruebasf= Issue.find(:all,
  :conditions => [" project_id = :proyecto"+
      " and tracker_id = '1' and "+vfecha( @pru.fecha_inicio,@pru.fecha_fin)+
      " value = :t_error",params[:pruebas]],
  :joins=>"INNER JOIN custom_values ON custom_values.customized_id = issues.id")
```

Figura 24. Consultas sql en RoR

En la variable @pruebasf se guarda un arreglo de objetos con todos los datos de la consulta. Las líneas de códigos anteriores significan lo mismo que:

@pruebasf = SELECT * FROM Issue INNER JOIN custom_values ON custom_values.customized_id = issue.id WHERE issue.project_id = #{:proyecto} AND issue.tracker_id = ‘1’ AND fecha_inicio = #{fecha_inicio} AND fecha_fin = #{fecha_fin} AND custom_values.value = #{t_error};

Todos los campos que se muestran encerrados entre el código #{variable} son parámetros que se obtienen de la vista.

3.9 Conclusiones

En el capítulo se pudieron analizar aspectos importantes del desarrollo del plugin como fueron el modelo de despliegue, diagrama de componentes así como una serie de elementos fundamentales relacionados con la implementación como son la descripción del framework Ruby on Rails así como su estructura. También se realizó una descripción de las principales clases y funcionalidades del plugin y se abordaron temas de tratamiento de errores, consultas a la base de datos y nomenclatura para los comentarios.

Capítulo IV:

Pruebas, validación e instalación

4.1 Introducción

Las aplicaciones informáticas no están exentas a errores por lo que a todas ellas es necesario aplicarles un conjunto de pruebas para garantizar que cuentan con la calidad requerida, este es el factor fundamental para lograr entregar un software que cumpla o supere las expectativas del cliente.

Para la realización del proceso de pruebas en la solución obtenida se pone en práctica la estrategia de pruebas definida por el nivel, tipo y métodos de prueba representados en los siguientes subepígrafes. Además se realiza la validación del diseño y se explica el proceso de instalación del plugin.

4.2 Nivel de prueba: Prueba de sistema

Las pruebas de sistema son imprescindibles a la hora de verificar el correcto desempeño y funcionamiento del sistema, valga la redundancia. Son pruebas que persiguen demostrar la fortaleza y robustez del sistema, aún en condiciones críticas. Las mismas verifican los requisitos tanto funcionales como no funcionales.

4.3 Tipo de prueba: Funcionalidad

Luego del desarrollo del plugin, es necesario verificar que su funcionamiento es correcto, que cumple los objetivos de su diseño y con las expectativas del cliente. Las pruebas de funcionalidad determinan la medida en la que la aplicación satisface los requisitos funcionales. Durante el proceso de pruebas se simulan varios escenarios con distintos juegos de datos para confirmar que todos los resultados son los esperados.

4.4 Métodos de pruebas

4.4.1 Pruebas de caja blanca o estructurales

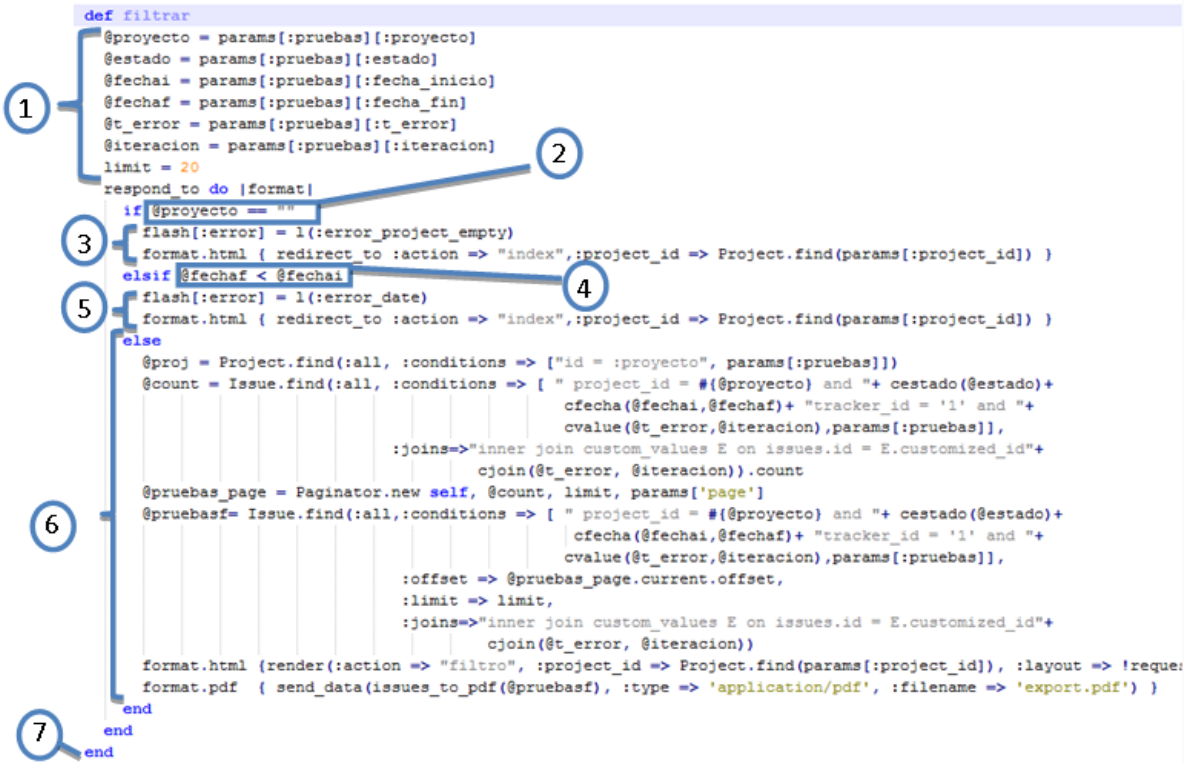
Se denomina pruebas de caja blanca a un tipo de prueba de software que se realiza sobre las funciones internas de un módulo, las mismas se llevan a cabo en primer lugar sobre un módulo concreto, para luego realizar las de caja negra sobre varios subsistemas.

Las pruebas de caja blanca intentan garantizar que:

- ✓ Se ejecutan al menos una vez todos los caminos independientes de cada módulo.

- ✓ Se utilizan las decisiones en su parte verdadera y en su parte falsa.
- ✓ Se ejecuten todos los bucles en sus límites.
- ✓ Se utilizan todas las estructuras de datos internas.

Para realizar la prueba es necesario realizar primeramente el análisis de complejidad del algoritmo sobre el que se va a realizar la prueba, con el propósito de calcular los valores de la complejidad ciclomática.




```
def filtrar
  @proyecto = params[:pruebas][:proyecto]
  @estado = params[:pruebas][:estado]
  @fechai = params[:pruebas][:fecha_inicio]
  @fechaf = params[:pruebas][:fecha_fin]
  @t_error = params[:pruebas][:t_error]
  @iteracion = params[:pruebas][:iteracion]
  limit = 20
  respond_to do |format|
    if @proyecto == ""
      flash[:error] = l(:error_project_empty)
      format.html { redirect_to :action => "index",:project_id => Project.find(params[:project_id]) }
    elsif @fechaf < @fechai
      flash[:error] = l(:error_date)
      format.html { redirect_to :action => "index",:project_id => Project.find(params[:project_id]) }
    else
      @proj = Project.find(:all, :conditions => ["id = :proyecto", params[:pruebas]])
      @count = Issue.find(:all, :conditions => [ " project_id = #{@proyecto} and "+ cestado(@estado)+
                                                 cfecha(@fechai,@fechaf)+ "tracker_id = '1' and "+
                                                 cvalue(@t_error,@iteracion),params[:pruebas]],
                                :joins=>"inner join custom_values E on issues.id = E.customized_id"+
                                        cjoin(@t_error, @iteracion)).count
      @pruebas_page = Paginator.new self, @count, limit, params['page']
      @pruebasf= Issue.find(:all,:conditions => [ " project_id = #{@proyecto} and "+ cestado(@estado)+
                                                  cfecha(@fechai,@fechaf)+ "tracker_id = '1' and "+
                                                  cvalue(@t_error,@iteracion),params[:pruebas]],
                                :offset => @pruebas_page.current.offset,
                                :limit => limit,
                                :joins=>"inner join custom_values E on issues.id = E.customized_id"+
                                        cjoin(@t_error, @iteracion))
      format.html {render(:action => "filtro", :project_id => Project.find(params[:project_id]), :layout => !reque
      format.pdf  { send_data(issues_to_pdf(@pruebasf), :type => 'application/pdf', :filename => 'export.pdf') }
    end
  end
end
```

Figura 25. Representación del algoritmo Filtrar ()

Lugo de este paso, es necesario representar el grafo de flujo asociado (figura 26), en el cual se representan distintos componentes como es el caso de:

Nodo: Son los círculos representados en el grafo de flujo, el cual representa una o más secuencias del procedimiento, donde un nodo corresponde a una secuencia de procesos o a una sentencia de decisión. Los nodos que no están asociados se utilizan al inicio y final del grafo.

Nodo predicado: Son los nodos que contienen una condición y se caracterizan porque de ellos salen dos o más aristas.

Aristas: Son constituidas por las flechas del grafo, son iguales a las representadas en un diagrama de flujo y constituyen el flujo de control del procedimiento. Las aristas terminan en un nodo, aún cuando el nodo no representa la sentencia de un procedimiento.

Regiones: Son las áreas delimitadas por las aristas y nodos donde se incluye el área exterior del grafo, como una región más. Las regiones se enumeran siendo la cantidad de regiones equivalente a la cantidad de caminos independientes del conjunto básico de un procedimiento.

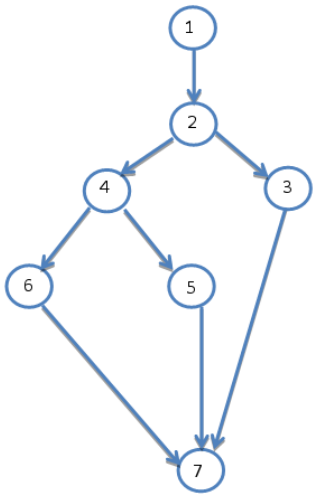


Figura 26. Grafo de Flujo asociado al algoritmo Filtrar ()

La complejidad ciclomática coincide con el número de regiones del grafo de flujo
La complejidad ciclomática, V (G), de un grafo de flujo G, se define como
V (G) = Aristas - Nodos + 2
La complejidad ciclomática, V (G), de un grafo de flujo G, también se define como
V (G) = Nodos de predicado + 1

A partir del grafo de flujo de la figura 26, la complejidad ciclomática sería:
Como el grafo tiene tres regiones, **V (G) = 3**
Como el grafo tiene 8 aristas y 7 nodos, **V (G) = 8 – 7 + 2 = 3**
Como el grafo tiene 2 nodos predicado, **V (G) = 2 + 1 = 3**

Teniendo la complejidad ciclomática obtenemos el número de caminos independientes, que nos dan un valor límite para el número de pruebas que tenemos que diseñar.
En el ejemplo, el número de caminos independientes es 3, y estos son:

- 1-2-3-7
- 1-2-4-5-7
- 1-2-4-6-7

Para cada camino se realiza un caso de prueba:

✓ **Caso de prueba para el Camino básico #1:**

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: El proyecto es un número entero.
Condición de ejecución: El proyecto es igual a vacío.
Entrada: @proyecto = “”
Resultados esperados: Se espera que el sistema lance un mensaje de error pues el proyecto es un campo obligatorio.

- ✓ **Caso de prueba para el Camino básico #2:**

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: El proyecto es un número entero, las fechas de inicio y fin deben ser números enteros.
Condición de ejecución: El proyecto es igual a 184, fecha inicio es igual a 2011-06-08 y fecha fin es igual a 2011-06-01.
Entrada: @proyecto = “184”, @fechai = “2011-06-08” y @fechaf = “2011-06-01”.
Resultados esperados: Se espera que el sistema lance un mensaje de error pues la fecha fin debe ser posterior o igual a la fecha de comienzo.

- ✓ **Caso de prueba para el Camino básico #3:**

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: El proyecto y el estado son números enteros, tipo de error e iteración son cadenas de caracteres, las fechas de inicio y fin deben ser números enteros.
Condición de ejecución: El proyecto es igual a 184, estado es igual 14, tipo de error es igual a De documento, iteración es igual a Iteración 1, fecha inicio es igual a 2011-06-01 y fecha fin es igual a 2011-06-08.
Entrada: @proyecto = “184”, @estado = “14”, @t_error = “De documento”, @iteracion = “Iteración 1”, @fechai = “2011-06-08” y @fechaf = “2011-06-01”.
Resultados esperados: Se espera que el sistema genere un reporte mostrando los datos que cumplan con los parámetros seleccionados.

Luego del equipo de desarrollo aplicar los casos de prueba descritos anteriormente se comprobó que el flujo de trabajo de la función es correcto pues cumplió con las condiciones necesarias de la prueba.

4.4.2 Pruebas de caja negra

Dentro de todo el proceso de pruebas las de caja negra sin lugar a dudas cumplen un papel fundamental, de las mismas nos interesará su forma de interactuar con el plugin, entendiendo qué es lo que hace, pero sin dar importancia a cómo lo hace, ya que este aspecto fue analizado en las pruebas de caja blanca. En una prueba de caja negra deben estar muy bien definidas sus entradas y salidas, es decir, su interfaz; en cambio, no se precisa definir ni conocer los detalles internos de su funcionamiento.

Se puede decir entonces que son pruebas funcionales que se aplican al sistema empleando un conjunto de datos de entrada y observando las salidas obtenidas para determinar si la función se está desarrollando correctamente por el sistema. Su realización parte de los requisitos funcionales. Algunos ejemplos básicos de pruebas de caja negra son la comprobación de valores límite, pruebas de integridad de la base de datos o pruebas de excepciones. Para las pruebas a la aplicación se introdujeron diferentes valores con el objetivo de verificar el correcto funcionamiento de la misma en diferentes escenarios, al introducir datos válidos e inválidos a partir de los requerimientos funcionales.

Con este tipo de pruebas se intenta detectar:

- ✓ Funciones incorrectas o ausentes.
- ✓ Errores de interfaz.
- ✓ Errores en estructuras de datos o en accesos a las bases de datos externas.
- ✓ Problemas de rendimiento.

Las pruebas de caja negra se realizaron utilizando el Diseño de caso de prueba que se muestra en el Anexo 5.

4.5 Entorno de prueba

Para demostrar la validez de la solución se ejecutó el plugin sobre el REDMINE del centro CEIGE, se decidió aplicarlo en el Departamento de Desarrollo de Productos específicamente la línea Capital Humano la cual contaba con no conformidades de liberación creadas previamente. Todo esto se realizó supervisado por el Grupo de Gestión de la Calidad de Software que se encarga de evaluar y verificar la calidad de los productos informáticos producidos por el centro.

4.6 Clases de prueba

Luego de haber analizado la estrategia para realizar las pruebas y seleccionado el tipo de prueba de caja negra para llevar a cabo en el proceso de validación de la solución se decide realizar distintas clases de prueba las cuales aportarán un conjunto de información sobre el funcionamiento del sistema. Estas clases son:

4.6.1 Prueba de interfaz

Las pruebas de interfaz tienen como objetivo verificar todas las interfaces del sistema en busca de posibles errores tanto de diseño como de funcionalidad. Para darle cumplimiento a esta prueba se ejecutaron todas las pantallas del plugin, comprobando así el correcto funcionamiento de las mismas y la existencia de todos los campos y atributos definidos.

Se ejecutó el plugin seleccionando la opción Gtest en el menú del REDMINE y se procedió a Filtrar peticiones por criterios de búsqueda para generar un Reporte de pruebas de liberación a partir de la pantalla perteneciente al mismo requisito funcional en la cual se pudo observar la existencia de campos que permiten definir todos los datos necesarios para crear un reporte, ya sea el proyecto (único campo obligatorio), el estado, el tipo de error, la iteración o la fecha de inicio y fin.

4.6.2 Prueba de validación

Mediante esta prueba se verifica el correcto tratamiento de los errores en la aplicación. Para obtener resultados en esta clase de prueba se procedió con la introducción de distintos datos en cada campo de la interfaz principal del plugin donde el usuario puede cometer errores a la hora de seleccionar los criterios por los cuales va a filtrar, con el objetivo de verificar que el sistema no permitiese la introducción de datos incorrectos, que pudiesen provocar un mal funcionamiento del mismo.

4.6.3 Prueba de cubrimiento

Las pruebas de cubrimiento junto a las de interfaz y de validación se llevaban a cabo conjuntamente, y se encargan de comprobar que todas las funciones se ejecuten correctamente.

En el caso de Filtrar peticiones por criterios de búsqueda después de introducir todos los datos para esta pantalla (ver anexo 6), funcionó correctamente mostrando el reporte generado según los criterios que especificó el usuario (ver anexo 7).

4.6.4 Prueba de rendimiento

El rendimiento del sistema es un eslabón clave dentro de toda aplicación informática, mediante las pruebas de rendimiento se determina lo rápido que realiza una tarea la aplicación y aseguran un mejor desempeño del producto acabado.

El rendimiento del plugin depende en gran medida del REDMINE pues es dentro de este donde se ejecutará. Por otra parte no presenta problemas de rendimiento llegando a ejecutarse rápidamente. Donde puede demorarse un poco es a la hora de generar el reporte pues el sistema solicita un gran cúmulo de información de la base de datos aunque esto se minimiza haciendo uso del paginado en la interfaz de reporte ya que solo se cargan 25 resultados a la vez, siendo el resto mostrado solamente cuando el usuario cambie de página.

Luego de haber realizado las pruebas se detectaron un total de cinco no conformidades las cuales fueron resueltas satisfactoriamente en solo tres iteraciones.

4.7 Validación del diseño

La validación del diseño es de vital importancia dentro del desarrollo de todo producto informático, a continuación se realiza una evaluación del diseño propuesto del plugin. Las métricas empleadas están diseñadas para evaluar los siguientes atributos de calidad:

- ✓ *Responsabilidad.* Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- ✓ *Complejidad de implementación.* Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- ✓ *Reutilización.* Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- ✓ *Acoplamiento.* Consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.
- ✓ *Complejidad del mantenimiento.* Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- ✓ *Cantidad de pruebas.* Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (componente, módulo, clase, conjunto de clases, etc.) diseñado.

Sin restarle importancia a la validación del diseño y realizando un análisis de los atributos de calidad tamaño operacional de clase (TOC) y relación entre clases (RC) los cuales tienen como atributo principal la cantidad de clases del sistema, se decidió no realizar la validación del diseño ya que la misma no cumple objetivo pues en el mejor de los casos su evaluación da 50% pues el plugin solamente presenta dos clases, pruebas_controller y Prueba_model las cuales contienen siete y un procedimientos respectivamente.

4.8 Manual de instalación del plugin

Como una parte fundamental del producto se presenta la instalación la cual representa un elemento fundamental y reviste una gran importancia ya que una mala instalación podría conllevar grandes problemas de funcionamiento, estabilidad y seguridad.

4.8.1 Pre-requisitos

Es necesario antes de instalar la aplicación verificar la existencia de los siguientes programas dentro del servidor:

- ✓ Ruby 1.8
- ✓ Base de datos PostgreSQL 8.0 o superior
- ✓ REDMINE 1.0.5 o inferior

4.8.2 Instalación

Primeramente se debe copiar la carpeta redmine_gtest (nombre de la carpeta del plugin) en el directorio /vendor/plugins (Ver Anexo 4) que es donde la herramienta para la gestión de proyecto REDMINE gestiona sus plugins y los ejecuta.

Para configurar el plugin primeramente se debe iniciar el servicio del REDMINE y comprobar que trabaja sin problemas.

4.8.3 Permisos

Para configurar los permisos del plugin se debe ingresar al REDMINE mediante la cuenta de administrador y acceder a la sección Administración que se encuentra en la parte superior de la página, luego seleccionar Perfiles y permisos y en la parte inferior de la página que se muestra acceder a Informe de permisos donde se encuentran los roles y permisos del sistema. Aquí se procede a dar autorización a los roles que tendrán acceso al plugin marcando el checkbox Ver gtest según el rol que se quiera aprobar. Ver figura 27.

Figura 27. Otorgamiento de permisos

4.8.4 Configuración del proyecto

Después de haber dado los permisos necesarios para la ejecución del plugin se debe configurar el proyecto para que el plugin sea visible. Autenticándose en el REDMINE con un usuario con privilegios para configurar el proyecto al cual se le quiere dar acceso al plugin se debe acceder al menú Configuración y en la pestaña Módulos marcar el plugin en cuestión tal como se muestra en la figura 28.

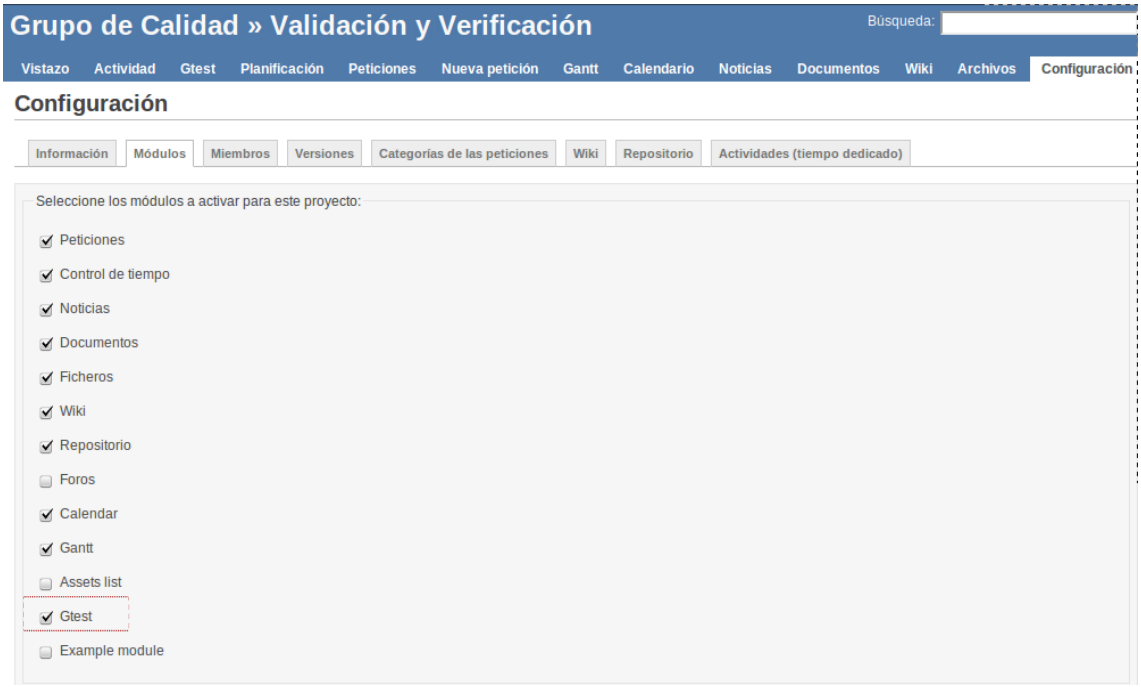


Figura 28. Configuración del proyecto

Por último comprobar que las configuraciones hayan tenido efecto y chequear que el plugin sea visible y funcione correctamente.

4.9 Conclusiones

Durante el desarrollo del capítulo se pudieron analizar las pruebas de software y luego de haber realizado las mismas al caso de uso del sistema y al plugin se puede asegurar que el sistema funciona correctamente ya que este da cumplimiento a todos los requerimientos funcionales definidos durante el análisis y diseño de la aplicación. No se asegura el cumplimiento de los requisitos no funcionales dado que el producto no fue sometido a pruebas para la verificación de los mismos.

Por último y no menos importante se abordó la instalación del plugin elemento fundamental para lograr un correcto funcionamiento y utilización del sistema.

Conclusiones generales

Una vez terminado el presente trabajo de diploma se puede concluir que se desarrollaron todas las tareas a fin de cumplir los objetivos propuestos, para ello:

- ✓ Se realizó un estudio de las tecnologías, herramientas y metodologías existentes en el mundo para el desarrollo de la solución, sirviendo este para seleccionar, como base para el proceso de desarrollo, la metodología ágil XP.
- ✓ Se analizaron los procesos de pruebas en el laboratorio de calidad para el levantamiento de los principales requisitos y funcionalidades para la elaboración del plugin.
- ✓ Se utilizó para el diseño e implementación el patrón arquitectónico Modelo-Vista-Controlador lo que facilitó un mejor rendimiento, reutilización del código y organización de las funcionalidades del sistema. Por lo que se garantiza el óptimo funcionamiento de la herramienta.
- ✓ Se utilizaron para la implementación del software un conjunto de herramientas y tecnologías la mayoría basadas en licencias de software libre, obteniendo un producto de alta independencia tecnológica y utilizable en diferentes plataformas.

Por todo lo antes planteado se puede concluir que este trabajo de diploma puede ser utilizado como línea de desarrollo para programadores de entornos web que utilicen el framework Ruby on Rails y fundamentalmente servirá de ejemplo para los interesados en el desarrollo de plugin para la herramienta de gestión de proyectos REDMINE. Como objetivo principal se logró obtener una herramienta para ser utilizada en los laboratorios de calidad cuya principal importancia radica en el análisis de los resultados obtenidos durante las pruebas realizadas permitiendo el ahorro de tiempo y recursos, así como el control eficiente de los cambios y la información manipulada.
Esta solución exhibe valor técnico, donde se destaca la incorporación de principios por los que se mide la factibilidad de un diseño de software, ejemplo: la utilización de patrones que posibilita la reutilización, garantizando la sostenibilidad y mantenimiento de la solución.

Recomendaciones

- ✓ Extender el uso del plugin a otros centros que hagan uso del REDMINE.
- ✓ Continuar con el desarrollo agregando nuevas funcionalidades para aumentar la calidad y robustez de la solución.
- ✓ Implementar las opciones de exportar a pdf y csv para lograr un mejor tratamiento de la información.
- ✓ Presentar este trabajo en futuros eventos científicos.

Bibliografía

1. **Stevens, Perdita, Pooley, Rob and Wesley, Addison.** *Utilización de UML en ingeniería de software con objetos y componentes.* 2007.

2. **Valencia, María Eugenia.** Universidad del Valle. [Online] 2011. http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software/PRUEBASoftware.pdf.

3. **Perissé, M. C.** *Una Metodología Simplificada.* 2001.

4. **Larman, Craig.** *UML y Patrones.* 2003.

5. **Fernandez, Obie.** *The Rails Way.* 2008.

6. **Perrotta, Paolo.** *The Pragmatic Bookshelf.* 2010.

7. **Hetzel, Bil.** *The Complete Guide to Software Testing.* 1983.

8. **Glenford, J. Myers.** *The Art of Software Testing.* 1979.

9. **Benson, Edward.** *The Art of Rails.* 2008.

10. **Feldt, Robert and Johnson, Lyle.** *Syngress - Ruby Developer's Guide.* 2002.

11. Sitio de Visual Paradigm. [Online] 2010. http://www.visual-paradigm.com/product/vpuml/.

12. Sitio de Ruby on Rails. [Online] 2010. http://rubyonrails.org/.

13. Sitio de Ruby Forge. [Online] 2011. http://www.rubyforge.org/.

14. Sitio de Redmine. [Online] 2006. http://www.redmine.org/.

15. Sitio de Net Beans. [Online] 2011. http://www.netbeans.org.

16. **Curt, Hibbs.** *Ruby on Rails: Up and Running.* 2006.

17. **Weiss, Jonathan.** *Ruby on Rails Security.* 2007.

18. **Velázquez, Esteban Manchado.** *Ruby on Rails avanzado.* 2005.

19. **Guillén, Diego F.** *Ruby Fácil.* 2007.

20. **Feldt, Robert, Johnson, Lyle and Neumann, Michael.** *Ruby Developer's Guide.* 2002.

21. **Williams, Justin.** *Rails Solutions Ruby on Rails Made Easy.* 2007.

22. **Dave, Thomas, Andy, Hunt and Chad, Fowler.** *Programming Ruby-The Pragmatic Programmer's Guide.* 2004.

23. **Alameda, Eldon.** *Practical Rails Projects.* 2007.

24. **Ola, Bini.** *Practical JRuby on Rails Web 2.0 Projects.* 2007.

25. **Letelier, Patricio and Penadés, Mª Carmen.** *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* 2003.

26. **PostgreSQL, Equipo de desarrolladores.** *Manual del usuario de PostgreSQL.* 2002.

27. **Sánchez, Jorge.** Jorge Sánchez. *Manual breve para el manejo de la base de datos de código.* [Online] 2004. http://www.jorgesanchez.net.

28. **Fernández Escribano, Gerardo.** *Introducción a Extreme Programming.* 2002.

29. **Pressman, Roger.** *Ingeniería de Software un enfoque práctico.* 1997.

30. **Santamaría, Ing. Mario.** Ingeniería de Software. [Online] 2011.
ftp://ftp.puce.edu.ec/.../Unidad%204%20Métodos%20convencionales%20IS.doc.

31. **Somerville, Ian.** *Ingeniería de Software.* 2002.

32. **IEEE.** *IEEE Guide to Software Requirements Specifications.* 1998.

33. Fondation Mozilla. *Sitio Mozilla.* [Online] 2010. http://www.mozilla.com/es-ES/.

34. **Beck, K.** *Extreme Programming Explained Embrace Change.* 1999.

35. Especificación de requisitos. [Online] 2011.
www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r55587.DOC.

36. **Chak, Dan.** *Enterprise Rails.* 2009.

37. **León, Luis Vinicio.** El contexto de la prueba de software. [Online] 2005.
http://www.softwareguru.com.mx.

38. **Velázquez, Esteban Manchado.** *Desarrollo web sobre ruedas Ruby on Rails (II).* 2010.

39. **Wirdemann, Ralf and Baustert, Thomas.** *Desarrollo REST con Rails.* 2007.

40. Aprenda en línea. *Arquitectura de software.* [Online] 2011.
http://www.aprendeenlinea.udea.edu.co/lms/moodle.

41. Api Rails. [Online] 2010. tp://api.rubyonrails.org/.

42. *Análisis, diseño e implementación de un sitio Web Departamental.* **García-Manso, Adolfo M. Catalán.** 2009.

43. **Dave, Thomas and David, Heinemeier.** *Agile Web Development with Rails.* 2005.

44. **Ediger, Brad.** *Advanced Rails.* 2010.

Referencias bibliográficas

1. **G.J. Myers**, *The Art of Software Testing*, 1979.
2. **Hetzel, Bill.** *The Complete Guide to Software Testing***.** 1988.
3. **María Eugenia Valencia.** *Prueba de Software.* [En línea] Mayo 2008 http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software/PRUEBASoftware.pdf
4. **Ing. Mario Santamaría.** *Métodos Convencionales de la Ingeniería de Software.* [En línea] 2010 ftp://ftp.puce.edu.ec/.../Unidad%204%20Métodos%20convencionales%20IS.doc
5. **Luis Vinicio León.** *El Contexto de la Prueba de* Software**.** [En línea] Mayo-Junio de 2005. http://www.softwareguru.com.mx
6. *STaaS - Software Testing as a Service*. [En línea] 2010. http://www.es.sogeti.com.
7. **Beck, K**. *Extreme Programming Explained Embrace Change, Pearson Education*. 1999
8. **Perissé, M C**. *Una Metodología Simplificada*. 2001.
9. **Larman, Craig**. *UML y Patrones*.
10. Especificación de requisitos. [En línea] 2011. www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r55587.DOC
11. Arquitectura de software. [En línea] 2011. http://www.aprendeenlinea.udea.edu.co/lms/moodle
12. **Somerville, Ian**. *Ingeniería de Software*. 2002

Anexos

Anexo 1: Estructura del patrón Modelo-Vista-Controlador

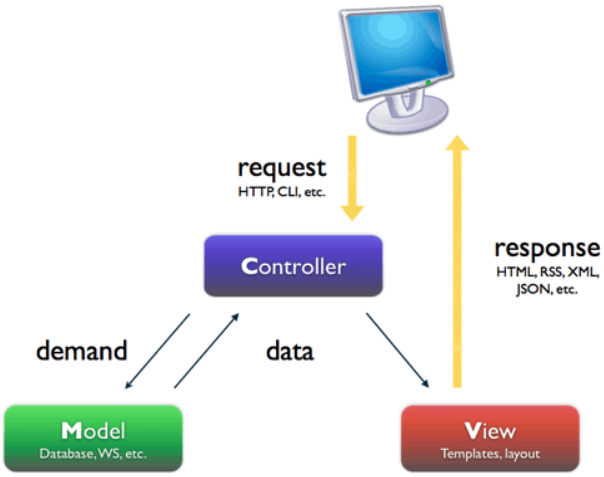


Figura 29. MVC

Anexo 2: Estructura del framework Ruby on Rails

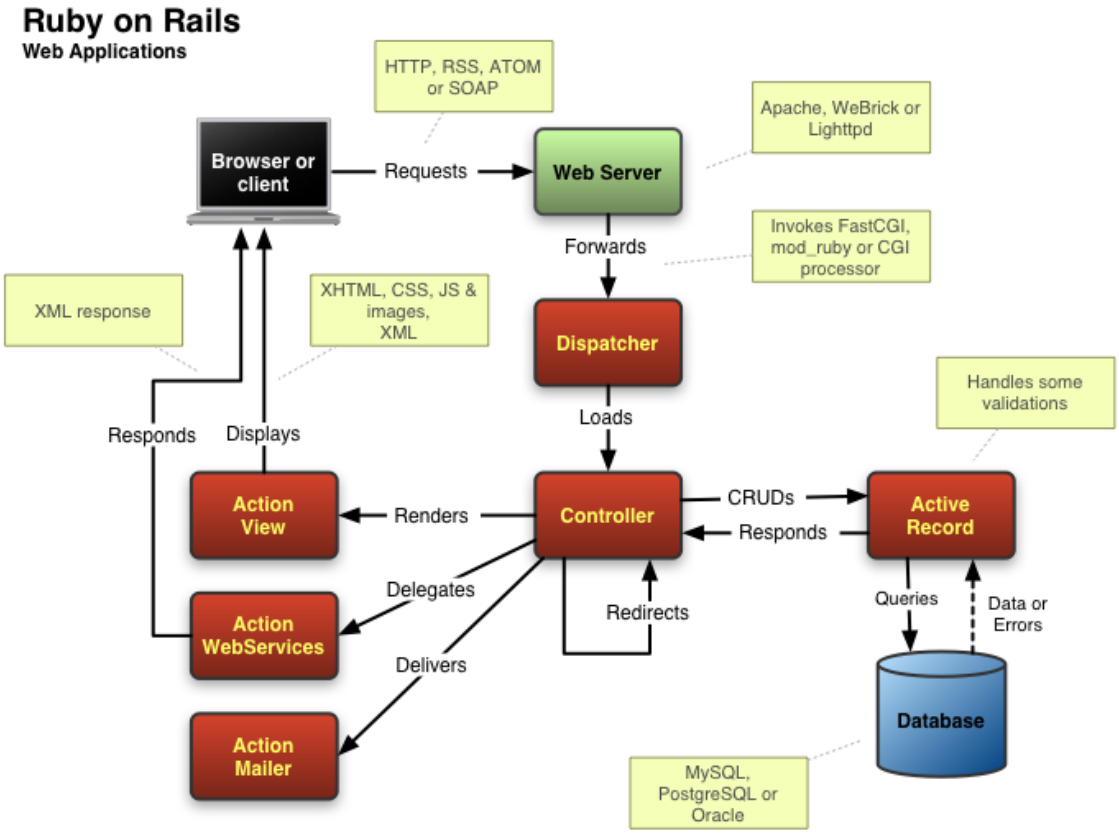


Figura 30. Estructura del framework RoR

Anexo 3: Estructura de carpetas del framework

Figura 31. Estructura de carpetas

Anexo 4: Estructura de carpetas dentro del REDMINE

Anexo 5: Diseño de Caso de Prueba

Condiciones de ejecución

- Se debe identificar y autenticar ante el sistema y además debe tener los permisos para ejecutar la acción.
- El proyecto al que pertenece debe tener el plugin activado.
- Se debe seleccionar la opción **Gtest** del menú del proyecto.

Requisitos a probar

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Elaborar reporte por criterio de búsqueda.	El sistema debe permitir filtrar las peticiones de liberación por los criterios de búsqueda mostrados.	EP 1.1: Buscar peticiones por criterios de búsqueda válidos.	– Se seleccionan y se introducen los criterios de búsqueda. – Se presiona el botón **Filtrar.**
		EP 1.2: Buscar peticiones por criterios de búsqueda inválidos.	– Se seleccionan y se introducen los criterios de búsqueda inválidos. – Se presiona el botón **Filtrar.**
		EP 1.3: Limpiar criterios de búsquedas seleccionados.	– Se seleccionan y se introducen los criterios de búsqueda. – Se presiona el botón **Limpiar.**

Descripción de variable

No	Nombre de campo	Tipo	Válido	Inválido	Inválido	Inválido	Inválido
1	Proyecto	Combo box.	Marcado.	No marcado.	N/A	N/A	N/A
2	Estado	Combo box.	N/A	N/A	N/A	N/A	N/A
3	Tipo de error	Combo box.	N/A	N/A	N/A	N/A	N/A
4	Iteración de liberación	Combo box.	N/A	N/A	N/A	N/A	N/A

5	Fecha de inicio	Date field	N/A	N/A	N/A	N/A	N/A
6	Fecha de fin	Date field	N/A	N/A	N/A	N/A	N/A

Juegos de datos a probar

Id del escenario	Escenario	Proyecto	Estado	Tipo de error	Iteración de liberación	Fecha de inicio	Fecha de fin	Respuesta del sistema
EP 1.1	Buscar peticiones por criterios de búsqueda válidos.	V(Capital Humano)	V(vacío)	V(vacío)	V(vacío)	V(vacío)	V(vacío)	El sistema muestra todas las peticiones de liberación existentes correspondientes con el criterio de búsqueda insertado.
EP 1.1	Buscar peticiones por criterios de búsqueda válidos.	V(Capital Humano)	V(Abierta)	V(De documento)	V(1ra iteración)	V(2011-06-08)	V(2011-06-22)	El sistema muestra todas las peticiones de liberación existentes correspondientes con los criterios de búsquedas insertados.
EP 1.2	Buscar peticiones por criterios de búsqueda inválidos.	I(vacío)	V(Abierta)	V(De documento)	V(1ra iteración)	V(2011-06-08)	V(2011-06-22)	El sistema muestra el siguiente mensaje de error: “Debe especificar un proyecto”.
EP 1.2	Buscar peticiones por criterios de búsqueda inválidos.	V(Capital Humano)	V(Abierta)	V(De documento)	V(1ra iteración)	V(2011-06-22)	V(2011-06-08)	El sistema muestra el siguiente mensaje de error: “Fecha fin debe ser posterior a la fecha de comienzo.”.

EP 1.3	Limpiar criterios de búsquedas seleccionados.	V(Capital Humano)	V(Abierta)	V(De documento)	V(1ra iteración)	V(2011-06-08)	V(2011-06-22)	El sistema limpia todos los campos que fueron seleccionados.

Anexo 6: Interfaz de usuario filtrar peticiones

Figura 32. Interfaz de usuario filtrar peticiones

Anexo 7: Interfaz de usuario filtrar reporte de peticiones

Figura 33. Interfaz de usuario reporte de peticiones