UNIVERSIDAD CENTRAL DE VENEZUELA

FACULTADA DE CIENCIAS

ESCUELA DE COMPUTACIÓN

DESARROLLO DE UNA APLICACIÓN WEB PARA LA GESTIÓN DE PROYECTOS DE DESARROLLO ÁGIL DE SOFTWARE

Trabajo Especial de Grado presentado ante la

Universidad Central de Venezuela

Por el Bachiller:

Daniel Ángel Guánchez Guánchez

C.I 16.509.636

Para optar por el título de

Licenciado en Computación

Tutores:

Profesor Andrés Sanoja

Profesor Eugenio Scalise

Caracas, Octubre 2013

Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

ACTA

Quienes suscriben, miembros del jurado designado por el Consejo de la Escuela de Computación, para examinar el Trabajo Especial de Grado titulado **"Desarrollo de una Aplicación Web para la Gestión de los Procesos Ágiles de Desarrollo de Software"** y presentado por el Br. Daniel Ángel Guánchez Guánchez (CI. 509.636), a los fines de optar al título de Licenciado en Computación, dejamos constancia de lo siguiente:

Leído como fue dicho trabajo, por cada uno de los miembros del jurado, se fijó el día 03 de Octubre del 2013, a las 11:00 a.m., para que su autor lo defendiera en forma pública, en la Sala 1, de la Escuela de Computación, Facultad de Ciencias de la Universidad Central de Venezuela, mediante una presentación oral de su contenido, luego de lo cual respondió a las preguntas formuladas. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobar con la nota de _____ puntos.

En fe de lo cual se levanta la presente Acta, en Caracas a los tres días del mes de octubre del año dos mil trece.

Profesor Eugenio Scalise
(Tutor)

Profesora Alecia E Acosta
(Jurado)

Profesor Antonio Leal
(Jurado)

Quiero dedicar este trabajo a mi familia

de la cual me siento muy orgulloso, Los Guánchez

Agradecimientos

A Dios, por todas las pruebas que me ha puesto a lo largo de mi vida y por la fuerza que me ha dado para superarlas. Por haberme permitido vivir cada momento, que bueno o malo me ha servido de aprendizaje y por demostrar cada día que está conmigo y no me abandona.

A la ilustre Universidad Central de Venezuela, cuna de grandes profesionales, por haberme permitido estar en su espacio, tus valores y lo que representan fueron estímulos para continuar cuando las adversidades y contratiempos se avecinaban, en mi espíritu vivirá por siempre el sentimiento Ucevista.

A mis padres Leida Guánchez y Rafael Guánchez y a mi padrino Royel Pérez, por haberme guiado por el mejor camino, por estar siempre presente en los buenos y malos momentos, gracias a ellos me encuentro ante ustedes, celebrando la finalización de una etapa que si bien ha sido larga y con muchos obstáculos, no ha dejado de ser gratificante y sin ustedes no hubiera podido llegar a ser la persona que soy hoy en día. Ustedes son mis pilares, sin sus enseñanzas, nada de esto podría ser posible.

A mi hermano Miguel Ángel Guánchez que ha estado presente siempre en todo momento, que me ha dado el afecto y apoyo para seguir siendo una persona mejor día a día.

A mis tutores Andrés Sanoja y Eugenio Scalise, por su excelente dedicación como guía, indicarme siempre el mejor camino a seguir y prestarme toda su colaboración para poder finalizar este proceso.

A mis abuelos, y mis tíos siempre atentos y dando su apoyo incondicional en las buenas y malas, por sus enseñanzas y ejemplo de superación que me ha dado cada uno de ustedes. A mis primos y primas que han sido como mis mejores amigos en todos los sentidos, siendo compañeros y hasta cómplices en todas mis vivencias, contarán conmigo para toda la vida.

A mis mejores amigas y compañeras durante toda esta aventura Ivana, Raizu y Yury, por todas esas palabras de aliento, consejos y confianza que siempre han tenido en mí, las quiero y les agradezco infinitamente su gran amistad.

A los profesores, Antonio Leal, Eleonora Acosta, Jossie Zambrano y Keyla Rivas, que me enseñaron a formarme como profesional y como persona, brindándome además de conocimientos, su amistad, consejos y apoyo, su carisma y respeto no podrán ser olvidados.

A mis amigos y amigas con quien compartí durante toda mi carrera académica, laboral y emocional, gracias por estar siempre presentes y contar con ustedes cada día.

Resumen

El objetivo principal de este trabajo consistió en desarrollar una aplicación web para la automatización de la gestión de los procesos de desarrollo de *software* basados en métodos ágiles, para ser utilizado como soporte para los Trabajos Especiales de Grado de la Escuela de Computación de la UCV. El método ágil utilizado para la construcción de esta aplicación fue una adaptación de la Programación Extrema (XP), la cual permite guiar la elaboración de la parte práctica del presente trabajo de investigación. Dentro de las necesidades contempladas en la aplicación, destaca el mantener informado a los actores involucrados del flujo de trabajo con la información pertinente, dependiendo del tipo de proceso que se lleve a cabo en determinado momento. La aplicación se implementó utilizando tecnologías actuales entre las cuales podemos mencionar: plataforma JEE, utilizando los *frameworks Hibernate y Struts*, el entorno de desarrollo *NetBeans*, el sistema manejador de base de datos MySql, entre los más importantes. Las pruebas permitieron evidenciar, no solo la correcta fluidez de la las actividades procesos de desarrollo, sino como la automatización del mismo mejora notable su ejecución.

Palabras Clave: Ingeniera del *Software*, Desarrollo de *Software*, Métodos Agiles, Programación Extrema, Gestión de Proyectos Ágiles, Documentación de *Software*.

Índice

Índice de Figuras

Índice de Tablas

Introducción

En la actualidad se requiere nuevas formas para el desarrollo eficaz y eficiente de proyectos y productos de *software* que reúnan las necesidades de los clientes. Para las organizaciones es cada vez más apremiante contar con las capacidades intelectuales y creativas de las personas involucradas y comprometidas en el logro de las metas planteadas y la serie de desafíos adicionales relativos a la naturaleza de los productos obtenidos durante el desarrollo de *software*. Un factor decisivo para asegurar el éxito durante el desarrollo de *software* no es suficiente contar con notaciones de modelado y herramientas, hace falta un elemento importante: el método de desarrollo, la cual nos provee de una dirección a seguir para la correcta aplicación de las actividades durante el proceso de desarrollo.

Generalmente el proceso de desarrollo llevaba asociado un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este enfoque "tradicional" para abordar el desarrollo de *software* ha demostrado ser efectivo y necesario en proyectos de gran tamaño, respecto a tiempo y recursos, donde los requerimientos están claramente definidos. Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. En este contexto de cambios se han engendrado los llamados métodos ágiles, cuyos recursos y herramientas avanzan como motores propulsores de desarrollo.

Los métodos ágiles emergen como una respuesta para llenar el vacío metodológico que dejan los métodos tradicionales en la incorporación de buenas prácticas de la Ingeniería de *Software* que por restricciones de tiempo y flexibilidad no han sido utilizadas. En general, los métodos ágiles se orientan para proyectos pequeños, constituyendo una solución a la medida del entorno, aportando una elevada simplificación sin renunciar a las prácticas esenciales para asegurar la calidad del producto.

Entre los denominados métodos ágiles de desarrollo de *software* se encuentra el llamado método Programación Extrema (XP) como uno los más exitosos en los tiempos recientes, diseñado para satisfacer las características del *software* que los clientes necesitan en el momento en que lo necesitan, alentando a los desarrolladores a responder a los requerimientos cambiantes de los clientes, aún en fases tardías del ciclo de vida del desarrollo.

Existen diversas alternativas a la hora de seleccionar el método ágil para el desarrollo de *software* y la escogencia de una u otra depende tanto de las características del proyecto como de la experiencia o preferencia del equipo de desarrollo. Los profesores e investigadores del Centro de Computación Paralela y Distribuida (C.C.P.D.) y del Centro de Ingeniería de *Software* y Sistemas (ISYS) de la Escuela de Computación de la Facultad de Ciencias de Universidad Central de Venezuela (UCV), tienen la tendencia a utilizar una adaptación del método ágil XP en los Trabajos Especiales de Grado y trabajos de investigación que involucren desarrollo de *software*, haciendo uso de ciertos artefactos que sirven como apoyo para llevar a cabo las diferentes etapas del proceso, registrando las necesidades del cliente (requerimientos del sistema) y facilitando la comunicación de las ideas entre los miembros del equipo. Entre estos se distinguen:

- Historias de Usuario: Constituyen el medio utilizado para especificar los requisitos del sistema a desarrollar.
- Tareas de Ingeniería: Surgen al tomar una historia y dividirla en funcionalidades más pequeñas que necesitan ser desarrolladas para llevar a cabo la historia.
- Pruebas de Aceptación: Son creadas en base a las historias de usuarios, en cada ciclo de la iteración del desarrollo. El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento.
- Plan de Entrega: Cronograma de entregas donde se establece la prioridad de las historias de usuario así como una estimación del esfuerzo necesario para cada una de ellas.

- Plan de Interacción: Las historias de usuarios seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo con el orden preestablecido.

Actualmente no existe ningún procedimiento estándar para la gestión del método durante el desarrollo de *software*, cada investigador utiliza la adaptación del método que más le convenga para aplicar en su proyecto. Por otra parte, no siempre logra el aprovechamiento máximo de los recursos, las actividades se llevan a cabo manualmente o semiautomática en pequeñas aplicaciones como hojas de cálculo, procesadores de palabras, uso de dispositivos externos y correo electrónico para el almacenamiento y el flujo de la información, lo que genera como desventajas, inconsistencia de datos, falta de automatización de las actividades y artefactos del proceso de desarrollo, aumentando la cantidad de tiempo en la que se llevan a cabo, dificultando el acceso a la información y la retroalimentación de los miembros del equipo de desarrollado, obteniendo resultados no lo bastante adecuados a las necesidades originales.

Otro de los problemas que se presenta es La falta de documentación, lo cual repercute de manera negativa en la productividad de los TEG y proyectos de investigación que involucren desarrollo de *software*, dificultando en gran medida su capacidad para adaptarse y anticiparse a los cambios. La documentación requiere ser registrada, ya que su propósito es enseñar a quienes no están familiarizados con un sistema, mostrar como este se estructura, funciona y los motivos que llevaron a decidirse por ese diseño.

Para la solución de la problemática planteada surge la propuesta de realizar una aplicación Web que, utilizando las tecnologías actuales, permita solventar los problemas existentes, en cuanto al manejo de los artefactos y documentación durante la gestión del proceso de desarrollo de *software*, apoyando al método utilizado en los trabajos de investigación permitiendo una planificación eficaz, proporcionando visibilidad en tiempo real, incorporando la retroalimentación temprana y facilitando una colaboración en el equipo de desarrollo.

El objetivo general de este trabajo consiste desarrollar una aplicación web que automatice la gestión de los procesos de desarrollo de *software* basados en métodos ágiles.

Los objetivos específicos necesarios para cumplir con el objetivo general antes planteado son los siguientes:

- Estudiar XP como método ágil para el desarrollo de *software*.
- Proponer la adaptación de XP a utilizar para los TEG y trabajos de investigación que involucren desarrollo de *software*.
- Seleccionar las tecnologías de desarrollo web a utilizar para la implementación de la aplicación.
- Implementar la herramienta para la gestión de los procesos de desarrollo de *software*, usando la configuración de XP propuesta.
- Realizar las pruebas para garantizar el correcto funcionamiento de la aplicación.

A continuación se describen los capítulos que conforman este documento:

- Capítulo 1: Se describe el contexto en el que surgen los métodos ágiles, sus valores, principios y comparaciones con los métodos tradicionales. Adicionalmente se describe con mayor detalle Programación Extrema (*eXtreme Programming*, XP) presentando sus características principales, el proceso que se sigue y las reglas y prácticas que propone.
- Capítulo 2: Se explican y documentan todos los pasos realizados para lograr el desarrollo del sistema siguiendo el ciclo de los procesos agiles adaptado a la programación extrema. También se presenta el análisis inicial del sistema de forma global, mostrando las historias de usuario, la metáfora del sistema y las especificaciones técnicas.
- Capítulo 3: Se precisan y detallan todos los pasos efectuados por medio de iteraciones para lograr el desarrollo de la aplicación.
- Conclusiones donde se dará muestra de los resultados, recomendaciones y aportes de la aplicación desarrollada para futuros trabajos relacionados.
- Finalmente se presentan las Referencias Bibliográficas consultadas durante el desarrollo del documento y aplicación.

Capítulo 1 : Desarrollo Ágil de Software y la Programación Extrema

El presente capítulo tiene la finalidad de exponer los fundamentos conceptuales utilizados durante el proceso de investigación y desarrollo. La primera parte introduce las principales características de los métodos ágiles recogidas en el manifiesto ágil. La segunda parte se centra en la Programación Extrema, presentando sus características principales, así como el proceso que se sigue, las reglas y prácticas que propone.

1.1 Desarrollo Ágil de *Software*

Se entiende como desarrollo ágil un paradigma de desarrollo de software basado en métodos ágiles, conocidos también como métodos livianos, que intentan evitar los tortuosos y burocráticos caminos de los métodos tradicionales enfocándose en la gente y los resultados, mediante la colaboración de grupos auto organizados y multidisciplinarios.

A principios de la década del 90, surge como un enfoque que fue bastante revolucionario para su momento ya que iba en contra de la creencia que mediante procesos altamente definidos se podía lograr obtener *software* en tiempo, costo y con la requerida calidad. El enfoque fue planteado por primera vez por James Martin (Martin, 1991) y se dio a conocer en la comunidad de Ingeniería de Software con el mismo nombre que su libro, *Rapid Application Development* (RAD). RAD consistía en un entorno de desarrollo altamente productivo, en el que participaban grupos pequeños de programadores utilizando herramientas que generaban código en forma automática tomando como entradas sintaxis de alto nivel. En general, se considera que este fue uno de los primeros hitos en pro de la agilidad en los procesos de desarrollo.

Durante 1996, Beck es llamado por Chrysler como asesor del proyecto *Chrysler Comprehensive Compensation (C3) payroll system*. Dada la poca calidad del sistema que se estaba desarrollando, Beck decide tirar todo el código y empezar de cero utilizando las prácticas que él había ido definiendo a lo largo del tiempo. Como consecuencia del éxito del proyecto C3, Kent Beck dio origen a eXtreme Programming (XP) iniciando el movimiento de metodologías ágiles al que se anexarían otras metodologías surgidas mucho antes que el propio Beck fuera convocado por Chrysler.

En febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace el término "ágil" aplicado al desarrollo de *software*. En esta reunión participan un grupo de diecisiete expertos de la industria del *software*, incluyendo algunos de los creadores o impulsores de metodologías de *software*. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar *software* rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de desarrollo de *software* tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

Tras esta reunión se creó *The Agile Alliance* (Alliance, 2001), una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de *software* y ayudar a las organizaciones para que adopten estos conceptos. El punto de partida es fue el Manifiesto Ágil (Beck, y otros, 2001), un documento que resume la filosofía "ágil".

1.1.1 El Manifiesto Ágil

El Manifiesto para el Desarrollo Ágil de *Software* es de suma importancia dentro del movimiento de los métodos ágiles. Uno de los principales objetivos del encuentro en que se generó el Manifiesto fue el de extraer un factor común de los principios esenciales que servirían de guía para cualquier metodología que se identifique como ágil. Esto concluyó en la declaración de lo que podríamos denominar el prólogo del Manifiesto: "Estamos descubriendo formas mejores de desarrollar *software* tanto por nuestra propia experiencia como ayudando a terceros" (Beck, y otros, 2001). A través de esta experiencia se valora:

- Individuos e interacciones sobre procesos y herramientas. La gente es el principal factor de éxito de un proyecto *software*. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo con base a sus necesidades.
- *Software* que funciona sobre documentación exhaustiva. La regla a seguir es no producir documentos a menos que sean necesarios de forma inmediata para tomar

una decisión importante. Estos documentos deben ser cortos y centrarse en lo fundamental.

- Colaboración con el cliente sobre negociación de contratos. Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.
- Responder ante el cambio sobre seguimiento de un plan. La habilidad de responder a los cambios que puedan surgir a los largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

Los valores anteriores inspiran los doce principios del manifiesto. Son características que diferencian un proceso ágil de uno tradicional. Los dos primeros principios son generales y resumen gran parte del espíritu ágil. El resto tienen que ver con el proceso a seguir y con el equipo de desarrollo, en cuanto metas a seguir y organización del mismo, estos son:

- La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de *software* que le aporte un valor.
- Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
- Entregar frecuentemente *software* que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
- La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
- Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
- El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
- El *software* que funciona es la medida principal de progreso.
- Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.

- La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
- La simplicidad es esencial.
- Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
- En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

1.1.2 Características del Desarrollo Ágil

Según (Miller, 2001) las características del proceso ágil de desarrollo de *software*, tomado desde el punto de vista del tiempo de entrega, son:

- Modularidad en el proceso de desarrollo.
- Iterativo, con cortos ciclos, permitiendo rápidas verificaciones y correcciones.
- Ciclos en un período de tiempo de una (1) a seis (6) semanas.
- Sencillez en el proceso de desarrollo, evitando todas las actividades innecesarias.
- Adaptativo, con la posibilidad de aparición de nuevos riesgos.
- Proceso incremental que permite segmentar la construcción de las aplicaciones.
- Orientado a las personas: se prefiere el proceso ágil enfocado a las personas que a los procesos y la tecnología.
- Un ambiente de trabajo comunicativo y de colaboración.

1.1.3 Métodos Ágiles

Los métodos agiles forman parte del movimiento de desarrollo ágil de *software*, que se basan en la adaptabilidad de cualquier cambio como medio para aumentar las posibilidades de éxito de un proyecto. Un método es ágil cuando el desarrollo de *software* es incremental (entregas pequeñas de *software*, con ciclos rápidos), cooperativo (cliente y desarrolladores trabajan juntos constantemente con una cercana comunicación), sencillo (el método en sí mismo es fácil de aprender y modificar, bien documentado) y adaptable (permite realizar cambios de último momento) (Abrahamsson, Salo, Ronkainen, & Warsta, 2002).

Las necesidades de un cliente pueden variar desde el momento de contratación de un proyecto hasta el momento de su entrega. Esto requiere métodos de desarrollo de *software*

diferentes, que en lugar de rechazar los cambios, sean capaces de adaptarse e incorporarlos. Estos métodos también deben:

- Producir versiones ejecutables en un lapso corto de tiempo con la finalidad de obtener por parte del cliente retroalimentación en cuanto al producto.
- Realizar soluciones sencillas con la finalidad de que el impacto de los cambios que puedan surgir se minimice.
- Mejorar la calidad del diseño para así lograr que las iteraciones posteriores requieran menos esfuerzos.
- Ejecutar pruebas continuas y desde temprano, para encontrar y solventar defectos antes que tengan un gran impacto.

1.1.4 Métodos Ágiles en comparación con Métodos Tradicionales

Los métodos no ágiles son aquellos que están guiados por una fuerte planificación durante todo el proceso de desarrollo, llamadas también métodos tradicionales o clásicos, donde se realiza una intensa etapa de análisis y diseño antes de la construcción del sistema. Los métodos tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo de *software*, con el simple objetivo de asegurar que el *software* que se obtenga satisfaga los requerimientos del usuario y reúna estándares aceptables de calidad. El trabajo de planificación es riguroso, aun cuando en la práctica muchas veces estas planificaciones no se respetan.

En contraposición a este tipo de métodos tradicionales, los métodos ágiles aportan nuevas formas de trabajo que apuestan por una cantidad apropiada de procesos. Es decir, no se desgastan con una excesiva cantidad de cuestiones administrativas, ni tampoco defienden la postura extremista de total falta de proceso. Ya que se tiene conciencia de que se producirán cambios, lo que se pretende es reducir el costo de rehacer el trabajo.

La Tabla 1.1 (Abián, 2003) recoge esquemáticamente las principales diferencias de los métodos ágiles con respecto a los tradicionales (“no ágiles”). Estas diferencias que afectan no sólo al proceso en sí, sino también al contexto del equipo así como a su organización.

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

Tabla 1.1 Diferencias entre Métodos Ágiles y Métodos Tradicionales.

Los métodos ágiles y los tradicionales no son estrictamente competidores directos, cada uno tiene su propio segmento de aplicación o terreno, usados en proyectos con diferentes características. Los métodos tradicionales son más adecuados en grandes proyectos con requerimientos estables, aplicaciones críticas, grandes equipos de desarrollo y/o equipos distribuidos geográficamente. Los métodos ágiles en cambio se adecuan mejor en ambientes dinámicos, con equipos de trabajo pequeños y produciendo aplicaciones no críticas. También son una buena elección cuando se trabaja con requerimientos desconocidos o inestables, garantizando un menor riesgo ante la posibilidad de cambio en los requerimientos.

1.1.5 Métodos Ágiles existentes

En los últimos años han existido métodos que fomentan las prácticas antes mencionadas, estos son conocidos como métodos agiles. A continuación se resumen los métodos más conocidos, dejando el análisis más detallado de XP para la siguiente sección:

- **Scrum:** Desarrollada por Ken Schwaber y Mike Beedle (Schwaber & Beedle, 2001). Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de *software* se realiza mediante iteraciones, denominadas *sprints*, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.
- **Crystal *Methodologies*:** La familia Crystal fue presentada por Alistair Cockburn (Cockburn, 2001). Se trata de un conjunto de métodos para el desarrollo de *software*, caracterizada por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. El desarrollo de *software* se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas.
- ***Feature-driven development (FDD):*** Sus impulsores son Peter Coad y Jeff De Luca (Coad, De Lucas, & Lefebvre, 1999). Define un proceso iterativo que consta de cinco pasos, desarrollar el modelo general, construir la lista de características, planear por características, diseñar por características y construir por características. Las iteraciones son cortas (hasta 2 semanas). Se centra en las fases de diseño e implementación del sistema partiendo de la lista de características que debe reunir el *software*.
- ***Adaptive Software Development (ASD):*** Desarrollado por James Highsmith (James, 2000). Se caracteriza por ser iterativo, orientado a los componentes de *software*

más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: especulación, colaboración y aprendizaje. En la primera de ellas se inicia el proyecto y se planifican las características del *software*; en la segunda desarrollan las características y finalmente en la tercera se revisa su calidad, y se entrega al cliente. La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

- ***Dynamic Systems Development Method (DSDM):*** Nace con el objetivo crear una método RAD unificada (Stapleton, 1997). La idea fundamental detrás de DSDM es que en vez de fijar la cantidad de funcionalidades en un producto, y luego ajustar el tiempo y los recursos para llegar a esa funcionalidad, es preferible fijar tiempo y recursos y, a continuación, ajustar la cantidad de funcionalidad en consecuencia. Propone cinco fases: estudio viabilidad, estudio del negocio, modelado funcional, diseño y construcción, y finalmente implementación.

1.2 La Programación Extrema (XP)

La programación extrema comúnmente llamado XP por sus siglas en ingles *eXtreme Programming*, es un método de desarrollo de *software* ágil basado en una serie buenas prácticas que persigue el objetivo de aumentar la productividad a la hora de desarrollar programas.

Kent Beck (Beck, 1999) define XP como “Un proceso ligero, de bajo riesgo, flexible, predecible, científico y divertido de desarrollar *software*”. XP fue concebido y desarrollado para satisfacer las necesidades específicas del desarrollo de *software*, conducido por pequeños grupos para enfrentar requerimientos que no están claramente definidos o que cambian constantemente. Este método desafía muchas concepciones tradicionales, incluyendo la creencia de que cambiar una pieza de *software* necesariamente implica un aumento dramático en el costo a través del tiempo. XP reconoce que en los proyectos se debe trabajar para alcanzar la reducción en el costo y explotar las ganancias una vez que se hayan obtenido (Beck, 1999).

XP fue introducido como método ágil de desarrollo de *software* sobre finales de los 1990s. Uno de los conocidos "caso de éxito" fue publicado a fines de 1998, cuando Kent Beck introdujo el nuevo método en el proyecto de desarrollo denominado *Chrysler Comprehensive Compensation (C3)* para la firma Chrysler.

XP está basado en ciertos valores, principios y buenas prácticas de desarrollo. Estas prácticas existen desde hace mucho tiempo, incluso algunas de éstas, desde principios de la ingeniería del *software*. La novedad del método, radica en la forma en que son utilizadas, combinadas y llevadas al extremo (de ahí el nombre de programación extrema) para lograr un *software* de calidad en el menor tiempo posible y minimizando los riesgos que se puedan presentar durante el desarrollo (como cambios en los requerimientos originales, retardos en el tiempo de entrega, altos costos de mantenimiento, tasa de defectos elevada, entre otros).

1.2.1 Los Valores

Los valores representan los aspectos considerados como fundamentales para garantizar el éxito de un proyecto de desarrollo de *software*, ya que darán consistencia y solidez al equipo de trabajo. Constituyen los elementos base sobre los cuales se fundamentan los principios y técnicas de XP. Son la esencia de la metodología. Desde sus inicios, la programación extrema se basó en cinco valores: comunicación, simplicidad, retroalimentación coraje y respeto. A continuación se describen cada uno de estos:

- **Comunicación:** Muchos de los problemas que existen en proyectos de *software* (así como en muchos otros ámbitos) se deben a problemas de comunicación entre las personas. La comunicación permanente es fundamental en XP. Dado que la documentación es escasa, el diálogo frontal, cara a cara, entre desarrolladores, gerentes y el cliente es el medio básico de comunicación. Una buena comunicación tiene que estar presente durante todo el proyecto.
- **Simplicidad:** XP, como método ágil, apuesta a la sencillez, en su máxima expresión. Sencillez en el diseño, en el código, en los procesos, etc. La sencillez es esencial para que todos puedan entender el código, y se trata de mejorar mediante recodificaciones continuas.

- **Retroalimentación:** Debe funcionar en forma permanente, el cliente debe brindar retroalimentación de las funciones desarrolladas, de manera de poder tomar sus comentarios para la próxima iteración y para comprender cada vez más sus necesidades. Los resultados de las pruebas unitarias son también una retroalimentación permanente que tienen los desarrolladores acerca de la calidad de su trabajo.
- **Coraje:** Cuando se encuentran problemas serios en el diseño, o en cualquier otro aspecto, se debe tener el coraje suficiente como para encarar su solución, sin importar que tan difícil sea. Si es necesario cambiar completamente parte del código, hay que hacerlo, sin importar cuanto tiempo se ha invertido previamente en el mismo.
- **Respeto:** Este último valor tiene que ver con el trato que debe existir entre los miembros del equipo. Cada uno debe recibir el respeto que merece como miembro valioso que es. Los desarrolladores deben respetar la experiencia de los clientes y viceversa, y los jefes de proyecto deben respetar el derecho de los programadores a aceptar responsabilidades.

1.2.2 Los Principios Básicos

Los valores fundamentan a XP y constituyen un criterio para una solución exitosa, no brindan la precisión o especificidad necesaria para decidir qué prácticas se deben emplear en el desarrollo del *software*. Se necesita refinar esos valores y convertirlos en principios concretos con los que se pueda trabajar (Beck, 1999). XP posee cinco principios fundamentales y cada uno abarca los valores tratados en la sección anterior, apoyándose en estos y actuando como puente entre los valores y las prácticas. A continuación se describe cada uno de los principios:

- **Retroalimentación rápida:** Es necesario que exista comunicación rápida en todas las etapas de desarrollo y entre todos los miembros del equipo. El tiempo entre una acción y su retroalimentación es fundamental para el aprendizaje. De esta forma, el negocio aprende cuál es la mejor forma en que el sistema puede contribuir, y los programadores aprenden la mejor forma de diseñar, implementar y probar el sistema.

- **Asunción de simplicidad:** Se debe tratar cada problema como si se pudiera resolver de una forma realmente sencilla. Solo se debe diseñar pensando en resolver la situación actual, sin tomar en cuenta las tareas futuras.
- **Modificaciones incrementales:** Se debe pensar en resolver los problemas realizando pequeños cambios que hagan la diferencia. Este principio se debe aplicar tanto en la planificación como en el diseño y desarrollo.
- **Adopción del cambio:** Consiste en adoptar una estrategia que preserve la mayor cantidad de opciones mientras resuelve los problemas más urgentes.
- **Trabajo de calidad:** La calidad del trabajo debe estar siempre presente. Este principio no se puede sacrificar o comprometer, debe prevalecer durante todo el proceso de desarrollo.

1.2.3 Actividades

Los valores y principios definidos anteriormente, constituyen las bases o fundamentos de XP, pero por sí solos no definen lo que se debe hacer para llevarla a cabo. Es necesario determinar las tareas a realizar para el desarrollo de *software*. El método XP comprende las siguientes actividades:

- **Codificar:** Los programas de *software* son creados a través de la codificación, por lo tanto, es la actividad básica o esencial que se debe realizar y no se puede obviar. Con una idea de lo que será el diseño del sistema, los desarrolladores de XP escriben código para expresar sus ideas. De esta forma, el código se convierte en una forma de comunicación y aprendizaje, ya que leyendo el código fuente, otros programadores pueden entender la lógica, los algoritmos y el flujo de las acciones. El código también se puede utilizar para las pruebas y proveer cierta especificación operacional del sistema.
- **Hacer pruebas:** Las pruebas constituyen un elemento importante en XP y deben estar presentes durante todo el proceso de desarrollo. Se diseñan incluso antes de la codificación y se prueba y depura el código constantemente para cumplir con los valores y principios de simplicidad y calidad. Programar y realizar pruebas al mismo tiempo es mucho más rápido que simplemente programar, ya que reduce el tiempo

requerido para la depuración. Se deben aplicar dos tipos de pruebas, una prueba realizada por el programador para asegurarse que el sistema hace lo que se requiere y una prueba funcional, realizada o especificada por el cliente para demostrarle que el sistema funciona como un todo.

- **Escuchar:** La realización de esta actividad tiene como objetivo garantizar una buena comunicación para que las ideas que se transmiten entre programadores y clientes sean perfectamente entendidas y atendidas.
- **Diseñar:** El diseño crea una estructura que organiza la lógica del sistema, un buen diseño permite que el sistema crezca con cambios en un solo lugar. Los diseños deben de ser sencillos, si alguna parte del sistema es de desarrollo complejo, lo apropiado es dividirla en varias. Se debe proveer un contexto en donde se creen buenos diseños, se corrijan los malos diseños, y se aprenda todo el que haga falta sobre el diseño actual.

1.2.4 El Proceso en XP

Un proyecto XP tiene éxito cuando el cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo para medir la funcionalidad que puede entregar a través del tiempo. El ciclo de vida ideal de XP consiste de seis fases (Beck, 1999): Exploración, Planificación de la Entrega, Iteraciones, Producción, Mantenimiento y Muerte del Proyecto (Figura 1.1).

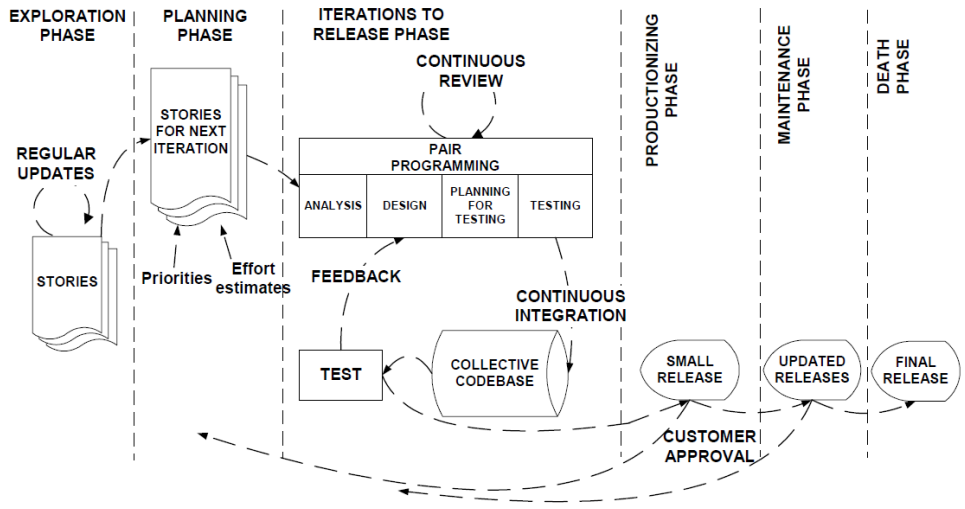


Figura 1.1 Ciclo de Vida del Proceso XP

1.2.4.1 Fase de Exploración

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

1.2.4.2 Fase de Planificación

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de éstas. Se llegan a los acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la "velocidad" de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración.

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

1.2.4.3 Fase de Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción.

Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de estas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores.

1.2.4.4 Fase de Producción

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase. Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación.

1.2.4.5 Fase de Mantenimiento

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

1.2.4.6 Fase de Muerte del Proyecto

Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

1.2.5 Los Roles

Para implementar el proceso de desarrollo de XP, las distintas tareas deben ser cubiertas por diferentes tipos de personas, es por esta razón que a continuación se presentan los roles de quienes formarán parte del equipo de trabajo. Para cada una de estos se detalla la responsabilidad, por lo tanto es necesario según ésta, colocar en el cargo a la persona más idónea. De acuerdo con la propuesta que realiza Beck en (Beck, 1999) se dividen en:

- **Cliente:** Escribe y determina las historias de usuario para cada iteración y define las prioridades de implementación según en el valor de negocio que aporta cada historia. También es responsable de diseñar y ejecutar las pruebas de aceptación.
- **Programador:** Es responsable de implementar las historias solicitadas por el cliente. Además, estima el tiempo de desarrollo de cada historia para que el cliente pueda asignarle prioridad dentro de alguna iteración. Cada iteración incorpora nueva funcionalidad de acuerdo con las prioridades establecidas por el cliente. El Programador también es responsable de diseñar y ejecutar las pruebas unitarias del código que ha implementado o modificado.
- ***Tracker:*** Una de las tareas más importantes del *tracker* consiste en seguir la evolución de las estimaciones realizadas por los programadores y compararlas con el tiempo real de desarrollo. De esta forma, puede brindar información estadística en lo que refiere a la calidad de las estimaciones para que puedan ser mejoradas.
- ***Coach:*** Es responsable del proceso en general. Se encarga de iniciar y de guiar a las personas del equipo en poner en marcha las prácticas. Es usualmente una persona con mucha experiencia en el desarrollo utilizando XP.

- ***Manager:*** Se encarga de organizar las reuniones, se asegura que el proceso de desarrollo se esté cumpliendo y registra los resultados de las reuniones para ser analizados en el futuro. Es de alguna forma el que responde al inversionista en lo que respecta a la evolución del desarrollo.

1.2.6 Reglas y Prácticas de XP

XP tiene un conjunto importante de reglas y prácticas. En forma genérica, se pueden agrupar en (Wells, 2009):

- Reglas y prácticas para la Planificación
- Reglas y prácticas para el Diseño
- Reglas y prácticas para el Desarrollo
- Reglas y prácticas para las Pruebas

1.2.6.1 Planificación

XP plantea la planificación como un diálogo continuo entre las partes involucradas en el proyecto, incluyendo al cliente, a los programadores y a los coordinadores o gerentes. El proyecto comienza recopilando historias de usuarios, las que sustituyen a los tradicionales casos de uso. Una vez obtenidas las "historias de usuarios", los programadores evalúan rápidamente el tiempo de desarrollo de cada una.

Si alguna de ellas tiene riesgos que no permiten establecer con certeza la complejidad del desarrollo, se realizan pequeños programas de prueba (*spikes*), para reducir estos riesgos. Una vez realizadas estas estimaciones, se organiza una reunión de planificación, con los diversos actores del proyecto, a los efectos de establecer un plan o cronograma de entregas (Release Plan) en los que todos estén de acuerdo. Una vez acordado este cronograma, comienza una fase de iteraciones, en donde en cada una de éstas se desarrolla, prueba e instala unas pocas historias de usuarios. Los conceptos básicos de esta planificación son los siguientes:

- **Historias de usuario:** sustituyen a los documentos de especificación funcional, y a los casos de uso. Estas historias son escritas por el cliente, en su propio lenguaje, como descripciones cortas de lo que el sistema debe realizar. La diferencia más

importante entre estas historias y los tradicionales documentos de especificación funcional se encuentra en el nivel de detalle requerido. Las historias de usuario deben tener el detalle mínimo como para que los programadores puedan realizar una estimación poco riesgosa del tiempo que llevará su desarrollo. Cuando llegue el momento de la implementación, los desarrolladores dialogarán directamente con el cliente para obtener todos los detalles necesarios. Las historias de usuarios deben poder ser programadas en un tiempo entre una y tres semanas.

- **Plan de entregas:** El cronograma de entregas establece qué historias de usuario serán agrupadas para conformar una entrega, y el orden de las mismas. Este cronograma será el resultado de una reunión entre todos los actores del proyecto. XP denomina a esta reunión Juego de planificación. El cliente ordenará y agrupará según sus prioridades las historias de usuario. El cronograma de entregas se realiza en base a las estimaciones de tiempos de desarrollo realizadas por los desarrolladores. Luego de algunas iteraciones es recomendable realizar nuevamente una reunión con los actores del proyecto, para evaluar nuevamente el plan de entregas y ajustarlo si es necesario.
- **Plan de iteraciones:** Las historias de usuarios seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo con el orden preestablecido. Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración. Cada historia de usuario se traduce en tareas específicas de programación. Asimismo, para cada historia de usuario se establecen las pruebas de aceptación. Estas pruebas se realizan al final del ciclo en el que se desarrollan, pero también al final de cada uno de los ciclos siguientes, para verificar que subsiguientes iteraciones no han afectado a las anteriores. Las pruebas de aceptación que hayan fallado en el ciclo anterior son analizadas para evaluar su corrección, así como para prever que no vuelvan a ocurrir.
- **Reuniones de seguimiento:** El objetivo de tener reuniones diarias es mantener la comunicación entre el equipo, y compartir problemas y soluciones. En la mayoría de estas reuniones, gran parte de los participantes simplemente escuchan, sin tener mucho que aportar.

- **Velocidad del proyecto:** La velocidad del proyecto es una medida que representa la rapidez con la que se desarrolla el proyecto. Se puede estimar tomando en cuenta cuántas historias pueden implementarse antes de una cierta fecha (planificación por alcance) o cuánto tiempo se requiere para desarrollar un conjunto de historias de usuario (planificación por tiempo). De esta forma, se sabrá la cantidad de historias que se pueden desarrollar en las distintas iteraciones, y así controlar que todas las tareas se puedan desarrollar en el tiempo que dure la iteración. En principio el tiempo se basa en una estimación, y al empezar a desarrollar las primeras historias de usuario se pueden cambiar los valores.

1.2.6.2 Diseño

El diseño crea una estructura que organiza la lógica del sistema, su correcta definición permite que los requerimientos crezcan con cambios controlados. Por lo tanto se plantea que el diseño debe ser revisado y mejorado de forma continua según se van añadiendo funcionalidades al sistema, ya que a priori no se tiene toda la información suficiente para diseñarlo en su totalidad.

Los diseños deben de ser sencillos, si alguna parte del sistema es de desarrollo complejo, hay que dividirla en varias. Si hay errores en el diseño o malos diseños, estos deben de ser corregidos cuanto antes. Para los diseños, sólo es necesario fichas, tarjetas o pizarras donde plantear las representaciones requeridas, ya que al implementar tareas pequeñas, no será necesario realizar inicialmente modelos complejos. Se propone invertir el tiempo en implementación y trabajar sobre el diseño antes de pasar a la próxima tarea. Los conceptos más importantes de diseño en este método son los siguientes:

- **Simplicidad:** Un diseño simple se implementa más rápidamente que uno complejo. Por ello XP propone implementar el diseño más simple posible que funcione. Se sugiere nunca adelantar la implementación de funcionalidades que no correspondan a la iteración en la que se esté trabajando.
- **Soluciones spike:** Cuando aparecen problemas técnicos, o cuando es difícil de estimar el tiempo para implementar una historia de usuario, pueden utilizarse pequeños programas de prueba (llamados spike), para explorar diferentes

soluciones. Estos programas son únicamente para probar o evaluar una solución, y suelen ser desechados luego de su evaluación.

- **Recodificación:** La recodificación consiste en escribir nuevamente parte del código de un programa, sin cambiar su funcionalidad, a los efectos de hacerlo más simple, conciso y/o entendible. Muchas veces, al terminar de escribir un código de programa, pensamos que, si lo comenzáramos de nuevo, lo hubiéramos hecho en forma diferente, más clara y eficientemente. Sin embargo, como ya está pronto y funciona, rara vez es reescrito. XP sugiere recodificar cada vez que sea necesario. Si bien, puede parecer una pérdida de tiempo innecesaria en el plazo inmediato, los resultados de ésta práctica tienen sus frutos en las siguientes iteraciones, cuando sea necesario ampliar o cambiar la funcionalidad. La filosofía que se persigue es, como ya se mencionó, tratar de mantener el código más simple posible que implemente la funcionalidad deseada.
- **Metáforas:** Una metáfora es algo que todos entienden, sin necesidad de mayores explicaciones. XP sugiere utilizar este concepto como una manera sencilla de explicar el propósito del proyecto, y guiar la estructura y arquitectura del mismo. Por ejemplo, puede ser una guía para la nomenclatura de los métodos y las clases utilizadas en el diseño del código. Tener nombres claros, que no requieran de mayores explicaciones, redunda en un ahorro de tiempo. Es muy importante que el cliente y el grupo de desarrolladores estén de acuerdo y compartan esta metáfora, para que puedan dialogar en un "mismo idioma". Una buena metáfora debe ser fácil de comprender para el cliente y a su vez debe tener suficiente contenido como para que sirva de guía a la arquitectura del proyecto.

1.2.6.3 Desarrollo del código

El desarrollo es la pieza clave de todo el proceso de programación extrema. En todas las tareas se promueve el desarrollo a máxima velocidad, sin interrupciones y siempre en la dirección correcta; en XP se programará solo la funcionalidad que es requerida para la entrega en cuestión. La presencia de la historia de usuario es necesaria a la hora de codificar. Antes del desarrollo de cada historia de usuario el cliente debe especificar detalladamente lo que se hará y también tendrá que estar presente cuando se realicen las

pruebas que verifiquen que la historia implementada cumple la funcionalidad especificada, dejando la optimización del código siempre para el final. Los conceptos más importantes del desarrollo del código en este método son los siguientes:

- **Disponibilidad del cliente:** Uno de los requerimientos de XP es tener al cliente disponible durante todo el proyecto. No solamente como apoyo a los desarrolladores, sino formando parte del grupo. El involucramiento del cliente es fundamental para que pueda desarrollarse un proyecto con el método XP. Al comienzo del proyecto, el cliente debe proporcionar las historias de usuarios. Pero, dado que estas historias son expresamente cortas y de "alto nivel", no contienen los detalles necesarios para realizar el desarrollo del código. Estos detalles deben ser proporcionados por el cliente, y discutidos con los desarrolladores, durante la etapa de desarrollo. No se requieren de largos documentos de especificaciones, sino que los detalles son proporcionados por el cliente, en el momento adecuado, cara a cara a los desarrolladores.
- **Uso de estándares:** Si bien esto no es una idea nueva, XP promueve la programación basada en estándares, de manera que sea fácilmente entendible por todo el equipo. La aplicación de estándares de código logra un ambiente de familiaridad con características comunes entre los archivos de código fuente creados por los desarrolladores. Además sin la aplicación de éstos será difícil la refactorización, construcción de pruebas y la comprensión de códigos por parte de otros programadores.
- **Programación dirigida por las pruebas:** En los métodos tradicionales, la fase de pruebas, incluyendo la definición de los casos de prueba, es usualmente realizada sobre el final del proyecto, o sobre el final del desarrollo de cada módulo. XP propone un modelo inverso, en el que, lo primero que se escribe son los test que el sistema debe pasar. Luego, el desarrollo debe ser el mínimo necesario para pasar las pruebas previamente definidas. Las pruebas a los que se refieren esta práctica, son las pruebas unitarias, realizados por los desarrolladores. La definición de estos test al comienzo, condiciona o dirige el desarrollo.

- **Programación en parejas:** Sugiere la producción de código por parejas de desarrolladores, ambos trabajando juntos en una misma computadora. Si bien parece que ésta práctica duplica el tiempo asignado al proyecto y por ende, los costos en recursos humanos, al trabajar en pares se minimizan los errores y se logran mejores diseños, compensando la inversión en horas.
- **Integraciones permanentes:** Todos los desarrolladores necesitan trabajar siempre con la "última versión". Realizar cambios o mejoras sobre versiones antiguas causan graves problemas, y retrasan al proyecto. Es por eso que XP promueve publicar lo antes posible las nuevas versiones, aunque no sean las últimas, siempre que estén libres de errores. Idealmente, todos los días deben existir nuevas versiones publicadas. Para evitar errores, solo una pareja de desarrolladores puede integrar su código a la vez.
- **Propiedad colectiva del código:** Establece que todo equipo puede contribuir con nuevas ideas que apliquen a cualquier parte o módulo del proyecto. Asimismo, cualquier pareja de programadores puede cambiar el código que sea necesario para corregir problemas, agregar funciones o recodificar. En este caso, quienes encuentran un problema, o necesitan desarrollar una nueva función, pueden resolverlo directamente, sin necesidad de negociar con el "dueño" o autor del módulo, ya que de hecho, este concepto no existe en XP.
- **Ritmo sostenido:** XP indica que debe llevarse un ritmo sostenido de trabajo. Anteriormente esta práctica se denominaba "Semana de 40 horas". Sin embargo, lo importante no es si se trabajan, 35, 40 o 42 horas por semana, el concepto que se desea establecer con esta práctica es el de planificar el trabajo de manera de mantener un ritmo constante y razonable, sin sobrecargar al equipo. Cuando un proyecto se retrasa, trabajar tiempo extra puede ser más perjudicial que beneficioso. El trabajo extra desmotiva inmediatamente al grupo e impacta en la calidad del producto. En la medida de lo posible, se debería renegociar el plan de entregas, realizando una nueva reunión de planificación con el cliente, los desarrolladores y los gerentes.

2.1.6.4 Pruebas

XP propone la división de las pruebas en dos grupos: pruebas unitarias y pruebas de aceptación. Los desarrolladores realizan las pruebas unitarias a medida que se escriba el código. Y el cliente escribe las pruebas de aceptación para evaluar el cumplimiento de los requerimientos del sistema. Así, definiendo una serie de pruebas durante la construcción del proyecto, y aplicándolas cada cierto período de tiempo para validar los resultados obtenidos de las entregas, se asegura mantener la calidad esperada del producto final. Los conceptos más importantes de las pruebas en este método son los siguientes:

- **Pruebas unitarias:** Las pruebas unitarias son una de las piedras angulares de XP. Todos los módulos deben pasar las pruebas unitarias antes de ser liberados o publicados. Por otra parte las pruebas deben ser definidas antes de realizar el código. Que todo código liberado pase correctamente las pruebas unitarias es lo que habilita que funcione la propiedad colectiva del código. En este sentido, el sistema y el conjunto de pruebas debe ser guardado junto con el código, para que pueda ser utilizado por otros desarrolladores, en caso de tener que corregir, cambiar o recodificar parte del mismo.
- **Detección y corrección de errores:** Cuando se encuentra un error (bug), éste debe ser corregido inmediatamente, y se debe tener precaución para que errores similares no vuelvan a ocurrir. Asimismo, se generan nuevas pruebas para verificar que el error haya sido resuelto.
- **Pruebas de aceptación:** Las pruebas de aceptación son creadas con base en las historias de usuarios, en cada ciclo de la iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada. Las pruebas de aceptación son consideradas como pruebas de caja negra. Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Asimismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación.

Capítulo 2 : Adaptación de Método de Desarrollo, Tecnologías y Requerimientos

En este capítulo se presenta el método utilizado para el desarrollo del sistema, el cual se basó en una adaptación de Programación Extrema (XP). Además se plantean y priorizan las historias de usuario y se describen las herramientas tecnológicas que se utilizan para la construcción del sistema.

2.1 Adaptación del Método de Desarrollo XP

El método ágil de desarrollo escogido es Programación Extrema (XP), la cual brinda bondades como la búsqueda de la simplicidad, constantes y rápidas respuestas al cliente buscando su mayor satisfacción, reducción de documentación, el estar orientada a equipos pequeños, entre otras. A continuación se especifican los aspectos más resaltantes de XP, su adaptación para el desarrollo del sistema. A continuación se especifican los aspectos más resaltantes de XP, su adaptación para el desarrollo del sistema.

2.1.1 Actores y Responsabilidades

Los actores son todas las personas involucradas en el desarrollo del proyecto, los cuales a su vez cumplen distintos roles o responsabilidades según su importancia y nivel de participación. A continuación se destacan los roles existentes en el presente proceso de desarrollo:

- **Programador:** Es el pilar fundamental del desarrollo en XP, tiene grandes habilidades en cuanto a la comunicación y al desarrollo en equipo. Adicionalmente, tiene la capacidad de poder abordar de forma simple y sencilla problemas complejos.
- **Cliente:** Es el encargado de proveer las historias de usuario, realizar las pruebas de aceptación, requisitos funcionales y no funcionales deseables en la aplicación y la toma de decisiones acertadas sobre las características esenciales de la aplicación.
- **Probador:** Su función se centra en realizar las pruebas de integración al sistema del código provisto por los programadores y de verificar el correcto funcionamiento de

la aplicación. También realiza pruebas regulares y da mantenimiento siempre sustentando los resultados con informes precisos.

- **Entrenador:** Se encarga de dar seguimiento al proceso general del grupo, calculando el tiempo que toman sus tareas y el progreso general a las metas que se quieren alcanzar. Realiza estimaciones de tiempo y da la retroalimentación al equipo con el fin de mejorar el rendimiento.

La Tabla 2.1 muestra los roles que cumplen los miembros de equipo involucrados en el desarrollo del proyecto.

	Programador	Cliente	Entrenador	Probador
Daniel Guánchez	X			X
Eugenio Scalise		X	X	X
Andrés Sanoja		X	X	X

Tabla 2.1 Actores y Roles que se desempeñan.

2.1.2 Actividades de XP

XP está compuesta por cuatro actividades fundamentales las cuales están contenidas en cada una de las iteraciones del proceso de desarrollo. A continuación una breve descripción y como es la adaptación de cada una de ellas.

2.1.2.1 Planificación

Como primer paso fue la de realizar el análisis global del sistema a desarrollar, creando una representación sencilla de las partes que lo conforman y como se comunican entre ellas. Esto sigue la idea de XP de crear una metáfora del sistema que represente de forma general cual es el resultado que se persigue en el desarrollo.

Además se crea una lista de historias de usuario durante la conversación inicial con el cliente, rol que desempeña el tutor a cargo del desarrollo de esta investigación, quien es responsable de observar y añadir todo tipo de modificación para su implementación

siguiendo una prioridad determinada para el aporte de la historia de usuario a la funcionalidad principal del sistema.

Se decidió trabajar en función del tiempo, utilizando los días como unidad de medida, así cada historia de usuario tiene una cantidad de días estipulados para realizar actividades que en conjunto dieran como resultado la implementación de la funcionalidad descrita en la historia en cuestión. Se utiliza un formato para registrar las historias de usuario el cual contiene: un número que servirá de identificador, un nombre, el tipo (nueva o correctiva/mejora), una prioridad (alta, media o baja), un riesgo técnico (alta, media o baja), una estimación del esfuerzo y una breve descripción sobre la historia de usuario. El riesgo técnico o riesgo de implementación de cada historia de usuario, fue determinado subjetivamente por los desarrolladores basándose en su experiencia con las tecnologías a utilizar y el trabajo que implicaba las actividades a realizar. El formato de presentación de las historias de usuario se observa en el Tabla 2.1

Número:	**Nombre:**
Tipo:	**Prioridad:**
Riesgo Técnico:	**Esfuerzo Estimado:**
Descripción:	

Tabla 2.2 Formato para registrar historias de usuario.

Se realiza la planificación de iteraciones en las cuales se divide la implementación de las historias de usuario definidas, cada una con una duración de entre 2 a 5 semanas, y las entregas al cliente para observaciones, correcciones, cambios y realización de pruebas que se llevaran a cabo al final de cada iteración. Se utiliza un formato al inicio de cada iteración el cual contendrá el número de la iteración, una descripción, el número y nombre de las historias de usuarios a desarrollar, la fecha de comienzo de cada historia, la fecha de inicio y la fecha de fin de la iteración. El formato de presentación de la planificación de cada iteración se observa en el Tabla 2.2

N° de Iteración	
Descripción	
Fecha Inicio – Fin	
Historias de Usuario	
Tiempo Estimado	

Tabla 2.3 Formato para registrar la planificación de las iteraciones.

2.1.2.2 Diseño

El diseño en XP sigue de forma rigurosa el principio de simplicidad, prefiriendo siempre un diseño simple respecto de una presentación más compleja. Además el diseño debe ofrecer una guía de implementación para una historia de usuario determinada.

Siguiendo la modelación ágil, se agregan los diagramas de clases UML y modelos de datos entidad relación (E/R) que se creyeron necesarios para el fácil entendimiento y documentación del sistema. En caso de haberlos creado en alguna iteración anterior, se actualizaron, desechando las versiones anteriores.

2.1.2.3 Codificación

En esta fase se realizan las instalaciones y configuraciones del ambiente que sean necesarias, además de toda la codificación que da solución a las historias de usuario de cada una de las iteraciones. Se definieron los estándares de Codificación para la programación en JEE, implementación del modelo de datos (MySQL) y la plataforma de desarrollo.

En cuanto a la codificación para la programación (JEE), el estándar de codificación empleado fue el descrito por Scott Hommel de Sun Microsystems Inc (Homme, 2007), esto con la finalidad de facilitar el entendimiento y mantenimiento del código.

En lo que se refiere a la implementación del modelo de datos, se siguieron los siguientes lineamientos:

- Todas las tablas sin excepción, constaron de 5 columnas adicionales para fines de auditoria: usuario_creador (usuario que crea el registro en la tabla), fecha_creacion (fecha en la que se crea el registro en la tabla), usuario_actualizador (usuario que actualiza la información del registro), fecha_ultima_actualizacion (fecha en la que se lleva a cabo la actualización del registro).
- Todas las tablas sin excepción, usaron como clave primaria una columna llamada Id_<tabla>, la cual consiste de un número entero que se incrementa con cada registro nuevo.

Las 40 horas semanales se distribuyen en los 7 días de la semana, teniendo jornadas nocturnas de lunes a viernes y diurnas los fines de semana. La aplicación Web es desarrollada por una sola persona, por lo que la Programación en Parejas no aplica; sin embargo, se revisa el código constantemente para obtener una aplicación lo más refinada y eficiente posible.

Dado que el desarrollo es realizado por una sola persona, no fue posible la aplicación de la práctica de programación en pareja, sin embargo, se revisa el código constantemente para obtener una aplicación lo más refinada y eficiente posible.

2.1.4.4 Pruebas

En esta fase se realizan las pruebas funcionales y de aceptación de cada Historia de Usuario, asegurando de esta forma, el buen funcionamiento de dicha implementación. Los restantes tipos de pruebas, como las unitarias y de integración, no se incluyen en esta fase para evitar retrasos en los tiempos de entrega en vista de ser una sola persona a cargo de la programación.

Al finalizar cada iteración, se escriben las pruebas de aceptación indicando los casos de prueba, sus resultados esperados y los obtenidos. Para cumplir este aspecto, se utiliza el siguiente formato (Ver Tabla 2.4).

N° Caso de Prueba	Historias de Usuario	Caso de Prueba	Resultado Esperado	Resultado Obtenido

Tabla 2.4 Formato para registrar los casos de prueba.

2.2 Tecnologías

Se procede a la creación de una aplicación web, de esta manera los usuarios del sistema podrán acceder a la aplicación desde cualquier lugar que disponga de conexión a Internet. Por lo tanto, se ha diseñado e implementado la aplicación siguiendo las especificaciones definidas por la plataforma JEE. Los principales motivos de esta elección para el desarrollo de la aplicación son:

- Abstracción al desarrollador de las tareas de bajo nivel, ya que se encuentran cubiertas por las propias APIs de la plataforma JEE.
- Existencia de una gran variedad de *Frameworks* que incrementan las utilidades o tareas de bajo nivel.
- Java como lenguaje de programación. Enfocado claramente a la programación orientada a objetos (POO) que favorece en gran medida la reutilización del código fuente y la portabilidad.
- Interoperable con otras tecnologías como XML, JavaScript, HTML entre otras.
- Fomenta el uso del patrón de diseño Modelo-Vista-Controlador, el cual permite separar la parte lógica de la presentación en una aplicación web.

2.2.1 Java Enterprise Edition (JEE)

JEE es una plataforma abierta y estándar diseñada por la reconocida empresa Sun Microsystems, que permite la construcción de aplicaciones empresariales y surge de la necesidad del mercado de desarrollo de *software*, respecto a contar con medios y herramientas que permitan construir aplicaciones para este rubro (Oracle, 2010).

Esta plataforma de desarrollo busca simplificar las aplicaciones empresariales basándolas en componentes modulares y estandarizados, brindando un completo conjunto de servicios a estos componentes, y manejando muchas de las funciones de la aplicación de manera automática, pudiendo realizar todo esto sin necesidad de recurrir a una programación muy compleja. Se le denomina plataforma porque proporciona especificaciones técnicas que describen el lenguaje pero, además, provee las herramientas para implementar productos de *software* (aplicaciones) basados en dichas especificaciones.

Con todo lo anteriormente descrito podemos concluir que JEE no es solo una tecnología para el desarrollo de aplicaciones empleando lenguaje Java, sino también un estándar de desarrollo, construcción y despliegue de aplicaciones.

2.2.2 *Struts*

Struts es un framework MVC desarrollado dentro de la Apache *Software* Foundation que proporciona soporte a la creación de las capas vista y controlador de aplicaciones web basadas en la arquitectura Model2 (Apache Struts, 1999). Está basado en tecnologías estándar como Servlets, JavaBeans y XML. Struts proporciona un controlador que se integra con una vista realizada con páginas JSP, incluyendo JSTL y Java Server Faces, entre otros. Este controlador evita la creación de Servlets y delega en acciones creadas por el desarrollador, simplificando el desarrollo de aplicaciones web. En cuanto a la capa vista, permite utilizar entre otras tecnologías páginas JSP para la realización de la interfaz. Para facilitar las tareas comunes en la creación de esta capa existen una serie de tecnologías que se integran con *Struts*:

- *Struts taglib*, una librería de etiquetas que proporciona numerosa funcionalidad evitando el escribir código Java en las páginas JSP.
- JSTL, la librería de etiquetas estándar de Java que añade funcionalidades a la librería de etiquetas de *Struts* y sustituye alguna de las ya presentes.
- Tiles, una extensión que permite dividir las páginas JSP en componentes reusables para la construcción de la interfaz.
- *Struts Validator*, proporciona validación de los formularios basándose en reglas fácilmente configurables.

2.2.3 *Hibernate*

Hibernate es un mapeador objeto/relacional y servicio de consultas para Java. Permite desarrollar clases persistentes a partir de clases comunes, incluyendo asociación, herencia, polimorfismo, composición y colecciones de objetos. Utiliza un lenguaje de consulta potente (HQL del inglés *Hibernate Query Language*) diseñado como una mínima extensión orientada a objetos de SQL, es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación. También permite expresar consultas utilizando SQL nativo o consultas basadas en criterios. Soporta todos los sistemas gestores de bases de datos SQL y se integra de manera elegante y sin restricciones con los más populares servidores de aplicaciones JEE y contenedores web, y por supuesto también puede utilizarse en aplicaciones de escritorio (Hibernate, 2013).

2.2.4 JasperReport

Es un motor de reportes de código abierto, puede ser embebido en todo tipo de aplicaciones, desde las que generan reportes a partir de una plantilla o modelo predeterminado hasta las que brindan más libertad al usuario para diseñar sus propios reportes y ejecutar otras operaciones complejas (JasperReports, 2013). Es una biblioteca implementada completamente en Java brindando el máximo nivel de portabilidad, con un amplio y expandible grupo de posibles fuentes de datos. Posee además una abundante variedad de formatos de salida, importación así como una gran comunidad mundial que mantiene y desarrolla la biblioteca. Genera informes para formatos de impresión predeterminados existentes o para reportes continuos a ser visualizados en la web, los reportes pueden ser exportados a formatos como: PDF, XML, HTML, CSV, XLS, RTF, TXT.

2.2.5 JQuery

JQuery es una librería Javascript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas (JQuery, 2012). JQuery no solo contiene un conjunto de funciones usadas comúnmente, sino que también provee de una forma de escribir código Javascript de forma reducida.

2.2.6 JQuery UI

JQuery UI es una librería de componentes para JQuery, desarrollada en gran parte por el mismo equipo de desarrollo de JQuery, que añade un conjunto de funcionalidades para la creación de Aplicaciones Web interactivas (JQuery UI, 2012). Esta librería proporciona abstracciones de bajo nivel de interacción y animación, efectos avanzados y de alto nivel, además de un conjunto completo de controles de interfaz de usuario conocido como widgets. Cada elemento tiene un conjunto de opciones configurables y se les pueden aplicar estilos CSS específicos.

2.2.7 MySQL

MySQL es un sistema de administración de bases de datos relacional que almacena y distribuye una gran cantidad de datos, típicos de una aplicación (MySQL, 2013). Está basado en la arquitectura cliente-servidor, por lo que el servidor de base de datos puede estar asociado a múltiples clientes. Utiliza el lenguaje de consulta estructurado (SQL, del inglés Structured Query Language) para el acceso y manipulación de los datos.

Su conectividad, velocidad, seguridad y licencia pública hacen de MySQL altamente apropiado para acceder bases de datos en Internet y el SGBD más conveniente para el desarrollo de aplicaciones en las que se buscan soluciones rápidas y de bajos costos.

2.2.8 Especificaciones Técnicas

El desarrollo de la aplicación se lleva a cabo utilizando las siguientes herramientas tecnológicas:

- MVC (Modelo-Vista-Controlador) como patrón de diseño de la aplicación.
- Java (JEE) como lenguaje de programación y patrón de diseño.
- NetBeans IDE 7.3 como entorno de programación.
- JasperReport 5.1 como motor para la presentación de reportes/informes.
- IReport Designer 5.1 como entorno para el diseño de reportes/informes.
- Struts 1.3 en la capa Control.
- Hibernate 3.2 para la capa Modelo.

- Tomcat 7.3 como servidor de la aplicación.
- MySQL 5.6 como sistema manejador de base de datos.

2.2.9 Arquitectura de Desarrollo

La arquitectura utilizada para el desarrollo de la aplicación, está basada en las siguientes herramientas que podemos observar en la Figura 2.1, las cuales están organizadas dentro del patrón de diseño MVC (Modelo – Vista - Controlador):

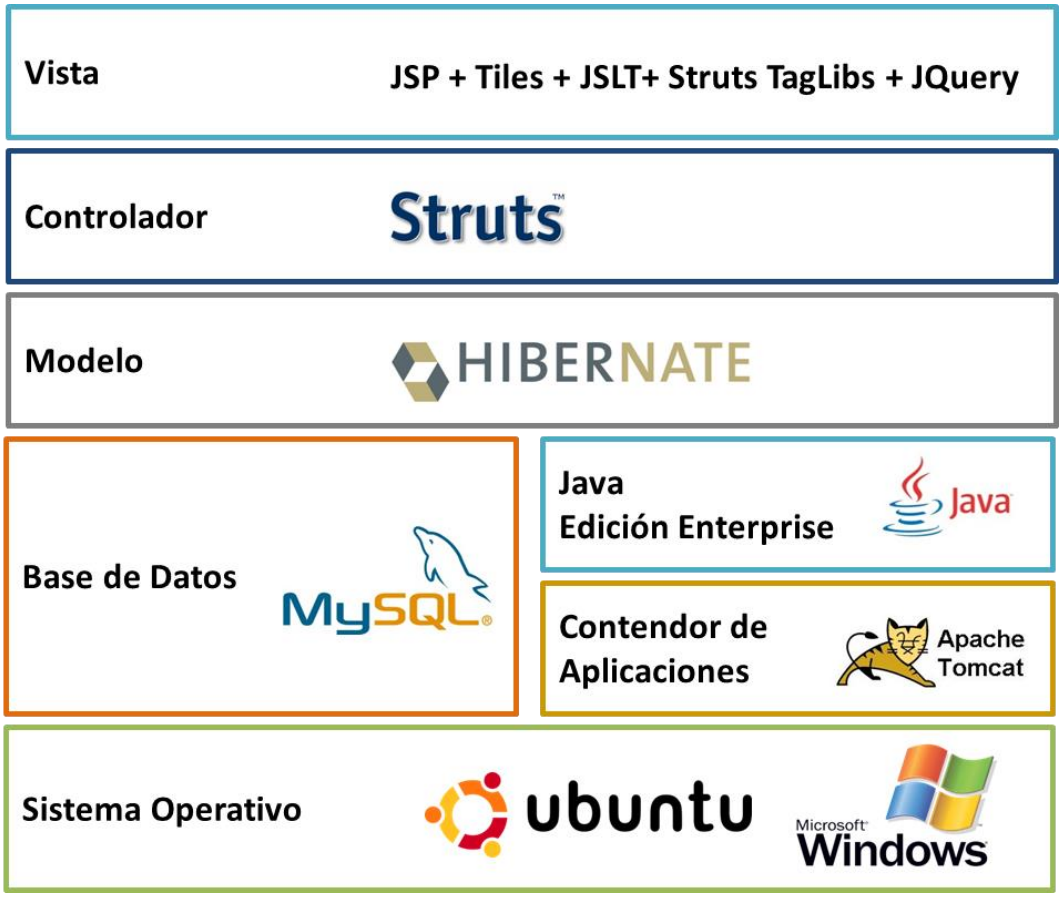


Figura 2.1 Arquitectura propuesta, considerando el patrón MVC

2.3 Análisis Global del Sistema

Este análisis global proviene desde el levantamiento de información que fue realizada al inicio del proyecto, tal como se describió en la sección 2.1.2.1, que conlleva a todos los requerimientos necesarios para el desarrollo de la aplicación. A partir de esto se obtienen las historias de usuario desde la primera reunión con el cliente y se construye un modelo del esquema de funcionamiento del sistema. Los resultados se presentan a continuación:

2.3.1 Historias de Usuario

Se crean las Historias de Usuario que se mencionan a continuación, que describen brevemente los requerimientos funcionales que la aplicación debe tener desde la perspectiva del cliente.

Número: 1	**Nombre**: Creación de cuentas de usuario
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 4 días
Descripción: El usuario Administrador puede crear cuentas de usuario para los miembros del equipo de trabajo. La aplicación maneja cuatro roles de usuario: • Visitante: Puede leer y revisar las historias de usuario en el sistema, pero no tiene permiso para agregar, editarlas o eliminarlas. • Miembro: Puede leer, crear y editar historias, tareas, casos de prueba, entregas e iteraciones, pero no puede eliminarlas. • Líder: Puede crear y editar, además puede eliminar historias, tareas, entregas e iteraciones. • Administrador: Todos los permisos de administrador del proyecto, además puede añadir, editar y eliminar proyectos y cuentas usuarios. Para crear una cuenta de usuario se introduce los siguientes atributos: el nombre de la persona, nombre de usuario, el tipo de usuario descrito anteriormente. Se establece la contraseña de usuario y vuelve a escribirla para confirmarla. Opcionalmente se puede especificar la dirección de correo electrónico. Una vez creado, el usuario debe ser visible desde la lista de usuarios. Las cuentas de usuario no se pueden eliminar, ya que se utilizan para rastrear cambios en un proyecto. Sin embargo, se puede desactivar una cuenta de usuario mediante la edición de la cuenta de usuario indicando un atributo de deshabilitado.	

Número: 2	**Nombre:** Editar las cuentas de usuario
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado:** 4 días
Descripción: El usuario administrador puede modificar los atributos de los usuarios registrados en la aplicación.	

Número: 3	**Nombre**: Control de acceso de usuarios
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 4 días
Descripción: Para acceder a la aplicación se debe solicitar el nombre de usuario y su contraseña para que tenga acceso a las funcionalidades que corresponden a su tipo de usuario.	

Número: 4	**Nombre**: Creación de proyectos
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 4 días
Descripción: El usuario administrador puede crear un proyecto para que él y los demás usuarios puedan comenzar a trabajar. Cada proyecto debe tener como atributos: un nombre que es utilizado para referirse a él, y opcionalmente una breve descripción del mismo. Una vez creado, el proyecto debe ser visible desde la lista de proyectos.	

Número: 5	**Nombre**: Editar los proyectos
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 4 días
Descripción: El usuario administrador puede modificar los atributos de un proyecto. Cuando un proyecto se ha culminado, se debe chequear un atributo de archivado y los usuario asignados no pueden seguir trabajando sobre el proyecto.	

Número: 6	**Nombre**: Eliminar los proyectos
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 2 días
Descripción: El usuario administrador puede eliminar los proyectos de la aplicación.	

Número: 7	**Nombre**: Restringir el acceso de un proyecto
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Medio	**Esfuerzo Estimado**: 4 días
Descripción: El usuario administrador puede asignar a los usuarios del equipo de trabajo de un proyecto. Los usuarios seleccionados podrán acceder al proyecto y pueden realizar sobre el las funciones correspondientes a su rol de usuario.	

Número: 8	Nombre: Selección de un proyecto
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Medio	**Esfuerzo Estimado**: 2 días
Descripción: El usuario puede seleccionar un proyecto para trabajar de los proyectos en los que se encuentra sido autorizado. Se debe mostrar el proyecto actual y permitir al usuario seleccionar un proyecto diferente.	

Número: 9	Nombre: Creación de historias
Tipo: Nueva	Prioridad: Alta
Riesgo Técnico: Bajo	Esfuerzo Estimado: 4 días

Descripción:

Las historias son una descripción de alto nivel de las funciones o requerimientos del proyecto, a partir de la perspectiva de los usuarios. Cada historia debe tener los siguientes atributos:

- Número: Número Identificador para la historia.
- Nombre: Nombre Identificador para la historia.
- Descripción: Breve descripción del requerimiento.
- Tipo: Indica si el tipo de historia es una nueva función, un defecto, una mejora.
- Responsable: El usuario que asumirá la responsabilidad para el seguimiento de los resultados.
- Prioridad: Importancia del requerimiento para la empresa o el cliente. Se establece para determinar el orden en el que trabajar en las historias. Los Valores que puede tomar son Bajo, Medio o Alto.
- Riesgo Técnico: El riesgo relativo o dificultad determinada por el equipo de desarrollo. Los valores que puede tomar son Bajo, Medio, Alto o Muy Alto.
- Esfuerzo Estimado: Tiempo necesario para implementar el requerimiento.
- Status: Indica si la historia está activa o completada.

Para crear una historia sólo el número y el nombre deben ser introducidos inicialmente, lo que permite capturar una solicitud de un requerimiento y volver posteriormente para profundizar en los detalles. Una vez creada, la historia debe ser visible desde la lista de historias del proyecto.

Número: 10	**Nombre**: Editar la historias
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 4 días

Descripción:

El usuario puede editar los atributos de las historias de usuario, lo que permite añadir más detalles para conseguir una mejor definición y adaptarla a sus necesidades.

Número: 11	**Nombre**: Eliminar las historias
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 2 día
Descripción: El usuario puede eliminar las historias del proyecto.	

Número: 12	**Nombre**: Copiar historia
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 2 días
Descripción: El usuario puede realizar una copia de una historia de usuario previamente definida en proyecto. La nueva historia se crea con los mismos atributos que posea la historia origen.	

Número: 13	**Nombre**: Continuar historia
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Medio	**Esfuerzo Estimado**: 4 días
Descripción: Cuando se llega al final de una iteración, pero no ha culminado una historia, se puede que mover a la siguiente iteración planificada. Para fines de seguimiento, se debe mantener el registro de la historia que se completó parcialmente en la iteración actual. El usuario debe especificar la iteración para continuar la historia, y qué nombre dar a la nueva historia de usuario (por defecto el nombre existente). Todas las tareas no completadas de la historia existente se mueven a la nueva historia.	

Número: 14	**Nombre**: Agregar tareas a una historia
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 4 días

Descripción:

Las historias se dividen en tareas concretas que se pueden ser realizadas por el equipo de desarrollo. El usuario puede agregar tareas para las historias de usuarios de proyecto, para cada tarea debe especificar los atributos:

- Número: Número identificador para la tarea.
- Nombre: Nombre identificador para la tarea.
- Descripción: Breve descripción de la tarea.
- Tipo: Indica si el tipo de tarea es de desarrollo, diseño, o documentación.
- Usuario Asignado: Usuario responsable de trabajar en la tarea.
- Usuario Par: Usuario adicional que también está trabajando en esta tarea.
- Horas Estimadas: Número de horas de desarrollo para llevar a cabo la tarea.
- Horas Completadas: Número de horas reales de trabajo realizado en la tarea.
- Status: Indica si la tarea está activa o completada.

Una vez agregada, la tarea debe ser visible desde la lista caso de tareas de la historia y la lista de tareas del proyecto.

Número: 15	**Nombre**: Editar las tareas
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 4 días

Descripción:

El usuario puede editar los atributos de las tareas agregadas en las historias de usuario, lo que permite añadir más detalles para conseguir una mejor definición y adaptarla a sus necesidades.

Número: 16	**Nombre**: Eliminar las tareas
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 2 días

Descripción:

El usuario puede eliminar las tareas de las historias de usuario del proyecto.

Número: 17	**Nombre**: Agregar casos de prueba a una historia
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 4 días

Descripción:

El usuario puede agregar casos de prueba a una historia de usuario. El caso de prueba debe contener los siguientes atributos:

- Número: Identificador del caso de prueba.
- Nombre: Nombre del caso de prueba.
- Descripción: Mayor descripción donde se debe indicar los pasos para realizar la prueba y el resultado esperado.
- Tipo: Indica si es una prueba de aceptación, de integración, unitaria, de desempeño o de usabilidad.
- Status: Indica si el caso de puesta está activo o inactivo.

Una vez agregado, el caso de prueba debe ser visible desde la lista caso de prueba de la historia y la lista de casos de prueba del proyecto.

Número: 18	**Nombre**: Editar los casos de prueba
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 4 días

Descripción:

El usuario puede editar los atributos de los casos de prueba agregados en las historias, lo que permite añadir más detalles para conseguir una mejor definición y adaptarla a sus necesidades.

Número: 19	**Nombre**: Eliminar los casos de prueba
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 2 día

Descripción:

El usuario puede eliminar casos de pruebas de las historias del proyecto.

Numero: 20	**Nombre**: Agregar resultados a casos de prueba
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 2 días
Descripción: El usuario puede agregar resultados obtenidos durante las pruebas. Se debe ingresar el tipo de resultado (Exitoso o Fallido) y una breve descripción o comentarios del resultado. Todos los resultados se almacenan como un historial para el caso de prueba, y pueden ser revisados en cualquier momento. El último resultado de la prueba se muestra en todas las listas de casos de prueba.	

Numero: 21	**Nombre**: Adjuntar archivos a las historias, tareas y casos de prueba
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 7 días
Descripción: El usuario puede adjuntar archivos en una historia, una tarea, o caso de prueba. Los archivos adjuntos pueden ser de cualquier tipo, y son útiles para el mantenimiento de los documentos, capturas de pantalla, planes de prueba, o cualquier otro archivo relacionado con una historia. Para agregar una archivo se debe especificar un título para el archivo y seleccionar el archivo que desea cargar desde el equipo local. Opcionalmente se puede proporcionar una descripción del archivo que va a cargar. Una vez agregado, el archivo debe ser visible desde la lista caso de archivos adjuntos.	

Número: 22	**Nombre**: Editar los archivos adjuntos
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 4 días
Descripción: El usuario puede editar los atributos de los archivos adjuntos en las historias, tareas o casos de prueba.	

Número: 23	**Nombre**: Eliminar los archivos adjuntos
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 3 días
Descripción: El usuario puede eliminar archivos adjuntos en las historias, tareas y casos de prueba.	

Número: 24	**Nombre**: Agregar comentarios en las historias, tareas
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 4 días
Descripción: El usuario puede agregar comentarios a una historia o tarea, para registrar la información relacionada con las notas y comentarios de los clientes. Los comentarios se muestran en orden cronológico y muestran los usuarios que los añadidos y la fecha y hora en que se introdujeron.	

Número: 25	**Nombre**: Eliminar los comentarios
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 3 días
Descripción: Solo el usuario que agrega el comentario puede eliminarlo.	

Número: 26	**Nombre**: Creación de entregas
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 3 días
Descripción: Una entrega representa la liberación de un proyecto de *software* a los usuarios finales. Esto podría ser un ciclo programado (por ejemplo, cada 6 meses), o con base en un conjunto de características. El usuario puede crear entregas o versiones para el proyecto, y asignarles historias de usuario para agrupar características que deben ser entregados juntas por razones de negocios. Se debe especificar un número o identificador de la entrega, un nombre, junto con una descripción y la fecha en que se espera que ocurra. También puede especificar una capacidad que indica la cantidad de esfuerzo estimado que el equipo puede completar. Una vez creada, la entrega debe ser visible desde la lista de entregas de la del proyecto.	

Número: 27	**Nombre**: Editar la entregas
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 3 días
Descripción: El usuario puede editar los atributos de las entregas y adaptarlas a sus necesidades.	

Número: 28	**Nombre**: Eliminar las entregas
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 1 día
Descripción: El usuario puede eliminar las entregas del proyecto.	

Número: 29	**Nombre**: Planificación de entregas
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Medio	**Esfuerzo Estimado**: 5 días
Descripción: La planificación de entrega prioriza las historias, y determinar conjunto de historias deseadas para la entrega de un proyecto. El usuario puede seleccionar las historias nos planificadas del proyecto (Historias no asignadas a una entrega) y asignarlas a la entrega. El plan de entrega debe mostrar del esfuerzo estimado planificado e indica si es superior a la capacidad de la entrega.	

Número: 30	**Nombre**: Visualización del resumen de la entrega
Tipo: Nueva	Prioridad: Media
Riesgo Técnico: Bajo	Esfuerzo Estimado: 2 días
Descripción: Se debe poder visualizar el contenido de la entrega, como una lista de las historias asignadas a la entrega, con descripción y estimación por cada historia. Se debe mostrar el esfuerzo total estimado para la entrega.	

Número: 31	**Nombre**: Creación de iteraciones
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 3 días
Descripción: La iteración es el período de tiempo determinado durante el cual un equipo de desarrollo realiza el diseño e implementación de *software* basado en las historias de usuario. El usuario puede crear interacciones para el proyecto. Se debe especificar un nombre para la iteración, junto con una descripción, una fecha de inicio y una fecha final. También puede especificar una capacidad que indica la cantidad de esfuerzo estimado que el equipo puede completar. Una vez creada, la iteración debe ser visible desde la lista caso de iteraciones de la del proyecto.	

Número: 32	**Nombre**: Editar las iteraciones
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 3 días
Descripción: El usuario puede editar los atributos de las iteraciones registradas y adaptarlas a sus necesidades.	

Número: 33	**Nombre**: Eliminar las iteraciones
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 1 días
Descripción: El usuario puede eliminar las iteraciones del proyecto.	

Número: 34	**Nombre**: Planificación de Iteraciones
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Media	**Esfuerzo Estimado**: 3 días
Descripción: El usuario puede elegir las historias no planificadas del proyecto (Historias no asignadas a una iteración) y asignarlas para la iteración, en función del tiempo estimado del desarrollador. El plan de iteración de debe mostrar el total del esfuerzo estimado e indica si es superior a la capacidad de la iteración.	

Número: 35	**Nombre**: Seguimiento de las iteraciones
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Bajo	**Esfuerzo Estimado**: 3 días
Descripción: El usuario puede consultar todas las historias planificadas para la iteración y para cada una, muestra todas las tareas organizadas en tres grupos en función de su condición: pendientes, en progreso, y completadas.	

Número: 36	**Nombre**: Lista de historia de usuarios
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Medio	**Esfuerzo Estimado**: 6 días
Descripción: El usuario puede visualizar la lista de las historias de usuario del proyecto que se encuentra activo. Desde aquí, se puede buscar, ordenar y filtrar la lista para incluir sólo las historias de su interés. Se permite filtrar las historias por los siguientes criterios: Iteración, Entrega, Tipo y Status. Para la lista de historias se mostrará las siguientes mediciones: total del tiempo estimado de todas las historias, total del tiempo de todas las historias completadas, total del tiempo de todas las historias incompletas o activas.	

Número: 37	**Nombre**: Exportar las historias de usuario
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Medio	**Esfuerzo Estimado**: 3 días
Descripción: Para cualquier lista de historias del proyecto, se puede exportar todos los datos a Excel, Word o PDF.	

Número: 38	**Nombre**: Lista de tareas
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Medio	**Esfuerzo Estimado**: 6 días
Descripción: El usuario puede visualizar la lista de las tareas del proyecto que se encuentra activo. Desde aquí se puede buscar, ordenar y filtrar la lista para incluir sólo las tareas de su interés. Se permite filtrar las tareas por los siguientes criterios: Iteración, Entrega, Tipo y Status. Para la lista de tareas se mostrará las siguientes mediciones: Total del tiempo estimado de todas las tareas, total del tiempo de todas las tareas completadas, total del tiempo de todas las tareas incompletas o activas.	

Número: 39	**Nombre**: Exportar las tareas
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Medio	**Esfuerzo Estimado**: 3 días
Descripción: Para cualquier lista de tareas del proyecto, se puede exportar todos los datos a Excel, Word o PDF.	

Número: 40	**Nombre**: Lista de casos de prueba
Tipo: Nueva	**Prioridad**: Media
Riesgo Técnico: Medio	**Esfuerzo Estimado**: 6 días
Descripción: El usuario puede visualizar la lista de los casos de prueba del proyecto que se encuentra activo. Desde aquí, se puede buscar, ordenar y filtrar la lista para incluir sólo los casos de prueba de su interés. Se permite filtrar las historias por los siguientes criterios: Iteración, Entrega, Tipo y Status.	

Número: 41	**Nombre**: Exportar los casos de prueba
Tipo: Nueva	**Prioridad**: Alta
Riesgo Técnico: Medio	**Esfuerzo Estimado**: 3 días
Descripción: Para cualquier lista de casos de prueba del proyecto, se puede exportar todos los datos a Excel, Word o PDF.	

2.3.2 Metáfora

Con el propósito de poder brindarle a los usuarios (Profesores e Investigadores del CCPS y Centro ISYS) la posibilidad de, gestionar, organizar y supervisar el proceso de desarrollo de software aplicado en los trabajos de investigación, se pone a su disposición una aplicación web basada en la Gestión de Proyectos Agiles. En la Figurar 2.2 se muestra la metáfora de la aplicación planteada a desarrollar.

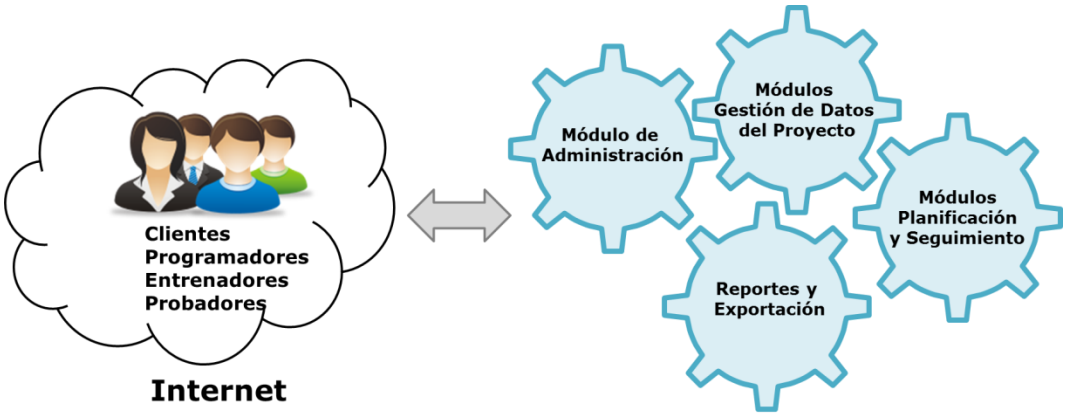


Figura 2.2 Metáfora de la Aplicación

La aplicación contiene varios módulos que trabajan en conjunto para ofrecer a los usuarios un enfoque integral y altamente eficaz para el ciclo de vida del desarrollo ágil de *software*:

- **Módulo de Administración:** Permite la gestión de proyectos y las cuentas de usuarios, y asignar los usuarios de los equipos de trabajo a los proyectos que correspondan.
- **Módulos de Gestión de Datos del Proyectos:** Permite capturar y organizar toda la información de los proyectos. Incluye la gestión de historias, tareas, casos de pruebas, archivos adjuntos y comentarios.
- **Módulos de Planificación y Seguimiento:** Permite la gestión interacciones y entregas. Planificar y priorización de las historias, tareas y los casos de prueba. Seguimiento del progreso de las tareas no iniciadas, en curso o completadas y monitoreo de las actividades durante el ciclo de desarrollo.
- **Reportes y Exportación:** Ordenar, buscar y filtrar las listas de historias de proyectos, tareas, o casos de prueba. Exportar datos del proyecto en Excel y PDF.

Capítulo 3 : Implementación

En este capítulo se precisan y describen todos los pasos efectuados por medio de iteraciones para lograr el desarrollo de la aplicación.

3.1 Plan de Entrega

En esta fase se establece la prioridad de cada historia de usuario así como una estimación del esfuerzo necesario de cada una de ellas con el fin de determinar un cronograma de entregas. Con base en esto partiendo de las historias descritas anteriormente se realiza una planificación en de cinco (5) iteraciones procurando agrupar las funcionalidades de cada módulo de la aplicación en una misma iteración:

- **Primera Iteración:** En esta primera iteración se desarrollan las funcionalidades necesarias para la gestión de proyectos, usuarios y la el acceso de los usuarios en la aplicación.
- **Segunda Iteración:** En esta segunda iteración se desarrollan las funcionalidades necesarias para la gestión de historias de usuarios, con sus tareas y casos de pruebas.
- **Tercera Iteración:** En esta tercera iteración se desarrollan las funcionalidades necesarias para agregar archivos y comentarios a las historias de usuarios, tareas y casos de pruebas.
- **Cuarta Iteración:** En una cuarta iteración se desarrollan la funcionalidad necesaria para la gestión, planificación y seguimientos de las entregas y las iteraciones de los proyectos.
- **Quinta Iteración:** En la quinta interacción se desarrollan las funciones necesarias para ordenar filtrar y exportar las listas de historias de usuarios, tareas y casos de prueba del proyecto.

3.2 Plan de Iteración

Debido al método de desarrollo utilizado para el presente Trabajo Especial de Grado, fue necesaria la definición y ejecución de un conjunto de iteraciones, las cuales se les estableció

fecha de inicio y fecha de culminación, con previa asesoría del tutor. Para cada iteración se realiza la adaptación de las actividades propias del proceso de desarrollo XP. A continuación se describen cada una de las cinto (5) iteraciones que conforman el proceso de desarrollo.

3.2.1 Primera Iteración: Gestión de Usuarios y Proyectos

En esta iteración se contemplan las funcionalidades que permiten la definición los usuarios y los de proyectos en la aplicación, así como también el acceso de los usuarios en la aplicación.

3.2.1.1 Planificación

Las historias de usuario a abordar se pueden ver en la Tabla 3.1.

N° de Iteración	1
Descripción	Desarrollo de los módulos para la gestión de usuarios, proyectos y acceso a la aplicación
Fecha Inicio – Fin	30/04/2012 -28/05/2012
Historias de Usuario	HU1: Crear cuentas de usuario HU2: Editar las cuentas de usuario HU3: Control de acceso a usuarios HU4: Creación de proyectos HU5: Editar los proyectos HU6: Eliminar los proyectos HU7: Restringir acceso a los proyectos HU8: Selección de un proyecto
Tiempo Estimado	28 días

Tabla 3.1 Planificación de la iteración: Gestión de Usuarios y Proyectos.

3.2.1.2 Diseño

En esta sección se describen los modelos de datos entidad relación (E/R) y diagramas de clases UML, que dan solución a las historias de usuario mencionadas en la planificación de la iteración, a través del uso de los patrones y framework elegidos.

Modelo de Datos

Para llevar a cabo la presente fase, se toma en cuenta lo conversado con el cliente referente a la definición de proyectos y cuentas de usuarios. En la Figura 3.1 se muestra el modelo datos E/R, que se desarrolla para la presente iteración en la que se puede destacar la incorporación de la tabla usuario y la tabla proyecto. Dentro de las reglas de negocio se destacan: cada usuario debe tener asociado un rol de usuario, un usuario puede estar asignado a más de un proyecto, y un proyecto puede tener acceso más un usuario.

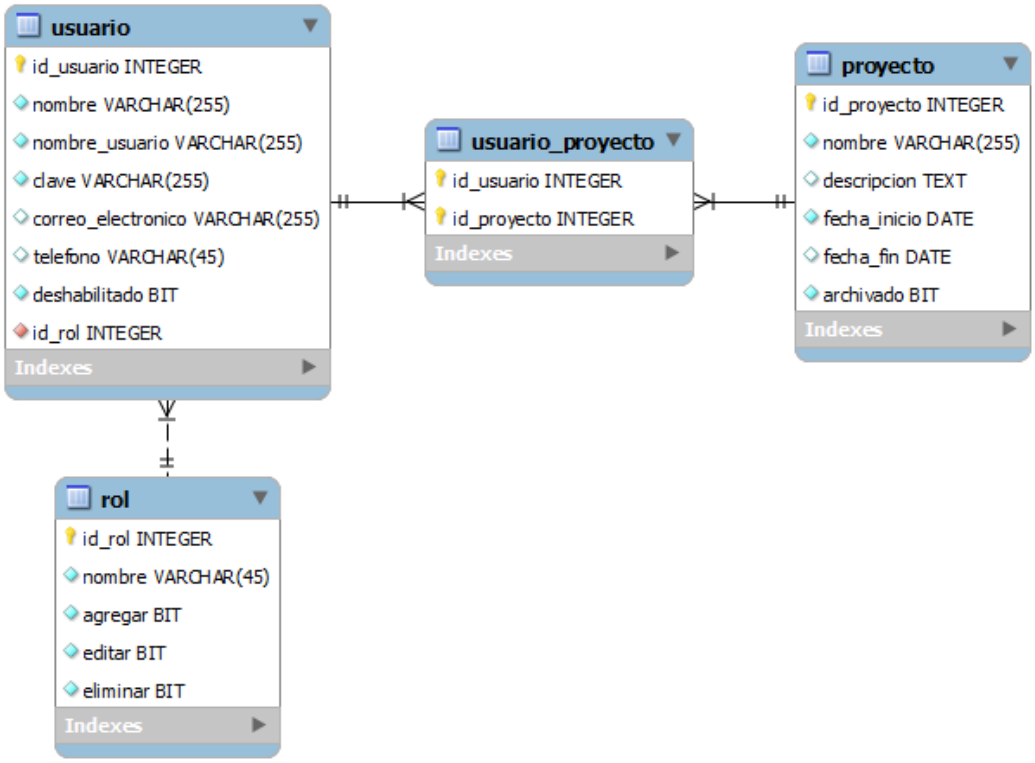


Figura 3.1 Modelo de datos E/R de la primera iteración.

Diagramas de Clases

Se comienza con los objetos del dominio que surgen en las historias relativas a la gestión de usuario y proyectos, se realiza el diagrama de clases UML que se puede ver en la Figura 3.2. Las clases del dominio, se utilizan para representar a los *ValueObjects* o *TransfertObejects* en el patrón *Data Access Object* (Data Access Object, 2012), son los objetos que están entre la lógica de la aplicación y la capa de datos. Estos objetos del dominio serán persistidos mediante *Hibernate,* sus atributos y sus relaciones pueden ser implementados directamente en la base de datos. Como son *bean* [1] tienen métodos *set* y *get*.

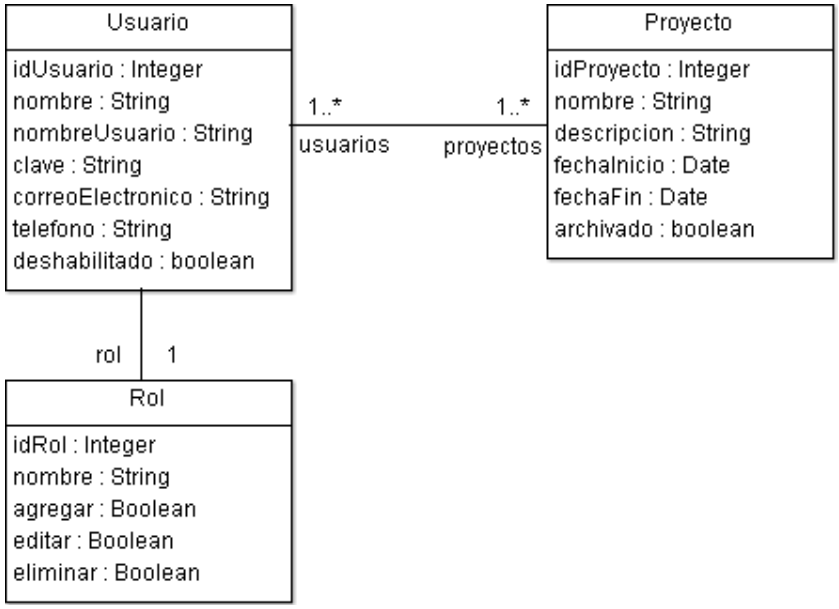


Figura 3.2 Diagrama de clases de los objetos del domino que modelan los usuarios y proyectos.

A continuación de describen cada uno de las clases de los objetos del dominio:

- *Proyecto***:** Esta clase represente un proyecto de trabajo dentro de la aplicación.
- *Usuario*: Esta clase representa los usuarios de la aplicación y las personas que trabajan dentro de un proyecto.
- *Rol*: Esta clase comprende el desempeña un usuario dentro de un proyecto, otorgándole las funciones correspondientes al tipo de rol, por ejemplo: Administrador, Líder, Miembro o Visitante.

[1] *Bean*: es un componente de software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java.

Para acceder a los datos se utiliza el patrón Data Access Object (DAO) (Data Access Object, 2012), que oculta la implementación utilizada en caso de necesitar cambiarla en el futuro. Dado que la persistencia será gestionada en principio con *Hibernate*, que permite el acceso a gran cantidad de metadatos, se realiza un DAO genérico que proporcione los servicios de persistencia para cualquier clase sin necesidad de escribir más líneas de código. Dado que el DAO será genérico es necesario indicarle como argumento la clase a la que proporcionará persistencia.

Utilizando *generics* característica introducida en Java SE 5.0 (The Java Tutorials, 2013), se puede diseñar un DAO genérico con las operaciones comunes para todas las clases persistentes del dominio. La Figura 3.3 muestra el diagrama de clases UML del DAO genérico.

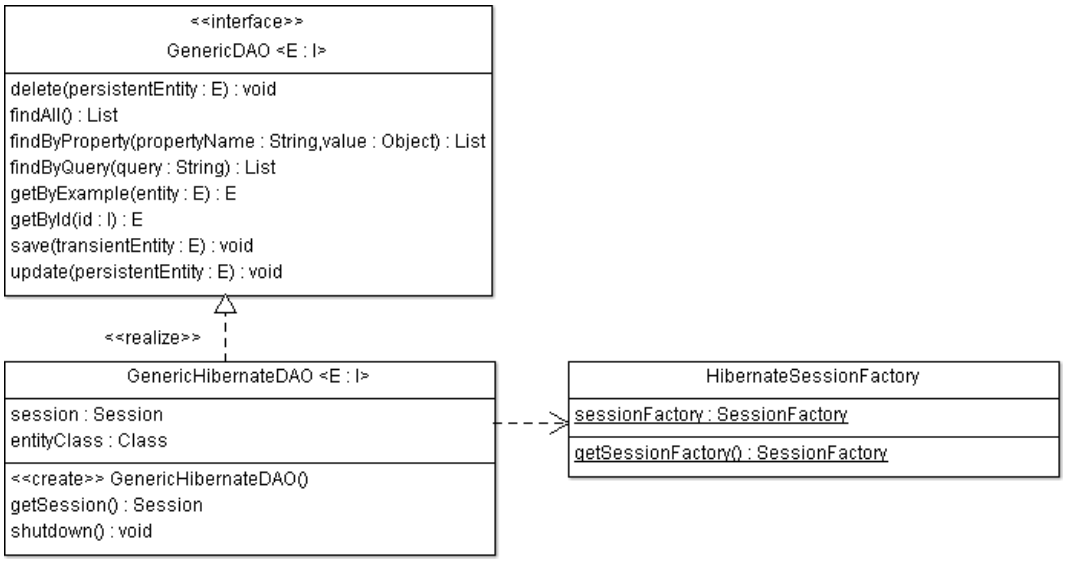


Figura 3.3 Diagrama de clases que definen el DAO genérico.

La interfaz parametrizada del DAO genérico *GenericDAO* recibe dos argumentos:

- El primer argumento (E) es la clase persistente para la que se implementa el DAO.
- El segundo argumento (I) define el tipo del identificador de la clase persistente.

La clase *GenericHibernateDAO* implementa utilizando *Hibernate* los métodos de la interfaz *GenericDAO*. La clase *HibernateSessionFactory* es la encargada de devolver las conexiones a la base de datos mediante *Hibernate*.

Cada clase del dominio tendrá su propio DAO específico, que extienden el DAO genérico proporcionando como argumentos el tipo de clase persistente, su clave primaria y los métodos adicionales. La Figura 3.4 muestra el diagrama de clases UML con los DAO específicos para las clases del dominio, definidas en la presente iteración.

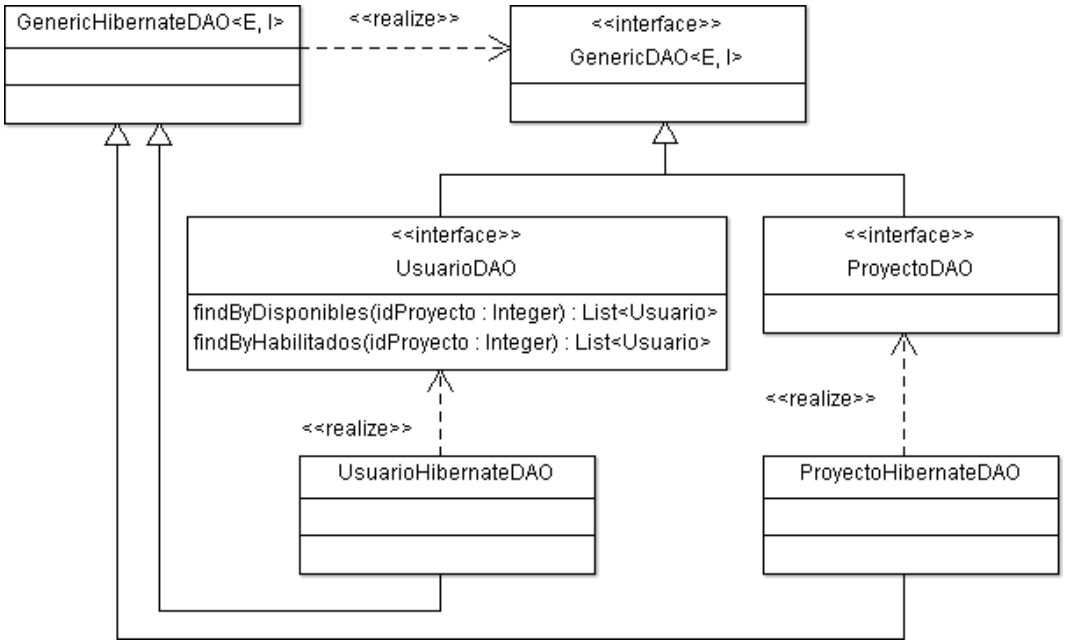


Figura 3.4 Diagrama de clases DAO específicas para acceder a los usuarios y proyectos

Para dar soporte al acceso de los datos de los usuarios y proyectos se definen las siguientes clases:

- *UsuarioDAO*: Interfaz con los métodos necesarios que debe implementar un DAO para persistir los usuarios, proporcionando como argumento el tipo de clase persistente *Usuario*. Proporciona los métodos adicionales:
    - *findByDisponibles*: Busca los usuarios habilitados que se pueden asignar a un proyecto.
    - *findByHabilitados*: Busca los usuarios que se encuentran habilitados asignados a un proyecto.

- *UsurioHibernateDAO:* Implementa utilizando *Hibernate* los métodos propios de la Interfaz *UsuarioDAO*, proporcionando como argumento la clase persistente *Usuario*.
- *ProyectoDAO*: Interfaz con los métodos necesarios que debe implementar un DAO para persistir los proyectos. proporcionando como argumento la clase persistente *Proyecto*.
- *ProyectoHibernateDAO: P*roporciona como argumento las clase persistente *Proyecto*.

Como interfaz de la capa modelo hacia capas superiores dentro de la arquitectura MVC se crean las fachadas para ocultar los detalles de implementación de la capa modelo, desempeñando el papel de los patrones *SessionFacade* (Session Facade, 2012) y *BusinessDelegate* (Business Delegate, 2012). Se crea una clase fachada por cada uno de los distintos módulos en los que se ha separado la aplicación, cada clase implementa una interfaz propia.

La Figura 3.5 muestra el diagrama de clases UML con la fachada para ocultar los detalles de implementación del módulo Usuarios. La fachada contiene los DAO responsables de la persistencia de la clase Usuario y sus subclases.

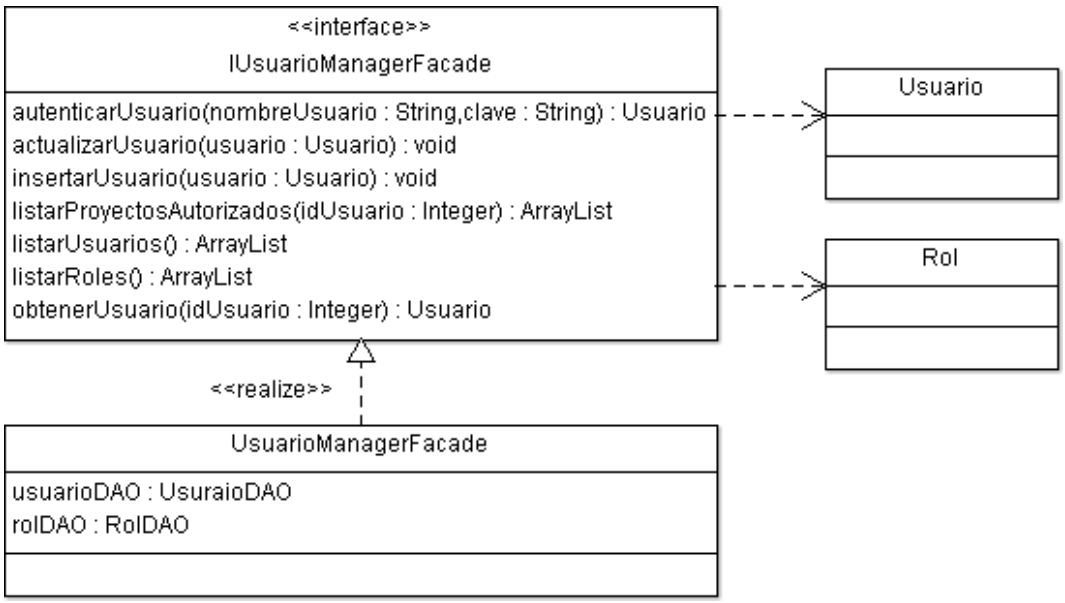


Figura 3.5 Diagrama de clases fachadas del módulo de usuarios.

La Figura 3.6 muestra el diagrama de clases UML con la fachada para ocultar los detalles de implementación del módulo Proyectos. La fachada contiene el DAO responsable de la persistencia de la clase Proyecto.

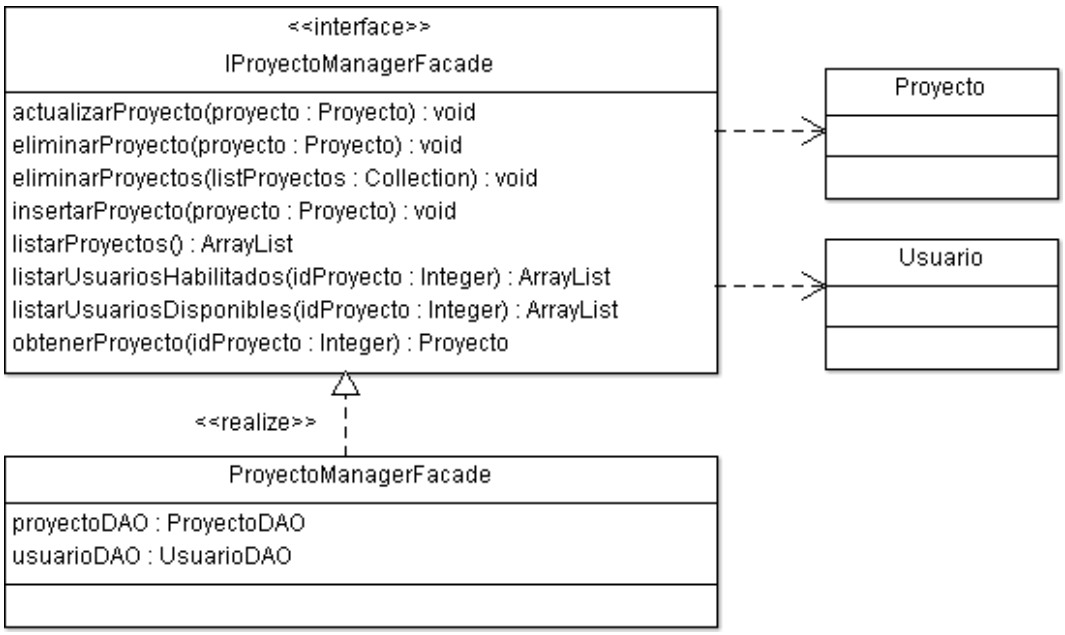


Figura 3.6 Diagrama de clases fachadas del módulo proyectos.

3.2.1.3 Codificación

Lo primero es instalar el entorno de desarrollo, siguiendo la arquitectura de desarrollo MVC, se optó por estructurar el subdirectorio *Source Packages* del proyecto en tres subdirectorios o paquetes, el paquete "controller" que contiene todas las acciones únicas (*Action*) y de grupo (DispatchAction), el paquete "view" que contiene los *ActionForms*, tanto estáticos como dinámicos (*DynaActionForm*), mientras que el paquete "model" contiene todas las clases de los objetos del dominio, las clases DAO, y las clases fachadas para manejar la lógica de negocio.

En una segunda instancia, para leer y almacenar los objetos en la base de datos, se tiene que iniciar *Hibernate* construyendo un objeto global *SessionFactory* a partir de un objeto *AnnotationConfiguration*, cuando se invoca el método *configure*, *Hibernate* busca el archivo de configuración *hibernate.cfg.xml*, de ahí obtiene el nombre de usuario, contraseña y la URL de la base de datos. A continuación se implementa la clase *HibernateSessionFactory*

que se encarga de iniciar y hacer accesible la *SessionFactory* de manera conveniente, como se muestra en la Figura 3.7.

```
package com.agilweb.model.daos.hibernate;

import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.SessionFactory;

public class HibernateSessionFactory {

    private static final SessionFactory sessionFactory;

    static {
        try {
            // Create the SessionFactory from standard (hibernate.cfg.xml)
            // config file.
            sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Figura 3.7 Implementación de la clase HibernateSessionFactory.

En un bloque estático se inicializa la instancia de *SessionFactory* que utiliza la aplicación. El método *getSessionFactory* devuelve siempre la misma instancia de este objeto.

El acceso a la información de la base de datos se realiza mediante los DAO, se implementa la interfaz parametrizada del DAO genérico la cual recibe los siguientes argumentos:

- E, es la clase persistente para la que se implementará el DAO
- I, define el tipo del identificador de la clase persistente. El identificador debe extender del objeto *Serializable* de Java.

Los métodos están definidos con base en estos parámetros y no están acoplados a ninguna tecnología de persistencia.

La Figura 3.8 muestra la implementación de la interfaz *GenericDAO*, donde se definen las operaciones comunes para manejar la persistencia de los objetos del dominio.

```
package com.agilweb.model.daos;

import java.io.Serializable;
import java.util.List;
import org.hibernate.HibernateException;

public interface GenericDAO <E, I extends Serializable> {

    public void delete(E persistentEntity) throws HibernateException;

    public List<E> findAll() throws HibernateException;

    public List<E> findByProperty(String propertyName, Object value) throws HibernateException;

    public List<E> findByQuery(String query) throws HibernateException;

    public E getByExample(E entity) throws HibernateException;

    public E getById(I id) throws HibernateException;

    public void save(E transientEntity) throws HibernateException;

    public void update(E persistentEntity) throws HibernateException;
}
```

Figura 3.8 Implementación de la interfaz del DAO Genérico.

La Figura 3.9 se presenta un fragmento del código de la clase *GericHibernateDAO* encargada de implementar los métodos de la interfaz *GenericDAO* utilizando *Hibernate*.

```
package com.agilweb.model.daos.hibernate;

public class GenericHibernateDAO <E, I extends Serializable> implements GenericDAO <E, I> {

    protected Session session;
    private Class<E>  entityClass;

    @SuppressWarnings("unchecked")
    public GenericHibernateDAO() {
        this.entityClass = (Class<E>) ((ParameterizedType) getClass().
                getGenericSuperclass()).getActualTypeArguments()[0];
    }

    public Session getSession() {
        return HibernateSessionFactory.getSessionFactory().getCurrentSession();
    }

    public void shutdown() {
        HibernateSessionFactory.getSessionFactory().close();
    }
}
```

Figura 3.9 Implementación con *Hibernate* del DAO genérico.

Para implementar con *Hibernate* el DAO genérico se requiere:

- Un objeto *Session* de *Hibernate*. El método *getSession* permite proporcionar la factoría a partir de la cual se obtendrá la sesión actual a través del método *getCurrentSession*. El método *shutdown* cierra la instancia única de *SessionFactory* y libera todos sus recursos (información de mapeo, conexiones, etc.)
- Conocer la clase persistente que es gestionada por el DAO. En el constructor del DAO se utiliza Java *reflection* para encontrar la clase del argumento genérico E y almacenarla en la propiedad *entityClass*. Una clase java parametrizada (*ParameterizedType*) dispone de métodos para obtener un arreglo con los tipos de sus argumentos (*getActualTypeArguments*), en este caso interesa el tipo del primer argumento (0).

A continuación se describe la implementación de los métodos comunes para para todas las clases persistentes:

- Método *delete*: Elimina el objeto dominio dado (Ver Figura 3.10).

```
@Override
public void delete(E persistentEntity) throws HibernateException {
    try {
        session = getSession();
        session.beginTransaction();
        session.delete(persistentEntity);
        session.getTransaction().commit();
    } catch (HibernateException he) {
        session.getTransaction().rollback();
        throw he;
    }
}
```

Figura 3.10 Implementación método *delete*.

- Método *findAll*: Recupera todos los objetos de dominio de tipo E (Ver Figura 3.11).

```
@Override
public List<E> findAll() throws HibernateException {
    try {
        session = getSession();
        session.beginTransaction();
        return session.createCriteria(entityClass).list();
    } catch (HibernateException he) {
        throw he;
    }
}
```

Figura 3.11 Implementación método *findAll*.

- Método ***findByProperty***: Recupera todos los objetos de dominio de tipo E por el valor de la propiedad (Ver Figura 3.12).

```
@Override
public List<E> findByProperty(String propertyName, Object value) throws HibernateException {
    try {
        session = getSession();
        session.beginTransaction();
        return session.createCriteria(entityClass)
                      .add(Restrictions.eq(propertyName, value))
                      .list();
     } catch (HibernateException he) {
         throw he;
     }
}
```

Figura 3.12 Implementación método findByProperty.

- Método *findByQuery*: Ejecuta una consulta para casos persistentes. Encuentra todas las instancias del objeto del dominio que coincida con el de consulta especificada (Ver Figura 3.13).

```
@Override
public List<E> findByQuery(String query) throws HibernateException {
    try {
        session = getSession();
        session.beginTransaction();
        return session.createSQLQuery(query)
                      .addEntity(entityClass)
                      .list();
    } catch(HibernateException he){
        throw he;
    }
}
```

Figura 3.13 Implementación método *findByQuery*.

- Método *getById*: Recupera un objeto de dominio por el identificador especificado (Ver Figura 3.14).

```
@Override
public E getById(I id) throws HibernateException {
    try {
        session = getSession();
        session.beginTransaction();
        return (E) session.get(entityClass, id);
    } catch (HibernateException he) {
        throw he;
    }
}
```

Figura 3.14 Implementación método *getById*.

- Método *getByExample*: Recupera un objeto de dominio que coincida con la instancia especificada (Ver Figura 3.15).

```
@Override
public E getByExample(E entity) throws HibernateException {
    try {
        session = getSession();
        session.beginTransaction();
        return (E) session.createCriteria(entityClass)
                          .add(Example.create(entity))
                          .uniqueResult();
    } catch (HibernateException he) {
        throw he;
    }
}
```

Figura 3.15 Implementación método *getByExample*.

- Método *save*: Guarda el objeto de dominio dado (Ver Figura 3.16).

```
 @Override
 public void save(E transientEntity) throws HibernateException {
     try {
         session = getSession();
         session.beginTransaction();
         session.save(transientEntity);
         session.getTransaction().commit();
     } catch (HibernateException he) {
         session.getTransaction().rollback();
         throw he;
     }
}
```

Figura 3.16 Implementación método *save*.

- Método *update*: Actualiza el objeto de dominio dado (Ver Figura 3.17)

```
@Override
public void update(E persistentEntity) throws HibernateException {
    try {
        session = getSession();
        session.beginTransaction();
        session.update(persistentEntity);
        session.getTransaction().commit();
    } catch (HibernateException he) {
        session.getTransaction().rollback();
        throw he;
    }
}
```

Figura 3.17 Implementación método *update*.

Los DAO específicos para una clase persistente, extienden el DAO genérico proporcionando como argumentos el tipo de clase persistente y de su clave primaria. La Figura 3.18 muestra la implementación del DAO específico para manejar la persistencia del objeto usuario.

```
package com.agilweb.model.daos.hibernate;

public class UsuarioHibernateDAO extends GenericHibernateDAO <Usuario, Integer> implements UsuarioDAO {

    @Override
    public List<Usuario> findHabilitadosByProyecto(Integer idProyecto) throws HibernateException {
        try {
            String query = "SELECT * FROM agilweb.usuario
                            JOIN agilweb.usuario_proyecto
                            ON usuario.id_usuario = usuario_proyecto.id_usuario
                            WHERE usuario.deshabilitado = 0 AND usuario_proyecto.id_proyecto ="+idProyecto;
            return findByQuery(query);
        } catch (HibernateException he) {
            throw he;
        }
    }

    @Override
    public List<Usuario> findDisponiblesByProyecto(Integer idProyecto) throws HibernateException {
        try {
            String query = "SELECT * FROM agilweb.usuario
                            WHERE deshabilitado = 0
                            AND id_usuario NOT IN (SELECT id_usuario FROM agilweb.usuario_proyecto
                                                   WHERE id_proyecto = "+idProyecto+")";
            return findByQuery(query);
        } catch (HibernateException he) {
            throw he;
        }
    }
}
```

Figura 3.18 Implementación DAO específico para el manejar los usuarios.

Para no tener que implementar una acción de *Struts* para cada acción de la vista se utiliza *MappingDispatchActions* que permite crear una única clase para manejar varias URL de peticiones, agrupando también las funcionalidades relacionadas. Basándose en la petición escoge automáticamente el método que debe ser llamado, por ejemplo peticiones del tipo create* o update* realizan llamadas a los métodos *create* o *update* respectivamente.

El controlador constará de las clases *UsuarioAction* y *ProyectoAction*, que gestionan todas aquellas acciones que realice el usuario a través de la vista y las transforman en invocaciones a la fachada de la capa modelo. En el controlador será necesario añadir los correspondientes métodos a la acción para listar, agregar, insertar, editar, actualizar, eliminar y visualizar la información de los Usuarios y los Proyectos, junto con la configuración necesaria de *Struts*. Las acciones (*Action*) tienen la responsabilidad de recibir los *ActionForm* a clases del dominio e invocan a la fachada, como se muestra en la Figura 3.19.

```
public ActionForward insertar(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
        try {

            ProyectoForm proyectoForm = (ProyectoForm)form;

            Proyecto proyecto = new Proyecto();
            proyecto.setNombre(proyectoForm.getNombre());
            proyecto.setDescripcion(proyectoForm.getDescripcion());
            proyecto.setFechaInicio(FormatUtil.dateFormat(proyectoForm.getFechaInicio()));
            proyecto.setFechaFin(FormatUtil.dateFormat(proyectoForm.getFechaFin()));
            proyecto.setArchivado(proyectoForm.isArchivado());
            proyecto.setFechaCreacion(new Date());

            HttpSession httpSession = request.getSession(true);
            Usuario usuarioSistema = (Usuario) httpSession.getAttribute("usuarioSistema");
            proyecto.setUsuarioCreador(usuarioSistema);

            IProyectoManagerFacade proyectoManager = new ProyectoManagerFacade();
            proyectoManager.insertarProyecto(proyecto);

            ActionMessages mensajes = new ActionMessages();
            mensajes.add("successful", new ActionMessage("proyecto.insert.successful"));
            saveMessages(httpSession, mensajes);

        } catch (Exception  ex) {
            ActionMessages mensajes = new ActionMessages();
            mensajes.add("error", new ActionMessage("proyecto.insert.error"));
            HttpSession httpSession = request.getSession(true);
            saveMessages(httpSession, mensajes);
        }
        return mapping.findForward(LISTAR_PROYECTOS);
    }
```

Figura 3.19 Ejemplo de implementación de la acción insertar proyecto.

Las clases que implementan la lógica de la aplicación se definen en cada módulo de la aplicación, implementando el patrón Fachada. Estas reciben los objetos del dominio que son usados por *Hibernate* mediante los DAO para insertar, modificar información o para mostrar datos referentes de la base de datos. Estas clases son invocadas desde en las acciones de *Struts*. Un ejemplo claro para ver la implementación de la fachada, se muestra en la Figura 3.20.

```
public void insertarProyecto(Proyecto proyecto) throws Exception {
    try {
      ProyectoHibernateDAO proyectoDAO = new ProyectoHibernateDAO();
      proyectoDAO.save(proyecto);
    } catch (Exception  ex) {
      throw ex;
    }
}
```

Figura 3.20 Implementación de la fachada.

Las vistas están formadas por las páginas JSP. Utilizando Tiles es posible utilizar una aproximación basada en componentes para realizar el interfaz web, separando las páginas JSP en porciones reutilizables que se ensamblarán en los archivos de configuración de Tiles mediante definiciones. Dentro de las vistas que se desarrollaron para esta iteración se encuentran las siguientes:

- Acceso a la aplicación.
- Menú principal de la aplicación.
- Definición y Consulta de usuarios.
- Definición y Consulta de proyectos.
- Asignación de los equipos de trabajo.

A continuación se describen las pantallas de las funcionales de los módulos de usuarios y proyectos de la aplicación.

Como se puede observar, en la Figura 3.21 se muestra el formulario básico de inicio de sesión para poder acceder a la aplicación. En caso de dejar en blanco el campo usuario o contraseña del formulario, se mostrará un mensaje de error que indica que debe colocar el usuario o contraseña.

Figura 3.21 Formulario de acceso al sistema.

En caso de colocar el usuario y/o la contraseña inválida, se mostrará un mensaje de error que indica el usuario y/o la contraseña introducidos no son correctos, como se muestra en la Figura 3.22.

Figura 3.22 En caso de colocar el usuario y/o contraseña inválidos.

Si el usuario y contraseña son correctos, ingresará a la aplicación donde podrá visualizar el menú con las opciones correspondientes al tipo de usuario y la lista de proyectos al que usuario se encuentra asignado. La Figura 3.23 muestra la cabecera y el menú principal de la aplicación.

Figura 3.23 Cabecera y menú principal de la aplicación.

Al seleccionar la opción “Usuarios” del menú principal se puede muestra el listado de los usuarios creados en la aplicación, en el que se puede seleccionar el usuario que se desea visualizar, editar, o se puede agregar uno nuevo. La Figura 3.24 muestra el listado de los usuarios creados.

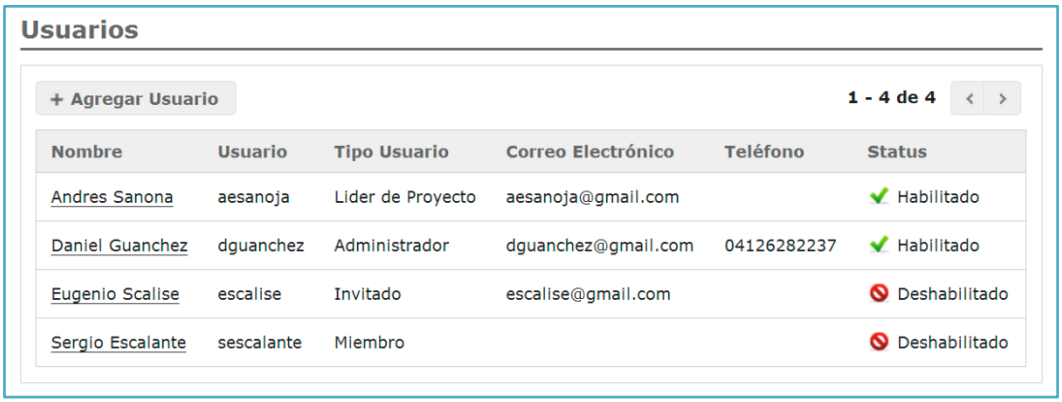


Figura 3.24 Listado de usuarios.

El nombre del usuario es un enlace que mostrara una pantalla con el detalle del usuario. Allí se podrá visualizar los datos básicos del usuario con las operaciones correspondientes. La Figura 4.26 muestra la pantalla de consulta de los detalles de un usuario.

Figura 3.25 Visualizar los detalles de un usuario.

La Figura 3.26 muestra la interfaz con el formulario que se utiliza tanto para crear o editar una cuenta de usuario.

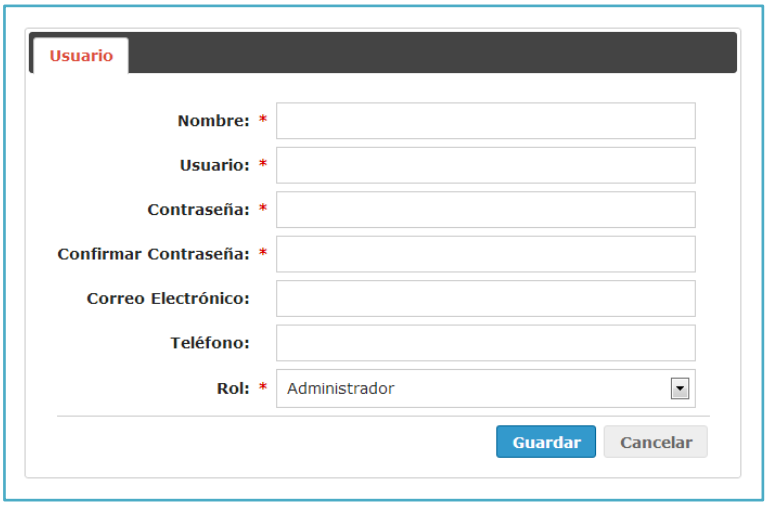


Figura 3.26 Formulario para agregar o editar un usuario.

Al seleccionar la opción "Proyectos" del menú principal se muestra el listado de los proyectos, en el que se puede seleccionar el proyecto que se desea editar, eliminar, o se puede agregar uno nuevo. La Figura 3.27 muestra la lista de todos los usuarios creados.

Figura 3.27 Listado de proyectos.

El nombre del proyecto es un enlace que mostrara una pantalla con el detalle del proyecto, allí se podrá visualizar los datos básicos del proyecto con las operaciones correspondientes. La Figura 3.28 muestra la pantalla de consulta de los detalles de un proyecto.

Figura 3.28 Visualizar los detalles de un proyecto.

En la Figura 3.29 se muestra la lista de usuarios asignados al proyecto, y un formulario para agregar nuevos usuario al proyecto, o remover usuarios ya asignados.

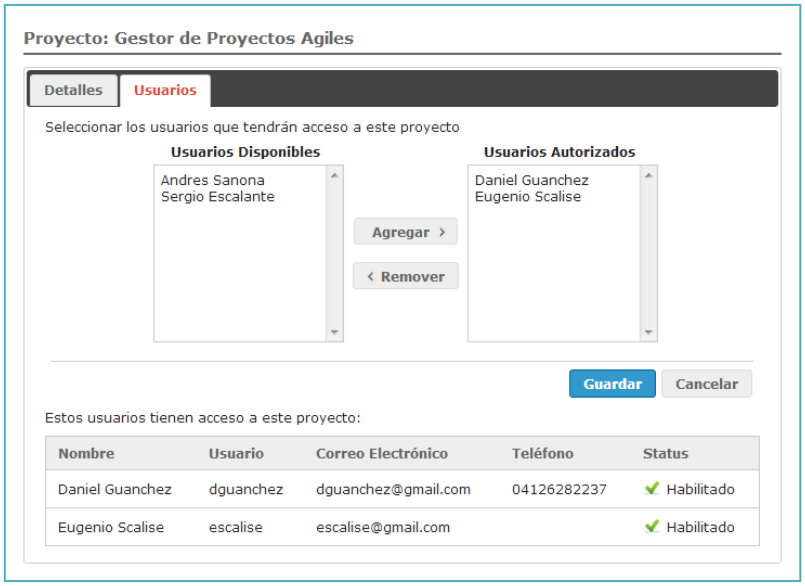


Figura 3.29 Formulario para asignar usuarios al proyecto.

La Figura 3.30 muestra el formulario que se utiliza tanto para crear o editar un proyecto.

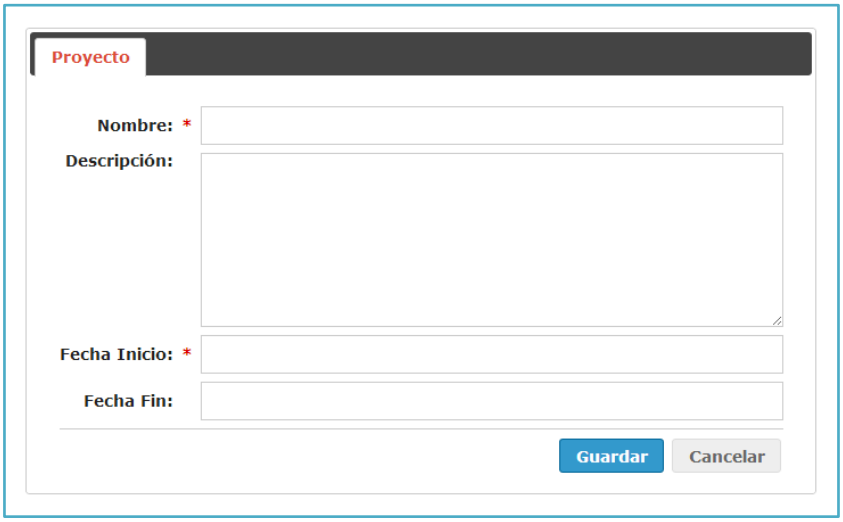


Figura 3.30 Formulario para agregar o editar un proyecto.

Cuando un proyecto se ha culminado, se debe marcar como archivado y los usuarios asignados no podrán seguir trabajando sobre él. Este atributo se muestra cuando se desea editar la información del proyecto.

3.2.1.4 Pruebas

En esta fase de la iteración, se busca probar cada uno de las historias de usuario asociadas del módulo usuarios y del módulo de proyectos. Para ello se contó con la presencia del cliente (Tutor) encargado de realizar cada una de las pruebas. Las pruebas fueron documentadas en bitácoras separadas por módulos:

- **Módulo de usuarios:** Las pruebas en este módulo se basan en la creación, edición de las cuentas de usuario y el acceso de los usuarios a la aplicación. La Tabla 3.2 describe los casos de prueba y resultados del módulo de usuarios.

N° Caso de Prueba	Historias de Usuario	Caso de Prueba	Resultado Esperado	Resultado Obtenido
1	HU1	Crear una cuenta de usuario	Creación de un nuevo usuario	Se guardó la información del usuario y se muestra en la lista principal de proyectos
2	HU2	Editar una cuenta de usuario	Actualización de la información de un usuario definido	Se actualizó la información del usuario y se muestra los cambios en la lista principal de usuarios
3	HU3	Acceder a la aplicación con usuario y previamente definidos	Otorgar acceso al usuario registrado, mostrando los datos del usuario y permitiendo el cierre de sesión	Se le concedió acceso a la aplicación
4	HU3	Mostrar menú principal	Se muestra el menú con las opciones correspondientes al rol usuario	Se mostró el menú con las opciones correspondientes al rol usuario

Tabla 3.2 Pruebas de aceptación del módulo de usuarios.

- **Módulo de proyectos:** Las pruebas en este módulo se basan en la creación, edición y eliminación de los proyectos, asignar los usuarios. La Tabla 3.3 describe los casos de prueba y resultados del módulo de proyectos.

N° Caso de Prueba	Historias de Usuario	Caso de Prueba	Resultado Esperado	Resultado Obtenido
4	HU4	Crear un proyecto	Creación de un nuevo proyecto	Se guardó la información del proyecto y se muestra en la lista principal de proyectos
5	HU5	Editar un proyecto	Actualización de la información de un proyecto existente	Se actualizó la información del proyecto y se muestra en la lista principal de proyectos
6	HU6	Eliminar un proyecto	Borrar el proyecto y toda la información asociada	Se borró el proyecto y toda la información asociada
7	HU7	Asignar usuarios al proyecto	Se agrega el usuario a la lista de usuarios autorizados	Se mostró el usuario, en la lista de usuarios autorizados
8	HU7	Remover usuarios del proyecto	Se elimina el usuarios de la lista de usuarios autorizados y se muestra en la lista de usuarios disponibles	Se eliminó de la lista de usuarios autorizados y se mostró en la lista de usuarios disponibles
9	HU8	Mostrar lista proyectos asignados	Al acceder a la aplicación, se muestra la lista desplegable con los proyectos asignados	Se mostró lista desplegable con los proyectos asignados

Tabla 3.3 Pruebas de aceptación del módulo de proyectos.

3.2.2 Segunda Iteración: Gestión de Historias, Tareas y Casos de Prueba

Para esta iteración se contempló el desarrollo de las funcionalidades que permiten crear, editar, y eliminar las historias de usuario con sus tareas y casos de prueba. Estas funcionalidades son unas de las más importantes dentro de la aplicación, porque permiten capturar, organizar y priorizar los requerimientos de un proyecto.

3.2.2.1 Planificación

Las historias de usuario a abordar se pueden ver en la Tabla 3.4.

N° de Iteración	2
Descripción	Desarrollo de los módulos para la gestión de historias de usuario, tareas y casos de prueba
Fecha Inicio – Fin	28/05/2012 - 02/07/2012
Historias de Usuario	HU9: Creación de historias HU10: Editar historias HU11: Eliminar historias HU12: Copiar historias HU14: Agregar tareas a una historia HU15: Editar tareas HU16: Eliminar tareas HU17: Agregar casos de prueba a una historia HU18: Editar caso de prueba HU19: Eliminar casos de prueba HU20: Agregar resultado al caso de prueba
Tiempo Estimado	35 días

Tabla 3.4 Planificación de la segunda iteración.

3.2.2.2 Diseño

En esta sección se describen los modelos de datos entidad relación (E/R) y diagramas de clases UML para dar solución a la gestión de historias de usuario, tareas y casos de prueba, a través del uso de los patrones y framework elegidos.

Modelo de Datos

Para llevar a cabo la presente fase, se toma en cuenta lo conversado con el cliente referente a la forma en que se puede organizar la información de los proyectos. En la Figura 3.31 se muestra el modelo datos E/R que se desarrolla para la presente iteración en el que se puede destacar la incorporación de las tablas *historia_usuario*, *tarea*, *caso_prueba*, y *resultado_caso_prueba*. Dentro de las reglas de negocio se destacan un proyecto puede tener varias historias de usuario, una historia de usuario debe estar asociada a un proyecto, una historia de usuario puede tener varias tareas y varios casos de prueba, una tarea o un caso de prueba deben estar asociados a una historia de usuario, un caso de prueba puede tener varios resultados.

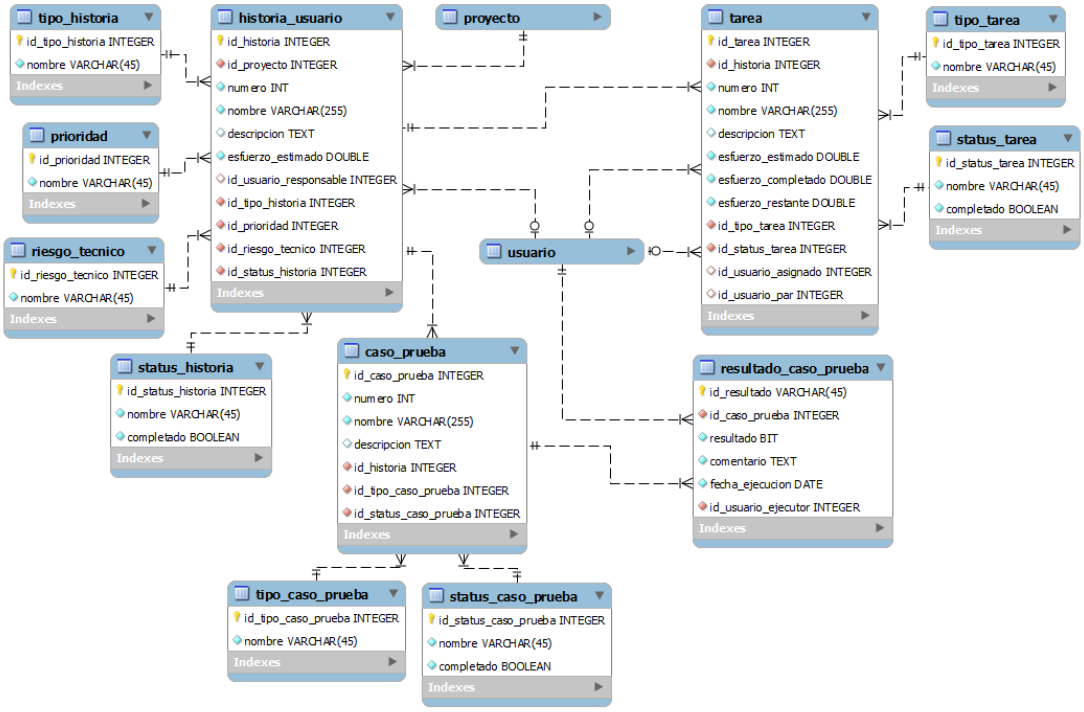


Figura 3.31 Modelo de datos E/R de la segunda iteración.

Diagrama de Clases

En primer lugar se diseñan los objetos del dominio que modelan la información del proyecto: historias, tipo de historia, prioridad, riesgo técnico, status de la historia, tareas, tipo de tarea, status de la tarea, casos de prueba, tipo de prueba, status de la prueba, resultados de la prueba. Como son *bean* tienen métodos *set* y *get*.

La Figura 3.32 muestra el diagrama de clases UML de los objetos del dominio que modelan las historias de usuario.

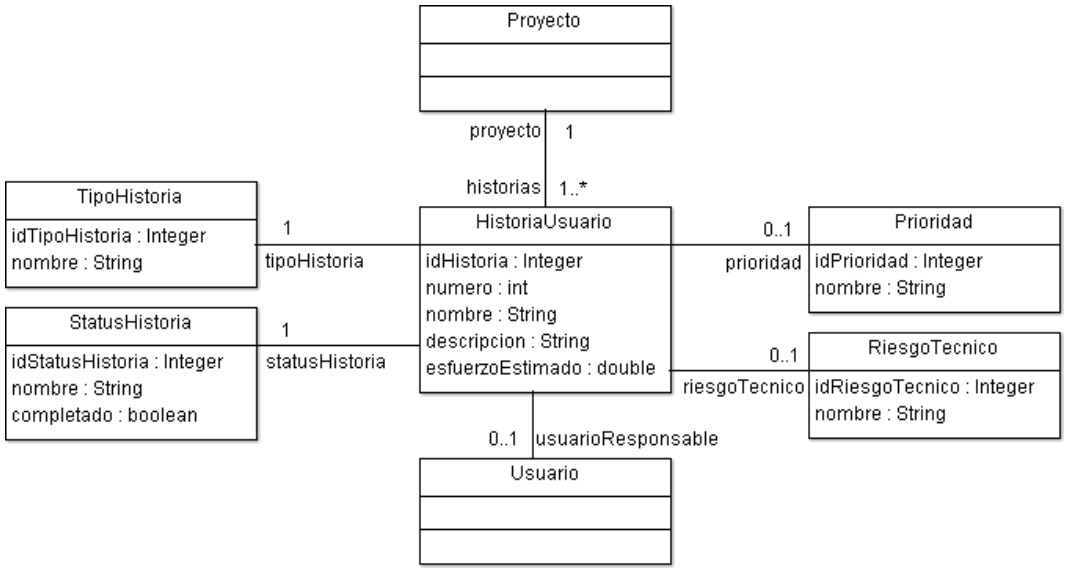


Figura 3.32 Diagrama de clases de los objetos del dominio que modelan las historias de usuario.

A continuación de describen cada uno de las clases de los objetos del dominio:

- *HistoriaUsuario*: Esta clase representa una descripción de las funciones o característica de un proyecto de desde la perspectiva del usuario. Los proyectos estas compuesto por varias historias de usuario. Un usuario puede ser el propietario o responsable varias historias de usuario.
- *TipoHistoria*: Esta clase corresponde a la categoría a la que puede pertenecer una historia de usuario, por ejemplo: Nueva Función, Mejora o un Defecto.
- *Prioridad*: Esta clase corresponde la importancia que puede tener una historia de usuario para el cliente, por ejemplo: Urgente, Alta, Media o Baja.

- *RiesgoTecnico*: Esta clase corresponde el riesgo técnico y dificultad de una historia, determinada por el equipo de desarrollo, por ejemplo: Alta, Media, Bajo.
- *StatusHistoria*: Esta clase corresponde el estado de la historia, si esta en activa o completada.

La Figura 3.33 muestra el diagrama de clases UML de los objetos del dominio que modelan las tareas.

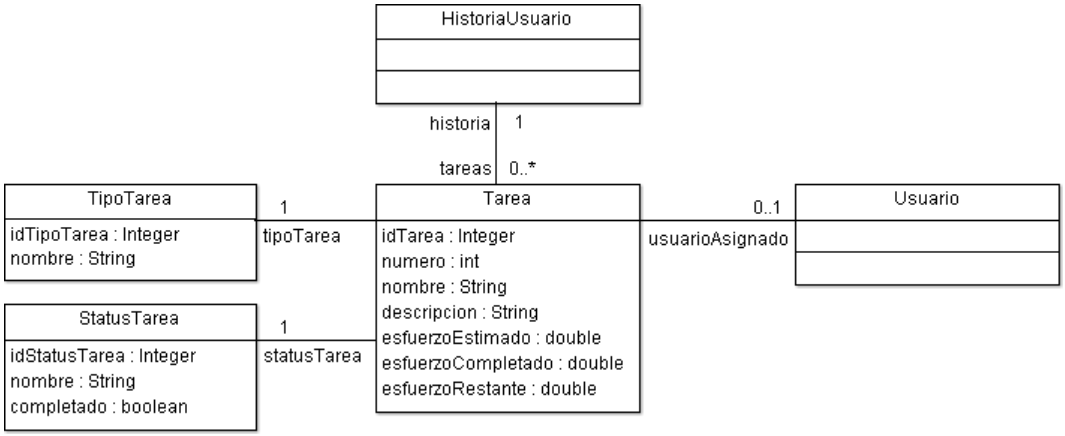


Figura 3.33 Diagrama de clases de los objetos del dominio que modelan las tareas.

A continuación de describen cada uno de las clases de los objetos del dominio:

- *Tarea*: Esta clase representa las actividades realzadas para implementar una historia de usuario. Una historia de usuario se divide en un conjunto de tareas concretas. Una tarea puede ser asignado a un usuario.
- *TipoTarea*: Esta clase corresponde a la categoría a la que puede pertenecer una tarea, por ejemplo: Planificación, Diseño, Desarrollo o Prueba.
- *StatusTarea*: Esta clase corresponde el estado de la tarea, si esta en activa o completada.

La Figura 3.34 muestra el diagrama de clases UML de los objetos del dominio que modelan los casos de prueba.

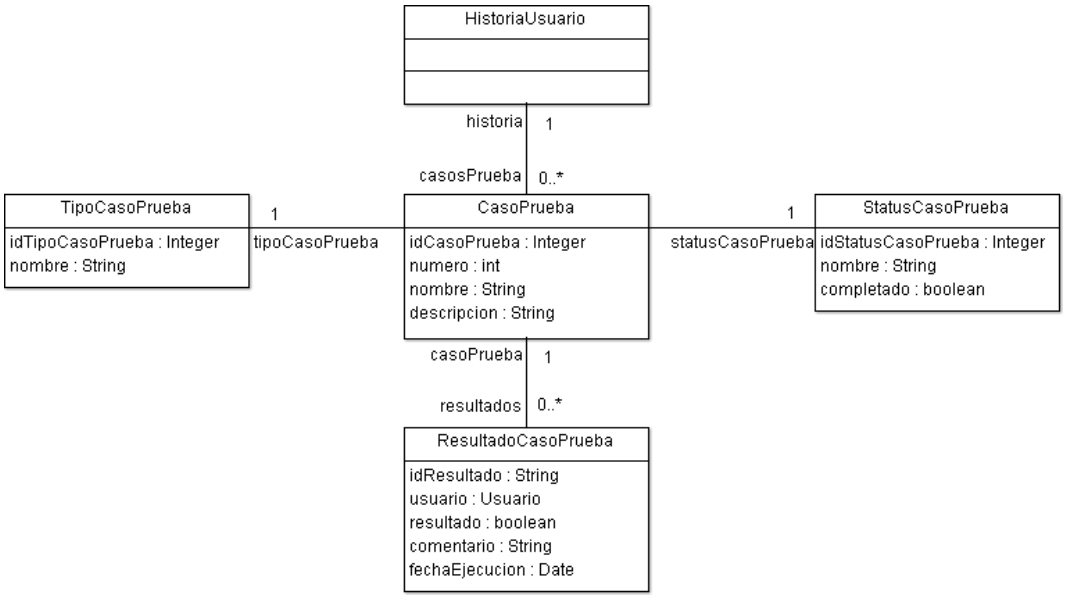


Figura 3.34 Diagrama de clases de los objetos dominio que modelan los casos de prueba.

A continuación de describen cada uno de las clases de los objetos del dominio:

- *CasoPrueba*: Esta clase representa los casos de pruebas para comprobar el comportamiento o funcionamiento de las historias de usuario. Una historia de usuario puede tener definidas varios casos de prueba.
- *TipoCasoPrueba*: Esta clase corresponde a la categoría a la que puede pertenecer un caso de prueba, por ejemplo: Aceptación, Unitaria, Usabilidad o Rendimiento.
- *StatusCasoPrueba*: Esta clase corresponde el estado del caso de prueba, si esta en activo o completado.
- *ResustadoCasoPrueba*: Esta clase representa el resultado obtenido de la ejecución de un caso de prueba.

Se definen las clases que implementan el patrón DAO específicos para de acceder a los datos de las historias de usuario, tareas y casos de prueba. El diagrama de clases UML se muestra en la Figura 3.35.

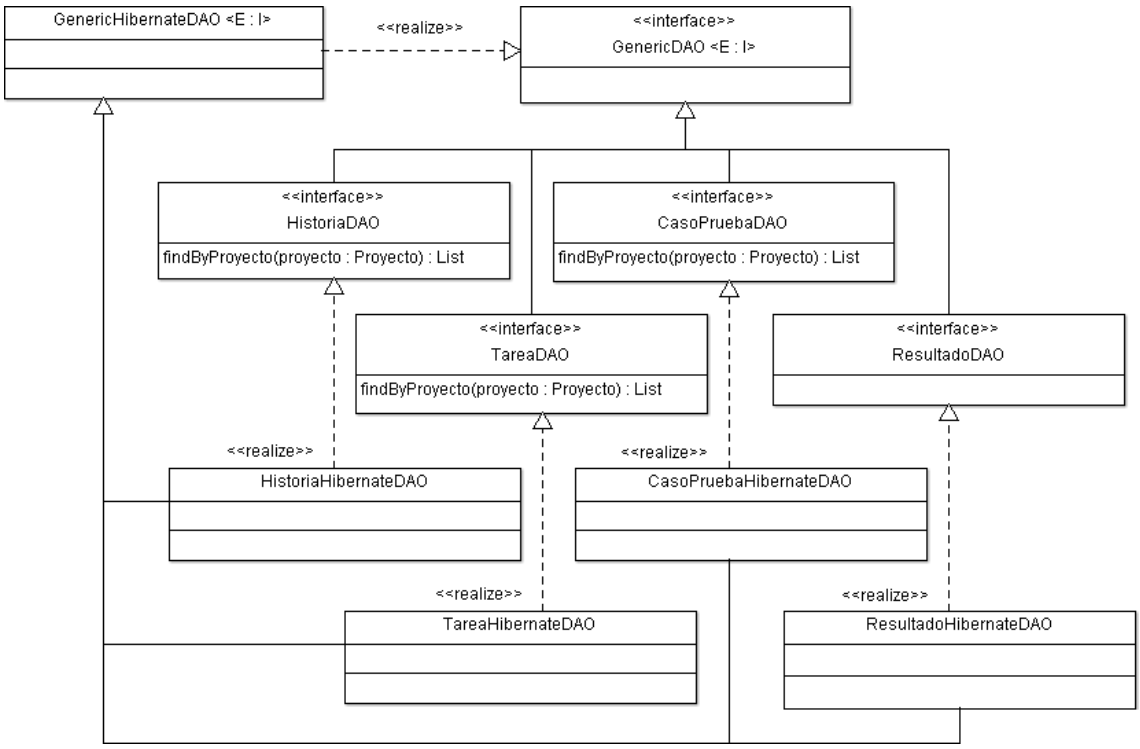


Figura 3.35 Diagrama de clases DAO específicas para acceder a las historias de usuario, tareas y casos de prueba.

Para ocultar detalles de implementación y dar soporte a la gestión de las historias de usuario, con sus tareas y casos de pruebas de un proyecto, se añaden las siguientes clases:

- *HistoriaDAO*: Interfaz con los métodos necesarios que debe implementar un DAO para persistir las historias de usuario, proporcionando como argumento la clase persistente *HistoriaUsuario*. Proporciona el métodos adicional:
    - findByProyecto: Retorna las historias de usuario de un proyecto dado.
- HistoriaHibernateDAO: Extiende de la clase *GenericHibernateDAO*, proporcionando como argumento la clase persistente *HistoriaUsuario.*
- *TareaDAO*: Interfaz con los métodos necesarios que debe implementar un DAO para persistir los tareas, proporcionando como argumento la clase persistente *Tarea*.

- *TareaHibernateDAO*: Extiende de la clase *GenericHibernateDAO*, proporcionando como argumento la clase persistente *Tarea.*
- *CasoPruebaDAO*: Interfaz con los métodos necesarios que debe implementar un DAO para persistir los casos de prueba, proporcionando como argumento la clase persistente *CasoPrueba*.
- *CasoPruebaHibernateDAO*: Extiende de la clase *GenericHibernateDAO*, proporcionando como argumento la clase persistente *CasosPrueba.*
- *ResultadoDAO*: Interfaz con los métodos necesarios que debe implementar un DAO para persistir los resultados de los casos de prueba, proporcionando como argumentos la clase persistente *ResultadoCasoPrueba.*
- *ResultadoHibernateDAO*: Extiende de la clase *GenericHibernateDAO*, proporcionando como argumento la clase persistente *ResultadoCasoPrueba.*

Se definen las fachadas *HistoriaManagerFacade*, *TareaManagerFacade* y *CasoPruebaManagerFacade* correspondientes a cada uno de los módulos como interfaz de la capa modelo hacia capas superiores ocultando los detalles de implementación de la capa. Cada clase implementa una clase interfaz *ManagerFacade* propia.

La Figura 3.36 muestra el diagrama de clases con la fachada para ocultar los detalles de implementación del módulo tareas. La fachada contiene los DAO responsables para la persistencia de la clase *Tarea* y sus subclases.

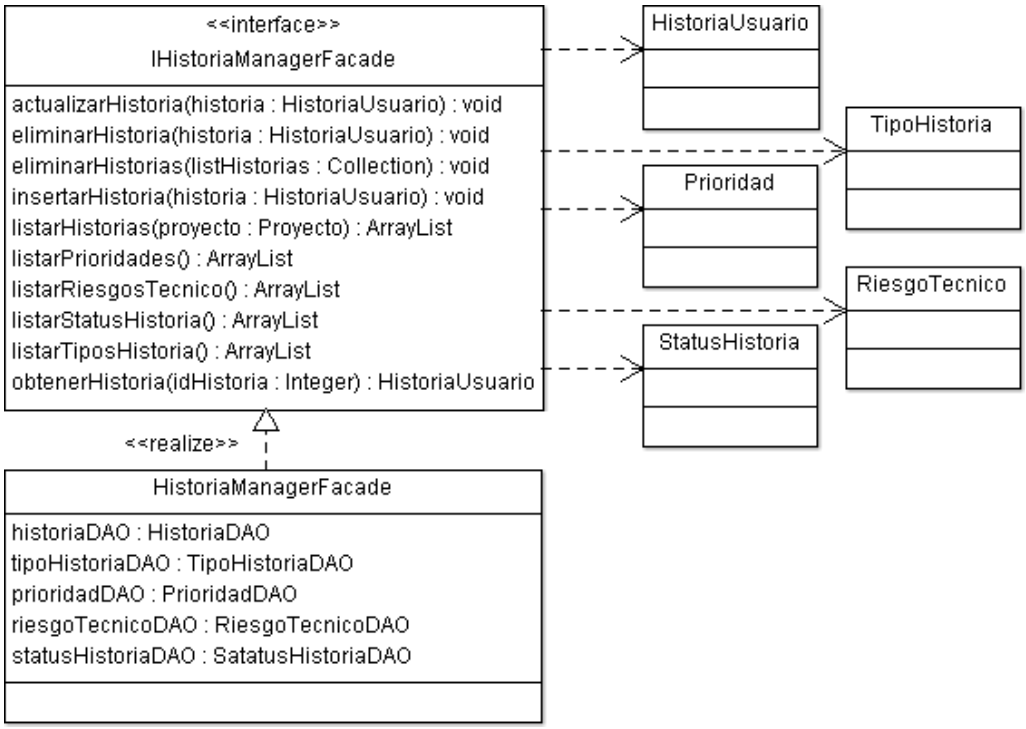


Figura 3.36 Diagrama de clases fachadas del módulo de historias.

La Figura 3.37 muestra el diagrama de clases con la fachada para ocultar los detalles de implementación del módulo tareas. La fachada contienen los DAO responsables para la persistencia de la clase *Tarea* y sus subclases.

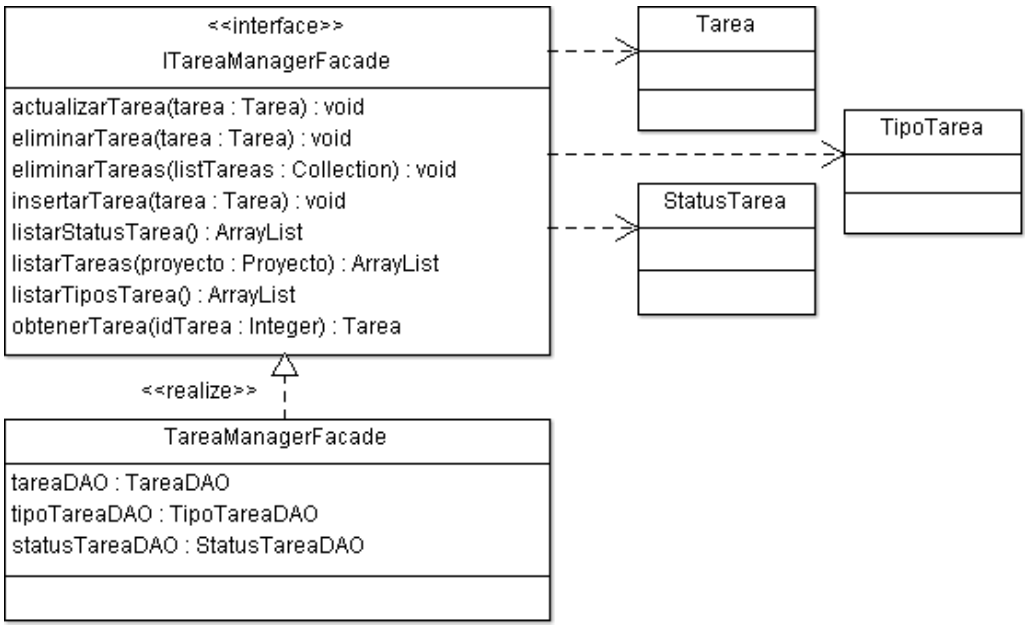


Figura 3.37 Diagrama de clases fachadas del módulo de tareas.

La Figura 3.38 muestra el diagrama de clases con la fachada para ocultar los detalles de implementación del módulo casos de prueba. La fachada contiene los DAO responsables para la persistencia de la clase *CasoPrueba* y sus subclases.

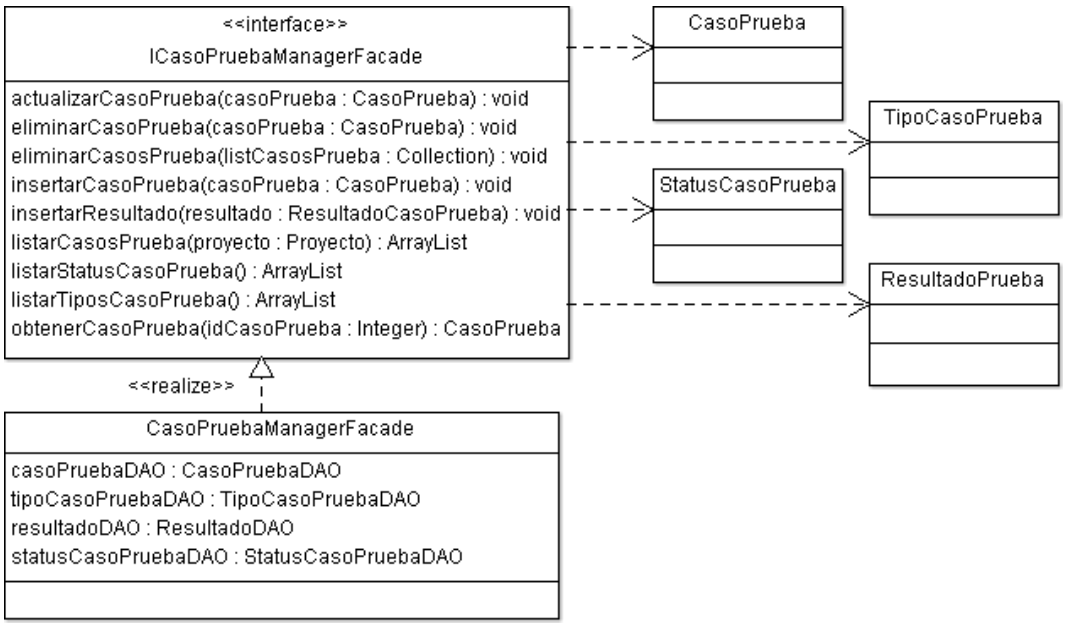


Figura 3.38 Diagrama de clases fachadas del módulo de tareas.

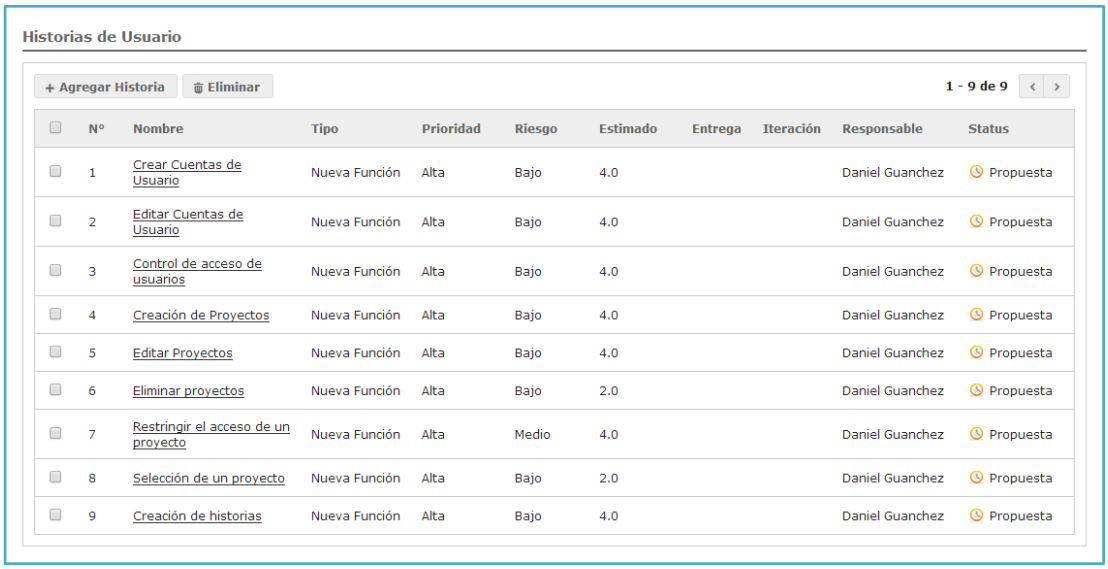
En la capa controlador se añaden las acciones de *Struts*, *HistoriasAction* e *TareasAction*, y *CasosPruebaAction*, con los métodos que darán soporte a la nueva funcionalidad (listar, visualizar, agregar, editar, eliminar), que hará de intermediario entre la vista y el modelo, con su correspondiente configuración.

Dentro de las vistas que se desarrollaron para esta iteración tenemos las siguientes:

- Mostrar la lista principal de historias de usuario del proyecto
- Definición y consultar de historias de usuario
- Definición y consulta de tareas y casos de pruebas a las historias
- Mostrar las listas tareas y casos de prueba de una historia
- Resultado de los casos de prueba

A continuación se describen las vistas de las funcionales para gestionar las historias de usuario con sus tareas y casos de prueba.

Al seleccionar la opción “Historias” del menú principal, se muestra el listado de las historias de usuarios del proyecto, en el que se puede seleccionar la historia que se desea visualizar, editar, eliminar, o se agregar una nueva. La Figura 3.39 muestra la lista de las historias de usuario creadas para un proyecto.



Nº	Nombre	Tipo	Prioridad	Riesgo	Estimado	Entrega	Iteración	Responsable	Status
1	Crear Cuentas de Usuario	Nueva Función	Alta	Bajo	4.0			Daniel Guanchez	Propuesta
2	Editar Cuentas de Usuario	Nueva Función	Alta	Bajo	4.0			Daniel Guanchez	Propuesta
3	Control de acceso de usuarios	Nueva Función	Alta	Bajo	4.0			Daniel Guanchez	Propuesta
4	Creación de Proyectos	Nueva Función	Alta	Bajo	4.0			Daniel Guanchez	Propuesta
5	Editar Proyectos	Nueva Función	Alta	Bajo	4.0			Daniel Guanchez	Propuesta
6	Eliminar proyectos	Nueva Función	Alta	Bajo	2.0			Daniel Guanchez	Propuesta
7	Restringir el acceso de un proyecto	Nueva Función	Alta	Medio	4.0			Daniel Guanchez	Propuesta
8	Selección de un proyecto	Nueva Función	Alta	Bajo	2.0			Daniel Guanchez	Propuesta
9	Creación de historias	Nueva Función	Alta	Bajo	4.0			Daniel Guanchez	Propuesta

Figura 3.39 Lista de historias de usuario del proyecto.

El nombre de la historia de usuario es un enlace que muestra una pantalla con el detalle de la historia. Allí se podrá visualizar los datos básicos con las operaciones correspondientes, las tareas y casos de pruebas creadas para la historia. La Figura 3.40 muestra la pantalla de consulta de los detalles de una historia de usuario.

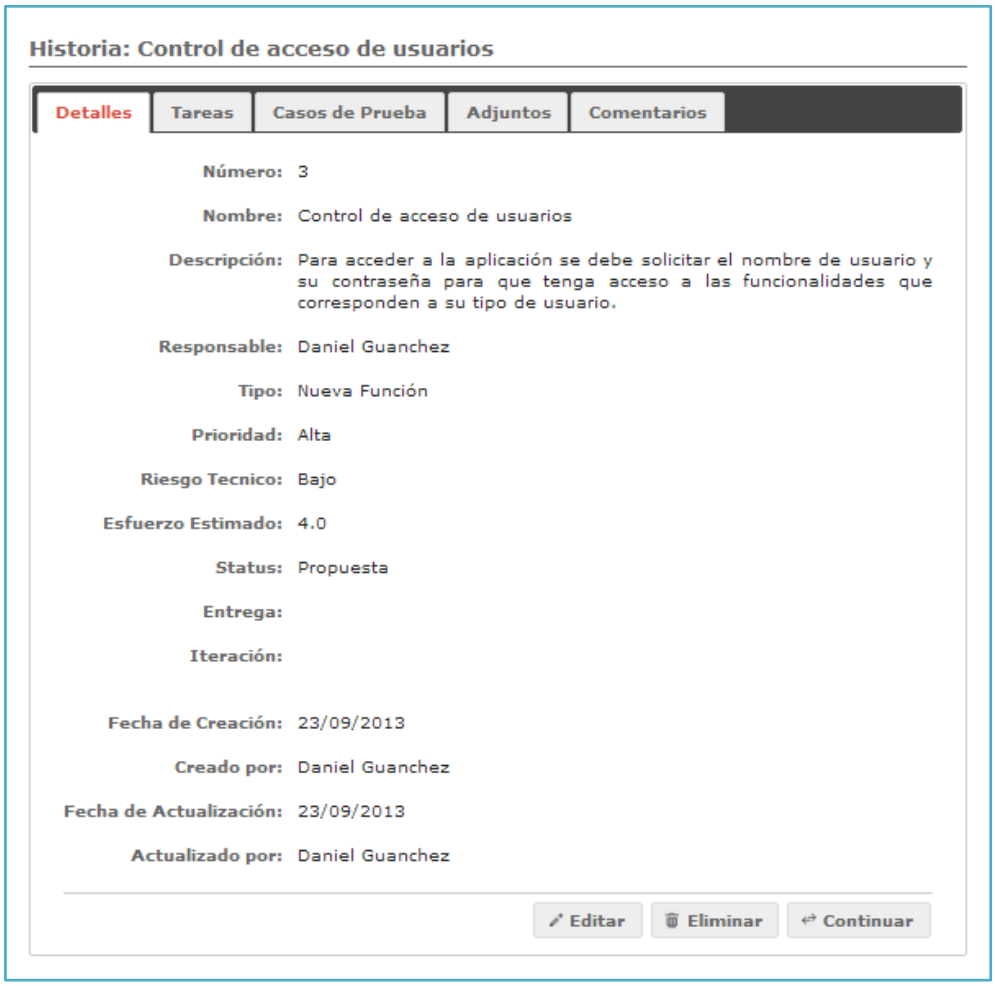


Figura 3.40 Visualizar los detalles de una historia.

La Figura 3.41 muestra el formulario que se utiliza tanto para crear o editar una historia de usuario.

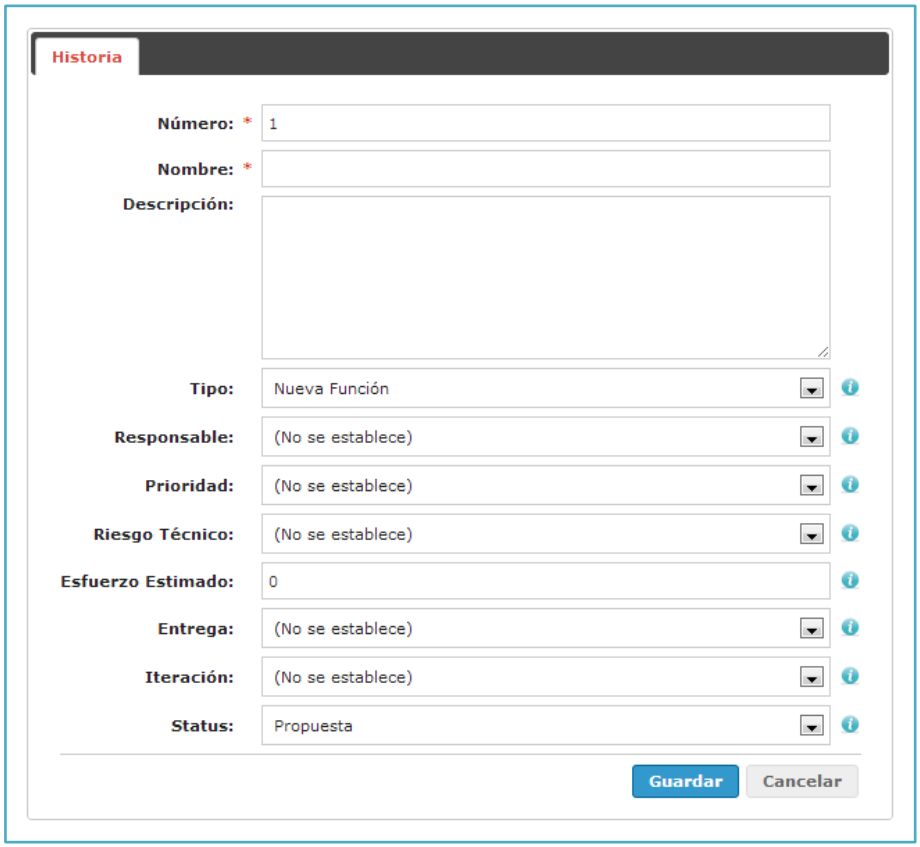


Figura 3.41 Formulario para agregar o editar una historia de usuario.

El proceso de definición de las tareas comienza a partir de la creación de una historia de usuario. Al visualizar la historia se muestra una pestaña con el listado de las tareas de la historia, en el que se puede seleccionar la tarea que se desea visualizar, editar, eliminar, o se puede agregar una nueva. La Figura 3.42 muestra la lista de las tareas creadas para la historia de usuario seleccionada.

Figura 3.42 Lista de tareas de una historia.

Al seleccionar la opción "Tareas" del menú principal se muestra el listado con todas las tareas del proyecto, desde allí también se puede editar, eliminar, o agregar una nueva tarea. La Figura 3.43 muestra la lista de las tareas del proyecto.

Figura 3.43 Lista de tareas de un proyecto.

La Figura 3.44 muestra el formulario que se utiliza tanto para crear o editar una tarea.

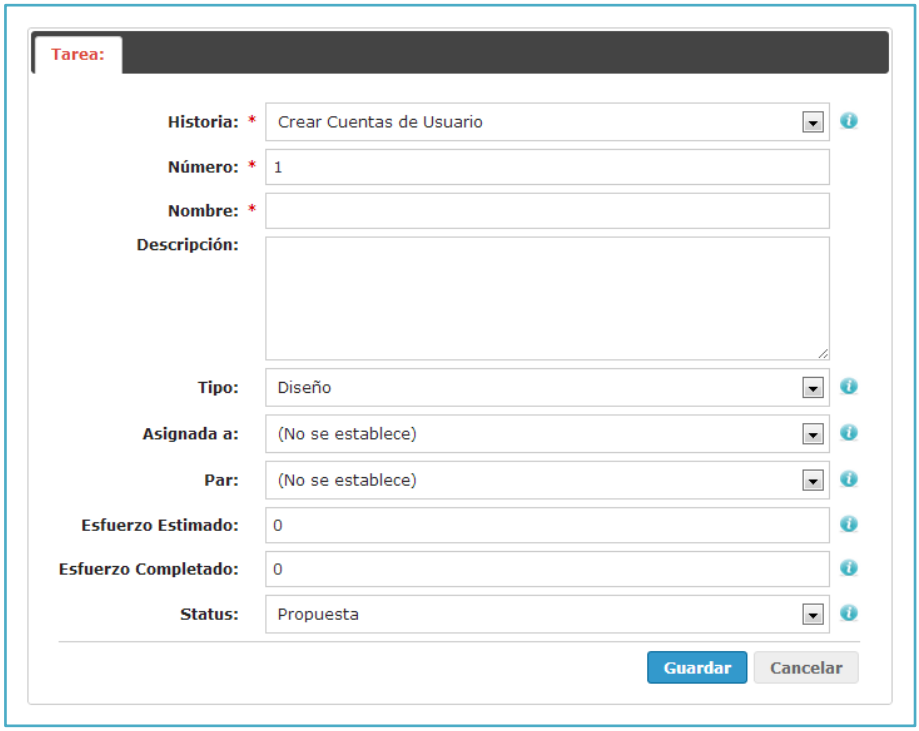


Figura 3.44 Formulario para agregar o editar una tarea.

Al igual que en las tareas, el proceso de definición de los casos de prueba comienza a partir de la creación de una historia de usuario. Al visualizar la historia se muestra una pestaña con el listado de los casos de prueba de la historia, en el que se puede seleccionar el caso de prueba que se desea visualizar, editar, eliminar, o se puede agregar uno nuevo. La Figura 3.45 muestra la lista de los casos de prueba creadas para la historia de usuario seleccionada.

Figura 3.45 Lista de casos de prueba de una historia.

Al seleccionar la opción "Casos de Prueba" del menú principal se muestra el listado con todas los casos de prueba del proyecto, desde allí también se puede editar, eliminar, o agregar un nuevo caso de prueba. La Figura 3.46 muestra la lista de los casos de prueba del proyecto.

Figura 3.46 Lista de casos de prueba de un proyecto.

La Figura 3.47 muestra el formulario que se utiliza tanto para crear o editar un caso de prueba.

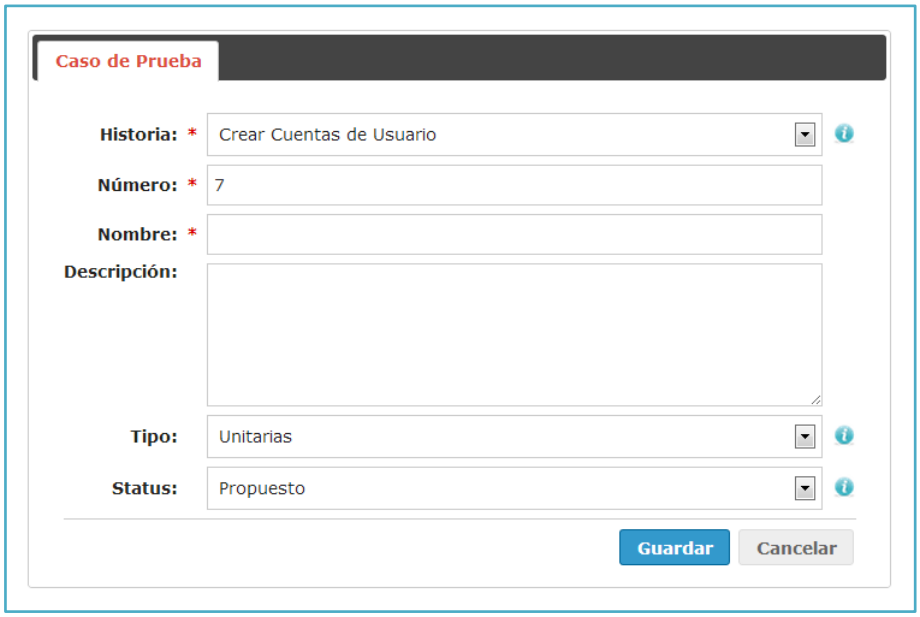


Figura 3.47 Formulario para agregar o editar un caso de prueba.

Una vez creado un caso de prueba se pueden agregar resultados desde la consulta del caso de prueba. Al visualizar un caso de prueba se muestra una pestaña con el listado cronológico de los resultados de las pruebas. La Figura 3.48 muestra la lista de resultados de un caso de prueba.

Figura 3.48 Listado de resultados de un caso de prueba.

3.2.2.4 Pruebas

Las pruebas realizadas para la verificación de todo lo desarrollado en la presente iteración, consistieron en la utilización y recorrido las funcionalidades con el propósito de corroborar que fuesen acordes a lo solicitado en las historias asociadas al módulos de historias, modulo, el módulo de tares y el módulo de casos de pruebas. Para la realización de las pruebas se tomó como muestra un grupo de estudiantes de la Escuela de Computación, que realizan trabajos de investigación utilizando la adaptación de XP como método de desarrollo. Las pruebas fueron documentadas en bitácoras separadas por módulos:

- **Módulo de historias:** Las pruebas en este módulo se basan en la creación, edición y eliminación de historias dentro de un proyecto. La Tabla 3.5 describe los casos de prueba y resultados del módulo de historias.

N° Caso de Prueba	Historias de Usuario	Caso de Prueba	Resultado Esperado	Resultado Obtenido
10	HU9	Crear una historia	Creación de una nueva historia	Se guardó la información de la historia y se muestra en la lista principal de historias del proyecto
11	HU10	Editar una historia	Modificar lo información de una historia existente	Se actualizó la información de la historia y se muestra en la lista principal de historias del proyecto
12	H11	Eliminar historias	Borrar las historia y toda la información asociada	Se borraron las historias y toda la información asociada
13	H12	Crear una copia de una historia	Creación de una historia con los atributos de la historia origen	Se creó la nueva historia con los atributos de la historia origen

Tabla 3.5 Pruebas de aceptación del módulo de historias.

- **Módulo de tareas:** Las pruebas en este módulo se basan en la creación, edición y eliminación de tareas a las historias de usuarios de un proyecto. Asignar y remover usuarios para restringir el acceso. La Tabla 3.6 describe los casos de prueba y resultados del módulo de tareas.

N° Caso de Prueba	Historias de Usuario	Caso de Prueba	Resultado Esperado	Resultado Obtenido
14	HU14	Agregar tarea a una historia	Creación de una nueva tarea	Se guardó la información de la tarea y se muestra en la lista principal de tareas de la historia
15	HU15	Editar una tarea	Modificar lo información de una tarea existente	Se actualizó la información de la tarea y se muestra en la lista principal de tareas de la historia
16	HU16	Eliminar tareas	Borrar las tareas y toda la información asociada	Se borraron las tareas y toda la información asociada

Tabla 3.6 Pruebas de aceptación del módulo de tareas.

- **Módulo de casos de prueba:** Las pruebas en este módulo se basan en la creación, edición y eliminación de casos de prueba en las historias de usuarios de un proyecto. La Tabla 3.7 describe los casos de prueba y resultados del módulo de casos de prueba.

N° Caso de Prueba	Historias de Usuario	Caso de Prueba	Resultado Esperado	Resultado Obtenido
17	HU17	Agregar caso de prueba a una historia	Creación de un nuevo caso de prueba	Se guardó la información del caso de prueba se muestra en la lista principal de casos de prueba de la historia
18	HU18	Editar un caso de prueba.	Modificar lo información de un caso de prueba existente	Se actualizó la información del caso de prueba y se muestra en la lista principal de casos de prueba de la historia
19	HU19	Eliminar casos de prueba	Borrar los casos de prueba y toda la información asociada	Se borraron los casos de prueba y toda la información asociada
20	HU20	Agregar resultados a un caso de prueba	Creación de un nuevo resultado en el caso de prueba	Se guardó la información del resultado y se mostró en la lista de resultados del caso de prueba. Se mostró el último resultado agregado en la lista de casos de prueba

Tabla 3.7 Pruebas de aceptación del módulo de casos de prueba.

3.2.3 Tercera Iteración: Gestión de Archivos Adjuntos y Comentarios

En esta tercera iteración se completará la organización de la información del proyecto, añadiendo las funcionalidades de adjuntar archivos y agregar comentarios en las historias de usuarios, tareas y casos de prueba.

3.2.3.1 Planificación

Las historias de usuario a abordar se pueden ver en la Tabla 3.8.

N° de Iteración	3
Descripción	Desarrollar las funcionalidades que permitan adjuntar archivos y agregar comentarios en las historias de usuarios, tareas y casos de prueba
Fecha Inicio – Fin	02/07/2012 – 16/07/2012
Historias de Usuario	HU21: Adjuntar archivos a las historias, tareas y casos de prueba HU22: Editar los archivos adjuntos HU23: El usuario puede eliminar archivos adjuntos en las historias, tareas y casos de prueba HU24: Agregar Comentarios en las historias, tareas HU25: Eliminar los comentarios
Tiempo Estimado	14 días

Tabla 3.8 Planificación de la tercera iteración.

3.2.3.2 Diseño

En esta sección se describen los modelos de datos entidad relación (E/R) y diagramas de clases UML, para dar solución a la gestión de adjuntos y comentarios, a través del uso de los patrones y *framework* elegidos.

Modelo de Datos

En la Figura 3.49 se muestra el modelo datos E/R que se desarrolla para la presente iteración donde destaca la incorporación de las tablas *adjunto* y *comentario*, y sus relaciones con las tablas previamente diseñadas en las iteraciones anteriores. Dentro de las reglas de negocio se destacan que las historias, tareas y casos de prueba pueden tener varios archivos adjuntos. Las historias de usuario y tareas pueden tener varios comentarios, cada comentario debe estar asociado a un usuario.

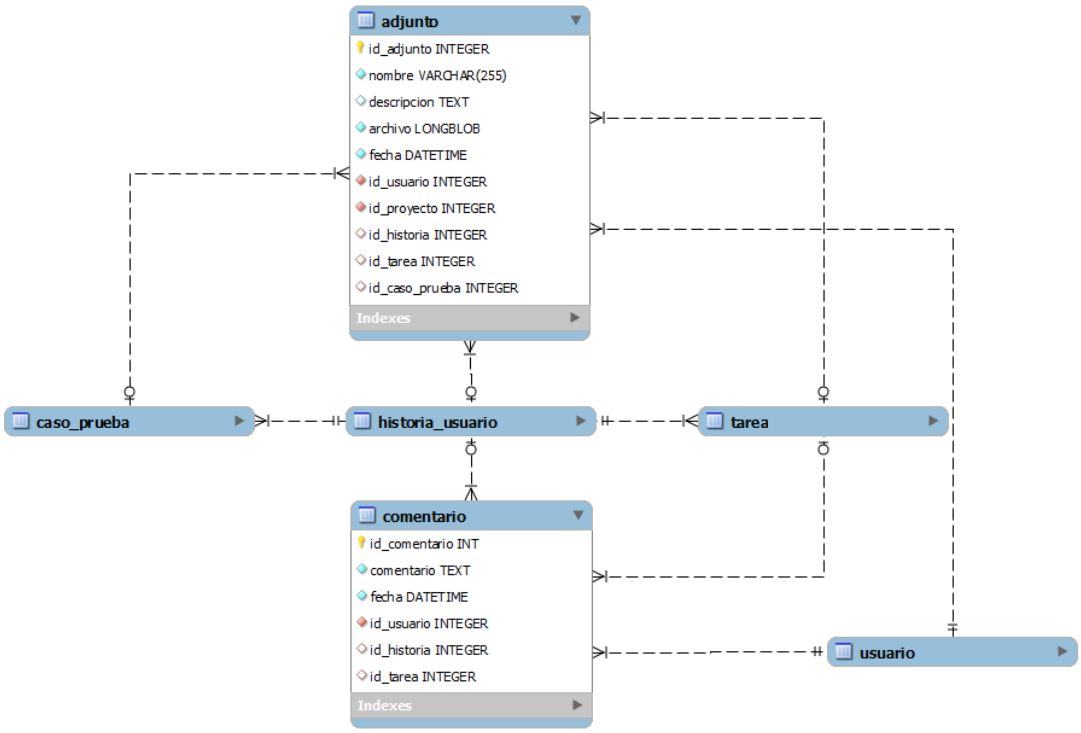


Figura 3.49 Modelo de datos E/R de la tercera iteración.

Diagrama de Clases

Para comenzar se actualiza el diagrama de clases UML que se puede ver en la Figura 3.50, con los objetos del dominio descritos en las historias de usuario, necesarios para incluir los archivos adjuntos y comentarios.

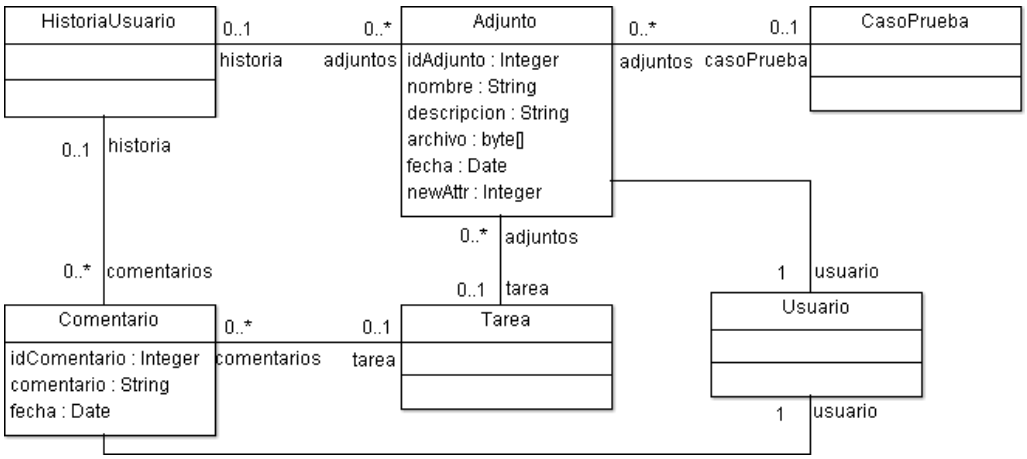


Figura 3.50 Diagrama de clases de los objetos del dominio que modelan los adjuntos y comentarios.

A continuación de describen cada uno de las clases de los objetos del dominio:

- *Adjunto*: Esta clase representa los archivos que se adjuntan los usuarios sobre una historia, tarea o casa de prueba.
- *Comentario*: Esta clase representa las notas o comentarios de los usuarios sobre una historia de usuario o tarea.

Se definen las clases que implementan el patrón DAO específico y se realiza el diagrama de clases UML que se puede ver en la Figura 3.51.

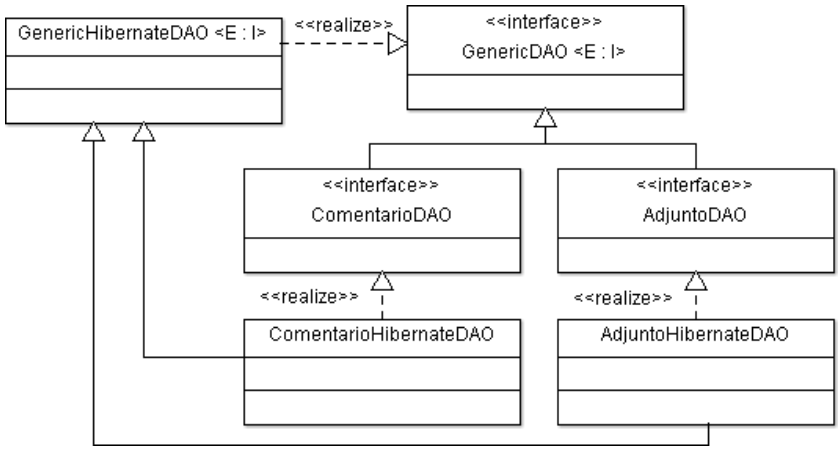


Figura 3.51 Diagrama de clases DAO específicas para acceder a los adjuntos y comentarios.

Para dar soporte se añaden las siguientes clases:

- *AdjuntoDAO*: Interfaz con los métodos necesarios que debe implementar un DAO para persistir los archivos adjuntos, proporcionando como argumento la clase persistente *Adjunto*.
- *AdjuntoHibernateDAO*: Extiende de la clase *GenericHibernateDAO*, proporcionando como argumento la clase persistente *Adjunto*.
- *ComentarioDAO*: Interfaz con los métodos necesarios que debe implementar un DAO para persistir los comentaros, proporcionando como argumento la clase persistente *Comentario*.
- *ComentarioHibernateDAO*: Extiende de la clase *GenericHibernateDAO*, proporcionando como argumento la clase persistente *Comentario*.

Al igual que en los módulos anteriores se crean las fachada para el módulo adjuntos y comentarios como interfaz de la capa modelo hacia capas superiores ocultando los detalles de implementación de la capa modelo. La Figura 3.52 muestra el diagrama de clases UML con las fachadas.

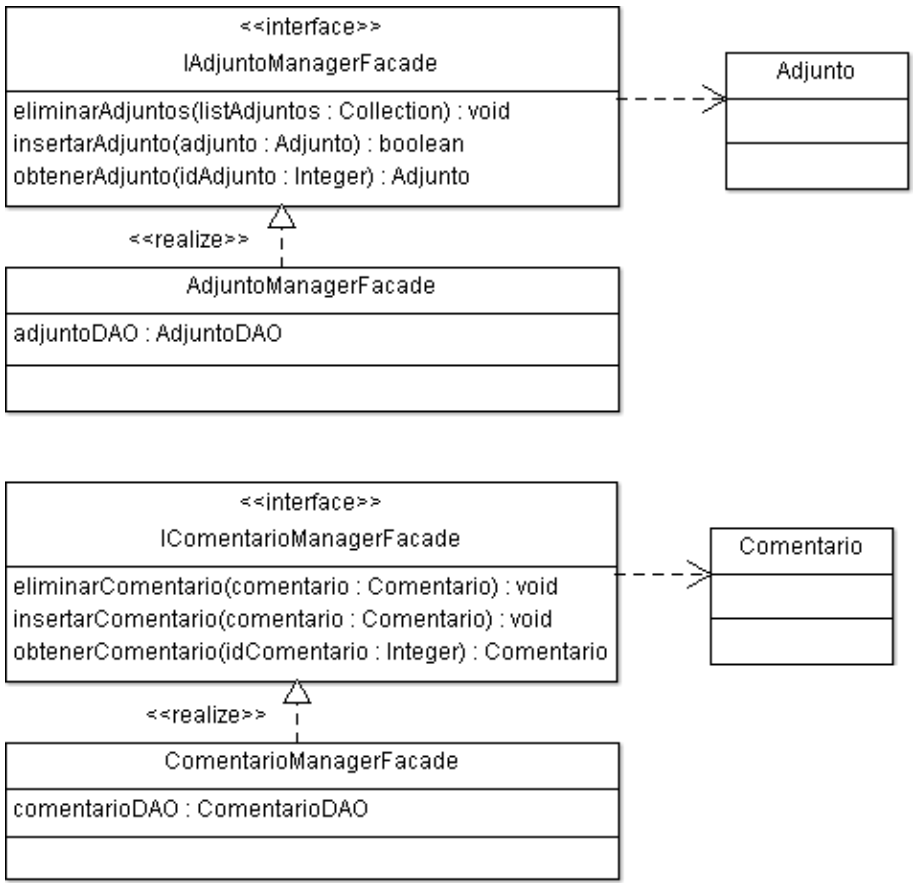


Figura 3.52 Diagrama de clases fachadas del módulo de archivos adjuntos y comentarios.

3.2.2.3 Codificación

En la capa controlador se añadirá las acciones de *Struts*, AdjuntoAction y ComentarioAction, que hará de intermediario entre la vista y el modelo, con su correspondiente configuración, y en la capa vista la página JSP para visualizar los adjuntos y los comentarios, así como los formularios para su creación.

A continuación se describen las pantallas de las funcionales para gestionar los archivos adjuntos y agregar comentarios.

Al visualizar una historia de usuario, una tarea o un caso de prueba, se muestra una pestaña con el listado de archivos adjuntos, en el que se puede seleccionar el archivo que se desea, descargar, eliminar, o se puede agregar uno nuevo. La Figura 3.53 muestra la pestaña para gestionar los archivos adjuntos.

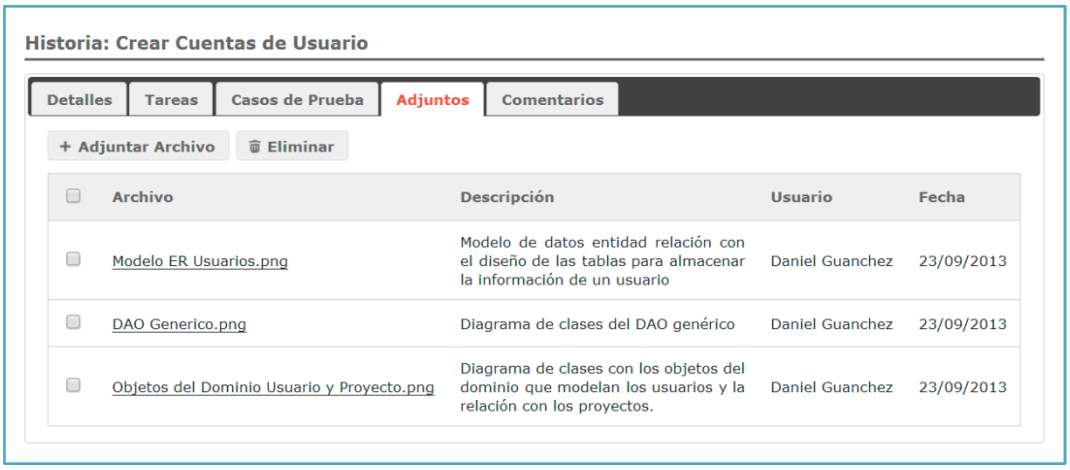


Figura 3.53 Lista de archivos adjuntos para una historia de usuario.

La Figura 3.54 muestra el formulario que se utiliza para adjuntar un archivo.

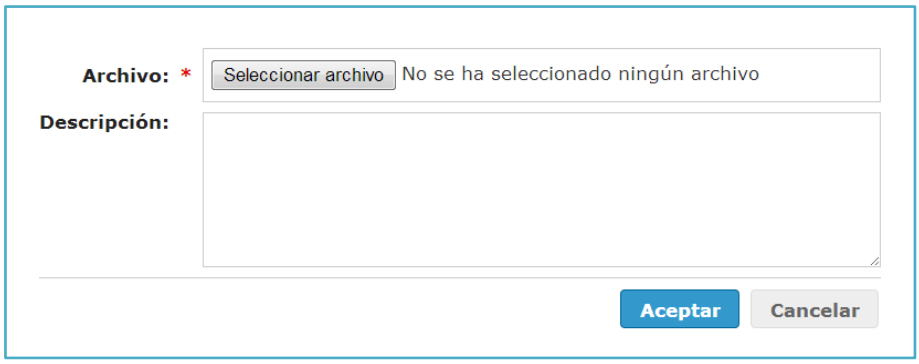


Figura 3.54 Formulario para adjuntar un archivo.

También al visualizar una historia de usuario o una tarea, se muestra una pestaña donde el usuario puede agregar comentarios, para registrar la información relacionada con las notas y comentarios de los clientes. Los comentarios se muestran en orden cronológico mostrando el usuario y la fecha y hora en que se agregó. La Figura 3.55 muestra la pestaña para registrar los comentarios sobre una historia o tarea del proyecto.

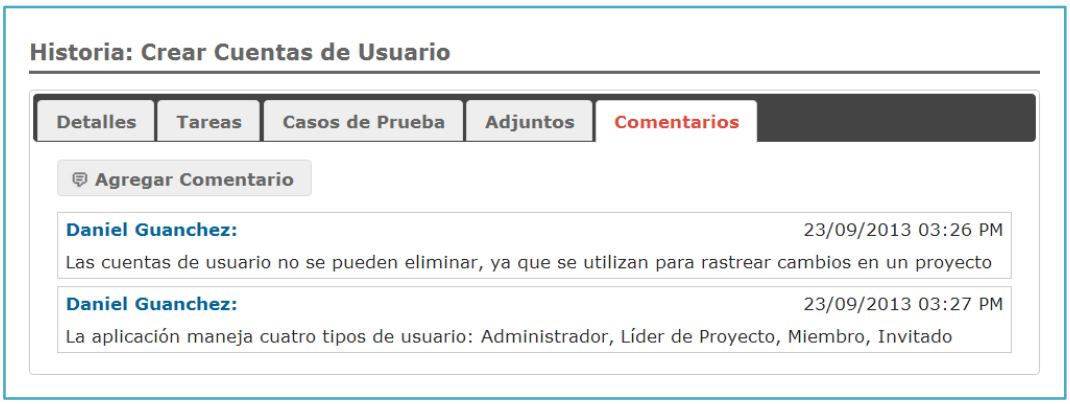


Figura 3.55 Lista de comentarios para una historia de usuario.

La Figura 3.55 muestra el formulario que se utiliza para agregar comentarios.

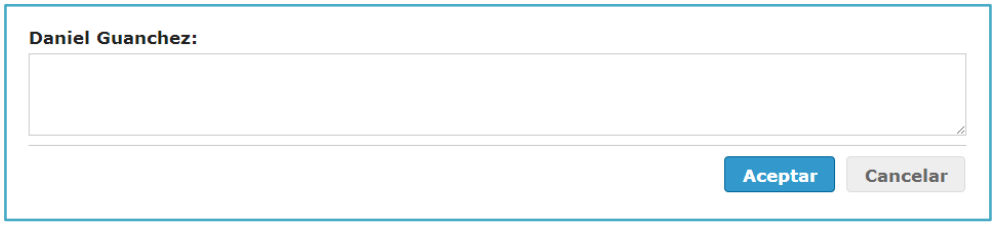


Figura 3.56 Formulario para agregar comentarios.

3.2.3.4 Prueba

Las pruebas realizadas para la verificación de todo lo desarrollado en la presente iteración consistieron en verificar la integración del módulo de archivos adjuntos y módulo de comentarios, con los módulos directamente relacionados, el módulo de historias de usuario, el módulo de tareas, y el módulo de casos de prueba, ya descritos previamente. La Tabla 3.9 describe los casos de prueba.

N° Caso de Prueba	Historias de Usuario	Caso de Prueba	Resultado Esperado	Resultado Obtenido
21	HU21	Adjuntar un archivo una historia de usuario	Se carga un nuevo archivo, asociado a la historia de usuario	Se cargó el archivo y se mostró en la lista adjuntos de la historia.
22	HU21	Adjuntar un archivo una tarea	Se carga un nuevo archivo, asociado a la tarea	Se cargó el archivo y se mostró en la lista adjuntos de la tarea
23	HU21	Adjuntar un archivo un caso de prueba	Se carga un nuevo archivo, asociado al caso de prueba	Se cargó el archivo y se mostró en la lista adjuntos del caso de prueba
24	HU22	Editar la información de un archivo adjunto	Modificar lo información de un archivo adjunto	Se actualizó la información del adjunto
25	HU23	Eliminar archivos	Borrar los archivos y la información asociada	Se borraron los archivo y toda la información asociada
26	HU24	Agregar comentario a una historia	Se crea un nuevo comentario	Se guardó el comentario muestra en la lista en orden cronológico

Tabla 3.9 Prueba de aceptación de los módulos de adjuntos y comentarios.

3.2.4 Cuarta Iteración: Gestión de Entregas e Iteraciones

Para esta iteración se contempló el desarrollo de las funcionalidades que permiten gestionar las entregas y las iteraciones en las que se divide un proyecto permitiendo la crear, editar y eliminar entregas o iteraciones. También en esta iteración se desarrollan las funcionalidades que permiten la planificación y seguimiento de las historias de usuarios en las entregas o las iteraciones definidas.

3.2.4.1 Planificación

Las historias de usuario a abordar se pueden ver en la Tabla 3.10.

N° Iteración	4
Descripción	Desarrollo de los módulos para la gestión de estregas e iteraciones para el proyecto. Planificación y seguimiento de las historias de usuario en entregas o las iteraciones.
Fecha Inicio – Fin	23/07/2012 - 27/08/2012
Historias de Usuario	HU13 Continuar Historia HU26 Creación de Entregas HU27 Editar Entregas HU28 Eliminar Entregas HU29 Planificación de Entregas HU30 Visualización del Resumen de la Entrega HU31 Creación de Iteraciones HU32 Editar Iteraciones HU33 Eliminar Iteraciones HU34 Planificar las Iteraciones HU35 Seguimiento de las Iteraciones
Tiempo Estimado	35 días

Tabla 3.10 Planificación de la cuarta iteración.

3.2.4.2 Diseño

En esta sección se describen los modelos de datos entidad relación (E/R) y los diagramas de clases UML para dar solución a la gestión y planificación de entregas e iteraciones, a través del uso de los patrones y *framework* elegidos.

Modelo de Datos

El diseño de la presente iteración lo conforma la creación de tablas en la base de datos referentes a la definición y planificación de entregas e iteraciones para un proyecto y la planificación de las historias de usuarios. En la Figura 3.57 se muestra el modelo datos E/R que se desarrolla para la presente iteración, donde destaca la incorporación de la tabla *entrega* y la tabla *iteracion*. Dentro de las reglas del negocio se destaca: un proyecto puede tener varias entregas e iteraciones, una entrega o iteración puede estar asociada a un solo proyecto, una entrega o iteraciones pueden tener asociadas varias historias de usuario, una historia de usuario puede estar asociada o no a una única entrega e iteración.

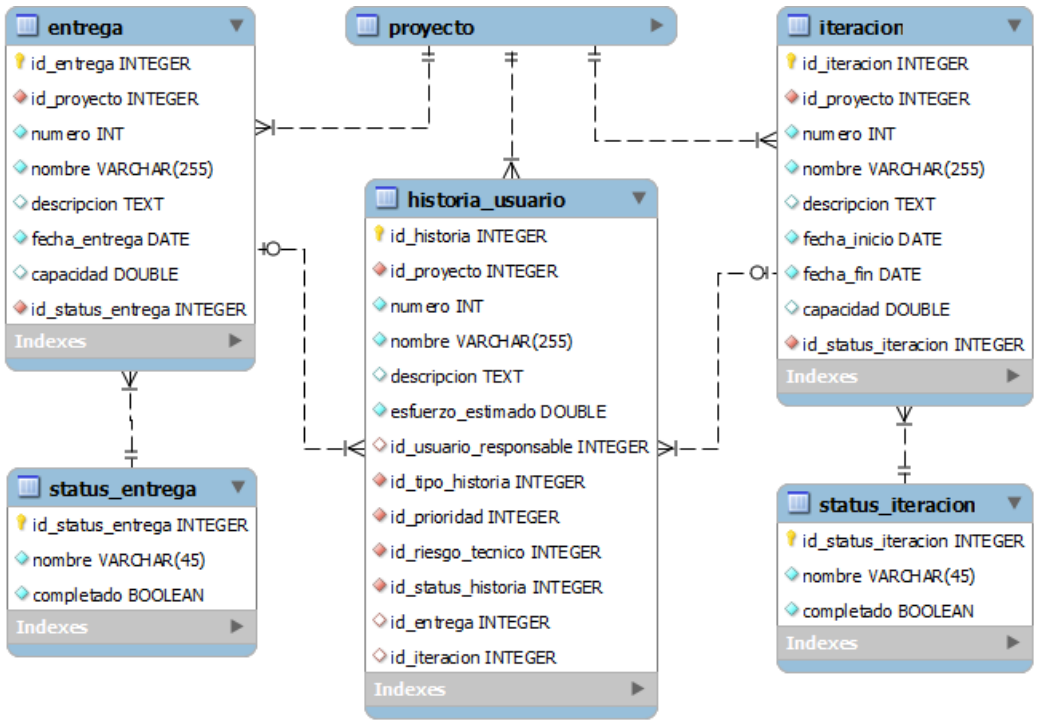


Figura 3.57 Modelo de datos E/R de la cuarta iteración.

Diagrama de Clases

Se comienza con los objetos del dominio que surgen en las historias relativas a las entregas e iteraciones y se realiza el diagrama de clases que se puede ver en la Figura 3.58. Estos objetos del dominio serán persistidos mediante *Hibernate*, como son *bean* tienen métodos *set* y *get* para cada atributo.

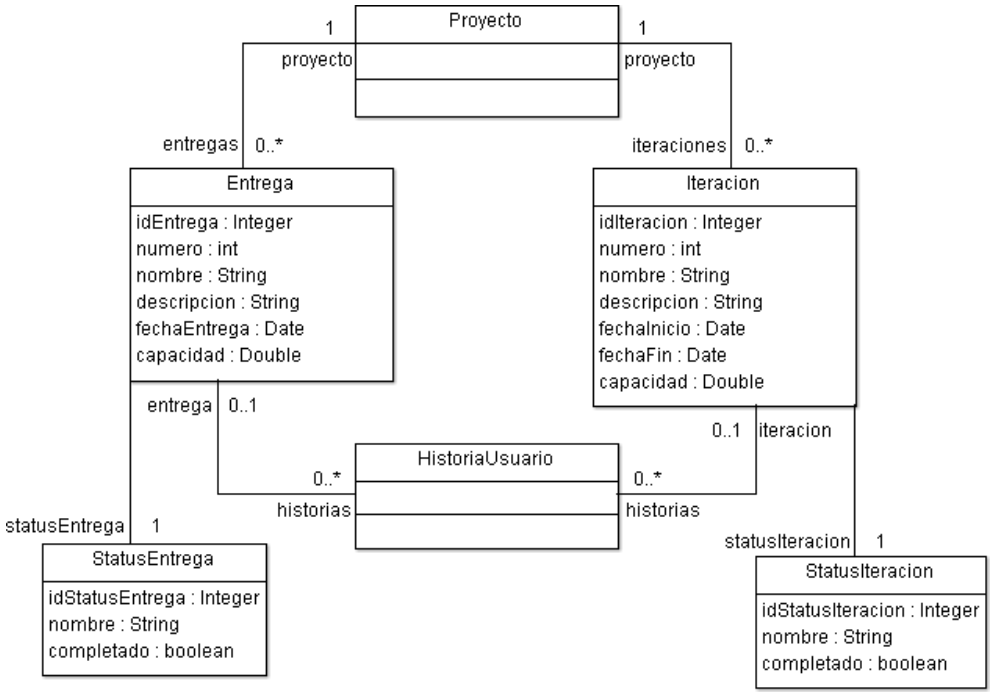


Figura 3.58 Diagrama de clases del dominio que modelan las iteraciones y entregas.

A continuación de describen cada uno de las clases de los objetos del dominio:

- *Entrega*: Esta clase representa las entregas o lanzamientos de un producto por el equipo de desarrollo. Un proyecto de desarrollo conjunto se compone de varias entregas.
- *StatusEntrega*: Esta clase corresponde el estado de la entrega, si esta en activa o entregada.
- *Iteracion*: Esta clase representa las iteraciones o ciclos de trabajo de duración fija. Un proyecto de desarrollo conjunto se compone de muchas de estas iteraciones.
- *StatusIteracion:* clase corresponde el estado de la iteración, si esta en activa o completada.

Se definen las clases que implementan el patrón DAO para de acceder a la base de datos de las entregas y las iteraciones. El diagrama de clases UML que se puede ver en la Figura 3.59.

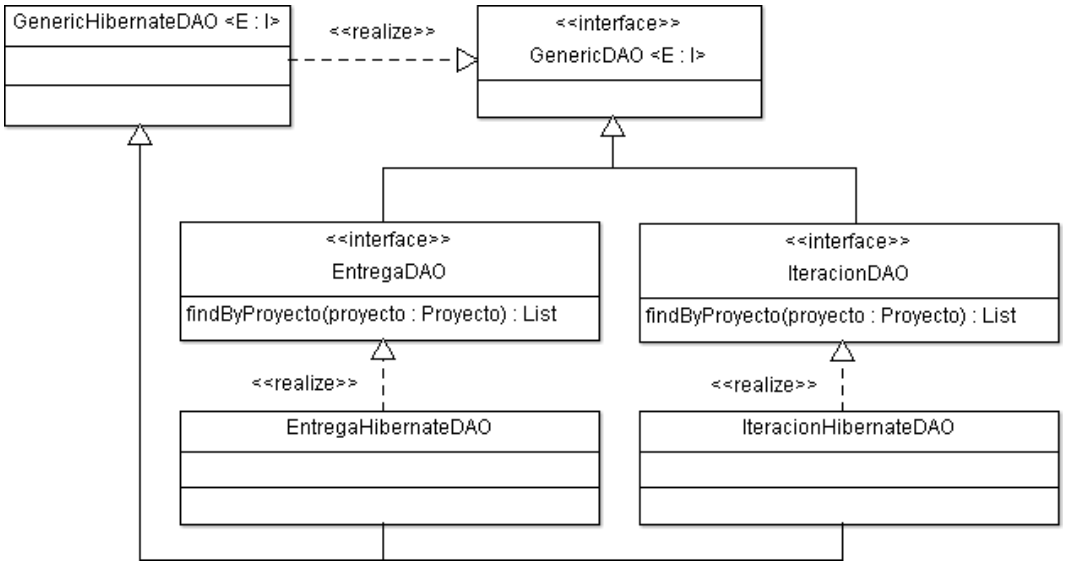


Figura 3.59 Diagrama de clases DAO específicas para acceder a las entregas e iteraciones.

Para ocultar detalles de implementación y dar soporte en la gestión y planificación de las entregas e interacciones se añaden las siguientes clases:

- *EntregaDAO*: Interfaz con los métodos necesarios que debe implementar un DAO para persistir las entregas, proporcionando como argumento la clase persistente *Entrega*. Proporciona el métodos adicional:
    - findByProyecto: Retorna las entregas de usuario de un proyecto dado.
- *EntregaHibernateDAO*: Extiende de la clase *GenericHibernateDAO*, proporcionando como argumento la clase persistente *Entrega*.
- *IteracionDAO*: Interfaz con los métodos necesarios que debe implementar un DAO para persistir las iteraciones, proporcionando como argumento la clase persistente *Iteracion*. Proporciona el métodos adicional:
    - findByProyecto: Retorna las iteraciones de usuario de un proyecto dado.
- *IteracionHibernateDAO*: Extiende de la clase *GenericHibernateDAO*, proporcionando como argumento la clase persistente *Iteracion*.

Para la gestión del plan de entrega y el plan de iteración, es necesario añadir en el DAO específico que maneja la persistencia de las historias de usuario, los métodos necesarios para obtener la lista de historias que no se han planificado dentro de una entrega o iteración según sea el caso. La Figura 3.60 muestra el diagrama de clases UML con la incorporación de los nuevos métodos en la clase *HistoriaDAO*.

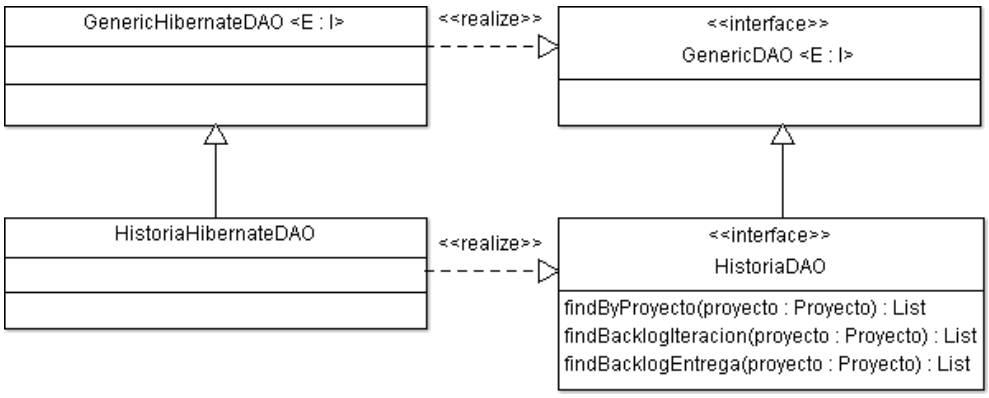


Figura 3.60 Diagrama de clases DAO específicas para acceder a las historias de usuario.

Para dar soporte a estas funcionalidades se agregan en la interfaz *HistoriaDAO* los siguientes métodos:

- *findBacklogIteracion*: Retorna la lista de historias de usuario, que no han sido asignadas a ninguna iteración, de un proyecto dado.
- *findBacklogRelease*: Retorna la lista de historias de usuario, que no han sido asignadas a ninguna entrega, de un proyecto dado.

Al igual que en los módulo anteriores se crean las fachadas *EntregaFacade* e *IteracionFacade* para los nuevos módulos, como interfaz de la capa modelo hacia capas superiores ocultando los detalles de implementación de la capa modelo. Cada clase implementa una clase interfaz *ManagerFacade* propia.

La Figura 3.61 muestra el diagrama de clases con la fachada para ocultar los detalles de implementación del módulo Historias de Usuario.

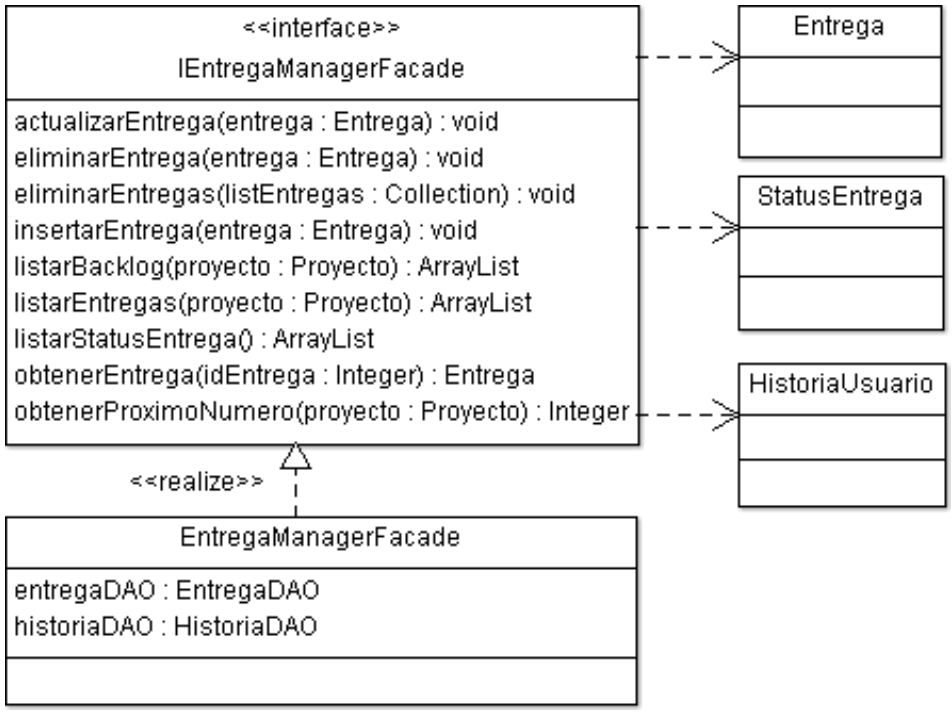


Figura 3.61 Diagrama de clases fachada del módulo de entregas.

La Figura 3.62 muestra el diagrama de clases con la fachada para ocultar los detalles de implementación del módulo Historias de Usuario.

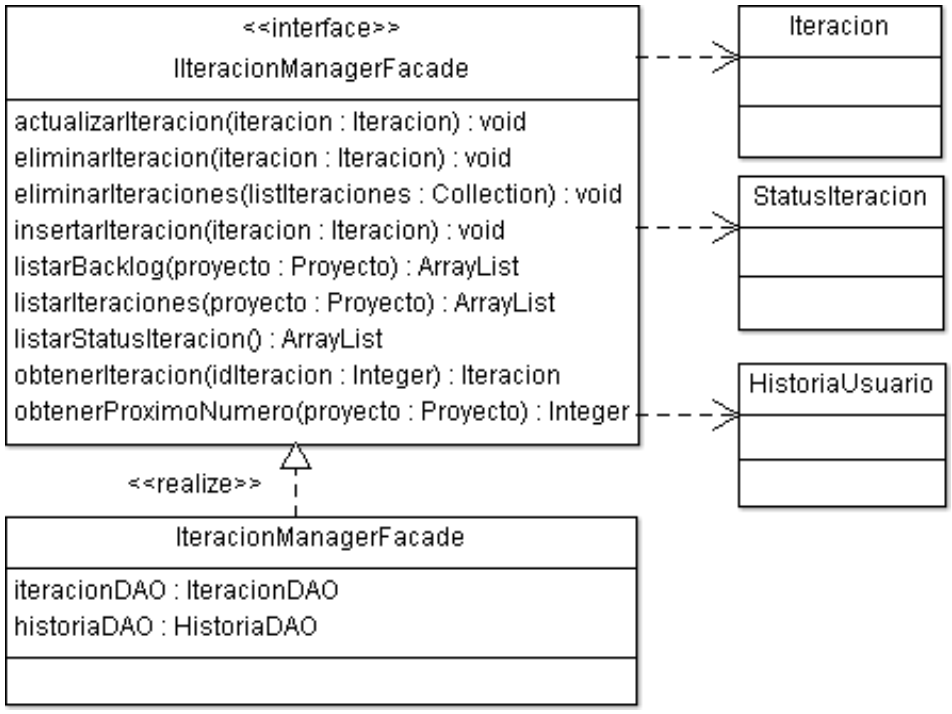


Figura 3.62 Diagrama de clases fachada del módulo de iteraciones.

3.2.4.3 Codificación

La fase de codificación se inicia con el desarrollo del método que permite obtener la lista de historias de usuario pendientes por planificar dentro de un una entrega o una iteración. El código puede apreciarse en la siguiente Figura 3.63.

```
public List<HistoriaUsuario> findBacklogIteracion(Proyecto proyecto) throws HibernateException {
    try {
         session = getSession();
         session.beginTransaction();
         List<HistoriaUsuario> list = session.createCriteria(HistoriaUsuario.class)
                                             .add(Restrictions.eq(PROYECTO, proyecto))
                                             .add(Restrictions.isNull(ITERACION))
                                             .list();
         session.getTransaction().commit();
         return list;
    } catch (HibernateException he) {
         session.getTransaction().rollback();
         throw he;
    }
}

public List<HistoriaUsuario> findBacklogEntrega(Proyecto proyecto) throws HibernateException {
    try {
         session = getSession();
         session.beginTransaction();
         List<HistoriaUsuario> list = session.createCriteria(HistoriaUsuario.class)
                                             .add(Restrictions.eq(PROYECTO, proyecto))
                                             .add(Restrictions.isNull(RELEASE))
                                             .list();
          session.getTransaction().commit();
          return list;
    } catch (HibernateException he) {
         session.getTransaction().rollback();
         throw he;
    }
}
```

Figura 3.63 Implementación de los métodos para obtener las historias no planificadas.

En la capa controlador se añaden las acciones de *Struts*, *ReleaseAction* e *IteracionAccion*, con los métodos que darán soporte a la nueva funcionalidad (listar, visualizar, agregar, editar, eliminar, planificar), que hacen de intermediario entre la vista y el modelo, con su correspondiente configuración. En la acción *HistoriaAction* se añade el método necesario para continuar una historia (continuar) junto con la configuración necesaria. Dentro de las vistas que se implementaron para esta iteración tenemos las siguientes:

- Definición de entregas e iteraciones para un proyecto
- Lista de entregas e iteraciones de un proyecto
- Continuar una historia de usuario
- Planificación y seguimiento de entregas e interacciones

A continuación se describe las pantallas con las funcionales para la planificación y seguimiento de las historias de usuario, en las entregas e iteraciones del proyecto.

Al seleccionar la opción “Entregas” del menú principal, se muestra el listado de las entregas del proyecto, en el que se puede seleccionar la entrega que se desea modificar, eliminar, o se puede agregar una nueva. La Figura 3.64 muestra la lista de las entregas planificadas para el proyecto seleccionado.

Figura 3.64 Lista de entregas planificadas para un proyecto.

La Figura 3.65 muestra el formulario que se utiliza tanto para crear o editar una entrega.

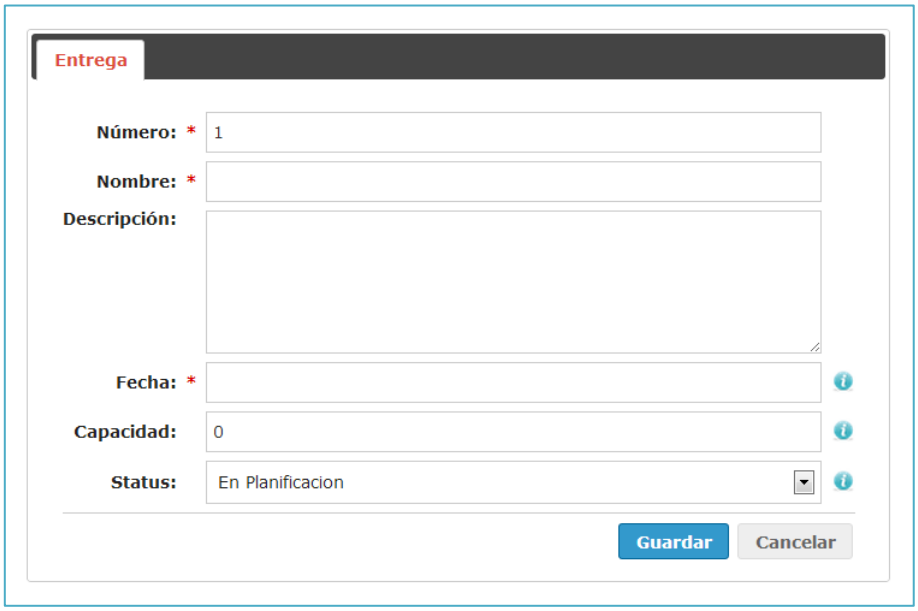


Figura 3.65 Formulario para agregar o editar una entrega.

Al seleccionar la opción "Iteraciones" del menú principal, se muestra el listado de las iteraciones del proyecto, en el que se puede seleccionar la entrega que se desea modificar, eliminar, o se puede agregar una nueva. La Figura 3.66 muestra la lista de las iteraciones planificadas para el proyecto seleccionado.

N°	Nombre	Descripción	Fecha de Inicio	Fecha de Fin	Capacidad	Status
1	Primera Iteración	Desarrollo de los módulos para la gestión de usuarios, proyectos y acceso a la aplicación	30/04/2012	28/05/2012	28.0	Completada
2	Segunda Iteracion	Desarrollo de los módulos para la gestión de historias de usuario, tareas y casos de prueba	28/05/2012	02/07/2012	35.0	Completada
3	Tercera Iteración	Desarrollar las funcionalidades que permitan adjuntar archivos y agregar comentarios en las historias de usuarios, tareas y casos de prueba	02/07/2012	16/07/2012	14.0	Completada
4	Cuarta Iteración	Desarrollo de los módulos para la gestión de estregas e iteraciones para el proyecto.	23/07/2012	27/08/2012	35.0	En Progreso
5	Quinta Iteración	Desarrollo de las funcionalidades que permiten el uso y exportación de las listas de historias de usuario, tareas y casos de prueba de un proyecto.	27/08/2012	24/09/2012	28.0	En Planificación

Figura 3.66 Lista de iteraciones planificadas para un proyecto.

La Figura 3.67 muestra el formulario que se utiliza tanto para crear o editar una iteración

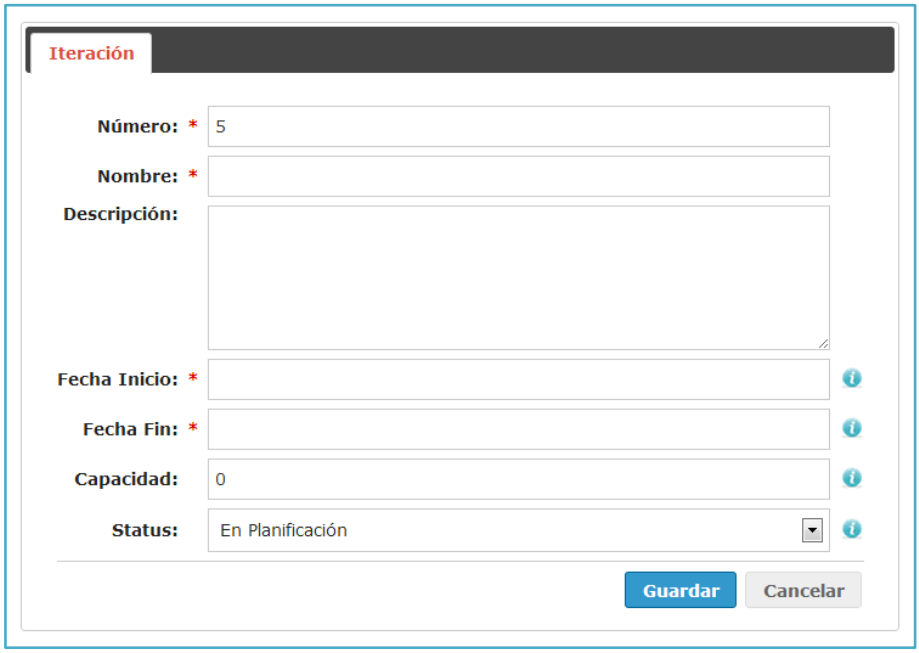


Figura 3.67 Formulario para agregar o editar una iteración.

En los formularios para la creación de entregas e iteraciones, se integra el componente JQuery llamado “Date picker”, permitiendo la selección de las fechas desde un calendario (Ver Figura 3.68).

Figura 3.68 Componente "Date Picker" para la selección de fechas.

En la lista de entregas e interacciones, el nombre es un enlace que muestra una pantalla con el detalle. Allí se podrá visualizar los datos básicos, con las operaciones correspondientes. La Figura 3.69 muestra la pantalla para consultar del detalle de una iteración.

Figura 3.69 Visualizar los detalles de una iteración.

La aplicación soporta la planificación de las entregas y las iteraciones, permitiendo al usuario seleccionar las historias que aún no han sido planificadas en alguna entrega o iteración. El plan de entrega o iteración se actualiza en tiempo real con el esfuerzo total estimado, indicado si el esfuerzo es mayor que la capacidad de la entrega o la iteración. La interfaz diferente dividida en dos (2) paneles:

- Panel Izquierdo (Backlog): La historias de usuario que no se han sido planificadas o asignadas, en alguna entrega o iteración según el caso (Ver Figura 3.70).

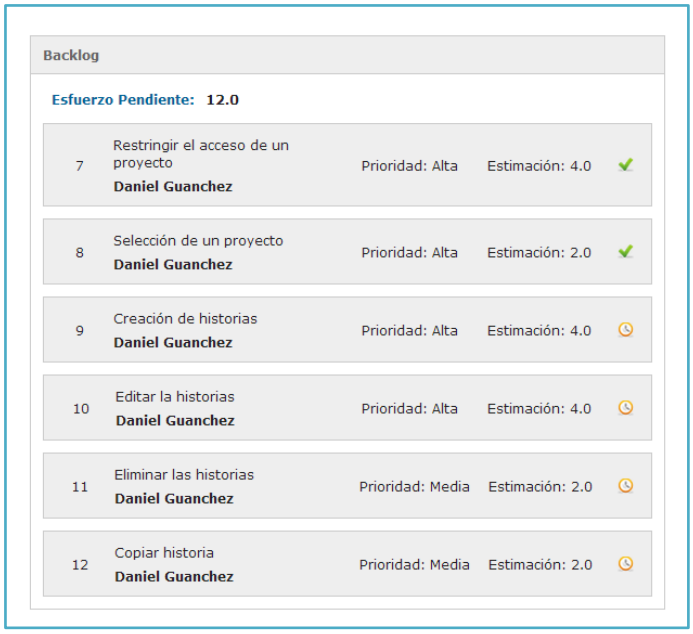


Figura 3.70 Panel Izquierdo: Backlog.

- Panel Derecho (Plan): Contiene la lista de historia de usuario planificadas para la entrega o iteración seleccionada (Ver Figura 3.71).

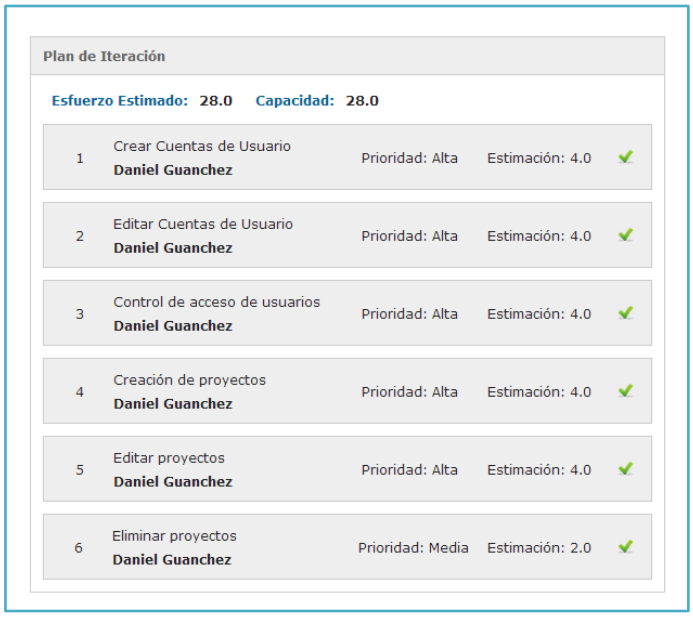


Figura 3.71 Panel Derecho: Plan de Iteración.

Al asignar historias de usuario a una entrega o interacción, es útil poder ver el contenido de diferentes formas. La lista principal de historias de usuario es una buena manera de ver la lista de historias planificadas para una entrega o una interacción, pero puede ser muy detallado si lo que se necesita es un resumen.

La aplicación ofrece como punto de vista del plan de entregas, un resumen con las listas de historias de usuario asignadas a la entrega con su descripción y estimaciones, como se muestra en la Figura 3.72.

Figura 3.72 Resumen de la planificación de una entrega.

La pantalla del estado de una iteración, se utiliza para el seguimiento de una iteración en curso, como se muestra en la Figura 3.73. Esta pantalla enumera todas las historias de usuarios planificadas para la iteración, mostrando para cada una las tareas organizadas en tres columnas, en función de su condición:

- Pendientes: Las tareas que no han iniciado (Rojo)
- En Progreso: Las tareas en curso (Amarillo)
- Completadas: Tareas que se han finalizado (Verde)

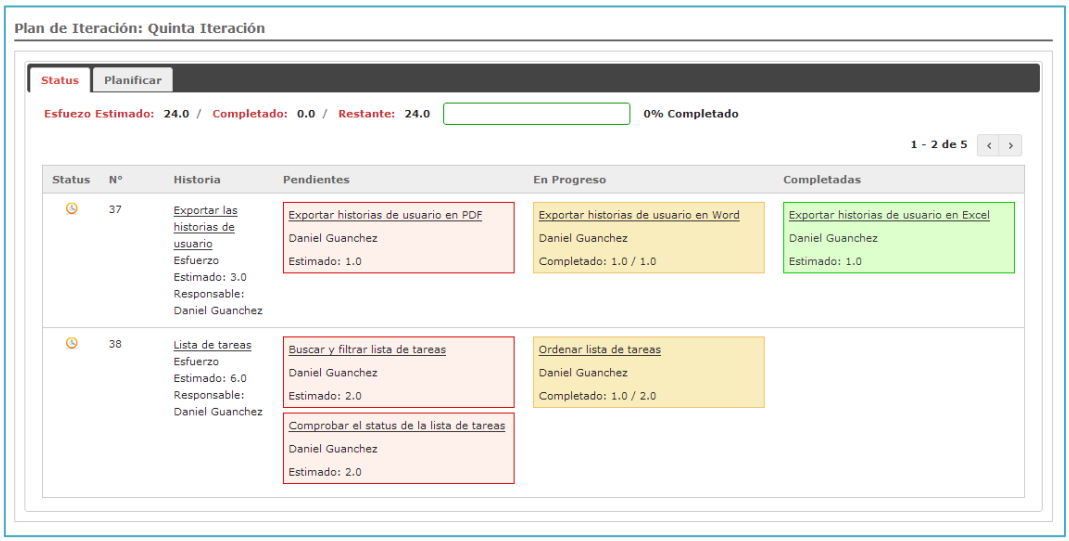


Figura 3.73 Estado de las historias y tareas de una iteración.

La Figura 3.74 la venta modal con el formulario que se utiliza para continuar una historia en otra iteración, donde se debe indicar el número, nombre de la nueva historia de usuario, y la iteración donde se le dará continuidad.

Figura 3.74 Formulario para dar continuidad a una historia.

3.2.4.4 Pruebas

En esta fase se llevan a cabo las pruebas para verificar el correcto funcionamiento de las historias de usuarios asociadas al módulo de entregas y al módulo de iteraciones. Las pruebas fueron documentadas en bitácoras separadas por módulos:

- **Módulo de entregas:** Las pruebas en este módulo se basan en la creación, edición y eliminación de entregas. Se realizan las pruebas de planificación de las historias de usuario en las entregas previamente definidas. La Tabla 3.11 describe los casos de prueba y resultados del módulo de entregas.

N° Caso de Prueba	Historias de Usuario	Caso de Prueba	Resultado Esperado	Resultado Obtenido
26	HU26	Crear una entrega	Creación de una nueva entrega	Se guardó la información de la entrega y se muestra en la lista principal de entregas del proyecto
27	HU27	Editar una entrega	Modificar lo información de una entrega existente	Se actualizó la información de la entrega y se muestra en la lista principal de entrega del proyecto
28	HU28	Eliminar entrega	Borrar las entrega y toda la información asociada	Se borró la entrega y toda la información asociada
29	HU29	Planificar historia en una entrega	Asignar la historia a la entrega y mostrar en el plan de entregas	Se actualizó la información de la historia. Se mostró la historia en el plan de entrega

Tabla 3.11 Pruebas de aceptación módulo de entregas.

- **Módulo de iteraciones:** Las pruebas en este módulo se basan en la creación, edición y eliminación de iteraciones. Se realizan las pruebas de planificación de las historias usuarios en las interacciones previamente definidas. La Tabla 3.12 describe los casos de prueba y resultados del módulo de iteraciones.

N° Caso de Prueba	Historias de Usuario	Caso de Prueba	Resultado Esperado	Resultado Obtenido
30	H31	Crear una iteración	Creación de una nueva iteración	Se guardó la información de la iteración y se muestra en la lista principal de iteraciones del proyecto.
31	HU32	Editar una iteración	Modificar lo información de una iteración existente	Se actualizó la información de la iteración y se muestra en la lista principal de entrega del proyecto.
32	H33	Eliminar iteración	Borrar las entrega y toda la información asociada	Se borró la iteración y toda la información asociada.
33	H34	Planificar historia en una iteración.	Asignar la historia a la entrega y mostrar en el plan de entregas.	Se actualizó la información de la historia. Se mostró la historia en el plan de iteraciones.
34	H13	Continuar Historia	Se crea una nueva historia con los atributos de la historia origen en la iteración seleccionada. Se trasladan las tareas no completadas de la historia.	Se creó la nueva historia, se trasladaron todas las tareas no completadas

Tabla 3.12 Pruebas de aceptación módulo de iteraciones.

3.2.5 Quinta Iteración: Reportes y Exportación

Para esta iteración se contempló el desarrollo de las funcionalidades que permiten ordenar, buscar y filtrar las listas de historias de usuario, tareas, y casos de prueba de un proyecto. Se incluye poder exportar la información del proyecto en formato Excel o PDF.

3.2.5.1 Planificación

Las historias de usuario a abordar se pueden ver en la Tabla 3.13.

N° Iteración	5
Descripción	Desarrollo de las funcionalidades que permiten el uso y exportación de las listas de historias de usuario, tareas y casos de prueba de un proyecto.
Fecha Inicio – Fin	27/08/2012 – 24/09/2012
Historias de Usuario	HU36 Lista de Historia de Usuarios HU37 Exportar Lista de Historias HU38 Lista de Tareas HU39 Exportar Lista de Tareas HU40 Lista de Casos de Prueba HU41 Exportar Lista de Casos de Prueba
Tiempo Estimado	28 días

Tabla 3.13 Planificación de la quinta iteración.

3.2.5.2 Diseño

En esta sección se describen los diagramas de clases a partir de la solución obtenida, para dar solución a la generación de reportes y exportación de las listas de historias, tareas y casos de prueba, a través del uso de los patrones y framework elegidos.

Diagrama de Clases

Como ya se modelaron los objetos del dominio en las iteraciones anteriores en cuanto a la capa modelo sólo será necesario añadir a la fachada los métodos que implementarán estas nuevas funcionalidades. Para poder listar las tareas y casos de prueba de un proyecto se

añaden los métodos *listarTareas*, *listarCasosPrueba*. Para realizar la búsqueda de las historias, tareas o casos de prueba del proyecto se añaden los métodos, *buscarHistorias*, *buscarTareas* y *buscarCasosPrueba* correspondiente a las fachadas de cada uno de los módulos. Adicionalmente se añade un método *calcularMatricas* que permite calcular los totales de los esfuerzos estimados, completado y restantes para resultado de la búsqueda.

La Figura 3.75 muestra el diagrama de clases con las modificaciones en la fachada del módulo de historias de usuario.

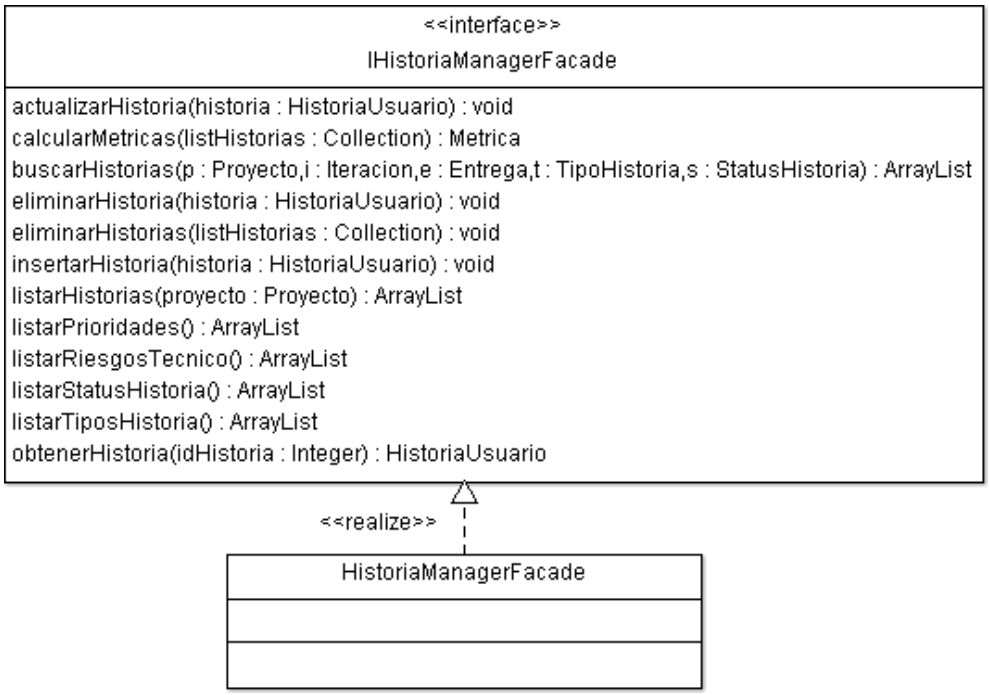


Figura 3.75 Modificación de las clases fachada del módulo de historias.

La Figura 3.76 muestra el diagrama de clases con las modificaciones en la fachada del módulo de tareas.

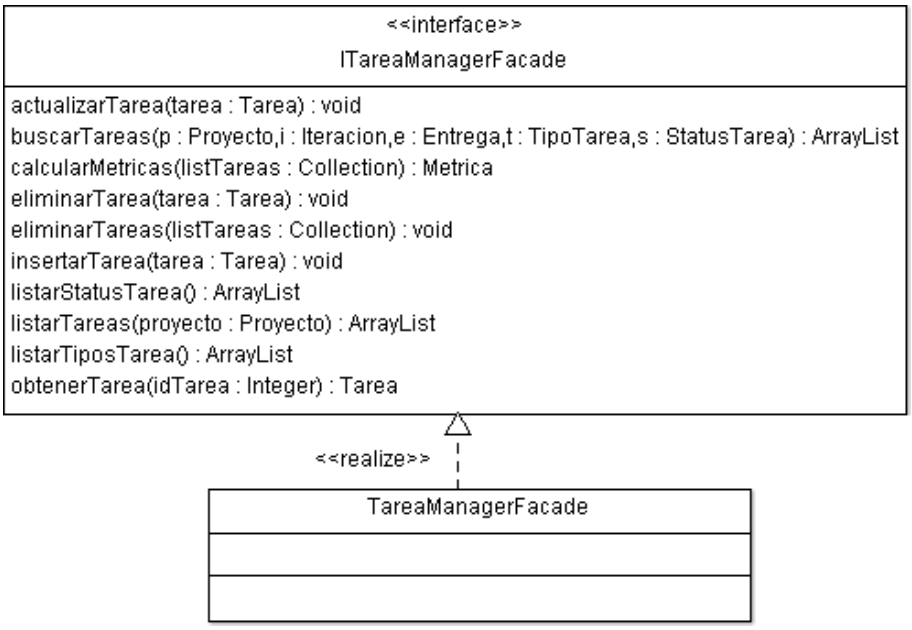


Figura 3.76 Modificación de las clases fachada del módulo de tareas.

La Figura 3.77 muestra el diagrama de clases con las modificaciones en la fachada del módulo de casos de prueba.

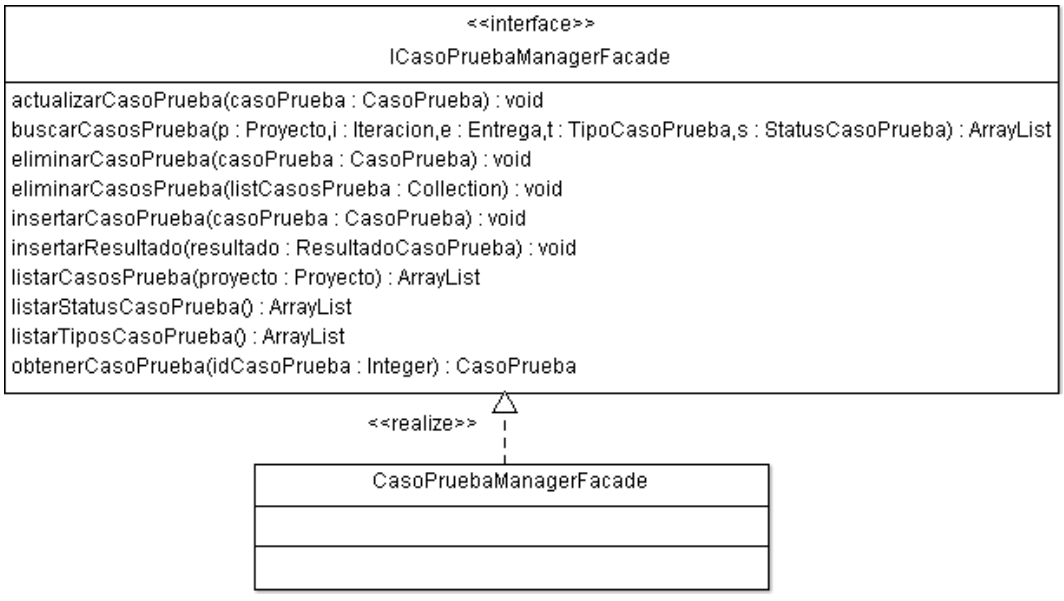


Figura 3.77 Modificación de las clases fachada del módulo de casos de prueba.

3.2.5.3 Codificación

En cuanto a las capa controlador se añade una acción para gestionar la información, en principio con el método necesario para realizar la búsqueda la información de (buscar) y un método para genera los reportes (exportar), junto con la configuración necesaria. En la capa viste se añaden los formatos de los reportes generados con iReport, y las páginas JSP con el formulario necesario para seleccionar los criterios de búsqueda y las listas con los resultados de una búsqueda. Utilizando los *taglibs* de la librera DisplayTag es posible utilizar componentes para ordenar, filtrar y paginar la lista de resultado.

Dentro de las pantallas que se implementaron para esta iteración se encuentra las siguientes:

- Lista principal de historias de usuario de un proyecto
- Lista principal de tareas de un proyecto
- Lista principal de casos de prueba de un proyecto

Al seleccionar un usuario para trabajar, desde el menú principal de la aplicación se puede acceder a cualquiera de las listas antes mencionadas. Desde allí se puede ordenar y filtrar la lista, para incluir solo la información del interés del usuario. También se puede comprobar el estado de progreso de la lista.

Para ordenar una lista se debe seleccionar cualquier columna del encabezado, permitiendo ordenarla de forma ascendente y descendente. De forma predeterminada la lista se ordena descendentemente por el número de identificación.

Estas listas cuenta con un formulario en la parte superior, que permite realizar búsquedas por los siguientes criterios: iteración, entrega, tipo y status (Ver Figura 3.78).

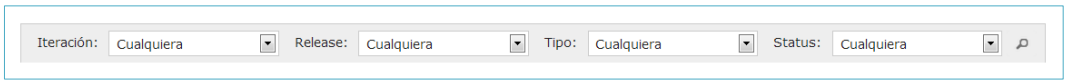


Figura 3.78 Formulario de búsqueda.

Para la lista de historias de usuario y tareas, se puede comprobar el progreso de toda la lista o de la lista filtrada (Ver Figura 3.79). Para la lista se debe mostrar las siguientes mediciones:

- Total del esfuerzo estimado
- Total de esfuerzo estimado completando
- Total del esfuerzo estimado incompleto o activo
- Barra de progreso con el porcentaje de esfuerzo completado

Figura 3.79 Mediciones para las listas.

La Figura 3.80 muestra la lista principal de historias de usuario.

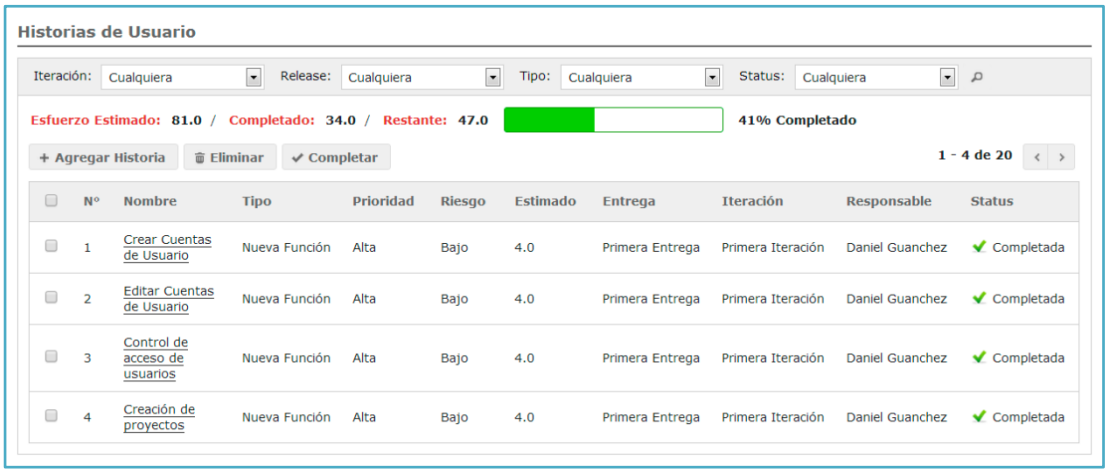


Figura 3.80 Lista principal de historias de usuario.

La Figura 3.81 muestra la lista principal de tareas.

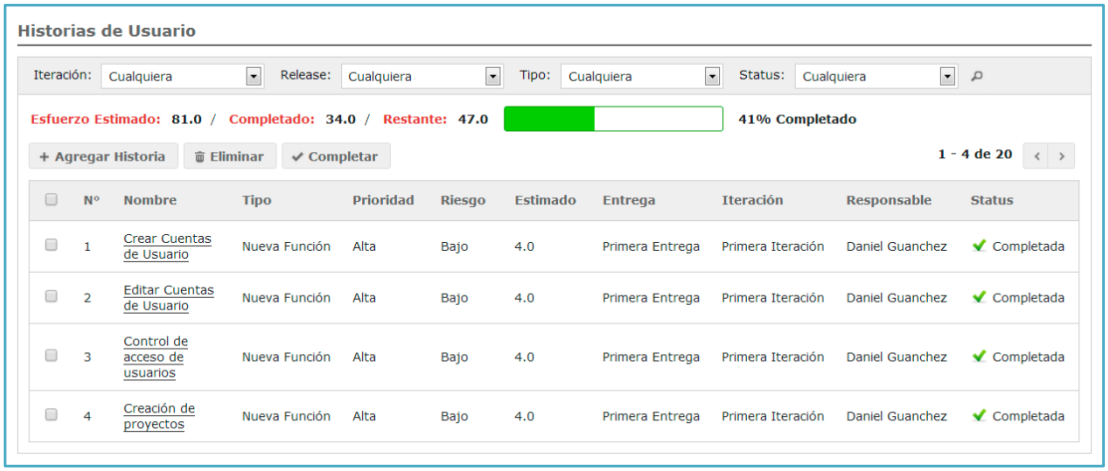
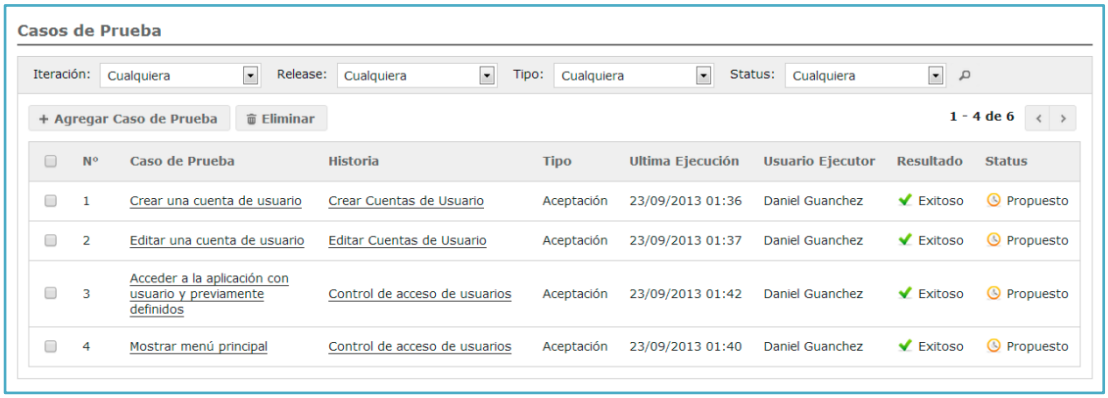


Figura 3.81 Lista principal de tareas.

La Figura 3.82 muestra la lista principal de casos de prueba.



3.2.5.4 Pruebas

Para esta fase de la iteración, se realizan las pruebas, descritas en la bitácora de la Tabla 3.14.

N° Caso de Prueba	Historias de Usuario	Caso de Prueba	Resultado Esperado	Resultado Obtenido
35	HU36	Buscar Historias	Mostrar la lista de historia que cumplan con los criterios seleccionados	Se pudo obtener la lista de historia que cumplen con los criterios seleccionados
36	H37	Generar el reporte en formato PDF de las historias de usuario de un proyecto	Se cargar en el navegador el reporte con las historias de usuario en el formato PDF	Se generó un archivo con el reporte de las historias de usuario en formato PDF
37	HU38	Buscar Tareas	Mostrar la lista de tareas que cumplan con los criterios seleccionados	Se pudo obtener la lista de tareas que cumplen con los criterios seleccionados
38	H39	Generar el reporte en formato PDF de las tareas de un proyecto	Se cargar en el navegador el reporte con las tareas en el formato PDF	Se generó un archivo con el reporte de las tareas en formato PDF
37	HU40	Buscar Casos de Prueba	Mostrar la lista de casos de prueba que cumplan con los criterios seleccionados	Se pudo obtener la lista de casos de prueba que cumplen con los criterios seleccionados

Tabla 3.14 Pruebas de aceptación reportes y exportación.

Conclusiones y Recomendaciones

El objetivo principal de este Trabajo Especial de Grado, consistió en desarrollar una aplicación web para la automatización de la gestión de los procesos de desarrollo de *software* basados en métodos ágiles, en particular para una configuración del método XP.

Como resultado se cuentan con una aplicación que se pueden acceder desde cualquier lugar que disponga de conexión a Internet, para capturar y organizar, priorizar y controlar todas las historias de usuario de un proyecto, dividir las historias en tareas y acceder a los progresos en tiempo real, adjuntar diferente tipos de archivos, agregar y compartir comentarios entre los miembros del equipo. Adicionalmente la aplicación cuenta con funcionalidades que permite preparar y publicar los planes de entrega y de iteración de un proyecto.

Utilizar el proceso de desarrollo ágil Programación Extrema (XP) permitió que la aplicación fuese sencilla de adaptar a los requerimientos y trabajar de manera iterativa e incremental en cuanto al desarrollo e implementación. Dividir los requerimientos en un conjunto de partes o iteraciones permitió que la atención se centrara en cumplir con los mismos, y luego componer el todo. Integrar a los usuarios durante la fase de pruebas permitió la validación de las implementaciones de cada una de las historias de usuarios.

Una de las prácticas de la Programación Extrema sugiere la codificación en parejas, lo cual no pudo llevarse a cabo ya que el desarrollo de la aplicación estuvo a cargo de una sola persona. Sin embargo, se obtuvo una aplicación adecuada a las necesidades del cliente, por lo cual puedo concluir que esta adaptación de la programación extrema puede resultar útil para el desarrollo de futuros proyectos.

La aplicación fue implementada utilizando tecnologías actuales entre las cuales cabe mencionar: plataforma JEE, utilización de los *framerworks Hibernate* y *Struts*, el entorno de desarrollo NetBeans, el sistema manejador de base de datos MySQL, integración de librerías o *frameworks* de desarrollo del lado cliente como fue el caso de JQuery, entre otras, lo que significó un reto personal en vista de la poca experiencia en el manejo y utilización de algunas de éstas, lo que motivo la investigación documental y la búsqueda de experiencias

relacionadas para aprender a utilizarlas y poder de esta manera llevar a cabo con éxito el desarrollo de la aplicación.

Las pruebas de aceptación realizadas en conjunto con los clientes y los proyectos pilotos, permitió la comprobación del flujo normal de las actividades del proceso de desarrollo, así mismo la corrección oportuna de los errores presentados durante la ejecución de las mencionadas pruebas. Como resultado de estas pruebas se pudo evidenciar, no solo la correcta fluidez de los procesos de desarrollo, sino como la automatización del mismo mejora notable y significativamente su ejecución, en cuanto a tiempo, reducción de errores humanos y redundancia en el manejo de información. Además de mantener la documentación sincronizada de las aplicaciones desarrolladas para los futuros responsables del mantenimiento.

Gracias a la elaboración de este Trabajo Especial de Grado se logró un aporte significativo para los profesores e investigadores del CCPD y el Centro ISYS, ya que hasta el momento no se contaba con una aplicación para mejorar la gestión de los procesos de desarrollo de *software* de los trabajados de investigación, Esto permite una planificación eficaz, proporcionando mayor acceso a la información en tiempo real, incorporando la retroalimentación temprana y facilitando una colaboración en el equipo de desarrollo, con el propósito de elevar la productividad y la eficiencia y con esto la calidad de los productos, a través de una herramienta basada en la gestión de métodos agiles.

Finalmente se presentan una serie de recomendaciones a ser tomadas en cuenta para realizar mejoras, agregar nuevas funcionalidades e incluso adaptar la aplicación para resolver otro tipo de problemáticas, permitiendo de esta manera el crecimiento constante del sistema realizado en este Trabajo Especial de Grado:

- Notificaciones automáticas por correo electrónico cuando se añada o modifique información al proyecto.
- Importar las historias de usuarios, tares y casos de prueba desde archivos planos u hojas de cálculos, utilizando un formato predeterminado.

- Incorporar un gráfico de progreso para cada iteración del proyecto. Esto se puede utilizar para determinar que tan bien se está progresando el proyecto. También es útil para detectar la desaceleración en el progreso del desarrollo.
- Elaborar un manual de usuario que se pueda consultar a medida que se haga uso de las funcionalidades de la aplicación, para así disminuir la posibilidad de incurrir en errores producidos por el mal uso o por el poco entendimiento de la misma.

Bibliografía

Abián, M. (2003). *Promesas incumplidas: sobre los métodos ágiles.*

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods. Espoo: VTT PUBLICATIONS.

Alliance, T. A. (2001). *Agile Alliance*. Obtenido de http://www.agilealliance.org/

Beck, K. (1999). *Extreme Programming Explained: Embrace Change.* Addison Wesley Professional.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., y otros. (2001). *Manifiesto por el Desarrollo Ágil de Software*. Obtenido de http://agilemanifesto.org/iso/es/

Coad, P., De Lucas, J., & Lefebvre, E. (1999). *Java Modeling in Color with UML.* Prentice Hall.

Cockburn, A. (2001). *Agile Software Development.* Addison-Wesley.

Homme, S. (2007). Convenciones de código para el lenguaje de programación JAVA. Sun Microsystems Inc.

James, H. (2000). *Adaptive Software Development.* Dorset House .

Jaspersoft Corporation. (2013). *JasperReports*. Obtenido de http://community.jaspersoft.com/project/jasperreports-library

JBoss Community. (2013). *Hibernate*. Obtenido de http://www.hibernate.org/

Martin, J. (1991). *Rapid Application Development.* New York: Macmillan Inc.

Miller, G. G. (2001). The Characteristics of Agile Software Processes. *Proceedings of the 39th Int'l Conf. and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS'01).* Santa Barbara.

MySQL. (2013). *Oracle Corporation*. Recuperado el 2013 de Mayo de 26, de http://www.mysql.com/about/

Oracle. (2010). Your First Cup: An Introduction to the Java EE Platform.

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit for Software Development Managers.* Addison Wesley.

Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum.* Prentice Hall.

Stapleton, J. (1997). *Dsdm Dynamic Systems Development Method: The Method in Practice.* Addison-Wesley.

Sun Microsystems. (2012). *Core J2EE Patterns - Business Delegate*. Obtenido de http://www.oracle.com/technetwork/java/businessdelegate-137562.html

Sun Microsystems. (2012). *Core J2EE Patterns - Data Access Object*. Obtenido de http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

Sun Microsystems. (2012). *Core J2EE Patterns - Session Facade*. Obtenido de http://www.oracle.com/technetwork/java/sessionfacade-141285.html

The Apache Software Foundation. (1999). *Apache Struts*. Obtenido de http://struts.apache.org/

The Java Tutorials. (2013). *Lesson: Generics.* Recuperado el Mayo de 2013, de http://docs.oracle.com/javase/tutorial/extra/generics/index.html

The jQuery Foundation. (2012). *JQuery*. Obtenido de http://jquery.com/

The jQuery Foundation. (2012). *JQuery UI*. Obtenido de http://jqueryui.com/

Wells, D. (2009). *Extreme Programming: A gentle introduction.* Obtenido de http://www.extremeprogramming.org/