UNIVERSIDAD CATÓLICA BOLIVIANA SAN PABLO
Unidad Académica Cochabamba
Ingeniería de Sistemas

Herramienta Web de apoyo al control de la calidad del software

Proyecto de Grado de Licenciatura en Ingeniería de Sistemas

Telma Carolina Frege Issa

Cochabamba - Bolivia
Junio 2004

TRIBUNAL EXAMINADOR

.....
Mgr. Alejandro Bedini G.
Profesor Guía

.....
Dr. Gustavo Calderón M.
Profesor Relator

.....
Mgr. Consuelo Puente C.
Jefe de Carrera

.....
Dr. René Santa Cruz R.
Rector Regional

AGRADECIMIENTOS

A Dios, el autor y protagonista de mi vida, para quien las palabras de gratitud nunca alcanzarán. 2 Corintios 2:14.

A mis padres y mejores amigos: Juan Carlos y Edith cuyo amor desmedido por mí y confianza plena en mis sueños me condujeron a quien soy hoy.

A mis hermanas y a toda mi familia por su amor y aliento constantes.

A Mgr. Consuelo Puente por brindarme su amistad, apoyo y confianza en todo momento.

A las personas que, conociéndome personalmente o no, me brindaron su ayuda de manera desinteresada.

A la empresa pirAMide Informatik GmbH por el impulso dado al inicio de este proyecto.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El control de calidad es la clave para obtener productos altamente competitivos en el mercado mundial, y en consecuencia, es también la clave del éxito de una empresa.

En el desarrollo de software, este control es una tarea compleja que engloba un conjunto de actividades que deben ser ejecutadas a lo largo de todo el proceso de desarrollo por personas dedicadas específicamente a esta labor.

Para el presente proyecto se propuso la elaboración de un documento que sea el resultado de la investigación de las distintas técnicas, modelos y estándares de control de calidad que existen en la actualidad en el área de la ingeniería del software y el desarrollo de un sistema que, basado en el resultado de la investigación, permita la sistematización de las herramientas más útiles que ayuden a las empresas de software en sus tareas de control de calidad.

Como resultado de este planteamiento, el presente documento ofrece una síntesis de la investigación realizada en el campo de la Ingeniería de Calidad del Software, presentando los distintos puntos de vista de los diferentes modelos y estándares que existen actualmente, como también las recomendaciones generales para la implementación de políticas de control de calidad en empresas dedicadas al desarrollo de software y de las causas y consecuencias que tienen los distintos problemas que surgen al no existir estas políticas en las organizaciones.

Asimismo, se presenta el sistema desarrollado en base al *Aseguramiento de la Calidad del Software (SQA)* que facilita el proceso de control de calidad durante el desarrollo del software mediante las técnicas recomendadas por el SQA, la estructuración y la sistematización del proceso completo de control (desde las fases más tempranas del desarrollo hasta las últimas).

Palabras clave: Calidad del Software, SQA, Pruebas de Software, Control, Métrica, Marcos de Trabajo, Estándar, Herramienta, Caso de Prueba, Proceso, Madurez, Defecto.

CAPÍTULO 1

INTRODUCCIÓN

El desarrollo de software ha evolucionado pasando del inicial estilo artesanal al industrial, donde los productos de software son desarrollados por equipos complejos de varios informáticos cuyo trabajo debe cumplir con fuertes exigencias presentadas por las normas de calidad. Las empresas de software abundan y tanto usuarios (consumidores) como desarrolladores (competencia) han hecho que el mercado se vuelva cada día más exigente, obligando a todas las empresas del área a implementar metodologías de control de calidad que les permitan ofrecer productos de software altamente competitivos.

1.1. Planteamiento del problema

La política de implementar metodologías para el control de la calidad en las empresas de desarrollo del software es estratégica. En la actualidad la calidad de un producto es tan importante como su funcionalidad. De hecho, si la calidad no figura entre los requisitos de cualquier proyecto, muy probablemente éste fracasará. La ausencia de métodos de control de calidad ocasiona en la mayoría de los casos retrasos en las entregas (desfases en los cronogramas), productos con fallas e incluso proyectos que nunca se concluyen, lo que deriva en problemas económicos, derroche innecesario de recursos y, lo que es peor, en el deterioro de la imagen de la organización.

A su vez, el control de la calidad del software no es una tarea fácil, requiere de expertos en el área y de instrumentos que les ayuden en su labor, por lo que surge la necesidad de contar con herramientas de apoyo en el proceso del Aseguramiento de la Calidad del Software (SQA).

Por otro lado, si bien ya existen herramientas (Test Director, QStat, QACenter Enterprise Edition, etc.) que ayudan al control de la calidad, éstas poseen las siguientes falencias:

- Su uso es complicado (son poco amigables).
- No son aptas para el trabajo en equipo, es decir, no permiten que más de un usuario las utilice al mismo tiempo.

- Requieren que sus usuarios sean expertos en Calidad.
- No se acomodan completamente a las necesidades de la empresa, como por ejemplo, el desarrollo distribuido y a distancia que es tan común actualmente.
- Su costo es elevado.

1.2. Objetivo general

Desarrollar una herramienta orientada a la World Wide Web que apoye al control de la calidad en proyectos de desarrollo de software.

1.3. Objetivos específicos

- Definir la manera en que el SQA puede ser sistematizado para poder ofrecer a los usuarios la posibilidad de implementar las técnicas de SQA de una manera sencilla.
- Implementar la herramienta de manera que ésta ofrezca a los usuarios la posibilidad de utilizar más de una técnica de control de calidad, de una manera amigable.
- Dotar a la herramienta con las funcionalidades necesarias para apoyar al usuario en el control de la calidad de un proyecto durante todas las etapas del desarrollo (desde los requerimientos hasta la prueba final).
- Proporcionar un sistema y documentación de base para los profesionales que quieran hacer un control serio de la calidad de sus productos durante su desarrollo a fin de asegurar que el mismo cumpla con las normas mínimas y se tenga un buen empleo de los recursos.
- Contribuir para que el control de la calidad del software deje de ser una teoría ajena en nuestro medio y comience a ser implementado como una práctica habitual en las empresas, de tal manera que éstas puedan ofrecer mejores productos.

1.4. Alcances y limitaciones

1.4.1. Alcances

- El trabajo contempla aspectos de investigación y de implementación.
- La herramienta es amigable para el usuario.

- La herramienta proporciona datos estadísticos básicos de los resultados obtenidos al terminar el control de la calidad de un determinado proyecto.
- El sistema está orientado al Web; es una herramienta ASP (Application Server Provider), lo que permite el acceso a la misma mediante una simple conexión al servidor, además de ser multiusuario y multiproyecto.
- La herramienta cuenta con un soporte multi-idioma. Inicialmente está disponible en español e inglés.
- El sistema emite mensajes de alerta a los usuarios cuando éstos tienen tareas pendientes.

1.4.2. Limitaciones

- El sistema no incluye el estudio ni la implementación de las especificaciones de documentación.
- El sistema es una herramienta de gestión, no cuenta con ningún módulo inteligente que evalúe de manera directa el cumplimiento de las métricas u otros criterios que son registrados en el mismo.
- La herramienta no está destinada a la administración de proyectos, solamente ofrece un marco de trabajo básico a este nivel.
- La herramienta fue implementada utilizando la arquitectura MVC (Model-View-Controller).

1.5. Justificación

La falta de énfasis en el tema de Control de Calidad durante la formación de los ingenieros en el país ocasiona que se le reste importancia a este aspecto y, por lo tanto, la mayoría de los productos no alcanzan el nivel de competitividad exigido por otras empresas del extranjero.

Tanto el documento como la herramienta ofrecerán a las empresas de software la oportunidad de hacer un control de la calidad de sus productos durante su desarrollo y, por consiguiente, las beneficia con respecto a la competencia y a sus clientes.

CAPÍTULO 2

MARCO TEÓRICO

2.1. Antecedentes

Una característica del ser humano siempre ha sido la búsqueda de la perfección en todos los aspectos de la vida, el interés por un buen desempeño en su labor y la necesidad de asumir responsabilidades sobre el trabajo efectuado. Este rasgo originó con el paso del tiempo el concepto de lo que hoy conocemos como Calidad.

Walter A. Shewhart, doctor en Física que trabajó como ingeniero para la Wester Electric Corporation, en 1924 introduce formalmente el uso de las normas de calidad en el control de procesos con las llamadas *cartas de control*, que eran utilizadas para detectar defectos en las líneas de producción antes de que éstos se generen [GON98].

A mediados del siglo XX, fueron los japoneses quienes dieron un verdadero énfasis a todo lo referente a la calidad. En 1950 la Unión de Científicos e Ingenieros Japoneses llevó a cabo un seminario donde el Dr. W. E. Deming desarrolló el tema: "Cómo mejorar la calidad mediante el ciclo de planear, hacer, verificar y actuar"[CER00], teoría que ha evolucionado y será explicada en el apartado 2.5.5.

En 1954, el Dr. J. M. Juran introdujo en Japón la idea de que la calidad de un producto o servicio depende directamente del grado de conciencia que tenga el personal involucrado en el mismo, lo que marcó una transición en las actividades de control de calidad y dio origen a un ambiente en el que se reconoció el Control de Calidad como un instrumento de gerencia [CER00].

Hacia fines de los años 70 e inicio de los 80, tuvo lugar la *crisis del software* debido a la falta de eficiencia de los procesos que eran utilizados durante el desarrollo de programas. Fue entonces que el Departamento de Defensa de los Estados Unidos encomendó al SEI (Software Engineering Institute) de la Universidad Carnegie Mellon de Pittsburg la elaboración de un modelo que tenía como propósito guiar a las organizaciones de desarrollo

de software durante la ejecución de sus proyectos a fin de emplear sus recursos de la mejor manera posible. De esta manera, en 1986, surgió el CMM© (Capability Madurity Model), acerca del cual se tratará en el apartado 2.5.1 [SAL00][BAC04].

2.2. Situación actual

En muchos países -incluyendo Bolivia- el desarrollo de software no termina de salir de su fase artesanal; aun existen empresas que delegan trabajos a personas individuales, quienes no aplican metodologías de desarrollo formales y mucho menos técnicas de control de calidad [BUA00].

La industria del software padece de la denominada prisa patológica, fruto de malas estimaciones y planificaciones, lo que deriva en el sacrificio de actividades importantes que forman parte del proceso de desarrollo del producto de software [BUA00].

En una organización inmadura [BUA00] si hay plazos, se sacrifican funcionalidad y calidad del producto para satisfacer el plan ("dejar de lado lo importante para hacer lo urgente") y cuando los proyectos se salen del plan, las revisiones y pruebas se recortan o inclusive se eliminan.

Todo lo anterior puede asociarse con [IBA03]:

- Ausencia de especificaciones completas, coherentes y precisas por parte del cliente y de los desarrolladores.
- Carencia de la aplicación sistemática de métodos y normas de ingeniería del software durante todo el proceso de desarrollo.
- Ausencia de entornos integrados de trabajo.
- Falta de uso de técnicas actuales y automatizadas para la gestión de proyectos.
- Carencia de procesos utilizados durante el desarrollo del software que estén bien definidos y asociados adecuadamente a las herramientas utilizadas.

Estas anomalías, denominadas en su conjunto "caos del software", ocasionan que [COR01] [BUA00]:

- Más del 30 por ciento de los proyectos se cancelen antes de ser finalizados.

- Alrededor de un 84 por ciento de los proyectos no se concreten en el tiempo y costos estimados.
- El 90 por ciento de los proyectos no alcancen sus objetivos.

Además de esto, el concepto de calidad utilizado en muchas organizaciones es errado. La mayoría de las empresas creen saber y practicar lo que es calidad implementando un limitado número de actividades relacionadas con ella pero sin establecer un control formal y sin asumir políticas específicas que establezcan una verdadera estrategia de control de calidad, la que es necesaria para garantizar el éxito del producto.

Ahora bien, implementar políticas de control de calidad en una empresa tiene como objetivo garantizar que el producto cumpla con los requisitos que le fueron asignados, pero no así que éste vaya a estar libre de errores. De hecho, muchos expertos opinan que el software libre de defectos no existe.

Esta última afirmación es evidente incluso en las grandes compañías de software donde los productos son comercializados cuando los mismos desarrolladores están concientes de que poseen fallas; esto ocurre comúnmente cuando el costo y tiempo invertidos en corregir estos errores no se justifican frente a las utilidades que se perciben si el producto es lanzado en el tiempo previsto aun teniendo algunos defectos. Estas anomalías pueden, por ejemplo, tratarse de detalles que no afecten a la funcionalidad en sí o de errores que ocurren dadas condiciones muy especiales y por lo tanto son poco probables (una vez cada mil casos por ejemplo).

2.3. Definiciones

2.3.1. Calidad del software

Existen muchas definiciones para este término, tres de las más precisas y ampliamente utilizadas son:

> *"Conjunto de propiedades y de características de un producto o servicio, que le confieren aptitud para satisfacer necesidades explícitas e/o implícitas."* (ISO 8402) [BUA00].
>
> *"La calidad del software es el grado en que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario"* (IEEE, Std. 610-1990) [BUA00].

"Concordancia del software producido con los requerimientos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requerimientos implícitos no establecidos formalmente" [PRE02].

En base a estas definiciones se resaltan los siguientes puntos [PRE02]:

- Los requisitos establecidos para el producto de software son la base de la calidad; es decir, la falta de concordancia entre los requisitos se traduce en falta de calidad.
- Los estándares existentes definen un conjunto de criterios de desarrollo que puedan guiar al equipo para aplicar la ingeniería del software. Por lo tanto, estos criterios también son necesarios para obtener un producto de calidad.
- Los requisitos implícitos deben cumplirse tanto como los explícitos pues en la mayoría de los casos son los más comprometidos con la calidad del producto (ej.: mantenibilidad).

2.3.2. Control de calidad

El control de calidad es una serie de actividades que incluyen inspecciones, revisiones y pruebas durante el ciclo de desarrollo para asegurar que cada producto cumpla con los requisitos que le fueron asignados [PRE02].

Esta tarea incluye un "bucle de retroalimentación" conocido como *feedback*, cuyo objetivo es que, junto con la medición, se pueda conseguir afinar el proceso [PRE02].

2.3.3. Garantía de calidad

Consiste en la auditoría y las funciones de información de la gestión con el objetivo de poder proporcionar los datos necesarios sobre la calidad del producto [PRE02].

2.4. Factores para determinar la calidad de un producto de software

Son muchos los criterios a considerar al momento de calificar la calidad de un producto de software. Básicamente éstos se clasifican en dos: internos y externos [CUE99] [GÓM01].

Los factores internos son aquellos que conciernen sobretodo a los desarrolladores. Se contemplan aspectos como la claridad y la eficiencia del código.

Los factores externos son percibidos por los usuarios. En esta categoría se encuentran criterios comoe velocidad, facilidad de uso, etc.

Cabe recalcar que los factores externos son más importantes que los internos a la hora de calificar la calidad de un producto. Por ejemplo, siempre es mejor (tiene mayor aceptación) un software rápido aunque su código no sea claro [GÓM01].

Factores de calidad externos

Exactitud

Medida en que los productos de software pueden realizar tareas específicas, tal como lo define su especificación [GÓM01].

Robustez

Capacidad de los sistemas de software para reaccionar apropiadamente a condiciones anormales [GÓM01].

Mientras más robusto sea el software, mejor tolerancia a fallas y tratamiento de errores ofrecerá, lo que contribuye a una mayor confianza en el producto por parte de los usuarios.

Seguridad

Capacidad del software para responder adecuadamente a posibles accesos no autorizados y cambios importantes que afecten negativamente a la información que maneja o a las funciones que desempeña.

Facilidad de uso

La facilidad de uso es la simplicidad con la que la gente de varios trasfondos y cualidades pueden aprender a utilizar productos de software y aplicarlos para resolver problemas. Esto también incluye la facilidad de instalación, operación y monitoreo [GÓM01].

Confiabilidad

Los cuatro términos mencionados anteriormente (exactitud, robustez, seguridad y facilidad de uso) son los que conforman un quinto concepto: la confiabilidad, es decir, la comodidad y seguridad que siente el usuario al utilizar el sistema. Mientras más resultados precisos arroje, mejor manipulación de errores provea, mejores controles de seguridad ofrezca y su uso sea más intuitivo, el usuario se sentirá más confiado de utilizarlo. Este es, sin duda, el concepto más importante en lo que a certificación de calidad por parte del usuario concierne.

Extensibilidad

Capacidad de adaptación del sistema hacia cambios de especificación (cambios en los requerimientos que impliquen modificaciones a la funcionalidad general del sistema, a su arquitectura, etc.) [GÓM01].

Reutilización

Capacidad de los elementos de software para servir en la construcción de muchas aplicaciones diferentes [GÓM01].

Compatibilidad

"Facilidad para combinar un elemento de software con otro" [GÓM01].

Eficiencia

Capacidad del software para utilizar la mínima cantidad de recursos de hardware y software [GÓM01].

Portabilidad

"Facilidad de transportar productos de software a varios ambientes de hardware y software" [GÓM01].

Funcionalidad

Cantidad de tareas que un sistema puede desempeñar y/o servicios que puede proveer [GÓM01].

Puntualidad

Capacidad del equipo de desarrollo de entregar el producto cuando sus usuarios lo esperan [GÓM01].

Reparabilidad

Facilidad con la que un desarrollador puede resolver los defectos de un sistema. Mientras más fácil sea esta tarea, mejor valoración por parte de los desarrolladores recibirá el software.

Algunos de estos factores externos son también denominados requisitos implícitos del producto de software, como es el caso de la eficiencia, la facilidad de uso, etc.

2.5. Marcos de Trabajo

> *"Los Marcos de Trabajo (MT) corresponden a estructuras escritas de una idea y/o conjunto de metas para facilitar a una organización la implementación de las mismas"* [BED98].

En el caso de la ingeniería del software, estos MT se clasifican según su propósito en [BED98]:

- Estándares y guías.
- Modelos de mejoramiento de procesos.
- Pautas de selección para la contratación de terceros.
- Premios de calidad.
- Modelos de ciclos de vida.
- Modelos de Ingeniería de Sistemas.

El propósito de los MT es el de proporcionar el soporte necesario para garantizar una mejora de los procesos de software, determinar las capacidades de cada uno de ellos y la madurez general de la organización.

Los principales MT en el área de ingeniería del software son mostrados en la figura 2.1 [BED98].

A continuación se describen los más importantes.

2.5.1. CMM©

Como se indicó en el apartado 2.1, el Capability Madurity Model (Modelo de Capacidad de Madurez) surgió a raíz de la crisis del software sufrida a principios de los años 80 y como encargo del Departamento de Defensa de los Estados Unidos al SEI de la Universidad Carnegie Mellon.

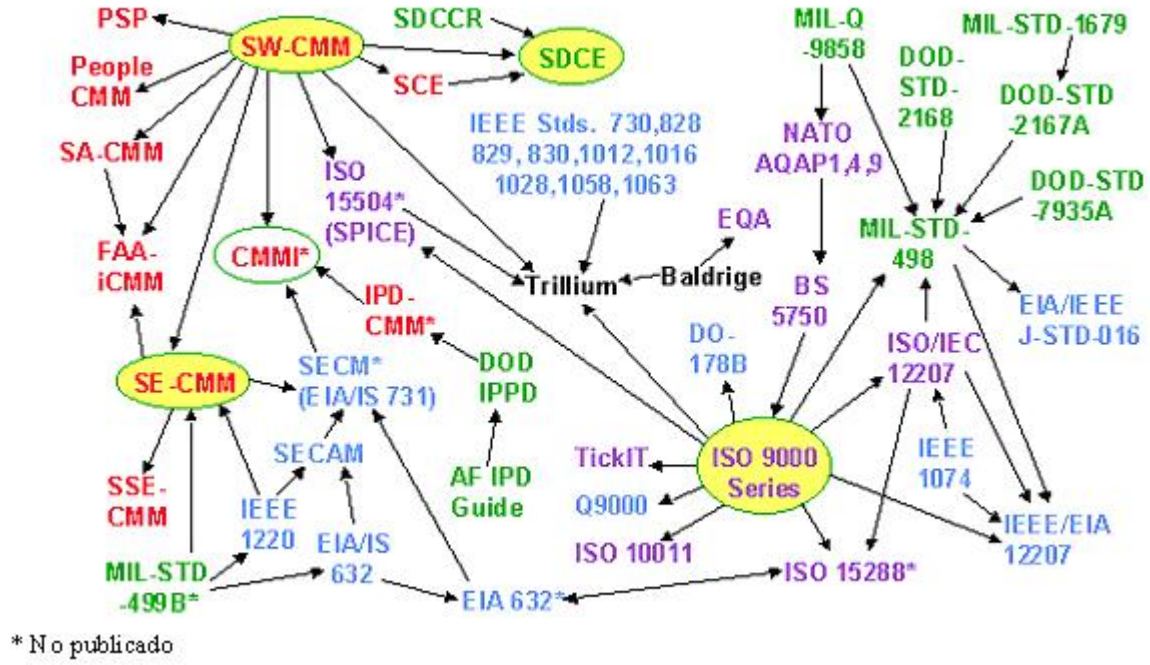


Figura 2.1: Principales Marcos de Trabajo en el área de ingeniería del software [BED98].

El principal autor del modelo es Watts Humphrey, quien trabajó en la elaboración del CMM basado en trabajos previos de Phil Crosby [BAC04].

Este modelo se basa en dos conceptos importantes: el proceso maduro y el nivel de madurez.

Un proceso se considera maduro si [SAL00]:

- Está definido: es claro y detallado.
- Está documentado.
- El personal ha sido entrenado en el proceso.
- Es mantenido: es revisado regularmente.
- Está controlado: cualquier cambio es revisado y comunicado a todo el personal.
- Se verifica: es claro si el proceso está envuelto en todos los proyectos actuales o no.
- Se valida: el proceso es coherente con los requerimientos y estándares.
- Se mide en términos de beneficio, utilización y rendimiento.

- Es mejorable.

La figura 2.2 muestra la manera en que un proceso alcanza la madurez deseada por una empresa.

Figura 2.2: Maduración de un proceso [BED98].

El nivel de madurez está definido como:

> *"La capacidad de los procesos de ingeniería del software y de administración de proyectos usados en una organización de desarrollo de software"* [SAL00].

La capacidad de los procesos en conjunto está determinada por la madurez de cada uno de ellos.

A su vez, el CMM© envuelve toda una familia de sub-modelos, cada uno de los cuales se enfoca en un área específica de la empresa. De esta manera se tienen [SAL00]:

- SW-CMM (Software CMM): Aplicado específicamente al ámbito del software.
- SE-CMM: (Systems Engineering CMM): Que cubre el ámbito de la Ingeniería de Sistemas propiamente dicha.
- P-CMM (Personal CMM): Se enfoca en los recursos humanos de la organización.
- SA-CMM (Software Acquisition CMM): El cual cubre las prácticas de adquisición de productos de software. Aunque suene irónico, es la actividad inevitable de las empresas de desarrollo de software; todas en determinado momento adquieren productos hechos por otras empresas a fin de poder desarrollar los propios.
- IPD-CMM (Integrated Product Development CMM): Aplicado al ámbito de la integración del producto.

El CMM© clasifica a las empresas involucradas en el desarrollo de software en cinco niveles de acuerdo a los criterios que maneja (la madurez de sus procesos y la calidad de los resultados que se obtienen de esos procesos). Entonces se tiene:

- Nivel 1: Inicial (Inmadurez)

También conocido como el "Nivel del Caos". Aquí se encuentran todas las empresas que no han logrado implementar prácticas básicas de Ingeniería del software y donde el personal coincide en que los proyectos no se pueden planear y los requerimientos se salen de control. Por lo general, la conclusión de un proyecto se logra a coste de muchas horas extra de trabajo y con una inversión injustificada [SAL00].

- Nivel 2: Repetible (El proyecto planificado)

Este nivel se caracteriza por la presencia en las empresas de prácticas mínimas de Ingeniería del software y donde la experiencia adquirida en proyectos anteriores es bien aprovechada por el personal. En este punto, no necesariamente todos los procesos tienen el mismo nivel de madurez [SAL00].

El mayor beneficio de este nivel es la planificación realista de los proyectos, la cual en general no expresa el deseo de la gerencia, factor que en el nivel 1 es el principal determinante para las malas estimaciones [SAL00].

- Nivel 3: El proceso definido (El proceso generalizado en todos los proyectos)

Para llegar a este nivel, la empresa tiene bien definido un conjunto de procesos y herramientas comunes a todos los proyectos. Cada proceso está debidamente documentado y el personal está entrenado en el manejo del mismo. El nivel de definición de cada proceso es detallado y completo. La dependencia en individuos "irreemplazables" es baja, al contrario que en los dos anteriores niveles[SAL00].

Una característica notoria del nivel 3 es la satisfacción del personal. Por lo general en las empresas del nivel 1 y algunas del 2 las quejas, acusaciones y frustración en el personal son características siempre presentes debido a los fracasos que acarrea la falta de

aplicación de prácticas formales de ingeniería del software y administración de proyectos.

Llegar a este nivel es considerado para muchos un lujo.

- Nivel 4: El proceso gestionado

En este nivel, tanto el proceso como el producto son cuidadosamente controlados mediante métricas precisas. Sin embargo, contar con un conjunto de métricas no significa necesariamente que la empresa pueda ser calificada en este nivel [SAL00].

La ventaja con la que cuentan las empresas del nivel 4 es la capacidad que tienen de poder medir su productividad y calidad; la capacidad de rendimiento de un proceso es previsible.

- Nivel 5: Mejoramiento permanente

Este nivel, denominado también "utópico" por algunos expertos, se basa en la realimentación cuantitativa e implementación de tecnologías innovadoras en la empresa. La organización entera se enfoca en el mejoramiento continuo de sus procesos[SAL00].

El último es considerado el nivel ideal pero prácticamente inalcanzable debido al nivel de madurez que se debe alcanzar. Tan sólo una veintena de empresas en todo el mundo han conseguido calificar en este nivel.

La figura 2.3 muestra a manera de esquema los niveles de madurez de CMM.

La figura 2.4 muestra la relación entre las empresas de software certificadas y los niveles del CMM©. Se destaca en el gráfico que la cantidad de empresas disminuye de forma considerable a medida que se sube de nivel, lo cual demuestra el enorme esfuerzo que se requiere para poder obtener una certificación aceptable (a partir del nivel 3). El detalle de las empresas certificadas por este modelo se encuentra en el Anexo A.

Figura 2.3: Niveles de madurez según el CMM [PIC03].

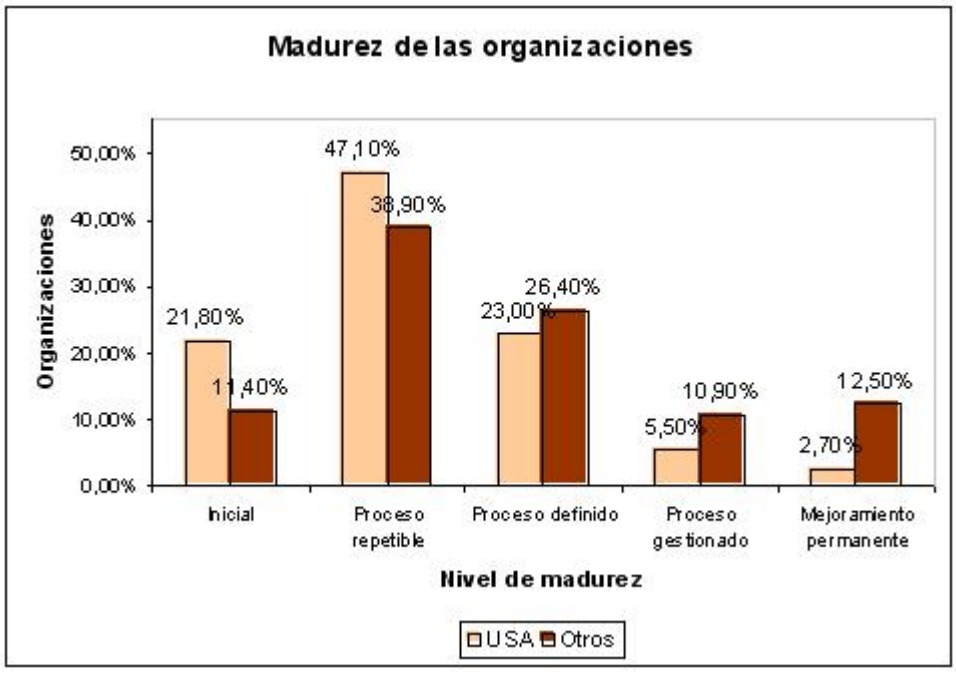


Figura 2.4: Nivel de madurez de las organizaciones en todo el mundo según el CMM© (a septiembre de 2003) [SEI03].

Desventajas

Este modelo ayudó a sacar a las empresas de desarrollo de software de la mencionada crisis en la que se vieron inmersas a inicios de los años 80 y planteó una nueva concepción del desarrollo de software al introducir el concepto de que la calidad de un producto depende del nivel de madurez de los procesos que estén involucrados en su desarrollo. Sin embargo, es un modelo muy inflexible, existen empresas que no califican en ningún nivel y otras que pueden perder su certificación o bajar de nivel si alguno de sus procesos desmejora.

Además de esta inflexibilidad, CMM© es criticado por algunos expertos, como James Bach de la Satisfice Inc. quien en su artículo "The Immaturity of CMM" (La Inmadurez del CMM) asegura que este modelo posee entre otras, las siguientes falencias [BAC04]:

- No posee bases teóricas formales, sino que simplemente está basado en la "extensa experiencia de la gente". De ser así, entonces se plantea la incógnita de porqué otros modelos fundados sobre las mismas bases no son considerados como alternativas válidas.
- El modelo posee un vago soporte empírico. Incluso Mark Paul del SEI reconoce que al modelo le falta un serio estudio de validación formal.
- CMM© valora cualquier proceso, pero ignora a la gente.
- CMM© proporciona muy poca información referente a la dinámica de los procesos, es decir, la manera en que estos trabajan y evolucionan.
- El modelo desplaza las metas que se enfocan en mejorar los procesos por la única finalidad de alcanzar un nivel más alto de madurez. A esto se le ha denominado el "nivel de envidia". Es decir, las empresas se ciegan en busca de un ascenso de nivel perdiendo por completo el verdadero sentido de aplicar el CMM©: la mejora de sus procesos. El mismo SEI ha reconocido esta falencia y ha manifestado su deseo de corregirla.

Dadas estas debilidades del modelo y puesto que en general las empresas nacionales no están preparadas para una certificación tan severa, no es recomendable considerar (por lo menos ahora) el CMM© como un modelo base para el inicio de implementación de técnicas formales de control de calidad en las empresas locales.

2.5.2. ISO

Esta organización certifica a las empresas mediante sus estándares 8402, 9000, 9001, 9002, 9003 y 9004, que han sido desarrollados desde 1979. Estos estándares imponen una serie de normas y directrices que deben ser seguidas por las empresas para conseguir la certificación [BUA00] [GON98]:

El primero (ISO-8420) se refiere solamente a la terminología utilizada en los demás estándares [BUA00].

ISO-9000 establece las normas referentes a la gestión y garantía de la calidad así como también algunas guías útiles para la aplicación de ISO-9001; es una guía básica de normas de control de calidad [TEC03].

En este estándar, ISO trata a las empresas como redes de procesos interconectados. Éstos deben ser identificados en áreas específicas definidas en los estándares y ser documentados y practicados como lo indica la especificación [PRE02].

ISO-9001 es el estándar que se aplica a la Ingeniería del Software y que otorga la garantía de calidad total. Este estándar está asociado a un conjunto de directrices para ayudar a las empresas en la interpretación y uso correcto del estándar. Estas directrices están especificadas en los estándares ISO-9000-3 [PRE02].

ISO-9002 a ISO-9003 establecen las normas y guías que conforman en modelo de calidad total, como ya se indicó líneas arriba [BUA00] [TEC03].

ISO-9004 contiene elementos de gestión del sistema de calidad, reglas generales y directrices para los elementos ya procesados y la mejora de la calidad [BUA00].

Básicamente estas normas establecen la misma secuencia descrita en el apartado 2.5 (un conjunto de actividades para cada fase del desarrollo del producto).

La lista de empresas certificadas por ISO se encuentra en el Anexo A.

2.5.3. SPICE

Análogo al CMM, el Software Process Improvement and Capability dEtermination es un modelo enfocado en la optimización (maduración) de los procesos que son utilizados en el desarrollo de software.

SPICE, en calidad de proyecto, fue aprobado por la ISO/IEC JTC1 en 1993 y tiene la finalidad de convertirse en un estándar de calidad llamado ISO/IEC 15504. Es un modelo de referencia para los procesos y sus capacidades, basado en la experiencias de empresas de software de distinta escala [BED97].

Entre los documentos de guía que ofrece, se encuentran [BED97]:

- Un marco de referencia para determinar las fortalezas y debilidades de cada proceso.
- Un marco de referencia para la mejora de los procesos de software y la cuantificación de la misma.
- Un marco de referencia para determinar los riesgos a los que se ve expuesto una empresa que planea desarrollar un nuevo producto de software.

La arquitectura del modelo SPICE organiza todas las actividades en [BED97]:

- Prácticas base o actividades esenciales para cada proceso.
- Prácticas genéricas: las que son aplicables a cualquier proceso.

A su vez, el modelo agrupa a los procesos en cinco categorías distintas [BED97]:

- Procesos Cliente - Proveedor: aquellos que tienen un impacto directo en el cliente y en la transición del producto del desarrollador al cliente.
- Procesos de Ingeniería: procesos relacionados con la Ingeniería del Software (planificación, desarrollo, documentación, mantenimiento, etc.).
- Procesos de Proyecto: aquellos involucrados en la elaboración del producto, como la administración de los recursos.
- Procesos de Soporte: se refiere a los procesos orientados a brindar soporte al desempeño de otros procesos del proyecto.

- Procesos de la Organización: propios a la organización. Son los procesos que determinan las metas de negocio de una empresa.

La figura 2.5 ilustra la interrelación que existe entre las cinco categorías mencionadas.

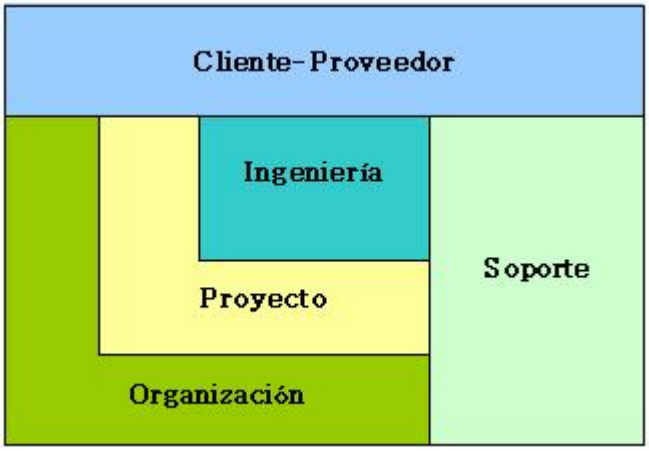


Figura 2.5: Interrelación de las categorías de procesos según SPICE [BED97].

Dependiendo de la madurez de los procesos de estas cinco categorías, una empresa puede ubicarse en uno de los seis niveles de SPICE [BED97]:

- Nivel 0: No realizado

En este nivel se clasifican las empresas que no han conseguido la implementación satisfactoria de sus procesos.

- Nivel 1: Realizado informalmente

Las empresas de este nivel ejecutan las prácticas base de sus procesos pero no necesariamente de manera planificada y sistemática. Es decir, la ejecución de estas prácticas depende directamente del esfuerzo y conocimiento del personal.

- Nivel 2: Planificado y seguido

A diferencia del Nivel 1, las prácticas base de las empresas de este nivel son planificadas y sistemáticas, lo que contribuye a una mejora progresiva del proceso que concluirá en la madurez del mismo.

- Nivel 3: Bien definido

En este nivel se clasifican las empresas cuyos procesos cuentan con un grado aceptable de madurez, es decir, son procesos definidos y documentados.

- Nivel 4: Cuantitativamente controlado

En este nivel, los procesos además de estar definidos y documentados, son medibles en términos de los resultados que producen al ser utilizados en un proyecto.

- Nivel 5: Mejoramiento continuo

Las empresas de este nivel cuentan con procesos maduros cuyo mejoramiento permanente se basa en la retroalimentación y en la implantación de nuevas tecnologías.

La figura 2.6 miestra el esquema de los seis niveles SPICE.

Figura 2.6: Niveles de madurez según SPICE [PIC03].

2.5.4. IEEE

El Institute of Electrical and Electronics Engineers certifica a las empresas que cumplen con los requisitos que plantea, tal como lo hace el ISO y el CMM©. Estos requisitos se encuentran documentados en los siguientes estándares:

- IEEE 828-1998: estándar para la configuración de planes de administración del software, que establece los requisitos mínimos con los que debe contar el Plan de Configuración y Administración del Software [TEC03].
- IEEE 1012-1998: un estándar para la verificación y validación del software, que consiste en la determinación de qué actividades involucradas en el desarrollo del mismo cumplen con los requisitos mínimos necesarios para ejecutarlas correctamente y si el producto final satisface las necesidades del usuario [TEC03].

Para complementar esta sección, se incluyen los siguientes apartados (que no corresponden a los MT):

2.5.5. Teoría del "Planear, hacer, revisar y actuar" (Plan, do, check, act)

Esta teoría se basa en los cuatro puntos mencionados:

- Planear: consiste en la etapa de planificación de los casos de prueba.
- Hacer: en esta fase se elaboran los casos de prueba.
- Revisar: es una evaluación previa de los casos diseñados antes de su ejecución.
- Actuar: consiste en la ejecución de los planes de testeo planificados anteriormente.

Estas tareas son realizadas en un ciclo permanente. Una vez terminada la fase de "actuar", se procede nuevamente a la de planear. Estas actividades son realizadas a lo largo de todo el proceso de desarrollo del software [LEW00].

2.5.6. Otros

Como las propuestas presentadas en este apartado, existen otros modelos y estándares menos conocidos pero útiles y que son aplicados en algunas empresas, como es el caso del Modelo McCall [CER00].

No existe un modelo perfecto, ninguno se acomoda al cien por ciento a las necesidades de una empresa y mucho menos garantiza ser la solución a sus problemas de calidad.

Cada modelo plantea un objetivo ideal y común, que debe ser moldeado de acuerdo a los problemas que se presenten en la organización.

> *"Todos los modelos son erróneos; algunos modelos son útiles"* George Box. [GAR01].

2.6. El costo de la calidad

La implementación de metodologías de control de calidad en una empresa acarrea su propio costo, que por lo general es elevado. Sin embargo, al ser prácticas que pretenden mejorar el rendimiento de los recursos invertidos, este costo resulta ser mucho menor al que la empresa asume cuando se trata de reparar fallas graves detectadas en las últimas fases del desarrollo o, aun peor, las reportadas por los clientes.

Estos costos están asociados con la prevención, la evaluación y la corrección de fallas.

Los costos de prevención involucran:

- Planificación de la calidad.
- Revisiones técnicas.
- Equipo de pruebas.
- Formación (capacitación).

Los costos de evaluación son los resultantes de las inspecciones en los procesos, el mantenimiento de equipos y las pruebas.

Finalmente, los costos de fallos son aquellos que no existirían si el producto final careciera de defectos. Estos costos pueden ser internos si los defectos son detectados antes de la entrega del producto y externos si son reportados por los clientes una vez que el producto ha sido entregado.

Los costos internos incluyen los costos de revisión y reparación.

En el caso de los fallos externos los costos se elevan puesto que implican además de las reparaciones, los costos de la devolución y sustitución de los productos.

Los costos de fallos se elevan a medida que el proyecto llega a sus últimas fases. Estudios realizados en base a experiencias de empresas como IBM demuestran este hecho [PRE02].

Impacto de los defectos del software sobre el costo

Una serie de estudios realizados por empresas como Nippon Electric y Mitre Corp. demostró que entre el 50 y 65 por ciento del total de errores se producen durante la etapa del diseño y que las actividades de revisión formales son efectivas en un 75 por ciento a la hora de detectar estos errores, lo que reduce sustancialmente el costo en los pasos siguientes [PRE02].

La figura 2.7 muestra el resultado de estos estudios (incluido el realizado a IBM), demostrando las proporciones del costo de arrastrar un error hasta las últimas fases.

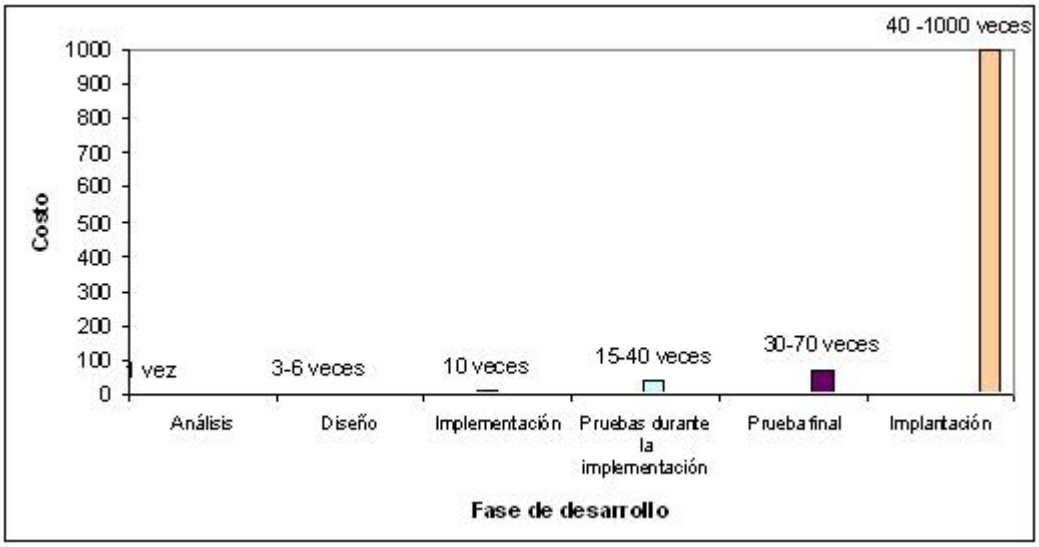


Figura 2.7: Costo relativo de corregir un error [PRE02].

2.7. Los beneficios de la calidad

Al tiempo que la implementación de métodos de control de calidad ayuda a mitigar (o en el mejor de los casos a eliminar por completo) los problemas descritos en apartados anteriores, la presencia de procesos maduros y prácticas de calidad formales en una empresa proporciona, entre otros, los siguientes beneficios [ASQ03]:

- En los empleados: la satisfacción de efectuar un buen trabajo, la mejora de la comu-

nicación interna, una mejor organización del personal, la reducción e incluso eliminación de la necesidad de realizar esfuerzos extra para terminar un proyecto.

- En la empresa: el uso eficiente de los recursos, la reducción de costos, la garantía de realizar buenas estimaciones, la seguridad de obtener resultados positivos al finalizar un proyecto.
- En los clientes: la confianza de que el producto adquirido satisface sus necesidades.
- En la sociedad: el crecimiento de la industria del software, lo que permite abrir el mercado y atraer a clientes tanto nacionales como internacionales y la reducción de la dependencia por el software externo.

CAPÍTULO 3

ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE

El Aseguramiento de la Calidad del Software (Software Quality Assurance, SQA) es el conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza de que el producto de software cumplirá con los requisitos dados de calidad. Es diseñado antes del desarrollo del proyecto y está presente en:

- Métodos y herramientas (de análisis, diseño, implementación y pruebas).
- Inspecciones técnicas formales en todos los pasos del proceso de desarrollo del producto.
- Estrategias de prueba.
- Control de la documentación del software y de los cambios realizados en el mismo.
- Procedimientos que permiten ajustarse a los estándares de calidad.
- Métricas.
- Informes realizados por las personas involucradas en el desarrollo (analistas, desarrolladores, testers, etc.).

Las actividades de las que el SQA está conformado son básicamente:

- Verificación y validación: la verificación es la actividad encargada de evaluar si el producto cumple con el desarrollo planificado ("se está construyendo adecuadamente") mientras que la validación comprueba si el software que se está construyendo es el correcto para las necesidades planteadas ("se está construyendo el adecuado"). Ambas tareas van de la mano y se aplican en cada fase del desarrollo [LEW00] [GER03].
- Gestión de la configuración del software.

3.1. Actividades del SQA

SQA abarca una amplia gama de actividades asociadas tanto a los desarrolladores como al propio equipo de calidad de la empresa.

El equipo de desarrolladores es responsable de aplicar métodos planificados y técnicas de revisión formales durante su trabajo. El equipo de SQA intenta ayudar al primero en la obtención de un software de alta calidad mediante la aplicación de técnicas de SQA como [PRE02]:

- Establecimiento de un plan de SQA para cada proyecto, el mismo que es revisado por todo el personal involucrado. Rige todas las actividades de ambos equipos. El plan define criterios como:
  - Las evaluaciones a realizar.
  - Los estándares a aplicar.
  - Los procedimientos a seguir para la documentación de los errores.
  - El *feedback* del proyecto.
- Participación en el desarrollo de la descripción del proceso de software en el proyecto.
- Revisión de las actividades de ingeniería del software para verificar que se ajusten al proceso de software definido.
- Asegurar que las desviaciones del trabajo y los productos se documenten y manejen de acuerdo con un procedimiento establecido.
- Registrar lo que no se ajuste a los requisitos e informar a los supervisores.

Revisiones Técnicas Formales

Una Revisión Técnica Formal (RTF) es una actividad de SQA que es llevada a cabo por los ingenieros del software y que pretende alcanzar los siguientes objetivos:

- Descubrir errores durante el desarrollo.
- Verificar que el software cumpla con sus requisitos.
- Verificar que el software cumpla con los estándares establecidos.
- Hacer que los proyectos sean manejables.
- Conseguir un desarrollo uniforme del proyecto.

Además de esto, una RTF permite promover la seguridad y continuidad permitiendo que todas las personas involucradas se familiaricen con todas las partes del software (ya que puede darse el caso en que una o más personas ignoren algunas) [PRE02].

Una RTF viene acompañada de las reuniones de revisión, donde tanto desarrolladores como ingenieros de calidad se concentran en la revisión de una etapa completa del desarrollo del producto o, si la complejidad lo amerita, en la etapa de desarrollo de un determinado módulo. Por ejemplo, una reunión puede estar destinada a la revisión del diseño de un solo módulo del sistema [PRE02].

Es en estas reuniones (denominadas también por algunos "War Meetings"[1]) donde los ingenieros de calidad y los desarrolladores deciden qué errores se corrigen y cuáles se omiten momentáneamente, puesto que existen errores cuya severidad no amerita la inversión de tiempo que requiere su reparación y mucho menos la postergación de otras tareas más importantes para concentrar al personal en resolverlo. Si bien el objetivo de estas reuniones es reducir al máximo la cantidad de errores que se encuentren, también debe haber un balance porque de todas formas siempre existirán defectos y la tarea de eliminarlos puede convertirse en algo eterno y a la larga perjudicial para la empresa.

Es necesario que todo el personal involucrado esté consciente de que el software perfecto no existe y que lo que se busca es que el producto final tenga la menor cantidad de defectos posible sin invertir más recursos de los destinados al proyecto.

3.2. SQA en las distintas fases del desarrollo

Como se dijo anteriormente, el control de calidad es una tarea que se realiza a lo largo de todo el proceso de desarrollo del producto de software. Es así que para cada fase de este proceso se tienen algunas consideraciones.

Herramientas

Para un efectivo control de calidad durante todas las fases del desarrollo, existen herramientas bastante recomendables. Algunas de ellas se describen a continuación:

[1] Extraído de la charla de Dr. Pavisic: "Tópicos de ingeniería del software" que tuvo lugar en mayo de 2003 en la Universidad Católica Boliviana, Regional Cochabamba.

- **Inspecciones**

  Esta es una técnica utilizada para evaluar permanentemente el avance de la documentación, el cumplimiento de los requerimientos y el apremio de la solución [LEW00].

- **Casos de Prueba**

  Son una herramienta bastante efectiva y común. Cada caso consta de especificaciones que el tester debe seguir para realizar una prueba específica sobre una funcionalidad determinada del sistema. Una vez hecha la prueba, se determina si el caso pasó o falló. De ocurrir lo último, se registra el defecto y se comunica el resultado al desarrollador para que éste haga las correcciones debidas.

  Un modelo de caso de prueba se presenta en el Anexo B.

- **Listas de Control**

  Más conocidas como *Checklists*, constan de una serie de preguntas que tienen como objetivo asegurar que tanto los requerimientos como la metodología se estén cumpliendo.

  Cada defecto que sea detectado mediante estas listas debe ser documentado [LEW00].

  Tanto para la elaboración del checklist como para la documentación de los defectos, se propone la utilización de herramientas como la que se muestran en el Anexo B, que combinan ambos reportes en uno solo.

- **Matriz de seguimiento de requisitos**

  Una matriz de seguimiento de requisitos es un documento que sirve para hacer un control de los requerimientos del software desde la etapa de análisis hasta la de implementación. Es utilizada para verificar que ningún requisito se haya "perdido" en el camino, que no existan características innecesarias y que la documentación esté completa [LEW00].

- **Plan de Testeo**

  Este documento, basado en el de especificación de requerimientos, se elabora durante todas las fases para verificar que el sistema cumple con los criterios de aceptación del usuario [LEW00].

  Un modelo de este documento es presentado en el Anexo C.

a) Fase de requerimientos

El control de calidad debe comenzar desde las fases más tempranas del desarrollo, no sólo durante la tradicional fase de pruebas donde pueden aparecer errores cuyo costo (en recursos humanos, tiempo y dinero) es elevado y puede evitarse si tales errores son detectados en fases previas.

Ésta es quizá la fase más importante en términos de calidad ya que durante las demás fases del desarrollo se busca que los requerimientos se cumplan; por lo tanto, malos requerimientos derivarán en un mal producto.

Los requerimientos pobres incluyen [LEW00]:

- Funciones parcialmente definidas.
- Omisión del performance.
- Requerimientos ambiguos, contradictorios o redundantes.
- Interfaces no documentadas.
- Requerimientos demasiado restrictivos (inflexibles).

Las secciones más importantes en una especificación son [LEW00]:

- Funcionalidad: el conjunto de tareas que el sistema debe ser capaz de realizar.
- Descripción de los datos.
- Descripción de las interfaces entre la función y las entidades externas.

Un modelo del documento de especificación de requerimientos se presenta en el Anexo C.

Plan de testeo - Prueba de aceptación

Para la ejecución de esta prueba, se realiza un testeo de caja negra[2] y, por lo general, es el usuario final quien participa activamente de ella. Naturalmente, se espera que para hacer este tipo de testeo el producto esté implementado o por lo menos una parte de él, lo que implica que el equipo de desarrollo se encuentre trabajando en las últimas fases. aun así es importante comenzar la construcción de esta prueba desde la fase de requerimientos y que el usuario evalúe si sus requerimientos se están cumpliendo o no [LEW00].

Una herramienta que facilita la ejecución de esta prueba es la Matriz de Testeo de Especificaciones, que se muestra a manera de ejemplo en la figura 3.1. Además, esta matriz presenta beneficios adicionales como [LEW00]:

- Permite establecer una correlación entre las pruebas y la documentación y los requisitos.
- Facilita el mantenimiento de los documentos de revisión.
- Es una herramienta que permite el seguimiento del desarrollo durante todas las etapas del mismo, incluyendo la implantación.

b) Fase de Diseño

Ésta es quizá la etapa que se encuentra más comprometida con la calidad del producto porque su resultado será la base para la implementación. Prácticamente todo diagrama, esquema o documento resultante de esta fase será convertido en código por los programadores, por lo que se debe hacer todo lo posible para que éstos carezcan de errores.

Se sabe que del 50 al 65 por ciento de los errores que se detectan en los productos de software provienen de su diseño, por lo que dedicar un énfasis especial a esta etapa puede garantizar una reducción (más adelante) del esfuerzo que se emplea para corregir los defectos que aparecerán [PRE02].

[2]Conjunto de pruebas donde lo único que interesa es verificar la funcionalidad y no así la manera en que el programa ejecuta las tareas.

Prueba	Caso de testeo									Comentario
Requerimiento	1	2	3	4	5	6	7	8	9	
Funcional										
1										
2		C					T			
3										
4	C			C						
Performance										
1	T									
2	U						T			
3										
Seguridad										
1			U							
2				C						

T – listo para ser testeado.
U – revisado por el usuario
C – revisado por el equipo de calidad

Figura 3.1: Matriz de testeo de especificaciones [LEW00].

En el apartado 2.6 se explica además porqué a nivel económico es importante evitar el acarreo de los defectos que aparecen en las fases tempranas.

Esta fase se divide en tres: el diseño lógico, el diseño físico y el diseño de unidad de la aplicación.

b.1) Evaluación de la fase de Diseño Lógico

Esta fase especifica lo dicho en los requerimientos funcionales, los requisitos que tienen que ver directamente con la lógica del negocio.

El diseño lógico de una aplicación establece un marco de trabajo (framework) detallado para la construcción del sistema. Los tres conceptos más importantes de este framework son el modelo de datos, el modelo del proceso y el enlace entre ambos [LEW00].

El modelo de datos es una representación de la información (el tipo de cada uno de los datos involucrados) necesaria para la aplicación. Permite definir tanto las entidades como las relaciones presentes en la aplicación [LEW00].

El modelo del proceso es una descomposición del negocio, es decir, es el desglose detallado de las actividades involucradas, desde su nivel más abstracto hasta el más

elemental.

El enlace entre ambos modelos se realiza mediante una representación gráfica de toda la lógica del negocio. Una de las técnicas para esta representación es la Matriz CRUD (para el detalle de cómo funciona consultar el Anexo D). Esta técnica puede ser además utilizada con fines de testeo, utilizando su lógica para formular una tabla como la que se muestra en la figura 3.2.

Objeto	C	R	U	D	Tester	Fecha	Comentario
Orden de Venta	✓			✗			
Orden de Compra		✗		✓			
Orden de Pago	✓	✓	✗				
...							

Figura 3.2: Matriz CRUD para el testeo [LEW00].

Plan de testeo - Prueba del sistema

En esta sección se prosigue con la elaboración del plan de prueba de aceptación que se inició en la fase anterior.

En esta fase se verifica que la lógica de la aplicación responda a los requisitos dados.

Para facilitar su ejecución, se pueden utilizar herramientas como la propuesta para la fase de requerimientos (figura 3.1) [LEW00].

b.2) Evaluación de la fase de Diseño Físico

Esta fase especifica la manera en que los requerimientos pueden ser automatizados, es decir, en esta etapa se diseña la arquitectura del sistema.

Mientras que el diseño lógico es funcional, el diseño físico es estructural y depende directamente del primero; por lo tanto, para esta fase, se asume que el diseño lógico es correcto. Algunas técnicas utilizadas para su representación son los denominados Charts y los Diagramas de Flujo de Datos (consultar Anexo D para su detalle). Estos esquemas proveen mecanismos para la especificación de los algoritmos a utilizar en algunos módulos del sistema. Es muy probable la presencia de inconsistencias en esta fase, sobretodo en la

especificación del flujo de la información entre los módulos [LEW00].

Plan de testeo - Prueba de integración

El testeo en esta etapa se realiza mediante revisiones técnicas estáticas, las cuales verifican que la arquitectura respete las convenciones establecidas, que el flujo de datos entre módulos no tenga inconsistencias y que la descomposición tanto de la información como de los procesos carezca de errores [LEW00].

b.3) Evaluación de la fase de Diseño de la Unidad del Programa

Esta fase corresponde al diseño detallado de la aplicación en base a los dos diseños anteriores y es donde los algoritmos y estructuras de datos son elegidos. Además, especifica el flujo de control que hará que el diseño sea fácilmente traducible a un lenguaje de programación específico [LEW00].

Plan de testeo - Prueba de Unidad

El testeo en esta etapa es estratégico ya que es el último que se hace al diseño antes de proceder a la implementación. Si existen errores que no son corregidos, al ser traducidos a código causarán defectos importantes (incluso de grandes proporciones) que tarde o temprano ocasionarán problemas al equipo de desarrollo y tiempos extra de implementación para su corrección [LEW00].

c) Fase de Implementación

Ésta es la fase donde el resultado del diseño se traduce en código de un determinado lenguaje de programación; por lo tanto, depende enteramente de las decisiones que hayan sido tomadas previamente.

La traducción a un lenguaje de programación es casi mecánica si el diseño es bueno. El desafío entonces para los programadores es garantizar que el código producido sea robusto y fácilmente mantenible a corto y largo plazo.

Plan de testeo - Conclusión

El control de calidad en esta etapa se realiza mediante revisiones técnicas.
Todo lo efectuado en el plan de testeo durante las fases previas se completa en ésta. Para esto, en cada fase se debe elaborar un conjunto de casos de prueba que serán verificados cuando se termine la fase de implementación [LEW00].

d) Fase de pruebas

En esta sección se realizan todas las pruebas posibles al sistema. No solo se verifica que éste cumpla con los requerimientos dados explícitamente sino también con los implícitos como la mantenibilidad, seguridad, exactitud, etc.

3.3. SQA y la herramienta desarrollada

Para el presente proyecto, se desarrolló una herramienta que, respondiendo a la teoría de SQA:

- Permite la aplicación de técnicas de control de calidad a través de todo el proceso de desarrollo de software. Es decir, la ejecución de pruebas en todas las fases del desarrollo.
- Permite la ejecución de algunas herramientas del SQA ya mencionadas: casos de prueba y listas de control.
- Permite la elaboración de estrategias de prueba.
- Permite el control de la documentación del software y de los cambios realizados en el mismo (de manera básica).
- Permite la definición de procedimientos básicos que a la larga contribuyan en el ajuste a los estándares de calidad.
- Permite la aplicación de métricas (expresadas en los casos de prueba y checklists) durante el control de calidad de los productos.

CAPÍTULO 4

MÉTODOS Y HERRAMIENTAS

4.1. Modelo de arquitectura MVC

La arquitectura MVC (Model View Controller) fue introducida como parte de la versión SmallTalk-80 del lenguaje de programación SmallTalk. Fue concebida con el objetivo de reducir el esfuerzo que es invertido durante la implementación de los sistemas múltiples y sincronizados con los mismos datos [BUR92].

Como su nombre indica, posee tres partes esenciales: el Modelo, la Vista y el Controlador. Cada una es una entidad independiente, lo que permite [BUR92] [APA03]:

- Establecer una clara separación entre los componentes de la aplicación, lo que a su vez permite implementarlos por separado.
- Hay un API bien definido; teóricamente, si se lo utiliza no existe ningún problema en hacer un reemplazo de cualquiera de las partes en cualquier momento.
- La conexión entre el View y el Model es dinámica, es decir, en tiempo de ejecución.

Modelo

Encapsula la lógica del negocio de la aplicación. Es el objeto que representa los datos del sistema. Maneja la información y sus transformaciones. No posee ningún conocimiento del Controlador o de la Vista y ni siquiera hace referencia a ellos (sobre este punto se detallará más adelante) [ANT03] [BUR92] [GOO02].

Internamente se divide en dos grandes subsistemas: Internal State (estado interno del sistema, de los objetos) y Actions, estas últimas son las que pueden cambiar al primero [APA03].

Una analogía clara que permite entender estos subsistemas es la de los objetos y los verbos en la gramática. En este caso el Internal State es el estado de cada uno de los

objetos del sistema y los Actions son los verbos que son capaces de cambiar el estado de estos objetos [APA03].

Vista

Es el objeto encargado de manejar las vistas de la aplicación. Genera una representación visual para el usuario de la información recibida desde el Modelo, con el que interactúa mediante una referencia directa [APA03] [GOO02].

Controlador

Es la parte de la aplicación que da sentido a las solicitudes del usuario. Todo el flujo de la aplicación está dirigido por él. Es el encargado de delegar cada solicitud a un manejador apropiado. Estos manejadores están unidos a la entidad Modelo [APA03].

Esta comunicación entre las capas del modelo capas se encuentra ilustrado en la figura 4.1.

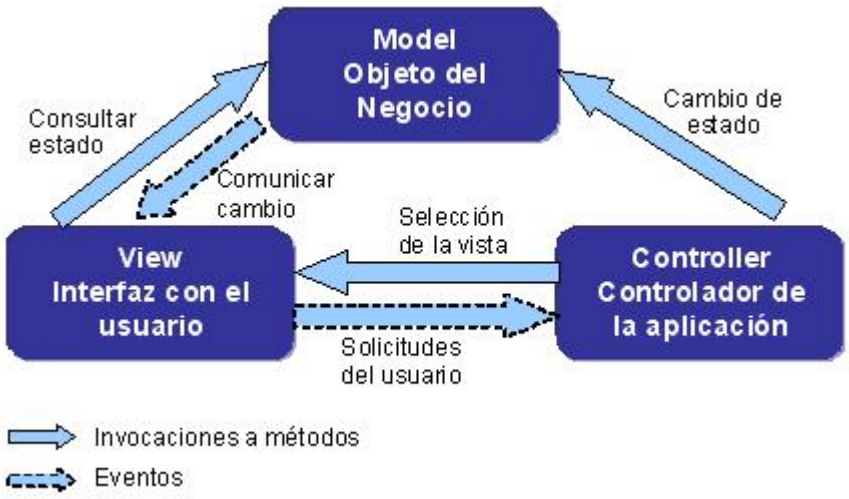


Figura 4.1: Arquitectura MVC [ANT03].

4.2. Metodología

Para el desarrollo de la "Herramienta Web de apoyo al control de la calidad del software", se utilizó la metodología RUP (Rational Unified Process).

Características del RUP [IBA03]:

- Es una guía que indica cómo utilizar de manera efectiva el UML (Unified Modeling Language).
- Es una metodología que envuelve una serie de prácticas utilizadas en el desarrollo moderno de software de una forma que es aplicable a una amplia gama de proyectos y organizaciones.
- Provee a cada miembro del equipo de desarrollo (analistas, diseñadores, desarrolladores, testers, etc.) un fácil acceso a una base de conocimientos con guías y herramientas para todas las actividades críticas de desarrollo.
- Crea y mantiene modelos que pueden ser aplicados en proyectos futuros en lugar de enfocarse en la producción de una gran cantidad de papeles de documentación.

RUP describe además la manera en que el equipo de desarrollo puede utilizar los procedimientos comerciales probados en el desarrollo de aplicaciones, conocidos como "mejores prácticas":

- Administración de requerimientos.
- Desarrollo iterativo.
- Modelamiento visual.
- Verificación de calidad.
- Arquitectura de componentes.
- Control de cambios.

Con estas características, RUP ofrece un incremento en la productividad en el trabajo tanto individual como grupal [IBA03].

Fases en RUP

RUP consta de cuatro fases [IBA03] [RAM01]:

1. Inicio

Su propósito es el de establecer los casos de negocio (casos de uso) para el nuevo sistema y el alcance del proyecto.

El resultado es una visión general de los requerimientos del proyecto y un caso de uso inicial que refleja la evaluación inicial de riesgos y la estimación de los recursos requeridos [IBA03].

2. Elaboración

Los objetivos de esta etapa son: el análisis del dominio del problema, el establecimiento de una buena arquitectura, el tratamiento de los elementos de riesgo más altos y la elaboración de un plan donde se muestre cómo el proyecto será finalizado [IBA03].

Los resultados se muestran reflejados en: un modelo del dominio de casos de uso completado en un ochenta por ciento, una lista de requerimientos suplementarios que capturan los requerimientos no funcionales y otros que no están asociados con un caso de uso específico y una lista de riesgos [IBA03].

3. Construcción

En esta fase tiene lugar la implementación del sistema.

Su resultado es el diseño completo de la aplicación, el producto desarrollado, la documentación pertinente y una liberación "beta" del producto [IBA03].

4. Transición

En esta fase se transfiere el producto al usuario (aplicación, manuales y documentación) y se hace una evaluación de los resultados midiendo el grado de satisfacción del usuario. Además, se realiza una evaluación interna en la empresa donde se determina si los gastos estimados resultaron siendo próximos o no a los gastos reales [IBA03].

La figura 4.2 muestra la distribución de trabajo de las fases de RUP con relación a las

etapas del desarrollo de una aplicación.

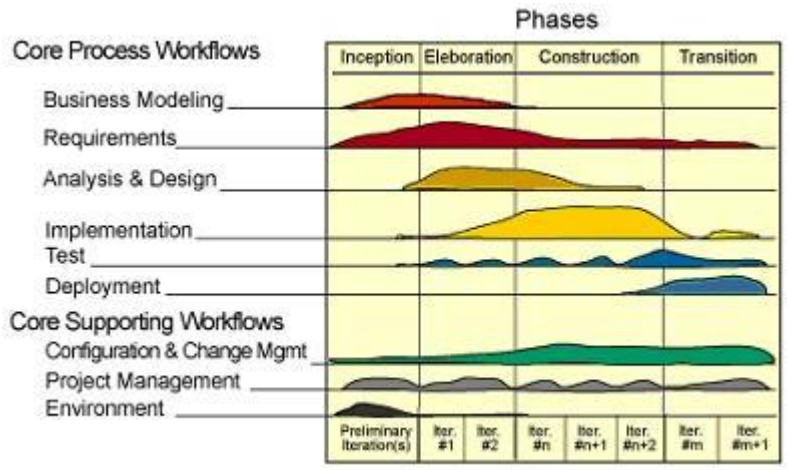


Figura 4.2: Fases de RUP y etapas del desarrollo de una aplicación [EVE03].

Iteraciones

Cada fase de RUP puede ser descompuesta en iteraciones. Cada una de estas iteraciones constituye un ciclo completo de la fase, cuyo resultado puede mejorarse siendo sometido a una nueva iteración u otorgar un producto útil para su empleo en la siguiente fase [IBA03].

Las iteraciones también pueden ser utilizadas para dividir el sistema en módulos, teniendo como objetivo la obtención de un módulo completo al final de cada iteración [RAM01].

Subproductos

Los subproductos (o artefactos como se denominan en RUP) que se obtienen de cada fase son los siguientes [RAM01]:

- **Fase: Inicio**

    1. Alcance del Sistema
    2. Arquitectura inicial
    3. Lista inicial de riesgos
    4. Entorno de desarrollo configurado

5. Plan inicial del proyecto
6. Caso inicial del negocio

- **Fase: Elaboración**

1. Contexto del sistema
2. Captura del 80 % de los requerimientos funcionales
3. Arquitectura de referencia
4. Lista de riesgos
5. Entorno de desarrollo adecuado
6. Caso del negocio completo

- **Fase: Construcción**

1. Modelos completos
2. Arquitectura íntegra
3. Riesgos mitigados
4. Manual inicial del usuario
5. Prototipo operacional (beta)
6. Caso del negocio actualizado

- **Fase: Transición**

1. Prototipo operacional
2. Documentos legales
3. Descripción de la arquitectura completa y corregida
4. Manuales

CAPÍTULO 5

ANÁLISIS

Como resultado de esta fase se pretende obtener los requerimientos tanto funcionales como no funcionales (recursos) con los que deberá contar la herramienta propuesta, para que a partir de éstos se pueda proceder al diseño e implementación.

5.1. Definición del problema

El control de la calidad del software no es una tarea fácil, se requiere de herramientas que ayuden a los expertos en su labor. Por eso, se plantea resolver una parte del problema desarrollando una herramienta de apoyo al control de la calidad del software.

El sistema debe ser capaz de ayudar a una empresa en:

- La coordinación de grupos de trabajo.
- La distinción de usuarios según su grupo (rol).
- La delegación de responsabilidades a los distintos grupos.
- El manejo de pruebas (herramientas del SQA) y resultados.
- El control de asignación de tareas.
- El mecanismo de alertas.
- El control de pruebas iterativas sobre un mismo trozo de la aplicación (fases del proyecto, módulos del proyecto, fases de los módulos ó trozos de código).
- La definición del nivel de detalle en las pruebas.
- La atención de aportes/observaciones del cliente.

5.2. Tabla de riesgos

Este apartado señala todos los posibles riesgos a los que el proyecto está sujeto, indicando el tipo de riesgo, la magnitud y el plan de contingencia. La magnitud puede tomar un valor de 1 a 10 dependiendo del grado del problema.

Magnitud	Descripción	Estrategia de mitigación / Plan de contingencia
9	Desarrollo no finalizado a tiempo.	Realizar cambios que reduzcan la cantidad de trabajo por realizar.
6	Problemas con el uso de la tecnología.	Recurrir a libros y/o sitios de Internet.
5	Falla a nivel de S.O. o DBMS al implantar la herramienta.	Realizar pruebas durante la implementación en el S.O. y el DBMS.

Cuadro 5.1: Tabla de riesgos

5.3. Usuarios del sistema

Se distinguen cinco tipos de usuarios:

- Usuario Administrador: que tendrá la capacidad de gestionar las entidades más importantes del sistema como proyectos, usuarios y clientes.
- Usuario Líder de proyectos: es el usuario que será responsable del control y supervisión del avance de los proyectos que tenga a su cargo.
- Usuario Tester: es el usuario encargado del control de calidad como tal. Es quien podrá ejecutar los casos de prueba y listas de control y reportar los resultados.
- Usuario Desarrollador: que recibirá del tester el reporte del resultado del control de calidad y será responsable de la corrección de los defectos que el tester le comunique mediante la herramienta.
- Usuario Cliente: es el dueño del sistema, quien podrá -en las aquellas fases que el administrador y líder de proyectos le permitan- aportar sus observaciones a medida que el sistema es desarrollado.

5.4. Requerimientos

5.4.1. Requerimientos funcionales

Los requerimientos funcionales definen todas las tareas que el sistema debe ser capaz de hacer (pero no cómo las hará, para ello está el diseño). La definición de estos

requerimientos es fundamental para el desarrollo del sistema ya que establecerá los límites de aquello que queda dentro y fuera del producto.

La herramienta del UML que permite la identificación de los requerimientos funcionales de manera sencilla son los diagramas de casos de uso.

De los diagramas presentados en las figuras 5.1-5.8 (y de su detalle expuesto en el Anexo D), se rescatan los siguientes requerimientos funcionales de acuerdo al tipo de usuario:

Actor general

El siguiente diagrama se aplica a administradores, líderes de proyectos, testers y desarrolladores.

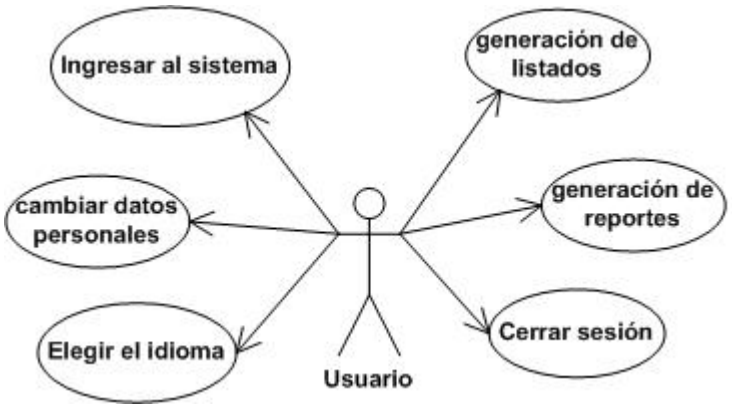


Figura 5.1: Diagrama de casos de uso para un usuario genérico de la herramienta.

- Ingreso al sistema, incluyendo la validación de los datos que el usuario ingrese.
- Elección del idioma con el que desea trabajar.
- Modificación sus datos personales (como la contraseña). A manera de ejemplo, la figura 5.2 muestra este caso de uso detallado.
- Generación de listas de datos.
- Generación de reportes estadísticos que estarán disponibles según el rol del usuario.
- Cerrar la sesión que inició al ingresar al sistema.

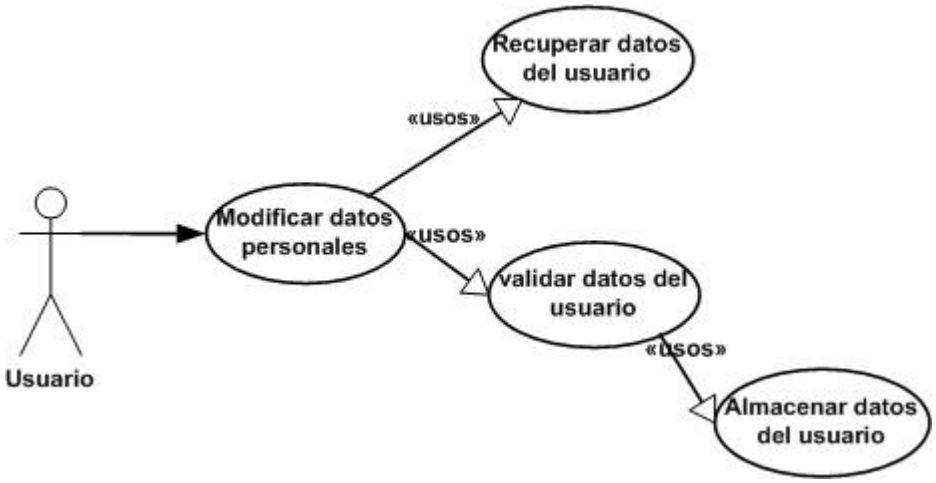


Figura 5.2: Caso de Uso: Modificar datos personales.

Actor administrador

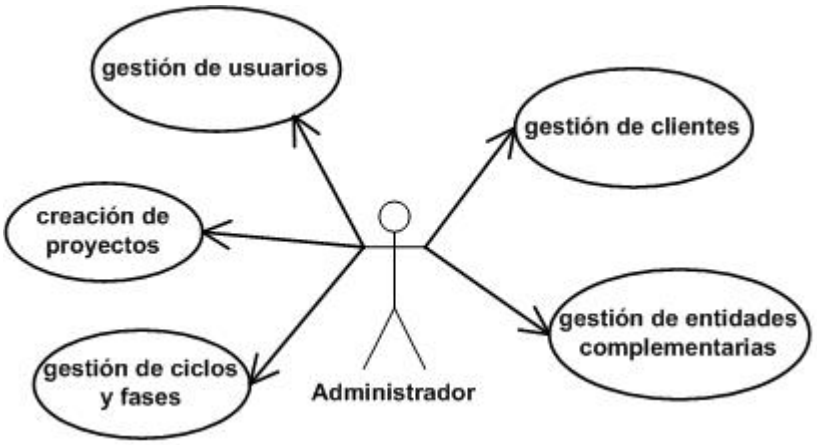


Figura 5.3: Diagrama general de casos de uso para un usuario de tipo Administrador.

- Gestión de usuarios: creación, incluyendo la asignación de un tipo (administrador, líder de proyectos, tester, desarrollador o cliente), modificación de datos, habilitación, bloqueo y eliminación de cuentas.
- Creación de proyectos lo que incluye la elección del ciclo de vida y asignación de: líder del proyecto, responsable del control de calidad y responsable del desarrollo.
- Gestión de clientes.
- Administración de ciclos de vida y fases.
- Gestión de las entidades secundarias como estados, prioridades, niveles, entornos, etc.

Actor líder de proyectos

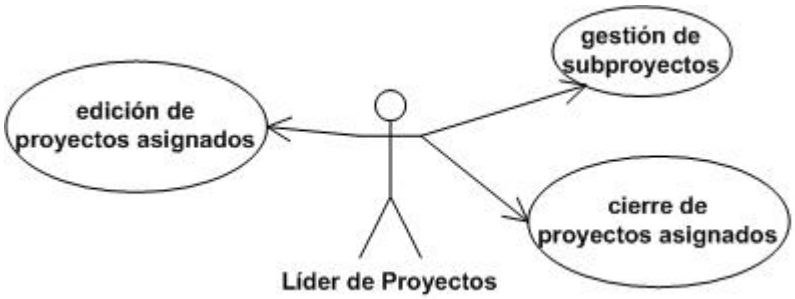


Figura 5.4: Diagrama general de casos de uso para un usuario de tipo Líder de Proyecto.

- Gestión de subproyectos, lo que incluye la asignación de responsable de control de calidad y desarrollo para cada uno.
- Edición de datos de los proyectos que tiene asignados.
- Cierre de los subproyectos y proyectos del que es responsable.

Actor tester

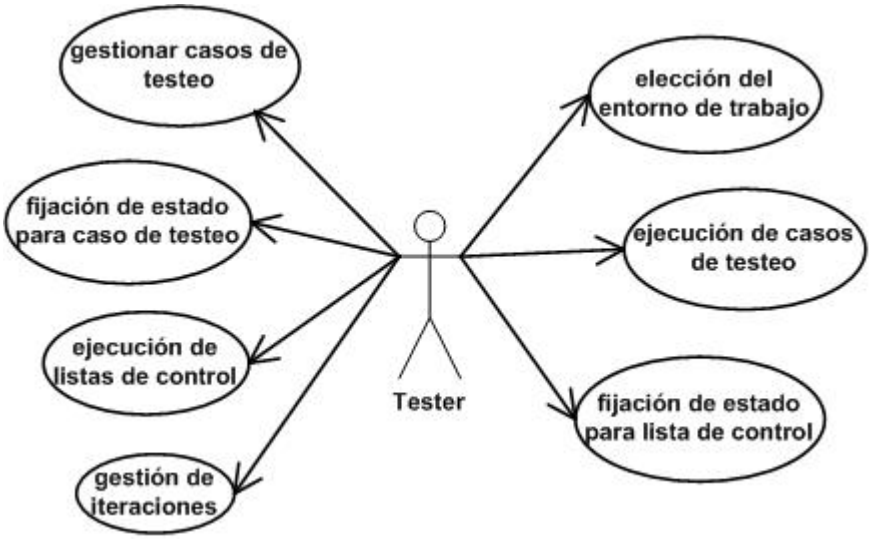


Figura 5.5: Diagrama general de casos de uso para un usuario de tipo Tester.

- Gestión de Casos de Prueba (creación, modificación y eliminación).
- Gestión de Listas de Control (creación, modificación y eliminación).

- Gestión de Iteraciones, que incluye la asignación de Casos de Prueba y Listas de Control durante su creación, así como también la adición y retiro de éstas herramientas durante la ejecución de las iteraciones.
- Ejecución de los Casos de Prueba (C.P.) y Listas de Control (L.C.) en las iteraciones.
- Elección de un estado resultante de la ejecución de cada C.P. y cada L.C. en las iteraciones.
- Asignación de tareas a desarrolladores cuando la ejecución de C.P. y L.C. en las iteraciones terminen con resultados negativos.
- Elección del entorno de trabajo para la ejecución de iteraciones (entorno de hardware y software, versión del sistema, etc.).

Actor desarrollador

Figura 5.6: Diagrama general de casos de uso para un usuario Desarrollador.

- Generación de reportes en caso de que el usuario sea responsable del desarrollo de un subproyecto.
- Listado de las tareas pendientes relacionadas a C.P. y L.C. que dieron resultados negativos al finalizar su ejecución en las iteraciones.
- Consulta del detalle de cada uno de los casos de prueba asignados.
- Cambio de estado de los casos de prueba y listas asignados.

Actor cliente

- Acceso a los proyectos de los que es dueño.
- Adición, modificación y eliminación de observaciones sobre una determinada fase de un subproyecto de un proyecto en particular.

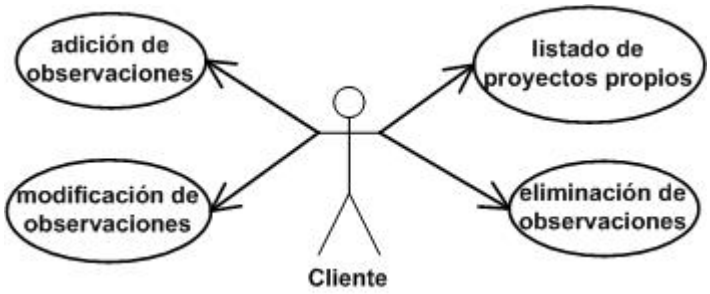


Figura 5.7: Diagrama general de Casos de Uso para un usuario Cliente.

- Listado de las observaciones expuestas previamente.

Actor sistema

A los cinco tipos de usuarios señalados previamente, se añade un sexto que es el Sistema, al cual se le atribuyen funciones que deberá efectuar automáticamente.

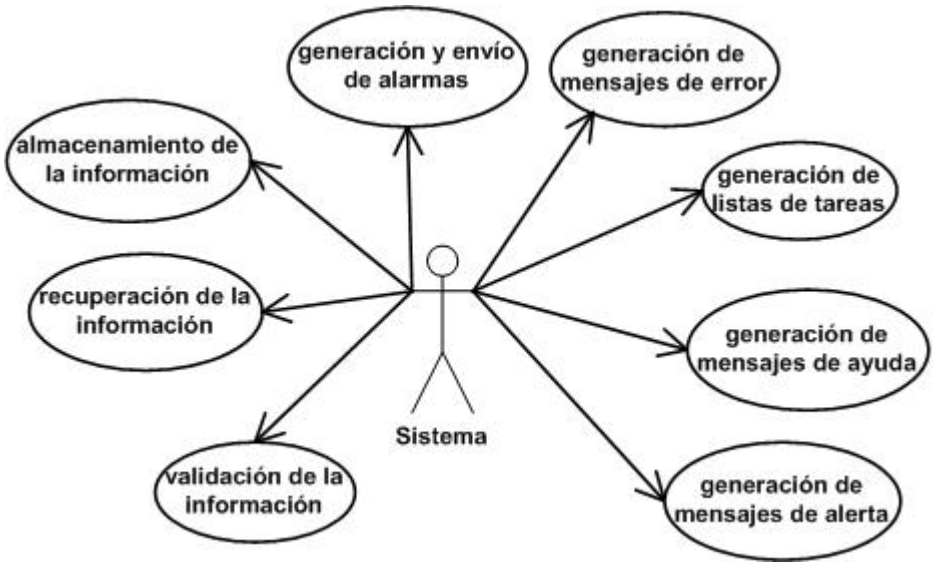


Figura 5.8: Diagrama de casos de uso para la herramienta.

- Generación de alarmas para líderes de proyectos, testers y desarrolladores sobre las tareas pendientes que tiene cada uno.
- Generación de alarmas para líderes de proyectos y administradores cuando los sub-proyectos y proyectos pueden cerrarse.
- Generación de listas sobre las tareas pendientes de cada usuario.
- Envío de alertas mediante distintos medios como el correo electrónico.

- Generación de mensajes al usuario en los casos en que una fase, un subproyecto o un proyecto estén en condiciones de ser cerrados.
- Control de errores durante el registro de los datos (controlando la integridad de los datos y la coherencia entre ellos).
- Recuperación de la información cuando se requiera (acceso a la Base de Datos y ejecución de las consultas necesarias según la solicitud del Controlador de cada entidad).
- Almacenamiento de la información cuando se le solicite (actualizaciones a las tablas de la Base de Datos como respuesta a solicitudes del Controlador de cada entidad).

5.4.2. Requerimientos no funcionales

Los requerimientos no funcionales definen básicamente las propiedades del sistema, sus restricciones y los factores externos al producto. Las propiedades del sistema pueden ser, por ejemplo, requerimientos implícitos que debe cumplir como la confiabilidad y la exactitud. Las restricciones son las capacidades de los recursos que el sistema utilizará, como la memoria, procesador, etc. [SOM96].

Requerimientos no funcionales del producto

- La herramienta debe contar con una documentación apropiada.
- Requerimientos implícitos (seguridad, robustez, exactitud, etc.).
- La herramienta debe ser fácil de usar y contar con manuales (ver sección 7.4.4) para cada tipo de usuario.

Restricciones

- El sistema deberá ejecutarse en un servidor que tenga los siguientes requerimientos:
  - Sistema Operativo: Windows 98 o Linux 7.0
  - Servidor Web: Apache 1.3 con la distribución de PHP 3.0 incluida, porque existen funciones del lenguaje utilizadas en la aplicación que exigen estas versiones de Apache y PHP como mínimo.
  - Servidor de SMTP instalado para el envío de alertas.
  - Memoria RAM de 128 MB.

- Procesador de 500 Mhz como mínimo. Se debe tomar en cuenta que el servidor atenderá a más de un usuario a la vez y mientras más lento sea el procesador, mayor será el tiempo de respuesta a cada solicitud. También hay que recordar que la aplicación está escrita en PHP y por lo tanto todo el procesamiento se hace del lado del servidor.
- Espacio libre mínimo en disco duro: 59 MB para:
  - Almacenamiento del código fuente de la aplicación: 10 MB.
  - Almacenamiento de la Base de Datos en caso de que ésta resida en el servidor y se utilice MySQL como DBMS: 10 MB.
  - 200 KB de espacio por archivo que sea subido al servidor desde un cliente. Un promedio de 10 archivos por proyecto, y un número aproximado de 20 proyectos hacen un total de 39 MB para el almacenamiento de documentos de los proyectos.

- Cada cliente que se conecte al servidor para hacer uso de la herramienta deberá contar con un mínimo de:
  - Sistema Operativo: Windows 98 o Linux 7.0.
  - Navegador: Internet Explorer 5.0 (si el S.O. es Windows) o Netscape 6.0.
  - Memoria RAM: 64 MB.
  - Procesador: 333 Mhz.
  - Espacio en Disco Duro: opcional según la cantidad de documentos que el usuario desee descargar de la aplicación.
  - Tarjeta de Red 10/100 o bien, una conexión a Internet con una velocidad mínima de transferencia de 50 Kbps (dependiendo del tipo de acceso que el cliente tenga al servidor).
  - Impresora en caso de desear tener los reportes impresos en papel.

5.5. Diagramas de colaboración

Estos diagramas muestran de manera gráfica las interacciones que existen alrededor de los roles. Permiten identificar mediante los denominados clasificadores las potenciales clases del sistema [RUM00].

A manera de ejemplo, se presenta el diagrama de la figura 5.9, que muestra las distintas interacciones (numeradas por orden de la secuencia que se sigue) que existirán entre el usuario, el sistema (en el esquema se presenta a éste desglosado en las tres capas del modelo MVC) y el repositorio de datos. Los clasificadores son:

- La Vista
- El controlador
- El validador
- El Caso de Testeo
- Base de Datos

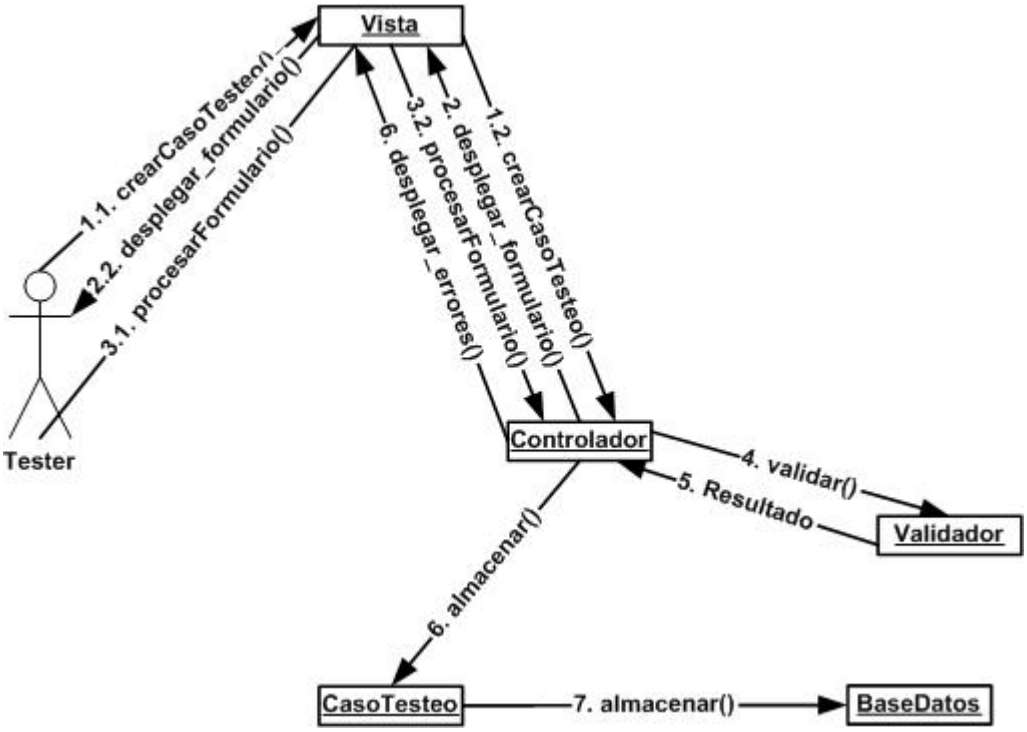


Figura 5.9: Diagrama de colaboración: crear un caso de testeo.

Estos clasificadores son comunes a casi todas las funciones de la aplicación, a excepción del cuarto que es reemplazado por las entidades que se vean involucradas en cada función.

5.6. Diagrama de estados

Muestra los distintos estados por los que pasa un objeto durante la ejecución de una función [RUM00]. El diagrama que se muestra en la figura 5.10 muestra todo el proceso

que se sigue desde que el usuario entra al sistema y elige el proyecto hasta que lo cierra y termina su sesión en la aplicación.

La secuencia que es descrita en el diagrama está explicada "en modo texto" en el algoritmo de la sección 6.4 (Algoritmos).

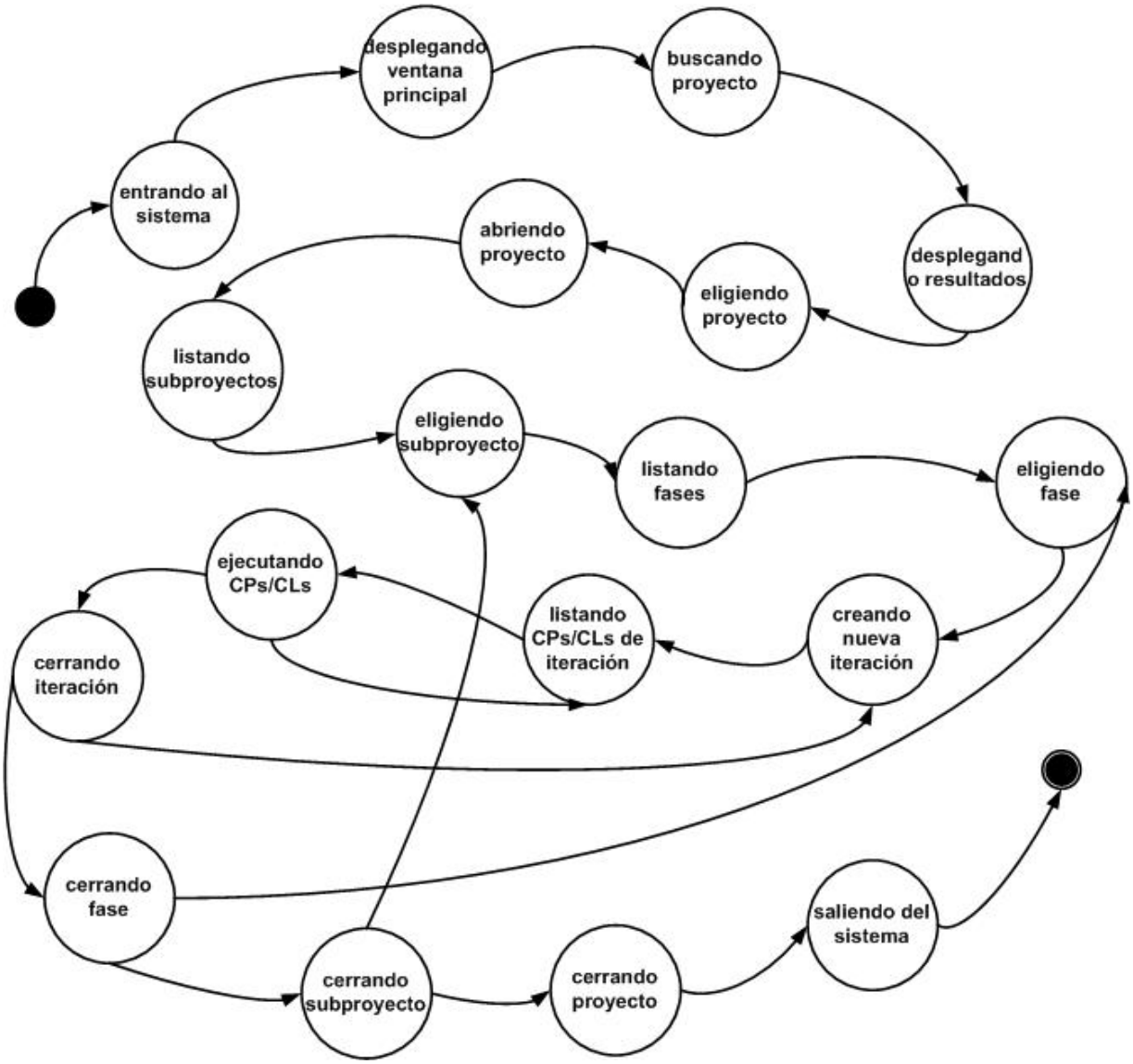


Figura 5.10: Diagrama de estados que ilustra el proceso de control de calidad de un proyecto.

Hasta aquí se hizo el análisis de la herramienta, extrayendo los requerimientos y una visión global de la lógica que ésta debe seguir.

CAPÍTULO 6

DISEÑO

Al finalizar esta etapa del desarrollo se cuenta con los diagramas completos de casos de uso, secuencia, estados, clases y otros complementarios, que den paso finalmente a la fase de implementación.

6.1. Diagramas de secuencia

Como continuación a los diagramas de colaboración y estado obtenidos del análisis es necesario crear los diagramas de secuencia. Éstos muestran las interacciones entre los distintos objetos organizadas en una secuencia temporal, lo que los hace ser en este aspecto diagramas más elaborados que los de Colaboración [RUM00]. Las interacciones son denominadas mensajes.

Los diagramas de secuencia constituyen un puente entre los diagramas de colaboración (donde se presentan los clasificadores o potenciales clases) y el diagrama de clases que será mostrado en la sección 6.2.

El diagrama mostrado en la figura 6.1 corresponde a la función "Ingresar al sistema".

En el diagrama se presentan los clasificadores mencionados en la sección 5.5 (Diagrama de colaboración).

La figura también describe la manera en que trabajan las capas del modelo MVC: Toda solicitud llega primero al Controlador mediante la Vista. Según el tipo de requerimiento, éste (Controller) solicita datos del Modelo. Los resultados de este último definen el tipo de respuesta que el Controlador ordenará a la Vista desplegar.

En el Anexo D se encuentran los diagramas de casos de uso, colaboración, estado y secuencia de las funciones del sistema según el tipo de usuario.

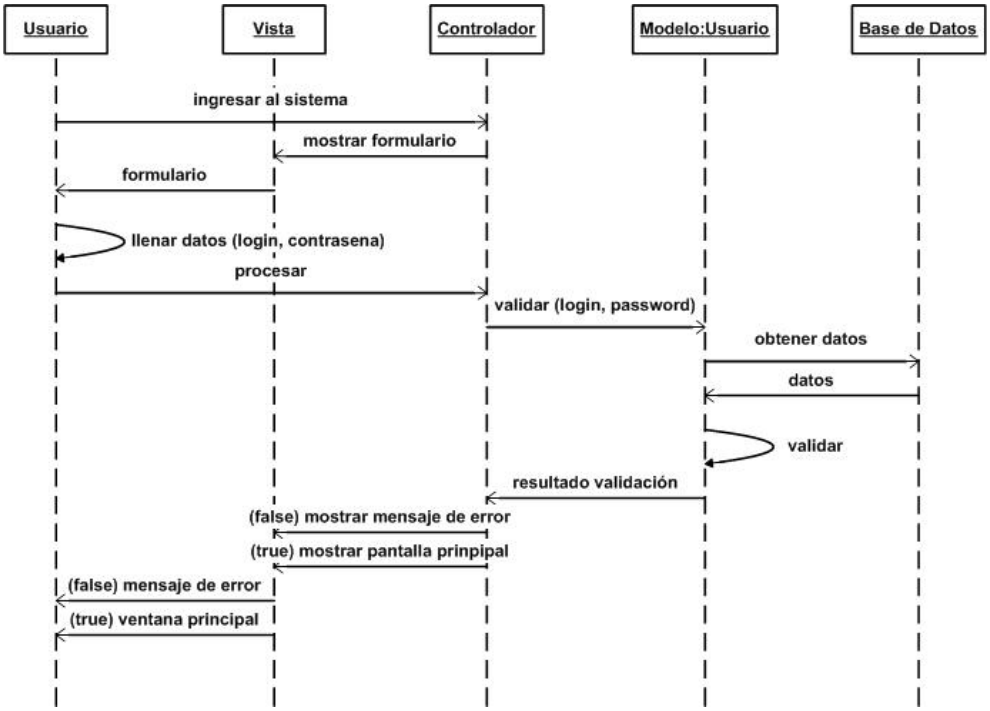


Figura 6.1: Diagrama de secuencia para la función Ingresar al Sistema.

6.2. Clases

El diagrama presenta las clases involucradas en el sistema en términos de estructura y de herencia. Como es de esperarse, la definición de toda clase trae consigo la definición de sus atributos y métodos, lo que aporta mucha información a los diseñadores para proceder a la elaboración de modelos como el presentado en el siguiente apartado (6.3 Diagrama entidad-relación) y de algoritmos [RUM00].

La figura 6.2 muestra las distintas clases involucradas en el sistema y las relaciones entre ellas. Las clases carecen de atributos y métodos en el gráfico con el objetivo de facilitar su visualización; el detalle de cada clase se encuentra en el Anexo E (Diccionario de Clases).

En base a éste diagrama y al de estados mostrado en la sección 5.6 se obtuvo el algoritmo mostrado en el apartado 6.4.

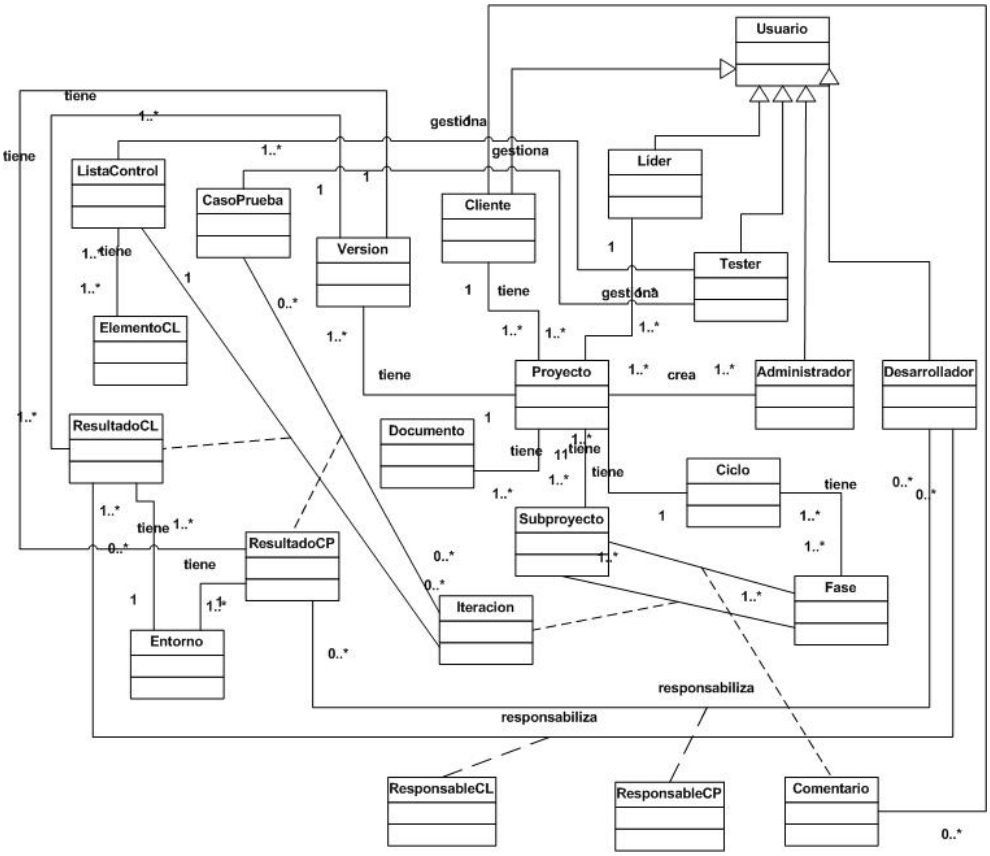


Figura 6.2: Diagrama de clases.

6.3. Diagrama entidad-relación

Este modelo fue obtenido en base al Modelo de Clases con el objetivo de obtener un modelo relacional para la creación de la Base de Datos.

Del diagrama de clases al diagrama E-R

Para hacer la conversión se utilizaron los siguientes criterios (sugeridos por [RUM96]):

- Cada clase se convierte en una o más tablas.
- Cada atributo de una clase se convierte en una columna de la tabla que corresponderá a la entidad. A esto se añaden uno o más campos (o se identifican de los ya existentes) que conformen la clave principal.
- De relaciones múltiples (n:n) surgen tablas intermedias como resultado de la normalización.

La figura 6.3 muestra el diagrama E-R de la aplicación. A pesar de que se obtuvo el modelo de Base de Datos relacional con este diagrama, las clases no desaparecen del Diseño de la aplicación puesto que serán utilizadas posteriormente durante la implementación de la Capa M del modelo MVC (véase el capítulo 7: Implementación).

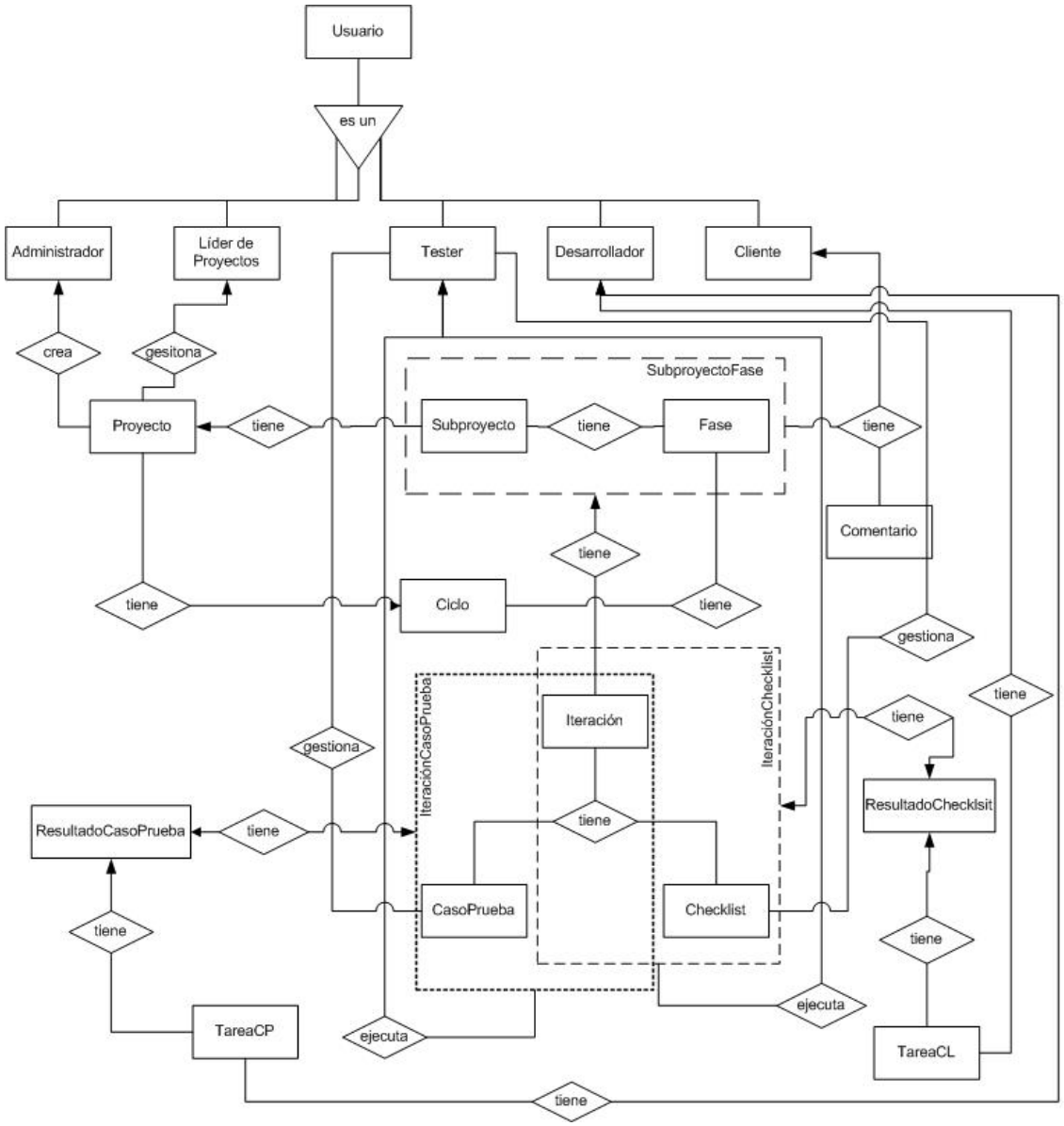


Figura 6.3: Diagrama entidad-relación.

El Diccionario de Datos que acompaña al diagrama presentado se encuentra detallado en el Anexo F.

6.4. Algoritmo

El algoritmo presentados a continuación es una contribución personal al diseño del sistema.

Controlar la calidad de un proyecto

El siguiente algoritmo muestra la manera en que los usuarios pueden efectuar el control de la calidad de un proyecto (en general). El algoritmo muestra toda la lógica de la aplicación en torno a un objeto de la clase Proyecto. En esta lógica intervienen los usuarios y el sistema.

```
1   controlar_calidad_proyecto( )
2
3   Si no ∃ P, entonces
4           P ← crearProyecto( )
5   elegir P
6
7   Si P no tiene SPs, entonces
8           SP ← crearSubproyecto( )
9
10  Para ∀ SP ∈ P, donde estado(SP) <> "cerrado", hacer
11          elegir SP
12          Para ∀ Fase ∈ SP, donde estado(Fase) <> "cerrado", hacer
13                  elegir Fase
14                  Si Fase no tiene Iteración con estado(Iteracion) <> "cerrado", hacer
15                          crear Iteración
16                          elegir Iteración
17                          Si Iteración no tiene herramientas asignadas, entonces
18                                  Asignar Casos de Testeo a Iteración
19                                  Asignar Checklists a Iteración
20
21                          Para ∀ CT ∈ Iteración, hacer
22                                  ejecutar CT
23                                  Si ResultadoObtenido(CT) = ResultadoEsperado(CT), entonces
24                                          Resultado(CT) = "pasó"
25                                  c/c
26                                          Resultado(CT) = "falló"
27                          Si para ∀ CT ∈ Iteración, Resultado(CT) = "pasó", hacer
28                                  CasosTesteo = true
29                          c/c
30                                  CasosTesteo = false
31
32                          Para ∀ CL ∈ Iteración, hacer
33                                  Para ∀ ECL ∈ CL, hacer
34                                          Ejecutar ECL
```

```
35                                          Si ResultadoObtenido(ECL) = "si"
36                                                  Resultado(ECL) = "pasó"
37                                          c/c
38                                                  Resultado(ECL) = "falló"
39                                  Si para ∀ ECL ∈ CL, Resultado(ECL) = "pasó", hacer
40                                          estado(CL) = true
41                                  c/c
42                                          estado(CL) = false
43                          Si para ∀ CL ∈ Iteración, estado(CL) <> "true", hacer
44                                  Checklists = true
45                          c/c
46                                  Checklists = false
47
48                          estado(Iteración) = "cerrado"
49                  Si CasosTesteo <> true o Checklists <> true, entonces
50                          Resultado(Iteración) = "falló"
51                          ir a línea 15
52                  c/c
53                          Resultado(Iteración) = "pasó"
54
55          Si para ∀ Iteración ∈ Fase, Resultado(Iteración) = "pasó", hacer
56                  estado(Fase) = "cerrado"
57                  ir a la línea 13
58          c/c
59                  estado(Fase) = "abierto"
60                  ir a la línea 15
61
62          Si para ∀ Fase ∈ SP, estado(Fase) = "cerrado", hacer
63                  estado(Subproyecto) = "cerrado"
64          c/c
65                  estado(Subproyecto) = "abierto"
66                  ir a la línea 11
67
68  Si para ∀ SP ∈ P, estado(SP) = "cerrado", hacer
69          estado(P) = "cerrado"
70  c/c
71          estado(P) = "abierto"
72          ir a la línea 10
73
74  fin_controlar_calidad_proyecto
```

6.5. Arquitectura

La arquitectura del sistema define la organización física y lógica que tendrá la aplicación.

6.5.1. Organización física

Esta organización se divide en dos: la organización general de la aplicación y la organización específica que ésta tendrá en el servidor.

a. Enfoque general

La figura que se muestra a continuación muestra un esquema de las tres capas que tendrá la aplicación (Cliente, Servidor y Base(s) de Datos).

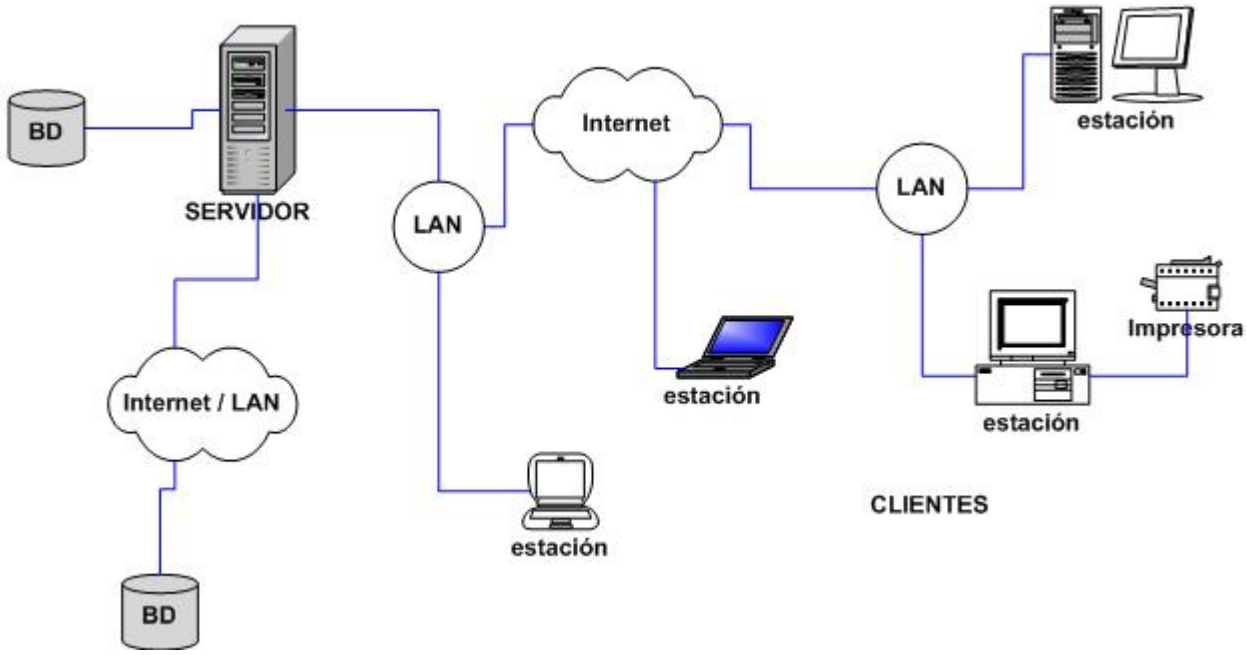


Figura 6.4: Arquitectura general de la aplicación (three-tier).

Como se ve en la figura, un cliente podrá acceder al servidor mediante una LAN (o cualquiera de las variantes conocidas de redes como MAN y WAN), directamente por Internet ó ambos (LAN y luego Internet). A su vez, del lado del servidor, se tendrá un esquema similar: el acceso al mismo podrá ser directo desde Internet o pasando por éste y luego por una subred.

De una manera análoga, la Base de Datos podrá residir en el mismo servidor o bien, en otra máquina, siendo accedida mediante una red (LAN, MAN, WAN o Internet). De todas formas, la arquitectura continuará siendo de tres capas.

b. Componentes

Un diagrama de componentes es utilizado para modelar la estructura del software incluyendo las dependencias entre los distintos componentes.

El diagrama de la figura 6.5 muestra los componentes de la aplicación y la relación de dependencia (flecha con trazo discontinuo) que existe entre ellos.

Básicamente, un componente se define como una parte física del sistema que contiene parte de la implementación del mismo. Puede o no depender de otros componentes y ser reemplazado si es necesario por uno o más nuevos. Contiene código fuente y su estructura interna y contenido puede responder a las convenciones del lenguaje de programación utilizado [RUM00].

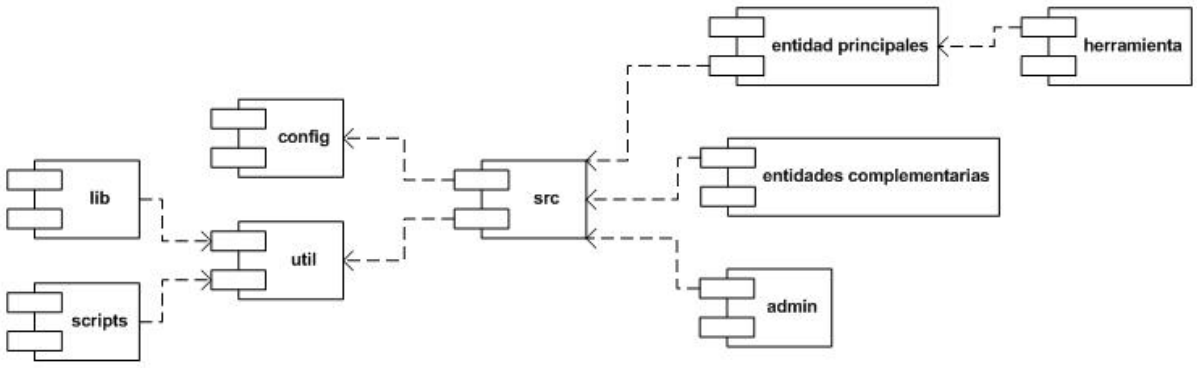


Figura 6.5: Diagrama general de componentes.

- Config.- contiene archivos de configuración del sistema
- Util.- contiene librerías y archivos complementarios que serán comunes a más de un objeto del sistema.
- Src.- Depende de los dos anteriores y contiene todo el código fuente de la aplicación, organizado por entidades.

Como complemento al diagrama de la figura 6.5, se presenta la figura 6.6 que detalla la organización de la aplicación a nivel físico dentro del servidor.

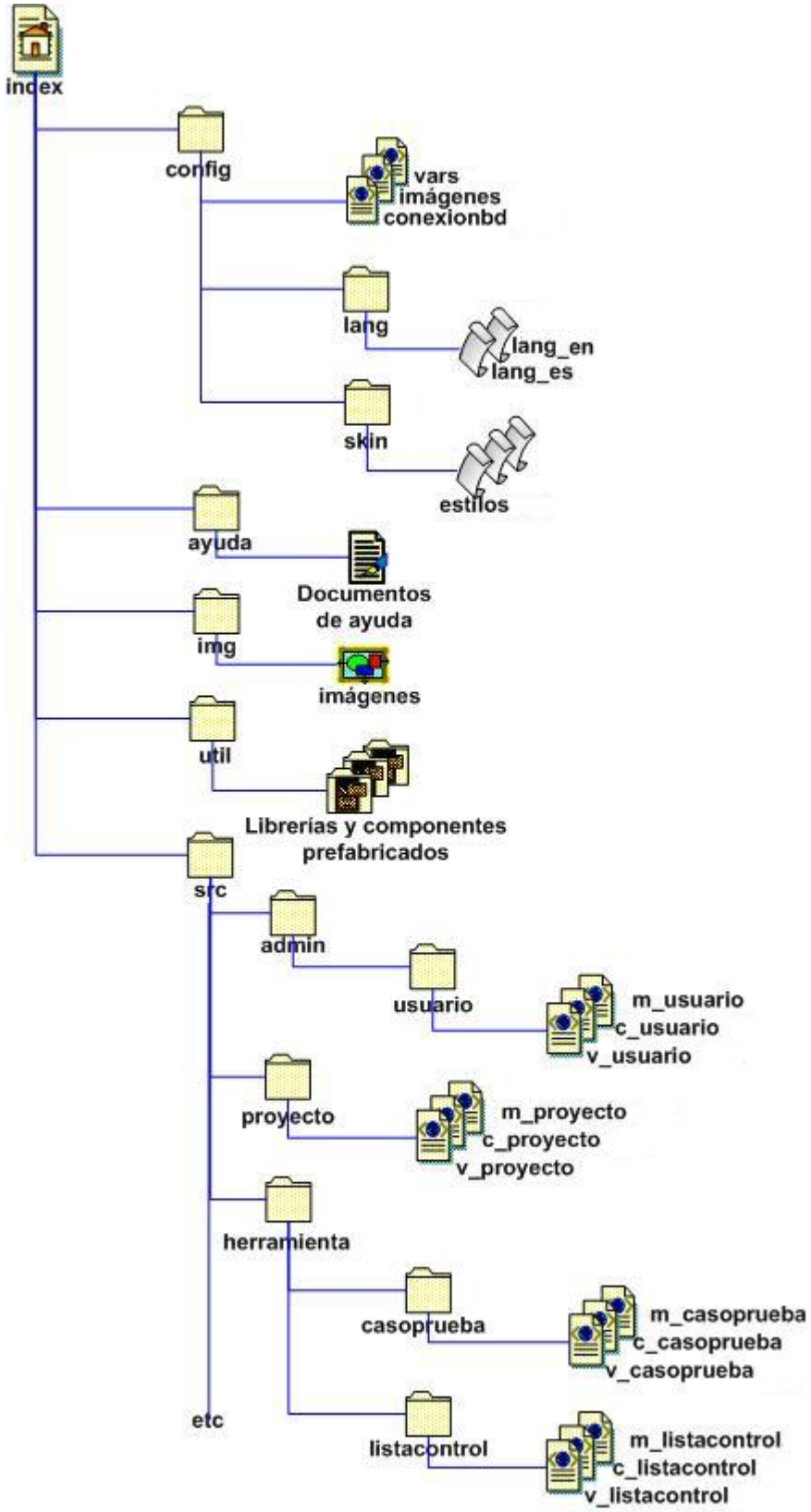


Figura 6.6: Organización física de la aplicación dentro del servidor.

6.5.2. Organización lógica

Muestra la manera en que los archivos se organizan para garantizar el correcto funcionamiento de la aplicación y una estructura lógica coherente y ordenada.

Esta organización está basada en la propuesta por la arquitectura MVC, que como se explicó en el capítulo 4, será utilizada para la implementación de la herramienta.

a. Paquetes

Un paquete es una parte un modelo. Cada paquete contiene un conjunto de partes más pequeñas del modelo, asignadas bajo un mismo principio (como la funcionalidad) [RUM00].

En la figura 6.7 se muestra el diagrama de paquetes de la aplicación y las clases más importantes que cada paquete contendrá.

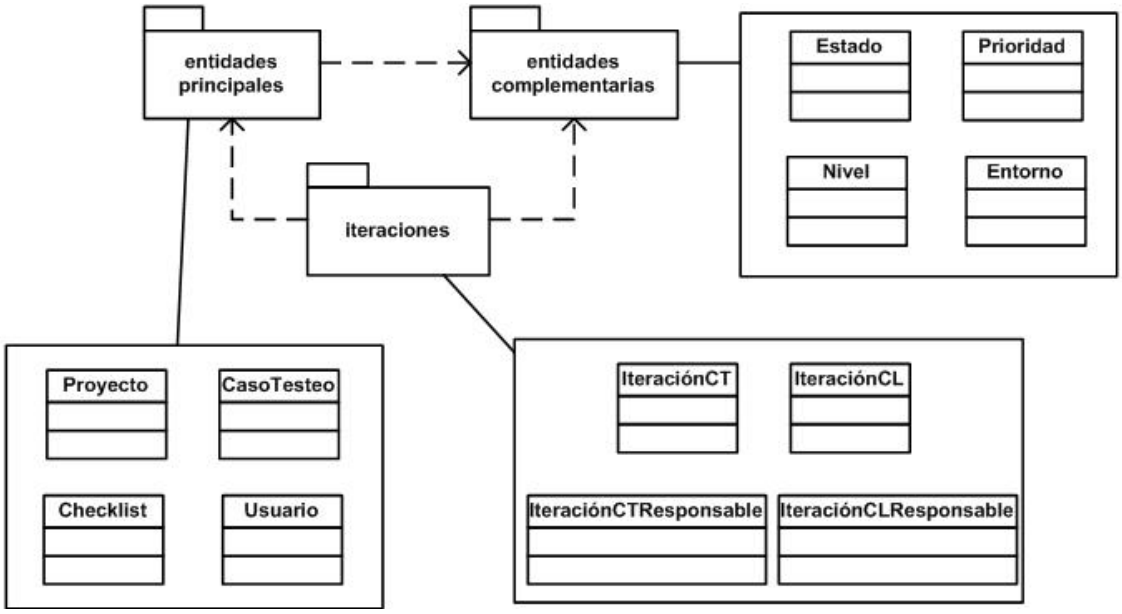


Figura 6.7: Diagrama de paquetes.

b. Capas MVC

La figura 6.8 ilustra la disposición de las capas del modelo MVC en la organización lógica de la aplicación. Como ejemplo, se muestra el caso de la entidad Proyecto que:

- En la capa de la Vista podrá tener archivos para listar los proyectos, crear nuevos, editar los existentes, eliminar, etc.
- En la capa del Controlador tendrá dos archivos: uno "maestro" que se ocupará de invocar a todos los archivos del Modelo que necesite (es por eso que en el ejemplo aparecen en la capa del Modelo las entidades proyecto, cliente, usuario y ciclo) y otro que servirá de soporte al primero durante la validación de la información.

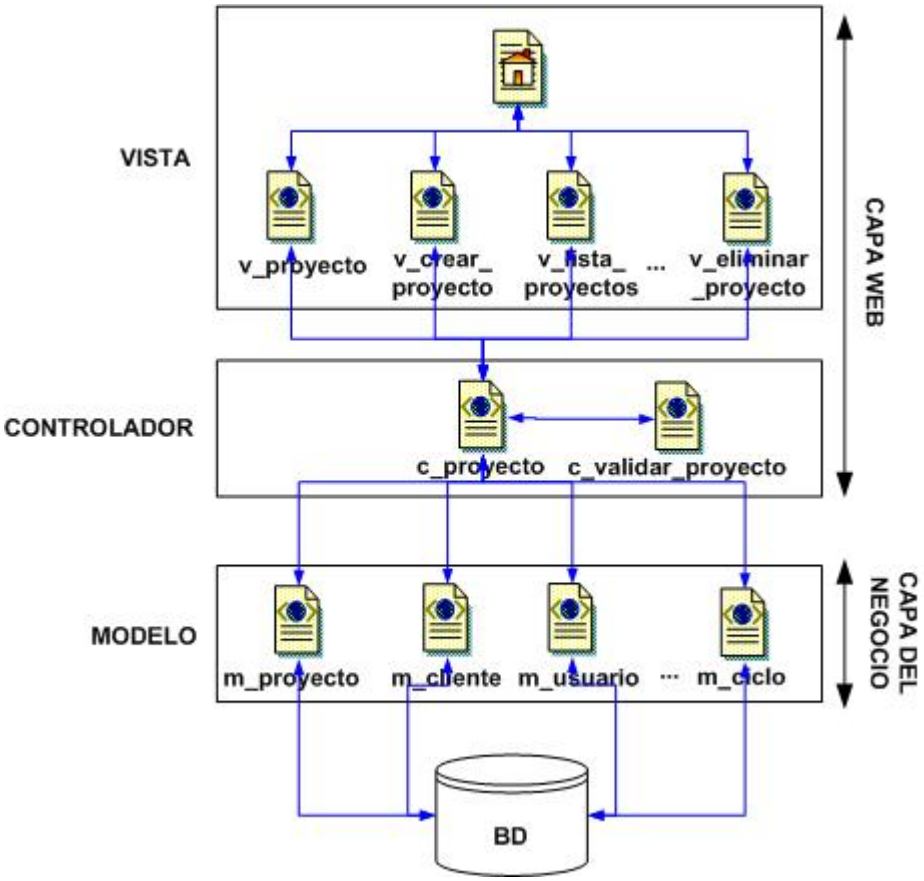


Figura 6.8: Organización lógica de la aplicación. En la figura se muestra como ejemplo la organización de archivos para la entidad Proyecto.

CAPÍTULO 7

IMPLEMENTACIÓN

En este capítulo se detallan los aspectos concernientes a la codificación de la herramienta y otros que tienen como objetivo alcanzar una versión operable del producto.

7.1. Elección de las herramientas de implementación

Para la implementación de la Herramienta Web de apoyo al control de la calidad del software, se decidió utilizar las siguientes herramientas:

- Lenguaje de programación: Hipertext PreProcessor (PHP).
- Servidor: Apache.
- Servidor de Base de Datos: MySQL.

PHP

PHP[1] es el acrónimo de Hypertext Preprocessor. Es un lenguaje "open source"[2] interpretado de alto nivel que es escrito en páginas HTML y ejecutado del lado del servidor.

La figura 7.1 muestra la manera en que se ejecutan las aplicaciones escritas en PHP.

Ventajas [IGN03][ASC03][PHP04]

1. Al heredar las características (sintaxis por ejemplo) de lenguajes tradicionales como C/C++, es sencillo de aprender para programadores que han tenido un mínimo de experiencia con tales lenguajes.
2. Ofrece un conjunto bastante amplio y completo de funciones que simplifican el trabajo del desarrollador. Estas funciones ofrecen la posibilidad de manejar clases, bases de datos, archivos, cadenas, etc.

[1] www.php.net
[2] Open source: Código abierto, gratuito.

Figura 7.1: Ejecución de un programa escrito en PHP [WEB03].

3. Cuenta con bastante documentación, fruto del trabajo de sus creadores, profesionales y aficionados que han puesto a disposición de los interesados los resultados de sus investigaciones, experiencias y trabajos.

4. Es multiplataforma, lo que permite que las aplicaciones escritas en este lenguaje sean independientes del Sistema Operativo y del Servidor.

5. Al ser un lenguaje interpretado, reduce el tiempo de desarrollo al permitir la ejecución inmediata del código sin previas compilaciones. Además resulta más ligero para el servidor y su ejecución es mucho más rápida.

6. Es robusto. Su sistema de control de errores es bastante bueno y detallado al momento de detectar y reportar fallas en el código.

7. Soporta (de manera básica) el paradigma de programación Orientada a Objetos.

8. El análisis léxico para el reconocimiento de variables que se pasan en la dirección es automático.

9. Permite la incrustación de etiquetas HTML en su código.

10. Su soporte de acceso a Bases de Datos es muy bueno.

11. No maneja el concepto de punteros como lo hacen otros lenguajes como C/C++, lo que facilita el trabajo al desarrollador, evita problemas de depuración y otros asociados con accesos a memoria.

12. El código PHP es mucho más legible e interpretable que el código de otros lenguajes.

13. Como se dijo en el párrafo introductorio, es un lenguaje “open source”, es decir, gratuito y de fácil acceso (esto incluye el acceso a un amplio conjunto de aplicaciones listas para reutilizarse que son ofrecidas por la comunidad que apoya PHP).

14. Su aceptación por parte de millones de desarrolladores lo han convertido en uno de los lenguajes para desarrollo Web más populares y ricos en documentación y recursos. La figura F.2 muestra el rápido de crecimiento de servidores que utilizan PHP.

15. Se ejecuta completamente del lado del servidor, con esto se garantiza que la aplicación no dependa de qué tecnología usa el cliente que acceda a la aplicación. Básicamente, éste sólo requiere de un navegador.

16. En comparación con otros lenguajes interpretados (como Perl), PHP garantiza una curva de aprendizaje mejor ya que es sencillo de aprender.

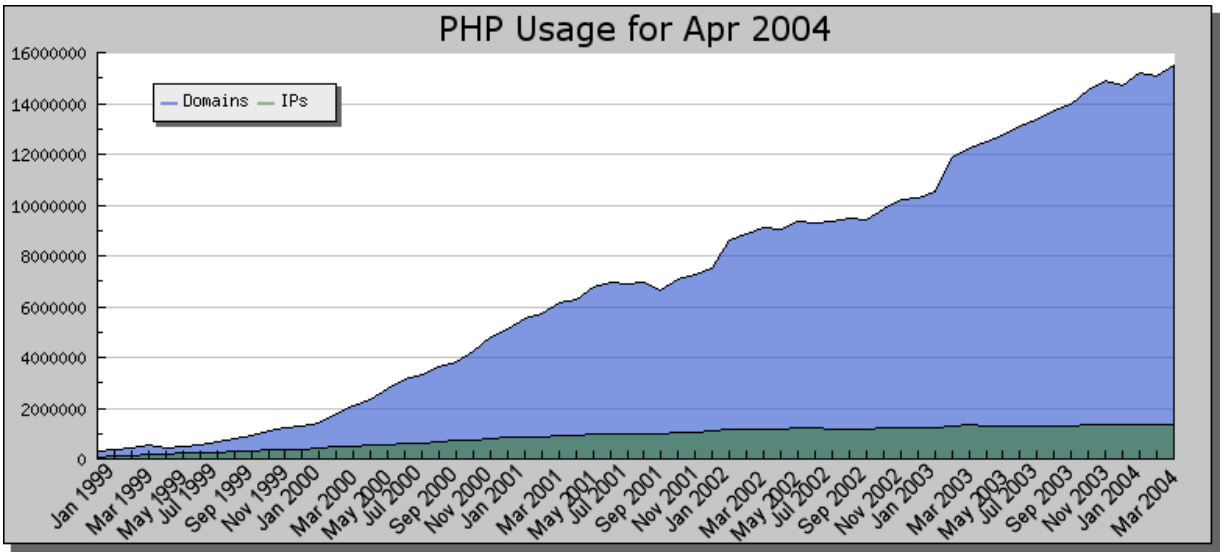


Figura 7.2: Gráfica del número de dominios y direcciones IP que utilizan PHP. Estadística de Netcraft [NET04].

Desventajas [ASC03]

1. No deja ningún tipo de tarea al cliente. Exige que todo el trabajo sea hecho en el servidor lo que en algunos casos resulta inconveniente debido al retardo que ocasiona en la respuesta de tareas que pueden perfectamente ser hechas del lado del cliente.

2. El hecho de poder mezclar código PHP con HTML puede resultar perjudicial si no se establece un buen conjunto de convenciones de programación.

3. La manipulación de Clases y Objetos es muy precaria aun y puede resultar perjudicial para sistemas muy grandes.

Para que estas desventajas no afecten el correcto y eficiente funcionamiento del sistema se establecieron las siguientes convenciones:

1. Debido a que se desea que el sistema sea multiplataforma será necesario que todas las tareas sean hechas por el Servidor, así no se dependerá del Cliente en absoluto. El costo de esta decisión es la velocidad de la aplicación. Sin embargo, debido a que PHP es interpretado y ofrece un excelente manejo de la memoria, se espera que el tiempo de respuesta sea mínimo.

2. Para no tener código PHP y HTML mezclado en toda la aplicación se decidió sólo utilizar éste último en la capa View de la arquitectura MVC.

3. En cuanto a las Clases y Objetos, el soporte y manipulación ofrecidos por PHP son suficientes para la aplicación debido a que sólo se utilizarán Clases en la capa Model de la arquitectura MVC.

PHP y otros lenguajes [ORA04][PHP04]

- PHP y ASP/ASP.NET.- Las ventajas que PHP presenta frente a ASP/ASP.NET son las siguientes:

    1. Es de libre distribución.
    2. Es multiplataforma (ASP y ASP.NET requieren de Win32 y el IIS).
    3. Es más eficiente en el uso de recursos.
    4. Es más seguro.
    5. Cuenta con código de libre distribución.

- PHP y ColdFusion.- Aunque ColdFusion tiene una mejor manipulación de errores y manejo de los datos, PHP corre en muchas más plataformas, es mucho más eficiente en tareas complejas, brinda una mayor estabilidad, flexibilidad y una mejor manipulación de estructuras anidadas. Además, PHP cuenta con mucha más documentación.

- PHP y Perl.- La gran ventaja de PHP frente a Perl es la curva de aprendizaje. Perl es mucho más complicado de aprender. Además, PHP es más rápido en tiempo de ejecución.

- PHP y Java.- Java fue considerado en un principio como el lenguaje de programación de la herramienta propuesta en el presente trabajo, sin embargo fue descartado porque presenta las siguientes desventajas frente a PHP:

  - Para este caso, Java exigía la utilización de diversas tecnologías como Struts (MVC en Java), JavaBeans, JSP y Servlets. Esto ocasionaba que el código de la aplicación crezca y se complique más de lo necesario y con ello ésta fuera más lenta. Además, la curva de aprendizaje no tenía comparación frente a la de PHP.
  - La combinación de distintas tecnologías de Java (Struts, JSP, Servlets, etc.) obliga el uso de otros lenguajes que hagan las veces de *engranaje* como XML, lo que complica aún más el código.
  - Java es compilado e interpretado, lo que duplica el tiempo de desarrollo y ejecución de una aplicación. Al ser PHP solo interpretado, permite un desarrollo más rápido y la ejecución inmediata sin previas compilaciones.
  - PHP maneja mejor los recursos que Java (memoria, procesador).
  - El código PHP puede ser escrito en cualquier editor de textos (incluso los ofrecidos de manera gratuita con los Sistemas Operativos) y luego ser llevado a un navegador para su ejecución. Si bien Java también puede ser escrito en un editor de textos, la complejidad de su código, las compilaciones previas y la inminente necesidad de contar con un ambiente de trabajo que organice los archivos y oriente a los desarrolladores (por la gran cantidad de archivos fuente que se van creando y el manejo de métodos y atributos en cada uno de ellos), obliga a éstos a adquirir herramientas adicionales (como JBuilder), cuyas versiones completas no son de libre distribución.

Apache

El NCSA (National Center for Super Computing Applications) desarrolló, en 1995, uno de los primeros servidores Web pero el proyecto fracasó.

Sin embargo, los usuarios comenzaron a crear parches para el servidor y a intercambiarlos, creando un foro para la administración de estos remiendos. Fue así que nació el grupo Apache, que sigue funcionando actualmente y se dedica a perfeccionar el servidor.

Ventajas

Las principales ventajas de Apache son [AEI02]:

1. Fiabilidad: Alrededor del 90 % de los servidores con más alta disponibilidad funcionan con Apache.

2. Gratuito: Apache es completamente gratuito y se distribuye bajo la licencia Apache Software License[3], que permite la modificación del código.

3. Extensibilidad: se pueden añadir módulos para ampliar las ya de por sí amplias capacidades de Apache.

La figura 7.3 muestra las estadísticas que comparan la aceptación de Apache frente a otros servidores Web [NET04].

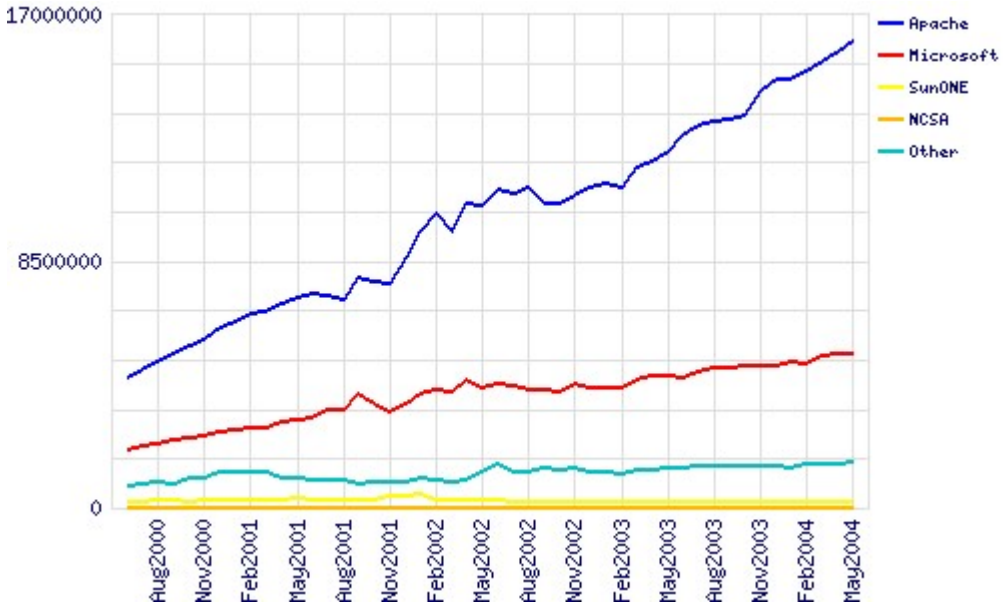


Figura 7.3: Servidores Web activos desde agosto de 2000 a mayo de 2004 [NET04].

[3]http://www.apache.org/LICENSE.txt

7.2. Marco de trabajo

El marco de trabajo definido une la arquitectura de desarrollo elegida (MVC) con el lenguaje de programación (PHP).

MVC en PHP

Para la implementación de MVC con PHP se determinó utilizar los siguientes recursos del lenguaje en cada capa del modelo:

- Vista: etiquetas HTML, hojas de estilo (css) y estructuras de control básicas de PHP (condicionales y ciclos).
- Controlador: estructuras condicionales para las acciones y redireccionamientos.
- Modelo: Clases y elementos de conexión a Bases de Datos de PHP.

Las tres capas están acompañadas de archivos PHP adicionales para la configuración de las variables de entorno, validaciones y otros recursos que deben ser compartidos por las tres capas.

La figura 7.4 ilustra el marco de trabajo definido.

Este marco de trabajo fue creado en base al documento de Jacobo Tarrío: "MVC en PHP" [TAR02]. A esta documentación se hicieron algunos cambios estructurales (como la adición de clases y objetos) dando como resultado un framework bastante completo, sencillo de manipular, flexible y robusto.

7.3. Hechos relevantes de la implementación

Como se estableció en la etapa de diseño y se explicó en el diagrama expuesto líneas arriba, la implementación utiliza las siguientes convenciones:

- Cada entidad tiene cinco archivos PHP base:
    - Para la Vista: los archivos v_entidad y v_entidad_editar. El primero para desplegar la lista de elementos que esa entidad tiene en la Base de Datos y el segundo para todas las acciones de edición (creación, modificación y eliminación).

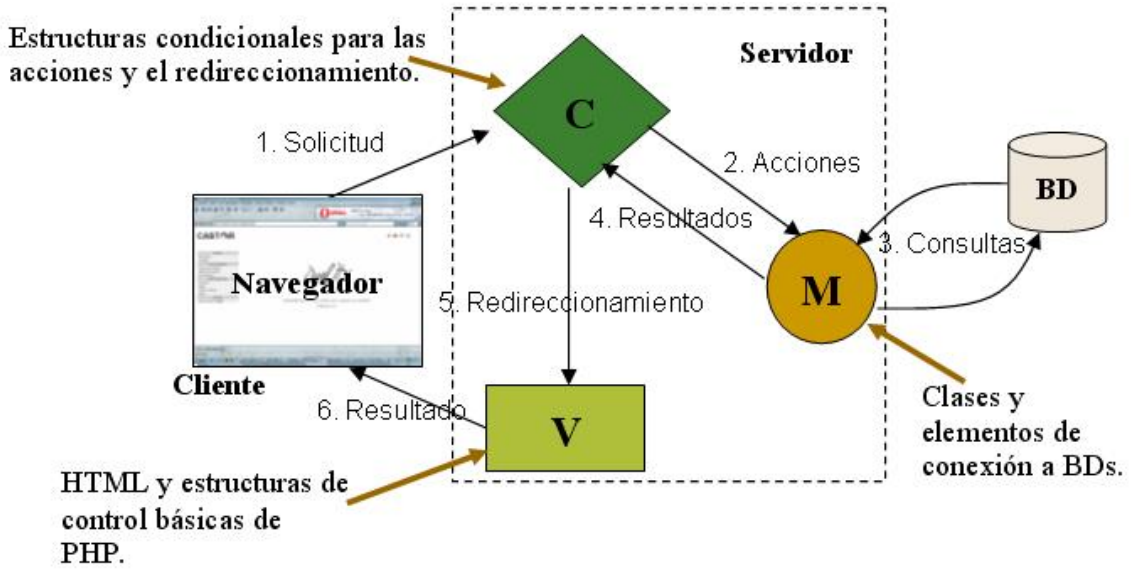


Figura 7.4: Recursos del lenguaje y secuencia de acciones en un marco de trabajo MVC con PHP.

- Para el Controlador: los archivos c_entidad y c_entidad_validar. El primero que es el encargado de recibir todas las solicitudes de la Vista, instanciar objetos de la capa del Modelo y redireccionar los resultados a nuevos archivos de la Vista. El segundo sirve de soporte al primero en las tareas de validación de datos (las funciones comunes de validación son reutilizadas por todos los archivos c_entidad_validar al compartir el archivo general "validador" del directorio "util").
- Para el modelo: el archivo m_entidad que contiene una clase con el nombre la entidad que maneja y es el encargado de todos los accesos a la Base de Datos que conciernan a esta.

▪ Se posee además, un conjunto de archivos de configuración entre los que se destacan:

- ConexionBD: archivo exclusivo para la conexión a la Base de Datos. Este script extrae los valores para la conexión (nombre de la Base de Datos y cuenta con la cual conectarse) del archivo "vars".
- Vars: para la declaración y asignación de valores de las variables de entorno (variables que son compartidas por la mayoría de los archivos de la aplicación).
- Mensaje: para la configuración de los mensajes de información, error y advertencia. Este archivo siempre recibe los datos a desplegar del archivo controlador que lo invoca cuando es necesario mostrar un mensaje al usuario.

7.4. La aplicación

El nombre asignado a la herramienta es CASTOR.

7.4.1. Descripción general

La pantalla inicial de la herramienta es la de Login, como se muestra en la figura 7.5. En ella se solicitan al usuario cuatro datos: el idioma que desea utilizar, su nombre de usuario, su contraseña y la apariencia que desea utilizar (colores de las ventanas). Además, el usuario puede elegir trabajar en modo simple o dos ventanas, haciendo clic en la última de las opciones que se despliegan en la pantalla.

Figura 7.5: Pantalla de Inicio de sesión.

Si el usuario ingresa el nombre de usuario o contraseña de manera incorrecta, el sistema despliega un mensaje como el que se muestra en la figura 7.6.

Si los datos provistos por el usuario son correctos, entonces se muestra la pantalla principal de la aplicación, como se muestra en la figura 7.7. Esta pantalla está compuesta por tres partes: el menú en la parte izquierda, la barra de funciones adicionales en la parte superior y los mensajes de ayuda (relacionados al tema del SQA) en la parte central.

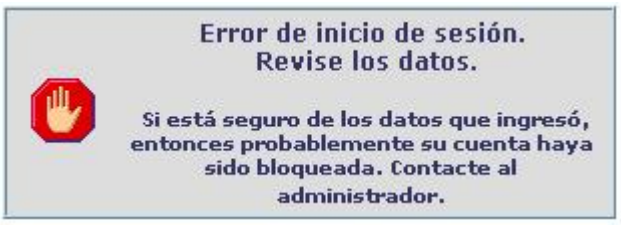


Figura 7.6: Mensaje de error de inicio de sesión.

Figura 7.7: Pantalla principal de la aplicación.

El menú principal conduce al usuario a las tareas más importantes que se pueden desempeñar utilizando la herramienta, como la gestión de proyectos, clientes, casos de prueba, etc., además de la gestión de entidades complementarias a las mencionadas, como estados, prioridades, entornos, ciclos y fases, etc.. La configuración de la aplicación es sólo accesible para Administradores.

En la parte superior derecha se encuentran enlaces a funciones adicionales de la herramienta, como la administración de la cuenta actual, el buscador, el acceso al documento de ayuda y el cierre de la sesión actual.

Figura 7.8: Menú principal.

Figura 7.9: Barra de funciones adicionales.

Cada ventana que es accedida desde los enlaces provistos en el menú posee en general las características descritas a continuación.

Una tabla que lista los registros de la entidad seleccionada. A manera de ejemplo, la figura 7.10 muestra la pantalla de administración de Casos de Prueba.

Junto a cada registro, se despliega un conjunto de iconos que conducen a distintas

Identificador	Nombre	Objetivo	Resultado esperado	Acción
1	Segunda forma normal	Verificar que las tablas estén en tercera forma normal.	No se encontraron los campos mencionados.	
2	Confiabilidad de los datos.	Verificar la confiabilidad de los datos que se almacenan.	Todos los datos almacenados están en los campos correctos.	
3	Requerimientos	Verificar que los requerimientos sean coherentes	No existen incoherencias entre los requerimientos.	
5	Análisis de requerimientos	Verificar que no existan requerimientos incoherentes	No existen requerimientos incoherentes	
6	Verificación de los requerimientos	Verificar que no existan requerimientos incoherentes.	No existen requerimientos incoherentes	
7	Verificación de ambigüedades en los requerimientos	Verificar que no existan requerimientos ambigüos	No existen requerimientos ambigüos	
8	Requerimientos completos	Verificar que cada requerimiento esté completo	Requerimientos completos (todos)	

Figura 7.10: Pantalla de administración de casos de prueba.

acciones a efectuar con éste. Estas acciones se habilitan a cada usuario de acuerdo al rol que desempeñe (por ejemplo, un Desarrollador no puede ejecutar Casos de Prueba pero sí un Tester). La figura 7.11 muestra todos los iconos que aparecen en distintos lugares de la aplicación y una breve descripción de la función que cada uno tiene asociada.

Figura 7.11: Acciones disponibles para los registros.

En el caso de que existan muchos registros para la entidad elegida, existen dos barras de navegación entre páginas: una según la página y otra según el orden alfabético (ver figura 7.12).

A su vez, en la parte superior de la pantalla (debajo de la barra de funciones adicionales) se habilita la barra de navegación entre partes de un proyecto (ver figura 7.13) que cambia a medida que el usuario elige el proyecto, subproyecto, fase e iteración

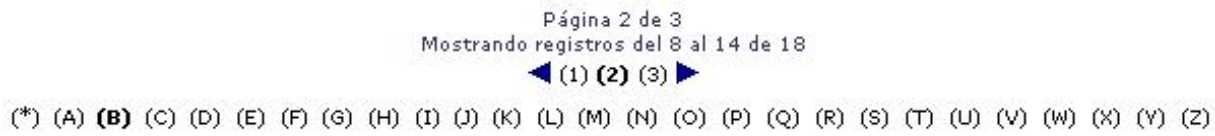


Figura 7.12: Barras de navegación entre registros de una misma entidad.

con los que desea trabajar, lo que facilita el cambiar de nivel de detalle en cualquier momento sin necesidad de ir saliendo de manera secuencial de la iteración, la fase, el subproyecto y el proyecto.

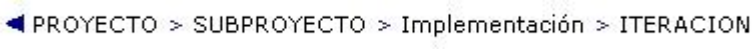


Figura 7.13: Barra de navegación entre partes de un proyecto.

Los formularios de inserción y edición de datos poseen un robusto control de errores que ayudan al usuario a efectuar un correcto registro de la información, evitando así posteriores problemas.

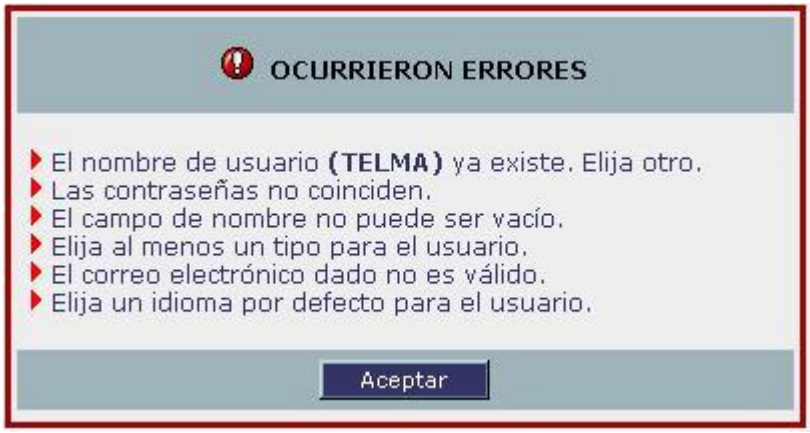


Figura 7.14: Ejemplo de mensaje de error durante el registro de un nuevo usuario.

Tareas pendientes

Cuando el usuario inicia la sesión y tiene tareas sin atender, el sistema le despliega un mensaje como el que se muestra en la figura 7.15.

El clic en el botón *Ver el detalle* conduce al usuario a una ventana con la lista de tareas pendientes, como se muestra en la figura 7.16.

Figura 7.15: Mensaje de tareas pendientes.

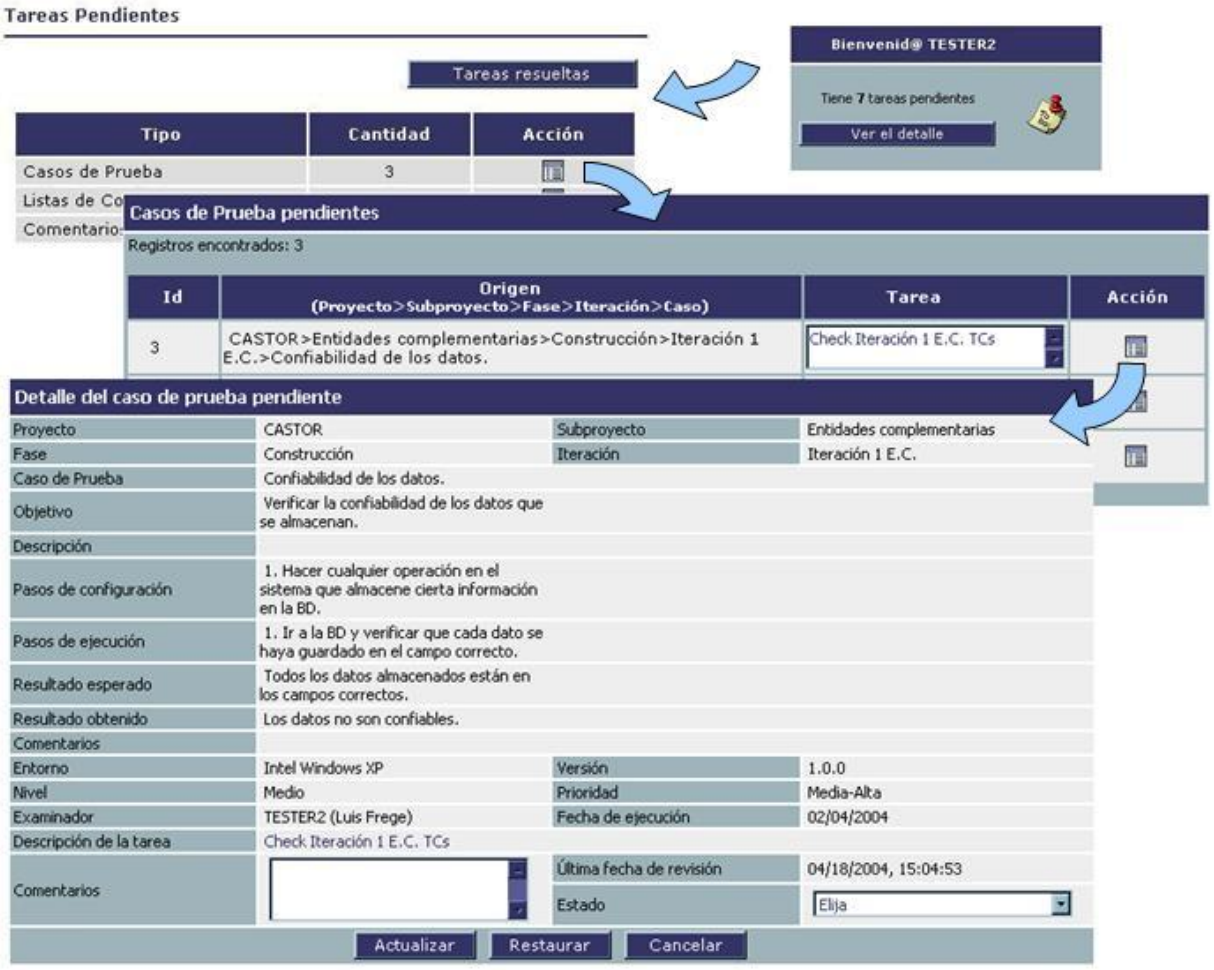


Figura 7.16: Detalle de tareas pendientes.

7.4.2. Envío de alertas

Para el envío de alertas se utilizan distintas tecnologías:

- Cron: programa residente en memoria que revisa cada minuto el archivo crontab. En este archivo se guardan las instrucciones que deben ser ejecutadas cada cierto

periodo de tiempo (también indicado en el archivo).

- PHP: se utilizan funciones del lenguaje para el envío de alertas al correo electrónico y teléfono móvil de cada usuario.
- Servidor SMTP: se utiliza un servidor de correo electrónico para el envío de los correos (sin este servidor las instrucciones de PHP no funcionan.

El modo de trabajo entre estas tecnologías es el siguiente: el cron revisa cada minuto el archivo crontab en el que está incluida la instrucción de ejecutar el script *enviar_alertas.php* una vez al día. Cuando la instrucción es ejecutada, el script utiliza la función *mail()* de PHP para enviar las alertas. Esta función utiliza el servidor SMTP para despachar las alarmas. Si el envío tuvo éxito, entonces el script almacena la fecha de envío en la Base de Datos. La figura 7.17 ilustra este modo de trabajo.

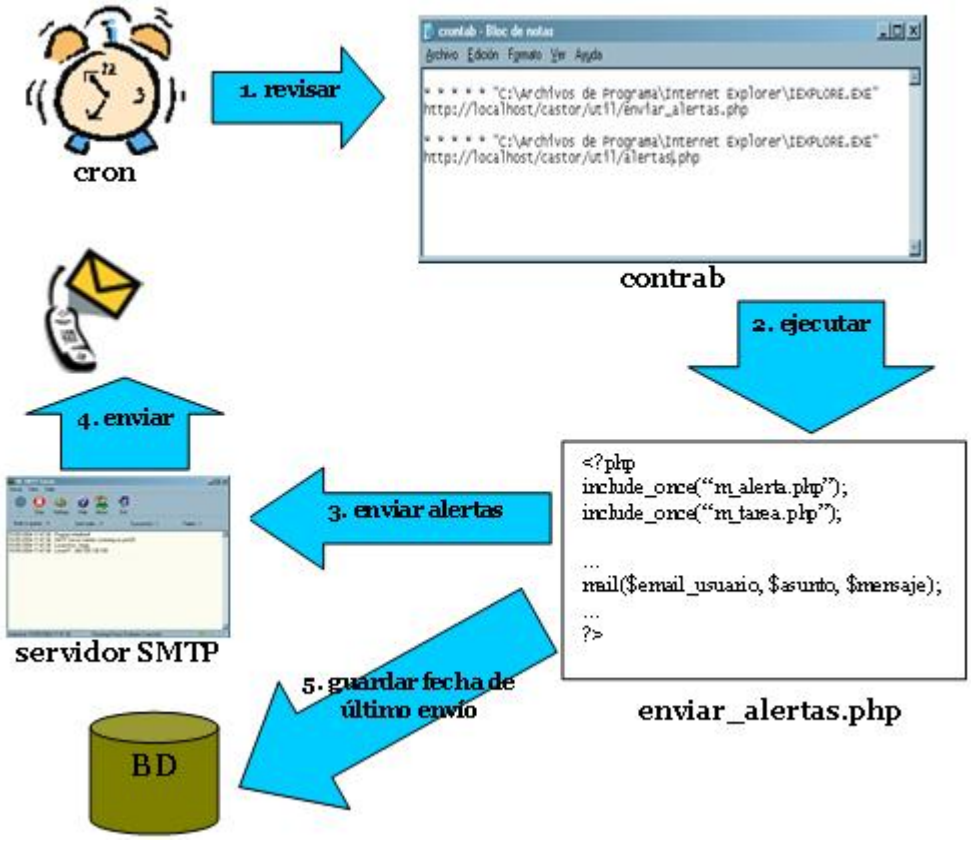


Figura 7.17: Modo de envío de alertas utilizando el Cron, PHP y un servidor SMTP.

7.4.3. Reportes

Una de las partes más importantes del sistema son los reportes. Éstos permiten la presentación de la información contenida en la Base de Datos de manera objetiva, lo que

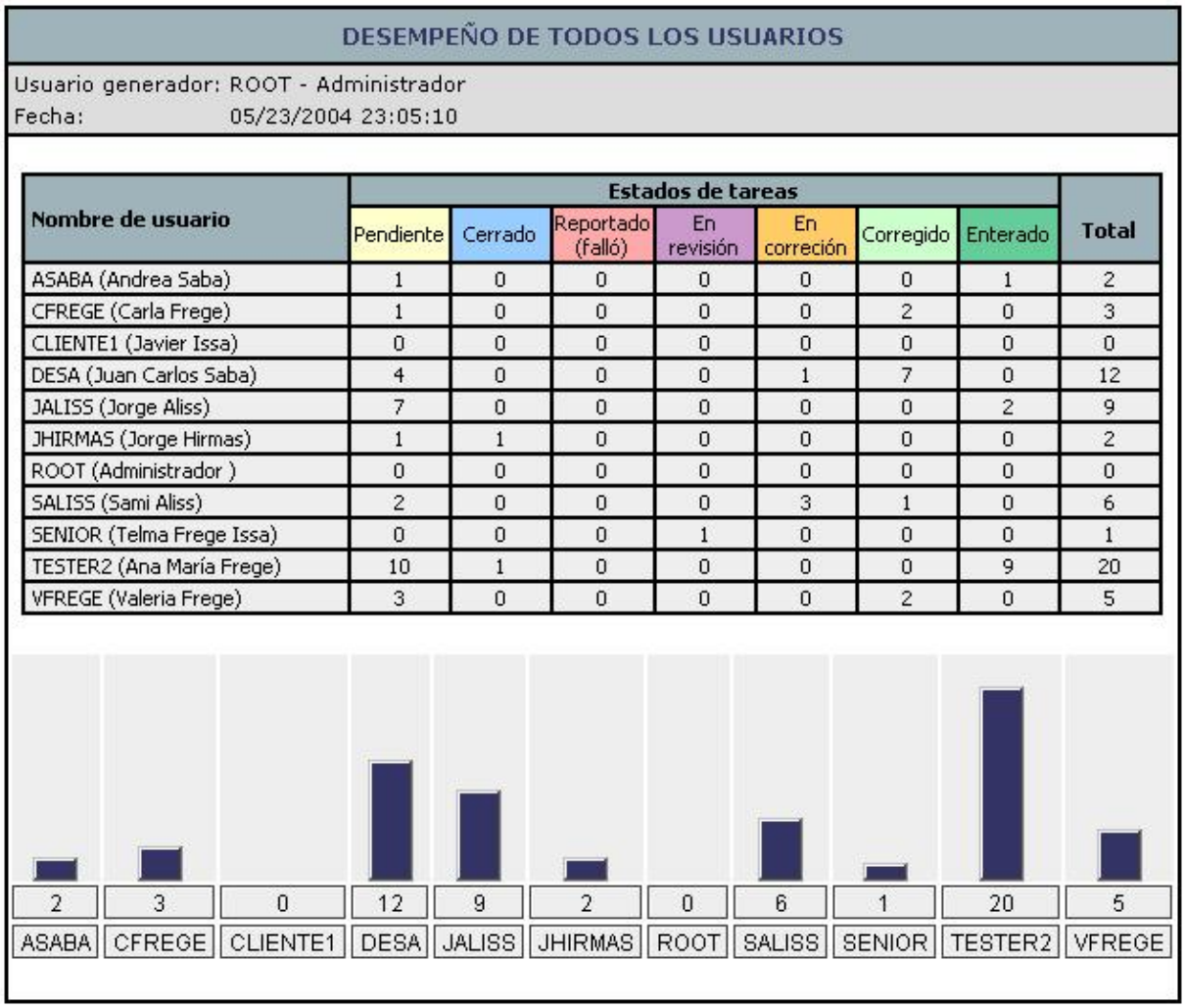
ayuda al usuario en la toma de decisiones. Entre los más importantes la herramienta ofrece los siguientes:

- Porcentaje de avance de los subproyectos de un proyecto dependiendo de los resultados de las últimas iteraciones de cada una de sus fases.
- Herramientas utilizadas por iteración.
- Tiempo que se demoró en cerrar una fase de un subproyecto, un subproyecto y un proyecto.
- Cantidad de personas involucradas en el proceso de control de calidad de un proyecto (testers y desarrolladores).
- Cantidad de errores encontrados por tester y cantidad de errores reportados a cada desarrollador.
- Fases de un proyecto (de todos sus subproyectos) que tuvieron más iteraciones (es decir, qué fases del proceso de desarrollo tuvieron más problemas durante el proceso de control de calidad).
- Reporte completo sobre un proyecto, usuario o cliente en particular.
- Desempeño de los usuarios del sistema con respecto a sus tareas asignadas (estados de las tareas). Como ejemplo se muestra este reporte generado en la figura 7.18.

7.4.4. Manuales

La aplicación ofrece como parte de la documentación seis tipos de manuales:

- Manual de Instalación y Configuración
- Manual del Administrador
- Manual del Líder de Proyectos
- Manual del Tester
- Manual del Desarrollador
- Manual del Cliente



DESEMPEÑO DE TODOS LOS USUARIOS

Usuario generador: ROOT - Administrador
Fecha: 05/23/2004 23:05:10

Nombre de usuario	Estados de tareas							Total
	Pendiente	Cerrado	Reportado (falló)	En revisión	En correción	Corregido	Enterado	
ASABA (Andrea Saba)	1	0	0	0	0	0	1	2
CFREGE (Carla Frege)	1	0	0	0	0	2	0	3
CLIENTE1 (Javier Issa)	0	0	0	0	0	0	0	0
DESA (Juan Carlos Saba)	4	0	0	0	1	7	0	12
JALISS (Jorge Aliss)	7	0	0	0	0	0	2	9
JHIRMAS (Jorge Hirmas)	1	1	0	0	0	0	0	2
ROOT (Administrador)	0	0	0	0	0	0	0	0
SALISS (Sami Aliss)	2	0	0	0	3	1	0	6
SENIOR (Telma Frege Issa)	0	0	0	1	0	0	0	1
TESTER2 (Ana María Frege)	10	1	0	0	0	0	9	20
VFREGE (Valeria Frege)	3	0	0	0	0	2	0	5

Figura 7.18: Reporte generado por CASTOR: desempeño de los usuarios.

Cada uno de los cuales explica al usuario -de acuerdo a su rol- las funciones que debe desempeñar dentro del sistema y la manera en que éste se comporta antes las distintas situaciones posibles. Estos manuales están en versión electrónica (formato PDF) y son accesibles desde el icono de ayuda de la aplicación. También vienen incluidos en el CD que está asjunto al documento principal de este proyecto.

CAPÍTULO 8

PRUEBAS

Las pruebas efectuadas al sistema se clasifican en: pruebas de verificación ("¿se está construyendo la herramienta de acuerdo a los requerimientos dados?") y pruebas de validación ("¿se está construyendo lo que el usuario necesita?").

8.1. Pruebas de verificación

8.1.1. Pruebas de unidad

Las pruebas de unidad se realizan concentrando la atención en un módulo en particular. En ellas se verifica que cada módulo, de manera independiente, haga lo que debe hacer.

Estas pruebas fueron efectuadas dividiendo la aplicación en las distintas entidades que la componen: proyectos, clientes, usuarios, casos de prueba, etc. y probando cada una de las funciones asociadas a estas entidades, como las de crear nuevo, modificar, eliminar, etc.

Las entidades fueron divididas en dos grupos básicos: "entidades principales" (proyectos, subproyectos, clientes, casos de prueba, listas de control, usuarios e iteraciones) y "entidades complementarias" (prioridades, estados, entornos, ciclos, fases, etc.). Estas últimas deben su nombre a la dependencia que tienen dentro del sistema de las primeras. Es decir, por sí solas no tienen utilidad en el sistema pero sí complementando la funcionalidad de las entidades principales (por ejemplo, la entidad "estado" complementa a las entidades iteración, usuario, proyecto, etc.).

Debido a esta dependencia, las entidades complementarias fueron las primeras en ser desarrolladas y luego las entidades principales.

8.1.2. Pruebas de integración

Las pruebas de integración sirven para verificar que no existan problemas en la interacción entre uno o más módulos, que las funciones de un módulo no anulen o interrumpan las funciones de otro y que las funciones compartidas se ejecuten correctamente.

En este caso, las pruebas de integración fueron ejecutadas de forma ascendente uniendo primero las entidades complementarias entre sí (como en el caso de ciclos y fases) y luego éstas a las principales a medida que las últimas eran desarrolladas. Luego se unieron las entidades principales entre sí para poder armar el flujo de trabajo (*workflow*) de la aplicación.

Se realizó una integración ascendente debido a lo que ya se explicó en el anterior apartado: para poder desarrollar las entidades principales se requería contar con las complementarias listas. De la misma forma, para poder armar el flujo de trabajo, se requería contar con las entidades principales ya implementadas.

8.1.3. Pruebas de Caja Blanca y Caja Negra

Pruebas de Caja Blanca

En las pruebas de verificación de caja blanca, lo que interesa es comprobar que no existan sentencias en el código que nunca sean ejecutadas (trozos de "código muerto"). Este tipo de pruebas se efectuó con las entidades principales (las entidades secundarias tienen un menor volumen de código y no requirieron de este tipo de verificación).

Pruebas de Caja Negra

En estas pruebas lo importante no es cómo se ejecuta, sino que se obtenga lo que se desea. Son pruebas ajenas al código involucrado durante la ejecución, lo único que interesa es que la función produzca el resultado esperado.

Ambos tipos de pruebas se llevaron a cabo dentro de las pruebas de unidad y de integración.

Resultados

Los trozos de la aplicación que fueron más difíciles de verificar a causa de la cantidad de código involucrado fueron las iteraciones (incluida la ejecución de casos de prueba y listas de control) y el control de tareas de los usuarios. Este gran volumen de código se debe a la necesidad de extensos controles que deben ser ejecutados durante el uso de las iteraciones y las tareas. Estos controles, a su vez, son necesarios ya que en estas funciones se manipulan grandes cantidades de datos debido a que prácticamente todas las entidades, tanto principales como complementarias, se ven involucradas en la ejecución -por ejemplo- de un caso de prueba en una iteración de una fase del subproyecto de un proyecto.

Los errores que se presentaban en estas secciones debían ser rastreados y corregidos, cuidando que los cambios hechos no afectaran a las demás funcionalidades que las mismas líneas de código podían desempeñar. Sin embargo, el hecho de contar con la aplicación implementada en MVC facilitó las tareas de identificación de los puntos críticos del código que causaban problemas y la corrección inmediata de éstos.

8.2. Pruebas de validación

Estas pruebas deben ser ejecutadas por los potenciales usuarios finales del sistema. Es decir, líderes de proyectos, testers y desarrolladores.

8.2.1. Pruebas Alfa

Las pruebas Alfa son ejecutadas por los usuarios y el equipo de desarrollo a fin de validar que el software adquirido alcanzó los objetivos planteados inicialmente.

Estas pruebas fueron llevadas a cabo cuando el sistema fue concluido, validando que:

- Ofrezca soporte al control de la calidad del software durante todas las fases de un proyecto.
- Sea amigable en su uso.
- Permita la personalización de las herramientas ofrecidas (casos de prueba y checklists).

La herramienta obtenida cumple con los objetivos que se trazaron en un principio.

8.2.2. Pruebas Beta - Aceptación del Usuario

Las pruebas de validación Beta son ejecutadas por los usuarios finales del sistema, ajenas a todo lo que fue el desarrollo del producto.

Para la ejecución de estas pruebas se recurrió a cinco empresas nacionales que están distribuidas en las tres ciudades principales del país: Cochabamba, La Paz y Santa Cruz:

- Intersoft (La Paz)
- Cybercia (Cochabamba)

- BZDesigners (Santa Cruz)
- E&M Computadoras y Programas (Santa Cruz)
- Axess Web (Santa Cruz)

Cada una de estas empresas (representadas para las pruebas por una a seis personas) participó del ciclo de pruebas del sistema que consistió en la presentación del mismo mediante la ejecución de un caso de ejemplo. Posteriormente, cada empresa tuvo a su disposición la aplicación por un periodo de diez días.

Entre las opiniones emitidas hacia la aplicación, se pueden mencionar:

> *"Producto excelente que apoya el control de la calidad asegurando un proceso formal."* (Patricia Aguilar, Jefe de Sistemas Cybercia)

> *"Herramienta de mucha utilidad para los procesos de desarrollo de software."* (Iván Montaño, Líder de Proyectos Intersoft)

> *"Herramienta muy útil que, debidamente utilizada, resulta de gran ayuda para testear QA en SW."* (Carlos Góngora, Líder de Proyectos Intersoft)

> *"Bastante bueno, mejor que otras herramientas similares."* (Aldo Gutiérrez, desarrollador Intersoft)

> *"Buen método de seguimiento para el control de calidad. Bastante amigable en su uso."* (Fernando Vargas, desarrollador Intersoft)

> *"Presenta grandes ventajas como herramienta de control de calidad. Adicionalmente, su flexibilidad en la configuración de casos de prueba y listas de control hace que sea una herramienta muy completa."* (Octavio Alzérreca, Jefe de proyectos Intersoft)

"Simplifica y ordena el proceso de control de calidad." (Sinchy Díaz, desarrolladora Intersoft)

"Excelente. Útil, práctico e independiente de la plataforma." (Branko Zabala, Gerente General BZDesigners)

"Bastante útil y necesaria para el desarrollo de software. Su flexibilidad permite a los usuarios definir sus propios parámetros de calidad." (Pamela Pacheco, Jefe de Sistemas Axess Web)

"Herramienta que viene a llenar un gran vacío en el medio que en tiempo y materia económica tiene un costo elevado. 100 % de aprobación." (Rubén Salazar, E&M Computadoras y Programas)

CAPÍTULO 9

CONCLUSIONES Y RECOMENDACIONES

9.1. Conclusiones

1. La ausencia de aplicación de métodos formales de control de calidad ocasiona en la mayoría de los casos retrasos en las entregas, fallas en los productos finales o incluso fracasos.

2. La Calidad de un producto es la coherencia que existe entre el software adquirido y los requerimientos fijados para éste antes de su desarrollo, tanto explícitos como implícitos. Por lo tanto, la calidad de un software es tan importante como su funcionalidad.

3. Calidad no es sinónimo de perfección. Un producto de alta calidad no necesariamente estará libre de errores. De hecho, el software libre de errores no existe, sino el que cumple con todos sus objetivos iniciales y con métricas bien definidas que se traducen en factores de calidad reconocidos.

4. Productos sometidos a procesos de control de calidad durante su desarrollo se traducen en uso eficiente de recursos y una alta probabilidad de terminar los planes a tiempo y cumpliendo con todos los objetivos propuestos.

5. Si el control de calidad se aplica desde las fases más tempranas del desarrollo, la cantidad de errores severos que se encuentran en las últimas fases se reduce de manera considerable.

6. Las empresas carentes de métodos básicos de control de calidad y procesos maduros experimentan problemas que acarrean consigo una alta dependencia de personas imprescindibles, el riesgo elevado de contar con malos productos que, de terminarse, lo harán utilizando más recursos de los previstos (tiempo, dinero, personas) y causando un deterioro progresivo de la imagen de la empresa ante la competencia y los clientes.

7. La calidad de un producto depende directamente del grado de compromiso que asuman las personas involucradas en su desarrollo.

8. Los modelos y estándares actuales deben ser utilizados como guías de orientación y fuentes de potenciales soluciones a los problemas más comunes de las empresas.

9. Cualquier aplicación de Control de Calidad disponible en la actualidad es tan eficiente como sus usuarios lo permitan. La definición personalizada de las herramientas (como los casos de prueba) causa que una aplicación llegue a ser muy potente en su uso o completamente inútil, dependiendo del grado de aprovechamiento del usuario (testers sobretodo).

10. La manera más práctica de implementar el SQA es la sistematización de sus herramientas: Casos de Prueba y Listas de Control.

11. CASTOR ofrece a los usuarios la posibilidad de implementar de manera formal las herramientas básicas del SQA (casos de prueba y listas de control), teniendo toda la información centralizada y estructurada.

12. La aplicación obtenida permite la ejecución de las herramientas mencionadas en todas las fases del desarrollo del producto.

13. El modelo de arquitectura MVC cumple con su objetivo principal: la reducción del esfuerzo invertido durante la implementación.

14. La definición de un buen Marco de Trabajo para la implementación del software simplifica el trabajo permitiendo el ahorro considerable del esfuerzo y los recursos invertidos. Además, facilita la definición de convenciones de programación, lo que a su vez contribuye a que el código sea claro, eficiente, escalable y reparable.

9.2. Recomendaciones

1. Profundizar la investigación realizada tomando como base un Marco de Trabajo específico y la manera en que éste puede acomodarse a la realidad nacional. Por ejemplo, se puede estudiar la factiblidad de la aplicación del modelo CMM en el ámbito local bajo ciertos criterios que el investigador considere oportunos.

2. Implementar un módulo (Interfaz) que permita al sistema la ejecución semi automática de Casos de Prueba.

3. Permitir que el sistema soporte iteraciones anidadas.

4. Manejo de comunicación más compleja entre los distintos usuarios del sistema. Por ejemplo, que permita la delegación de subtareas entre desarrolladores.

5. Implementar un módulo que permita el seguimiento completo de los requerimientos durante el proceso de control de calidad.

6. Permitir que la aplicación no sólo implemente las herramientas del SQA (casos de prueba y checklists) sino también sus prácticas más recomendadas, como las RTFs.

GLOSARIO

Algoritmo.- Secuencia finita de instrucciones dada para resolver un problema específico. Cada una de estas instrucciones tiene un significado claro y es ejecutada con una cantidad de esfuerzo finita.

ANSI.- El "American National Standards Institute", al igual que el SEI, la ISO y el IEEE, está vinculado con la calidad mediante sus publicaciones de estándares.

API.- Application Programming Interface. Conjunto de rutinas o funciones que un programa puede utilizar para que el Sistema Operativo realice alguna tarea.

Caja Blanca, Testeo de.- Conjunto de pruebas donde el tester es responsable de verificar la ejecución de cada prueba hasta el más mínimo nivel de detalle, incluyendo una revisión del código involucrado.

Caja Gris, Testeo de.- Híbrido entre el testeo de caja blanca y el testeo de caja negra. El tester verifica el código sin mucho nivel de detalle.

Caja Negra, Testeo de.- Conjunto de pruebas donde lo único que interesa es verificar la funcionalidad y no así la manera en que el programa ejecuta las tareas. Al tester no le interesa si el código está bien escrito, simplemente si la tarea es bien realizada o no.

CGI.- Common Gateway Interface. Es un protocolo genérico que permite extender las capacidades de HTTP. Los programas CGI añaden funcionalidad al servidor Web.

Cliente.- Ordenador que funciona localmente y se comunica con el servidor para solicitar información.

Defecto.- Anomalía de un producto.

EIS.- Enterprise Information System. Se refiere a cualquier sistema que proporciona

una infraestructura para el soporte de la información de la empresa.

Escalabilidad.- Es la medida en que un sistema es capaz de soportar y servir a más de un usuario, entendiéndose como usuario a cualquier agente que haga uso del sistema, ya sea una persona como otro sistema.

Framework.- Es la extensión de un lenguaje mediante una o más jerarquías de clases que implementan una funcionalidad y que (opcionalmente) pueden ser extendidas.

HTML.- HiperText Markup Language. Conjunto de marcas (más conocidas como etiquetas o "tags.[en] inglés) que permite básicamente el formateo de documentos de hipertexto para su divulgación en el World Wide Web.

HTTP.- HiperText Transfer Protocol. Principal protocolo de Internet para la transferencia de archivos.

IEEE.- "Institute of Electrical and Electronics Engineers". Entre otras publicaciones, se encuentra relacionado a la calidad del software mediante sus estándares "IEEE Standard for Software Test Documentation" (IEEE/ANSI 829), "IEEE Standard of Software Unit Testing" (IEEE/ANSI Standard 1008) y "IEEE Standard for Software Quality Assurance Plans" (IEEE/ANSI Standard 730).

IIS.- El Internet Information Server es el conjunto de herramientas ofrecidas por Microsoft para la administración y control de sitios Web, FTP, SMTP y Servicio de noticias.

Interface.- Parte de una aplicación capaz de interactuar con el usuario.

ISO.- "Internacional Organization for Standarization". Esta organización está involucrada con la calidad mediante sus estándares 9001, 9002 y 9003.

Lógica del Negocio.- Porción de la arquitectura realizada por componentes que es utilizada para crear y garantizar las reglas establecidas.

Proceso.- Un proceso define quién hace qué, cuándo y cómo para lograr un objetivo. Es una secuencia de tareas básicas que algún individuo de la empresa desempeña con el fin de alcanzar un objetivo.

SEI.- "Software Engineering Institute" de la Universidad Carnegie Mellon de Pittsburgh Pennsylvania, Estados Unidos. Es el responsable del desarrollo del CMM© - Capability Madurity Model.

Servidor.- Ordenador remoto (en algún lugar de la red) cuya tarea es proporcionar información a los clientes según peticiones.

SGML.- Standard Generalized Markup Language.

Shell.- Interface de un programa.

Smalltalk.- Lenguaje de programación Orientado a Objetos que nació como fruto de una investigación de XEROX en 1972.

Tester.- Persona encargada de hacer las pruebas al software.

BIBLIOGRAFÍA

[AEI02] AEI, Asociación Española de Internet
2002. Apache 2.0. www.ciberaula.com/curso/apache/. Fecha de consulta: 23 de octubre de 2003

[ANT03] ANTONIUCCI, Javier
2003. Manual Básico de Struts. programacion.com/java/tutorial/. Fecha de consulta: 8 de mayo de 2003

[APA03] APACHE, Foundation
2003. The Struts User's Guide - Introduction. jakarta.apache.org/struts/userGuide/introduction.html. Fecha de consulta: 3 de junio de 2003

[ASC03] ASCII, ASociación para el Conocimiento y la Innovación de la Informática
2003. Web dinámicas con PHP. ascii.eii.us.es/cursos/PHP/PHP4.html. Fecha de consulta: 22 de octubre de 2003

[ASQ03] ASQ, American Society for Quality
2003. Quality basics. www.asq.org. Fecha de consulta: 26 de noviembre de 2003

[BAC04] BACH, James
2004. The Immadurity of CMM©. www.satisfice.com/articles/CMM.htm. Fecha de consulta: 4 de junio de 2003

[BED97] BEDINI, Alejandro
1997 SPICE. Software Process Improvement and Capability Determination. En *1er. Simposio Latinoamericano de calidad y productividad en el desarrollo del software. SINTEC. Santiago, Chile*, tomo 1

[BED98] BEDINI, Alejandro
1998 Lo bueno, lo malo y lo feo de los Marcos de Trabajo. En *Congreso SoST'98. Universidad de Buenos Aires. Buenos Aires, Argentina*, tomo 1

[BUA00] BUADES, Gabriel
2000. Calidad en Ingeniería del Software.

dmi.uib.es/ bbuades/calidad/index.htm. Fecha de consulta: 15 de junio de 2003

[BUR92] BURBECK, Steven
1992. How to use Model-View-Controller (MVC). st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html. Fecha de consulta: 3 de junio de 2003

[CER00] CERVERA, Ángel
2000. El modelo McCall como aplicación de la calidad a la revisión del software de gestión empresarial. www.monografias.com/trabajos5/call/call.shtml. Fecha de consulta: 24 de mayo de 2003

[COR01] CORTES, Gloria
2001. Desarrollo de Software con Calidad. www.ubiquando.com.co. Fecha de consulta: 21 de marzo de 2003

[CUE99] CUEVA, Juan Manuel
1999. Calidad del Software. gidis.ing.unlpam.edu.ar/downloads/pdfs. Fecha de consulta: 22 de marzo de 2003

[EVE03] EVERLASTING, Technologies
2003. Rational Unified Process. www.everlastingtec.com. Fecha de consulta: 26 de enero de 2004

[GAR01] GARCÍA, Claudia GOMEZ Pilar
2001. El Modelo de Capacidad de Madurez. mailweb.udlap.mx. Fecha de consulta: 3 de mayo de 2003

[GER03] GERVÁS, Pablo
2003. Capítulo 11 Prueba. calisto.sip.ucm.es/people/pablo/teaching/tp0203/cap10.pdf. Fecha de consulta: 27 de junio de 2003

[GÓM01] GÓMEZ, Mateo
2001. Calidad del Software. www.losparciales.com.ar/examen/kenpaper02200103.pdf. Fecha de consulta: 12 de marzo de 2003

[GON98] GONZÁLES, Carlos
1998 *ISO 9000, QS 9000, ISO 14000. Normas internacionales de adminis-*

tración de calidad y sistemas ambientales.. McGraw Hill, México, primera edición

[GOO02] GOODWILL, James
2002 *Mastering Jakarta Struts*. Wiley Publishing, Estados Unidos, primera edición

[IBA03] IBARRA, Armando
2003. Rational Unified Process. delta.cs.cinvestav.mx/ pmejia/softeng. Fecha de consulta: 7 de julio de 2003

[IGN03] IGNSIDE.NET
2003. Apuntes de PHP. www.ignside.net/man/PHP/. Fecha de consulta: 23 de octubre de 2003

[LEW00] LEWIS, William
2000 *Software Testing and Continuous Quality Improvement*. CRC Press LLC, Estados Unidos, primera edición

[NAS03] NASSCOM, National Association of Software y Companies, Service
2003. List of Quality Certified Companies. www.nasscom.org/qualitycertified.asp. Fecha de consulta: 5 de mayo de 2003

[NET04] NETCRAFT
2004. May 2004 Web Server Survey. news.netcraft.com. Fecha de consulta: 22 de mayo de 2004

[ORA04] ORACLE
2004. PHP and ASP.NET Go Head-to-Head. otn.oracle.com/pub/articles/. Fecha de consulta: 28 de abril de 2004

[PHP04] PHP.NET
2004. Usage Stats for 2004. www.php.net/usage.php. Fecha de consulta: 22 de mayo de 2004

[PIC03] PICS, Process Improvement Center
2003. Grundsteine für Assessments. pics.arcs.ac.at/synbench/Assessments2.htm. Fecha de consulta: 27 de septiembre de 2003

[PRE02] PRESSMAN, Roger
2002 *Ingeniería del Software - Un enfoque práctico*. McGraw Hill, España, quinta edición

[RAM01] RAMIREZ, Gustavo
2001. Apuntes RUP. atenea.ucauca.edu.co/ gramirez/archivos/AnotacionesRUP.pdf. Fecha de consulta: 1 de julio de 2003

[RUM96] RUMBAUGH, James BLAHA Michael PREMERLANI William EDDY Frederick LORENSEN William
1996 *Modelado y Diseño Orientado a Objetos. Metodología OMT*. Prentice-Hall, España, primera edición

[RUM00] RUMBAUGH, James JACOBSON Ivan BOOCH Grady
2000 *El lenguaje Unificado de Modelado. Manual de Referencia*. Addison-Wesley, Estados Unidos

[SAL00] SALESA, Inma PALACIOS Ester
2000. Capability Madurity Model (CMM). courses.cs.tamu.edu/cpsc431/lively/. Fecha de consulta: 5 de mayo de 2003

[SEI03] SEI, Software Engineering Institute. Carneguie Mellon University
2003. Process Madurity Profile. www.sei.cmu.edu/sema/pdf/SW-CMM/2003apr.pdf. Fecha de consulta: 16 de septiembre de 2003

[SOM96] SOMMERVILLE, Ian
1996 *Ingeniería del Software*. Addison-Wesley, Estados Unidos, sexta edición

[TAR02] TARRIO, Jacobo
2002. MVC en PHP. www.xente.mundo-r.com/jtarrio/txt/mvc.html. Fecha de consulta: 23 de octubre de 2003

[TEC03] TECHSTREET
2003. IEEE and ISO Standards Catalogs. www.techstreet.com/. Fecha de consulta: 26 de junio de 2003

[WEB03] WEBESTILO
2003. Manual de PHP. www.webestilo.com/PHP/. Fecha de consulta: 23 de octubre de 2003

ANEXO A

EMPRESAS CERTIFICADAS POR LA ISO Y EL CMM©

Origen de esta información: [NAS03].

Empresa	ISO	CMM©
24/7 Customer.com	9002:1994	
3Com India Pvt Ltd	9000	
A G Technologies Pvt Ltd	9001:2000	
Accel Software Solutions Limited	9001	
Ace Software Exports Ltd	9002	
Adroit Computer Technique Pvt Ltd	9001:2000	
Aithent Technologies Pvt Ltd.		Nivel 5
Alcatel India Limited.	9001	
ALSTOM Systems Limited	9001	
American Express (India) Pvt. Ltd.		Nivel 2
AmitySoft Technologies Pvt Ltd	9001:2000	
Ampersand Software Applications Ltd.	9001	
Apex Technologies Pvt. Ltd.	9001	
Aptech Limited.	9001	
Atos Origin India Private Limited	9001:2000	Nivel 5
Aviation Software Development Consultancy India Ltd.		Nivel 3
AXA Business Services Pvt Ltd	9002	
Axes Technologies (I) Pvt Ltd	9001	
BAeHAL Software Limited	9001	
Bangalore Software Services Ltd	9001	
Bells Softech Limited	9001:1994	
Beni Softwares Pvt Ltd	9001:2000	
Bhari Information Technology Systems Pvt. Ltd.	9001	
Bharti Telesoft Ltd.	9001	
Birla Technologies Ltd	9001	
Birlasoft Limited	9001	

Botree Software International Ltd	9001	
BPL Telecom Ltd.	9001	Nivel 3
BT (Worldwide) Ltd.	9001	
Canbank Computer Services Ltd.	9001	
CA-TCG Software Pvt. Ltd.	9001	
CDS International Ltd	9001	
Cellnext Solutions Ltd	9001:2000	
Celstream Technologies Pvt Ltd	9001:2000	
Cerebra Integrated Technologies Ltd	9001:2000	
CG Maersk Information Technologies Pvt. Ltd.	9001	
CGI Information Systems and Management Consultants Pvt Ltd	9001	Nivel 4
CG-Smith Software Limited		Nivel 5
CG-VAK Software Exports Ltd.	9001	
Chandigarh Infotech Centre Ltd.	9001:2000	
Churchill India Pvt. Ltd.	9002	
CMC Limited	9001:1994	Nivel 3
Cognizant Technology Solutions India Pvt. Ltd.	9001	Nivel 5
Compudyne Winfosystems Limited	9001:2000	
Compulink Systems Pvt Ltd	9001	
Computer Clinic India Pvt. Ltd.	9001	
Congruent Solutions Pvt. Ltd.	9001	
Consoliintd Cybernetics Co. Pvt. Ltd.	9001:2000	
Consoliintd Futuristic Solutions Ltd	9001	
Consulting Engineering Services (I) Ltd	9001	
Contech Software Ltd.	9001	
Convergent Software Limited		Nivel 3
Covansys India (Private) Ltd.	9001	Nivel 5
Crossword Software Pvt Ltd	9001	
Datamatics Ltd.	9001	PCMM Nivel 2
Datamatics Technologies Pvt. Ltd.	9001:2000	PCMM Nivel 2
DCM Datasystems Ltd.	9001	
DCM Technologies Limited		Nivel 5
DDE ORG Systems Ltd.		9001:2000
Differential Technologies Limited	9001	

Digital GlobalSoft Ltd.	9001	Nivel 4
Divine India Limited	9001:2000	
DPS Technologies India Pvt Ltd	9002:1994	
DSL Software Ltd.	9001	
DSQ Software Limited	9001	Nivel 5
DSS Infotech International Ltd.	9001	
Duncan Infotech	9001	
Dusk Valley Technologies Ltd	9001:2000	
Eastern Software Systems Ltd.	9001	Nivel 5
E-Brahma Technologies (P) Ltd.	9001:1994	
eFunds International Private Limited		Nivel 3
Eonour Software Ltd.	9001	
Eximsoft Technologies Pvt. Ltd.	9001	
EXplora InfoTech Limited	9001:2000	
FCS Software Solutions Ltd	9001	
Financial Technologies (India) Ltd.	9001	Nivel 3
Future Software Limited	9001:2000	Nivel 5, PCMM Nivel 3
GAVS Information Services Pvt. Ltd.	9001:2000	
Genisys Integrating Systems (I) Pvt. Ltd.	9001	
Global Dialnet Limited	9001	
Global Infosystems Ltd	9001	
Globalsoft Pvt. Ltd.	9001	
Globsyn Technologies Ltd.	9001	
Godrej Infotech Ltd	9001, 9002	Nivel 4
Goldstone Technologies Ltd	9001:2000	
Growth Compusoft Exports Ltd.	9002	
GTL Limited	9001:2000	Nivel 4
HCL Infosystems Ltd	9001	Nivel 4
HCL Perot Systems Pvt Ltd	9001	Nivel 5
HCL Technologies Ltd	9001	Nivel 5
Hewlett-Packard (India) Software Operations Pvt Ltd	9001:2000	Nivel 5
Hexaware Technologies Limited	9001	Nivel 5
Hinduja TMT Ltd	9001:1994	
Honeywell India Software Operations Pvt. Ltd.	9001	Nivel 4

HTC Software Development Centre	9001	Nivel 3
Hughes Software Systems Ltd 9001		Nivel 4
i2 Technologies India Pvt. Ltd.	9001	
IBILT Technologies Ltd.	9001:9002	
IBM Global Services India Pvt. Ltd.	9001:2000	Nivel 5
ICICI Infotech Services Ltd.	9001	Nivel 3
iCope Technologies Private Limited	9001	
Ideaspace Solutions Ltd.	9001:2000	
i-flex solutions ltd.		Nivel 5
IIS Scientific Computing Ltd	9001:2000	
India Education Centre Softwares Limited	9001	
Indus Software Pvt. Ltd.	9001	
Infinite Computer Solutions (India) Pvt. Ltd.		Nivel 3
Information Technologies (India) Limited	9001	Nivel 5
Information Technology Park Ltd.	9000	
Infosys Technologies Ltd.	9001	Nivel 5
Infotech Enterprises Ltd.	9001,9002	Nivel 5
Infozech Software (P) Ltd.	9001	
InfraSoft Technologies Limited	9001:2000	
Intelligroup Asia Pvt. Ltd.	9001	Nivel 5, PCMM Nivel 2
InteQ Software Limited	9001	
IonIdea Enterprise Solutions Pvt Ltd	9001	Nivel 3
iS3C Consultancy Services Ltd.	9001	Nivel 4
iSeva Systems Pvt Ltd	9001:2000	
IT Solutions (India) Pvt. Ltd.	9001	Nivel 5
ITC Infotech India Ltd	9001	Nivel 5
ITTI Limited	9001	
Ivega Corporation	9001	Nivel 4
IVL India Pvt. Ltd.	9001	
JK Technosoft Ltd	9001	
JP Mobile (India) Limited	9001:2000	
Kale Consultants Pvt. Ltd.	9001	
Kals Information Systems Ltd	9001	
Kanbay Software (India) Pvt. Ltd.	9001:1994	
Kaneriya Technologies Limited	9002	

Karvy Consultants Ltd	9002	
Keane India Ltd	9001	Nivel 5
KLG Systel Ltd.	9001	
KPIT Infosystems Limited.	9001	Nivel 4
Larsen and Toubro Infotech Limited	9001:2000	Nivel 4
LG Soft India Pvt. Ltd.	9001	Nivel 5
Linc Software Services Pvt. Ltd.	9001:1994	
Live Tech Solutions (P) Ltd	9000	
Lucent Technologies India (P) Ltd.	9001	
Magic Software Pvt. Ltd.	9001	
Mahindra British Telecom Ltd.	9001	Nivel 3
Majoris Systems Pvt. Ltd.	9001	
MASCON Global Limited	9000	
Mascot Systems Ltd.	9001:2000 CMM©	Nivel 4
MASTEK Ltd.	9001	PCMM Nivel 3, Nivel 5
MECON Limited	9001	
Medicom Solutions (P) Ltd	9001	
Megasoft Limited	9001:1994	
Melstar Information Technologies Ltd	9001:2000	Nivel 3
Micro Technologies (India) Ltd	9001:1994	
Mindteck (India) Ltd	9001:1994	
Momentum India Private Limited	9001	
Motorola India Electronics Private Ltd.		Nivel 5
Mphasis BFL Ltd.	9001	Nivel 5
MTC (India) Pvt. Ltd.	9001:2000	
NatureSoft Private Limited	9001	
Navayuga Infotech Pvt. Ltd.	9001	
Neptune Information Solutions Limited	9001	
Network Programs (India) Ltd.		Nivel 3
Network Systems and Technologies (P) Ltd.	9001	Nivel 5
Newgen Imaging Systems (P) Ltd.	9001	
NIIT Ltd.	9001	Nivel 5
Nokia Telecommunications Pvt. Ltd.	9000	
Onward Technologies Limited	9002	
Opusasia Technologies Pvt. Ltd	9001	

Oracle India Private Limited.	9001	Nivel 4
Oracle Solution Services (India) Pvt Ltd	9001	Nivel 4
OrbiTech Solutions Limited		Nivel 5
Orient Information Technology Ltd.	9002	
Paharpur Business Centre	9002	
Paragon Solutions (I) Pvt. Ltd.		Nivel 3
Patni Computer Systems Ltd.	9001:2000	Nivel 5
Pentamedia Graphics Limited	9001	Nivel 4
Pentasoft Technologies Limited	9001	Nivel 4
Philips Software Centre Pvt. Ltd.	9001	Nivel 5
Phoenix Global Solutioins (India) Pvt. Ltd.	9001	
Planet PCI Infotech Ltd	9001	
PlanetAsia.com Limited		Nivel 4
Polaris Software Lab Ltd	9001	CMMI Nivel 5
Premier Technology Group Pvt. Ltd.	9001	
Princeton Software Exports Pvt. Ltd.	9001:2000	
PSI Data Systems Ltd.	9001	
Pyxis Technology Solutions Ltd	9001	
QAI (India) Limited		PCMM Nivel 2
Quinnox Consultancy Services Ltd		Nivel 5
R S Software (India) Ltd.	9001	PCMM Nivel 3, Nivel 4
R Systems International Limited	9001:2000	Nivel 4
Ram Informatics Ltd.	9001	
Ramco Systems Ltd	9001:1994	
RelQ Software Pvt Ltd	9001:2000	
Resonance Technologies (P) Ltd.	9001	
Rishabh Software Pvt Ltd	9001:2000	
River Run Software Group	9001	
Robert BOSCH India Limited	9001:1994	Nivel 3
Rolta India Ltd.	9001:2000	
SAP India Pvt. Ltd.	9001,9002	
Sar Softech Pvt Ltd	9001	
Sasken Communication Technologies Limited	9001:1994	
Satyam Computer Services Ltd.	9001:2000	Nivel 5
Selectronic Equipment and Services Pvt. Ltd.	9002	

Shyama Software Solutions (I) Pvt Ltd	9001	
Siemens Information Systems Ltd.	9001	Nivel 5, PCMM Nivel 3
Siemens Public Communications Network Ltd.	9001	
Sierra Optima Ltd.	9001:2000	
Silverline Technologies Ltd.	9001	Nivel 4
Siri Technologies Private Limited	9001	
Sitel India Pvt Ltd	9000	
Softek Limited	9001:1994	
Solitar Systems (India) Pvt Ltd	9001:2000	
SolutionNET India (Pvt) Ltd	9001:2000	
Sonata Software Limited	9001	Nivel 5
Sovika Infotek Ltd.	9001	
Speck Systems Limited	9001	
SQL Star International Limited	9001:2000	
SRA Systems Ltd.	9001	Nivel 4
Srishti Software Private Limited		Nivel 3
SSI Technologies	9001,9002	Nivel 5
Subex Systems Limited	9001	
Summit Information Technologies Ltd	9001:2000	
Syntel (India) Ltd.	9001	Nivel 5
Systems and Software	9001	
Tata Consultancy Services	9001	Nivel 5, PCMM Nivel 4
Tata Elxsi (India) Ltd.	9001,9002	Nivel 5
Tata Infotech Ltd.	9001,9002	Nivel 4
Tata Interactive Systems	9001	Nivel 5
Tata Technologies Limit ed	9001:2000	
TCG Software Services Pvt. Ltd.	9001	
TCIL BellSouth Ltd.	9001	
TechSpan India Ltd.		PCMM Nivel 3
Tektronix Engineering Development (I) Ltd.	9000:2001	
Telecommunications Consultants India Ltd.	9001	
Temenos India Pvt. Ltd.	9000	
Thermax Systems and Software	9001	
Three S Solutions Ltd	9002	
Trident Infotech Corporation Ltd.	9001:2000	

Trigent Software Ltd.	9001	
UshaComm India Pvt. Ltd.	9001	
Ushus Technologies Pvt Ltd	9001	
Value Software Technologies Pvt. Ltd.	9001	
Visesh Infosystems Limited	9001	
Vispark Solutions (I) Pvt Ltd	9001	
VJIL Consulting Limited	9001	
VXL eTech Ltd		Nivel 3
Web Spiders (I) Pvt Ltd	9001	
Webify Services (India) Private Limited	9001	
West Bengal Electronics Industry Development Corporation Ltd.	9002	
Wipro Technologies	9001	Nivel 5
WNS Global Services	9001	
Xansa (India) Ltd	9001	Nivel 5
Xcelvision Technologies Limited	9001:2000	
Xerox Modicorp Limited		Nivel 3
Zenith Computers Ltd.	9000	
Zensar Technologies Limited	9001	Nivel 5, EFQM Nivel 2

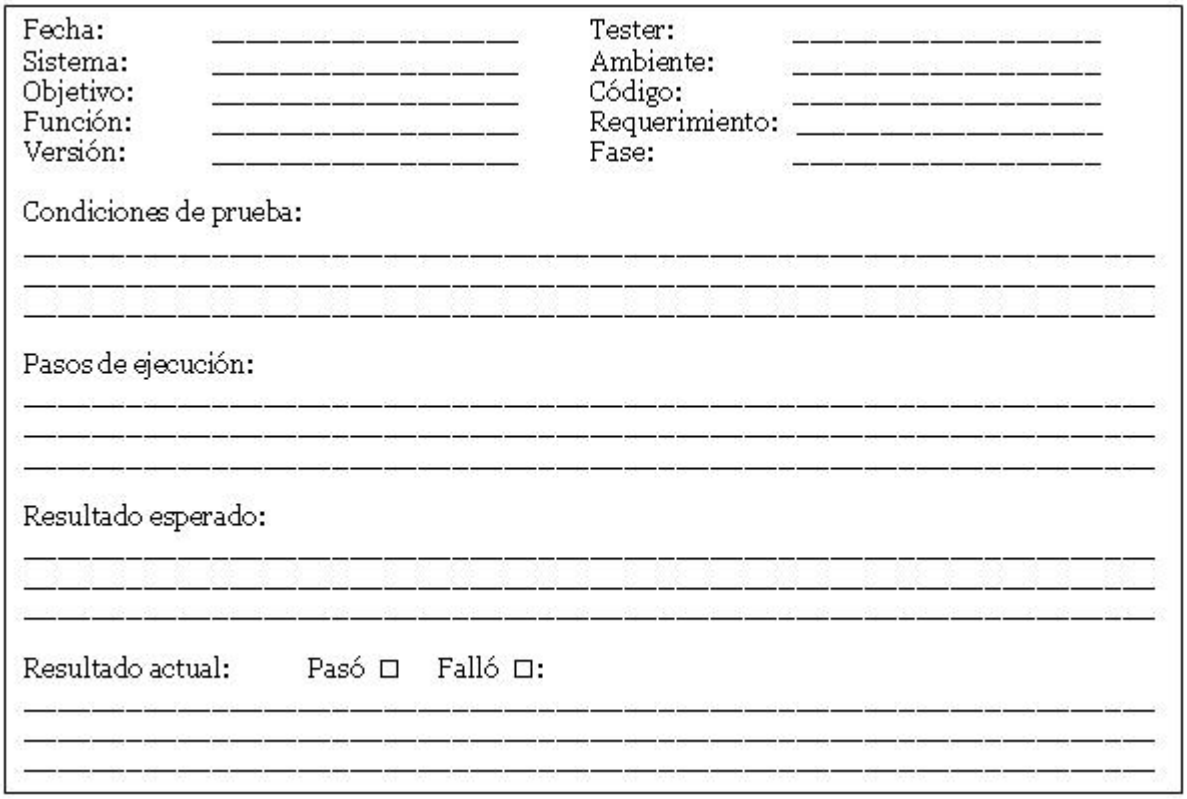
Cuadro A.1: Empresas certificadas por la ISO y el CMM©

ANEXO B

CASOS DE PRUEBA Y LISTAS DE CONTROL

Los siguientes son modelos de casos de prueba y extractos de listas de control basadas en las que William Lewis presenta en su libro *Software Testing and Continuous Quality Improvement.* [LEW00].

Caso de Prueba


Fecha: _____ Tester: _____
Sistema: _____ Ambiente: _____
Objetivo: _____ Código: _____
Función: _____ Requerimiento: _____
Versión: _____ Fase: _____

Condiciones de prueba:

Pasos de ejecución:

Resultado esperado:

Resultado actual: Pasó ☐ Falló ☐:

Figura B.1: Modelo de Caso de Prueba [LEW00].

Listas de Control

Sí: La sentencia se cumple.

No: La sentencia no se cumple.

Om: La sentencia fue omitida.

NP: La sentencia aún no fue probada.

1. **Fase de requerimientos**

Defecto	Sí	No	Om	NP
1. La lógica del negocio está siendo utilizada inadecuadamente.				
2. El criterio para el performance está siendo mal utilizado.				
3. La información del entorno es errónea, insuficiente o irrelevante.				
4. El objetivo del sistema no ha sido bien planteado.				
5. Los requerimientos son incompatibles.				
6. Los requerimientos son incompletos.				
7. Faltan algunos requerimientos.				
8. Los requerimientos son erróneos.				
9. La exactitud especificada no está conforme con la necesidad actual.				
10. El entorno de los datos no está descrito de forma adecuada.				
11. La definición de interfaces externas es errónea.				
12. El entrenamiento para el usuario no ha sido bien considerado.				
13. El estado del sistema al ser inicializado no ha sido bien considerado.				
14. Las funciones no han sido bien definidas.				
15. Las necesidades del usuario no han sido bien planteadas.				
16. Las métricas de calidad no han sido bien especificadas.				

Cuadro B.1: Criterios a considerar en el testeo de requerimientos

2. **Fase de diseño lógico**

Defecto	Sí	No	Om	NP
1. Los datos no han sido definidos apropiadamente.				
2. La definición de la entidad está incompleta.				
3. La cardinalidad de la entidad es incorrecta.				

4. Los atributos de la entidad están incompletos.				
5. La normalización no es correcta.				
6. La clave principal no es correcta.				
7. La clave foránea no es correcta.				
8. La clave compuesta no es correcta.				
9. El sub-tipo de la entidad es incorrecto.				
10. El proceso no ha sido bien definido.				
11. Los "procesos padre" no están bien definidos / están incompletos.				
12. Los "procesos hijo" no están bien definidos / están incompletos.				
13. Las salidas/entradas no están bien definidas.				
14. Los procesos elementales no están bien definidos.				
15. Existe el problema de exclusión mutua.				
16. La activación de eventos no está bien definida.				
17. La creación de asociaciones entre entidades/procesos es incorrecta.				
18. La lectura de asociaciones entre entidades/procesos es incorrecta.				
19. La actualización de asociaciones entre entidades/procesos es incorrecta.				
20. La eliminación de asociaciones entre entidades/procesos es incorrecta.				

Cuadro B.2: Criterios a considerar en el testeo del diseño lógico de la aplicación

3. **Fase de diseño físico**

Defecto	Sí	No	Om	NP
1. La lógica y/o secuencia es errónea.				
2. El procesamiento es inexacto.				
3. Una rutina no tiene requisitos de entrada/salida.				
4. La rutina no acepta los datos con los rangos establecidos.				
5. La validación está hecha en los datos de entrada.				
6. Los procedimientos no son los adecuados.				
7. Falta un procesamiento correcto.				
8. Los valores son erróneos o ambiguos.				
9. Los datos almacenados son inadecuados.				
10. Faltan algunas variables.				
11. Los requerimientos de diseño han sido malinterpretados / no están presentes.				
12. La Base de Datos no es compatible con el entorno de los datos.				

13. La descomposición modular refleja una alta dependencia intermodular.				
14. Los algoritmos más importantes no han sido evaluados en función de su exactitud y velocidad.				
15. La estructura de control no es extensible.				
16. La estructura de control omite las prioridades de procesamiento.				
17. Los protocolos de las interfaces son incorrectos.				
18. La lógica utilizada para la implementación de los algoritmos es incorrecta.				
19. Los datos no son formateados de la forma correcta.				
20. No se han considerado los efectos del trunqueo de datos.				
21. Los índices no son validados.				
22. Se permiten ciclos infinitos.				
23. La especificación de los módulos ha sido malinterpretada.				
24. Las reglas de la Base de Datos han sido violadas.				
25. La lógica está incompleta para todos los casos.				
26. Los casos especiales son ignorados.				
27. El manejo de errores es deficiente.				
28. Las consideraciones de tiempo han sido ignoradas.				
29. Los requerimientos han sido cambiados en los módulos.				
30. Las especificaciones de las interfaces han sido malinterpretadas.				
31. El sistema funciona bien pero no cumple con los requerimientos de performance.				
32. El sistema no es lo suficientemente completo como para satisfacer la necesidad planteada.				
33. El “overflow” de operaciones aritméticas no ha sido considerado apropiadamente.				
34. Las acciones para las entradas dadas no son correctas.				
35. La aproximación de los algoritmos no provee una exactitud adecuada.				
36. Existen errores en el diseño detallado para resolver un problema en particular.				
37. Existen casos particulares de entradas cuyo resultado luego de un procesamiento es una salida no considerada.				
38. Existen algoritmos que pueden ser reemplazados por otros más eficientes.				

Cuadro B.3: Criterios a considerar en el testeo del diseño físico de la aplicación

4. **Fase de diseño de unidad de programa**

Defecto	Sí	No	Om	NP
1. ¿Las estructuras "if-then-else" son usadas correctamente?				
2. ¿Los ciclos ("while", "for") son utilizados correctamente?				
3. ¿Existen ciclos infinitos?				
4. ¿El programa es fácilmente legible?				
5. ¿El programa es eficiente?				
6. ¿Las estructuras "case" contemplan todos los casos?				
7. ¿Existe trozos de "código muerto"?				
8. ¿Los algoritmos están bien escritos?				
9. ¿Existen demasiadas anidaciones?				
10. ¿Existen expresiones booleanas demasiado complejas?				

Cuadro B.4: Criterios a considerar en el testeo del diseño de unidad de programa

5. **Fase de implementación**

Defecto	Sí	No	Om	NP
1. La lógica para la toma de decisiones es inadecuada.				
2. Los cálculos aritméticos son erróneos.				
3. Las ramificaciones están mal.				
4. Existen ciclos infinitos porque están mal definidos.				
5. Las reglas del lenguaje de programación son violadas.				
6. Los estándares de programación no son respetados.				
7. Existen errores de typeo.				
8. Existen errores en el formato de las entradas y salidas.				
9. Existen errores en la invocación a otros programas.				
10. Existen errores en los datos.				
11. Existen subprogramas inconclusos.				
12. El manejo de errores es incorrecto.				
13. Existen errores en el procesamiento de datos para las salidas.				
14. Existen errores en las interfaces.				
15. Existen errores de sintaxis.				
16. Existen errores de inicialización.				
17. Existe confusión en el uso de los parámetros.				

18. Los contadores en los ciclos no funcionan correctamente.				
19. El resultado de las decisiones que se toman no es manejado correctamente.				
20. Los nombres de las variables no son apropiados.				
21. Las librerías a utilizar no están claramente definidas.				
22. El tratamiento de caracteres especiales es incorrecto.				
23. Existen errores de compilación.				
24. Existen problemas de "overflow".				
25. El software para interactuar con el hardware no funciona correctamente.				
26. Hay problemas a la hora de modificar los registros de la Base de Datos.				

Cuadro B.5: Criterios a considerar en el testeo de la fase de implementación de la aplicación

6. **Fase de pruebas**

Defecto	Sí	No	Om	NP
1. ¿Los códigos son validados correctamente?				
2. ¿Los campos pueden ser actualizados sin problemas?				
3. ¿El largo de los campos es el adecuado?				
4. ¿La descripción de los campos es adecuada?				
5. ¿Los campos son inicializados correctamente?				
6. ¿Todas las referencias a un campo determinado son correctas?				
7. ¿Las restricciones de cada campo son validadas correctamente?				
8. ¿Existe un control de errores que verifique que las reglas establecidas sean respetadas?				
9. Para valores numéricos, ¿se hace un control de valores positivos y negativos?				
10. Para valores alfanuméricos, ¿el caso de "vacío" ha sido contemplado?				
11. ¿Los propietarios de la información han testeado estos casos?				
12. ¿Los propietarios de la información han otorgado el resultado de sus pruebas?				

Cuadro B.6: Criterios a considerar en el testeo de la aplicación en la fase de pruebas

ANEXO C

ESPECIFICACIÓN DE REQUERIMIENTOS Y PLAN DE PRUEBAS

Especificación de Requerimientos

El siguiente modelo es el recomendado por el IEEE y fue extraído del libro de William Lewis (2000).

1.- Introducción

- 1.1.- Propósito
- 1.2.- Alcance
- 1.3.- Definiciones: Acrónimos y Abreviaciones
- 1.4.- Referencias
- 1.5.- Resumen

2.- Descripción general

- 2.1.- Perspectiva del producto
  - 2.1.1.- Interfaces del sistema
  - 2.1.2.- Interfaces de usuario
  - 2.1.3.- Interfaces del Hardware
  - 2.1.4.- Interfaces del Software
  - 2.1.5.- Comunicación entre las interfaces
  - 2.1.6.- Limitaciones de memoria
  - 2.1.7.- Operaciones
  - 2.1.8.- Requerimientos de adaptación
- 2.2.- Funciones del producto
- 2.3.- Características del usuario
- 2.4.- Limitaciones
- 2.5.- Suposiciones y Dependencias.
- 2.6.- Reparto de Requerimientos

3.- Especificación de Requerimientos

- 3.1.- Requerimientos de la Interface Externa
  - 3.1.1.- Interfaces de usuario
  - 3.1.2.- Interfaces del Hardware
  - 3.1.3.- Interfaces del Software
  - 3.1.4.- Interfaces de comunicación
- 3.2.- Características del Sistema
  - 3.2.1.- Característica 1
    - a' Introducción/Objetivo de la característica.
    - b' Secuencia Estímulo/Respuesta
    - c' Requerimientos funcionales asociados
  - 3.2.2.- Característica 2
  - 3.2.3.- ...
- 3.3.- Requerimientos de Performance
- 3.4.- Limitantes del Diseño
- 3.5.- Atributos del Software
- 3.6.- Otros Requerimientos

4.- Información de soporte

- 4.1.- Tabla de Contenidos
- 4.2.- Apéndices

Plan de Pruebas

1.- Introducción

- 1.1.- Estrategia de testeo.
- 1.2.- Alcance del testeo.

2.- Testeo estático

- 2.1.- Defectos descubiertos y corregidos
- 2.2.- Ideas de mejora
- 2.3.- Estándares del lenguaje utilizado.
- 2.4.- Estándares de la documentación del desarrollo.

3.- Casos de Testeo (Testeo dinámico)

- 3.1.- Datos de entrada
- 3.2.- Condiciones iniciales
- 3.3.- Resultados esperados
- 3.4.- Historial

4.- Requerimientos del entorno

- 4.1.- Estrategia de testeo
- 4.2.- Plataforma
- 4.3.- Librerías
- 4.4.- Herramientas
- 4.5.- Procedimientos
- 4.6.- Reportes

ANEXO D

DIAGRAMAS Y FUNCIONES

En este anexo se detallan los diagramas (casos de uso, colaboración, estado y secuencia) y funciones que el sistema desempeñará organizados según el tipo de usuario.

1.- Usuario Genérico

1.1.- Diagrama de casos de uso

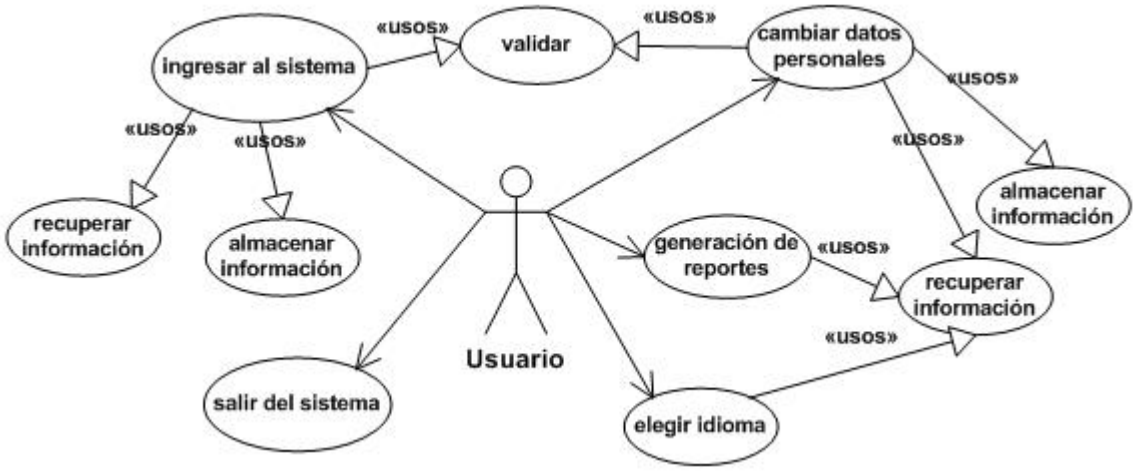


Figura D.1: Diagrama de casos de uso detallado para un usuario genérico del sistema.

1.2.- Funciones y diagramas de colaboración, estado y secuencia

Función:	**Ingreso de un usuario al sistema**
Descripción:	Esta función permite al usuario ingresar al sistema luego de registrar sus datos.
Entrada:	Nombre de usuario y contraseña pertenecientes al usuario.
Fuente:	Usuario
Salida:	Pantalla principal de la herramienta si los datos son correctos. Ventana de aviso de error en caso contrario.
Destino:	El mismo usuario.
Precondición:	Que el usuario tenga una cuenta creada en el sistema.

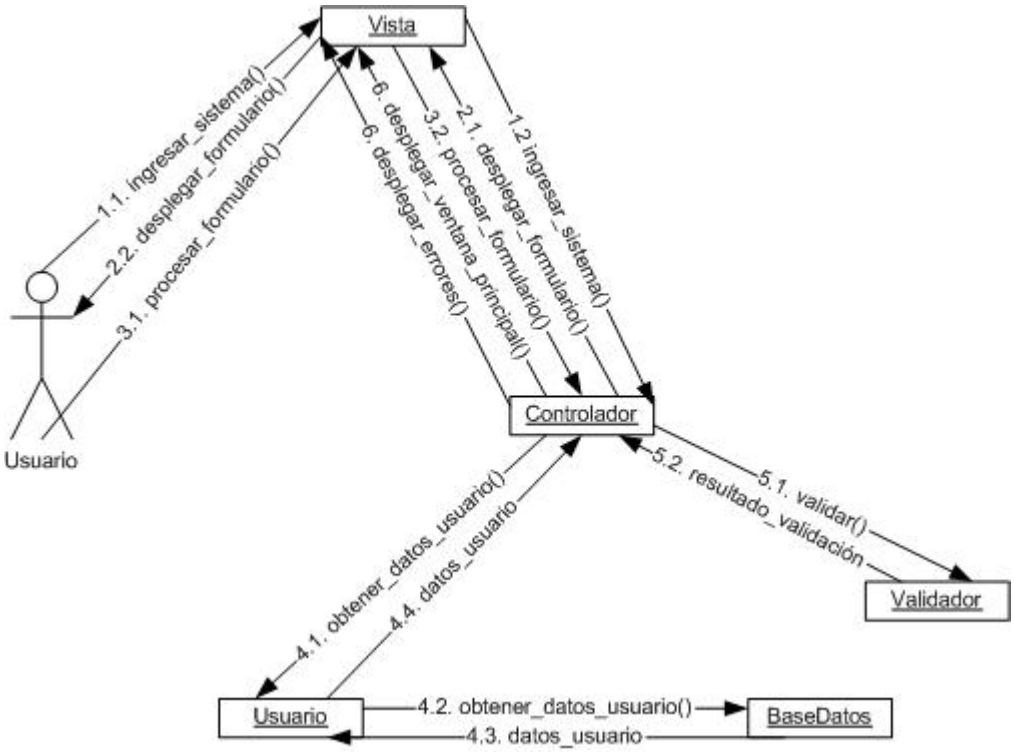


Figura D.2: Diagrama de colaboración para la función "ingresar al sistema".

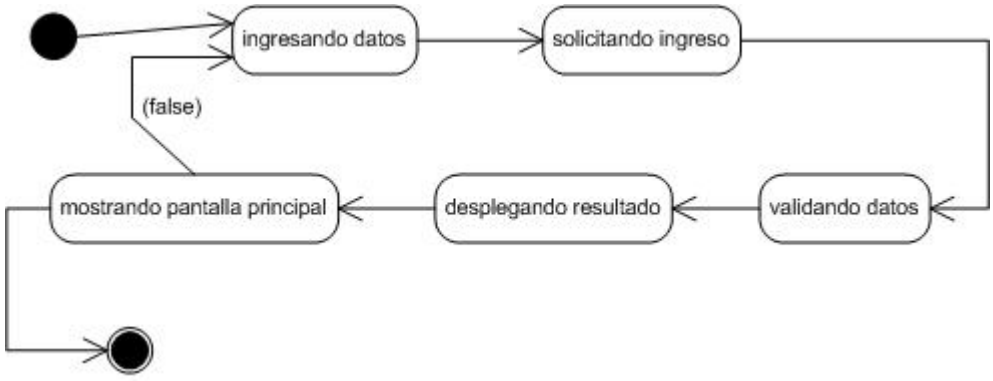


Figura D.3: Diagrama de estados para la función "ingresar al sistema".

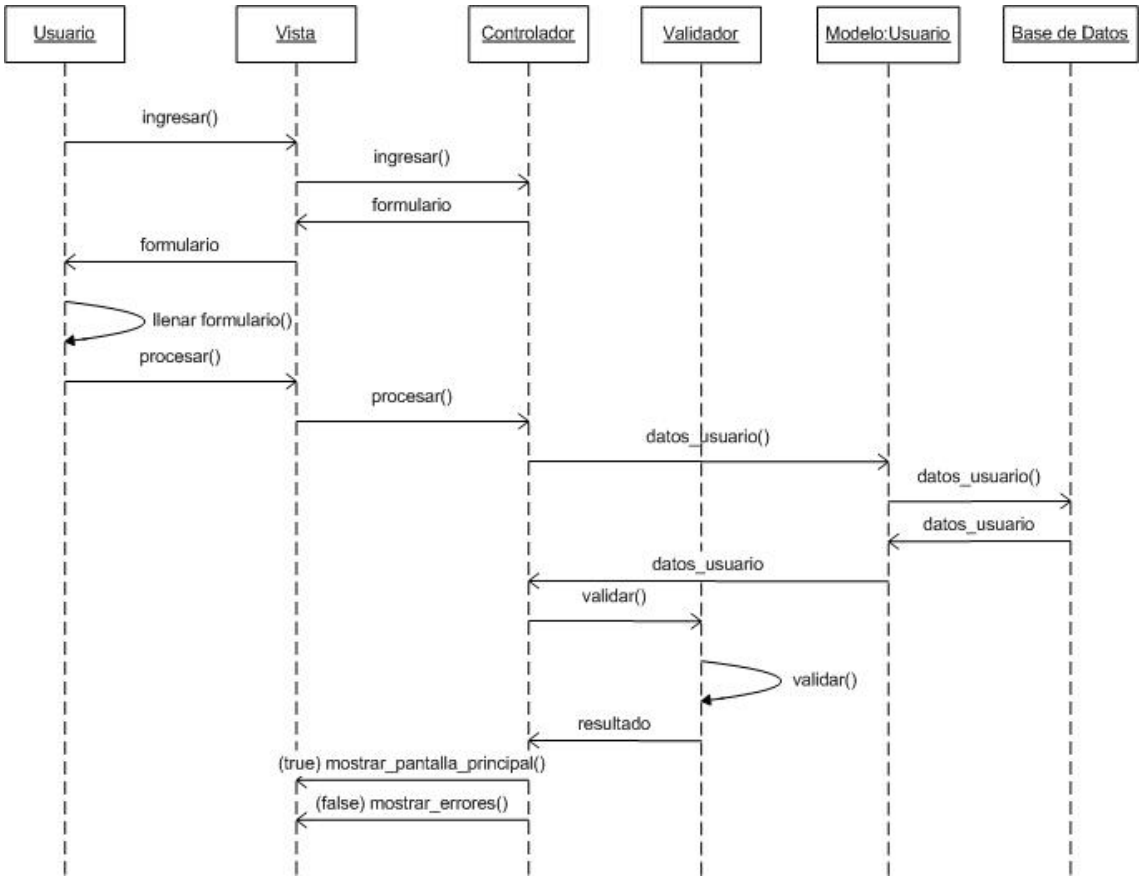


Figura D.4: Diagrama de secuencia para la función “ingresar al sistema”.

Función:	**Elección del idioma**
Descripción:	Esta función permite al usuario la elección del idioma con el que desea trabajar en la sesión que iniciará.
Entrada:	Identificador del idioma a utilizar.
Fuente:	Usuario (sin importar el tipo).
Salida:	La aplicación en el idioma elegido.
Destino:	El mismo usuario.
Precondición:	.

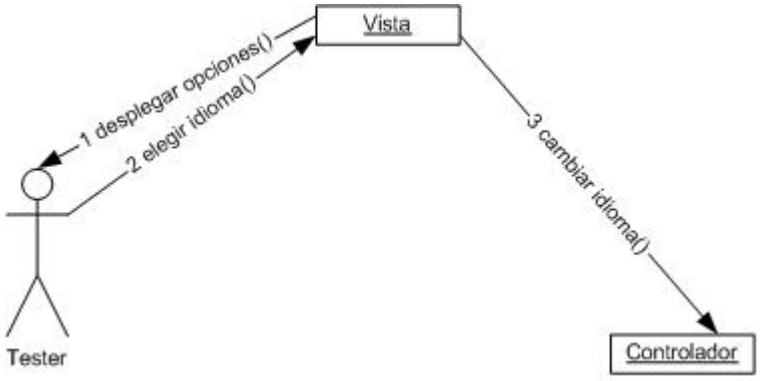


Figura D.5: Diagrama de colaboración para la función de "elegir idioma".

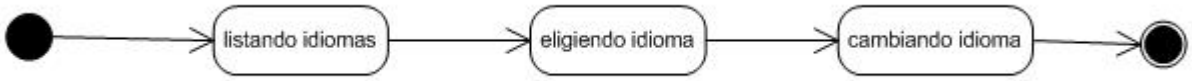


Figura D.6: Diagrama de estados para la función "elegir idioma".

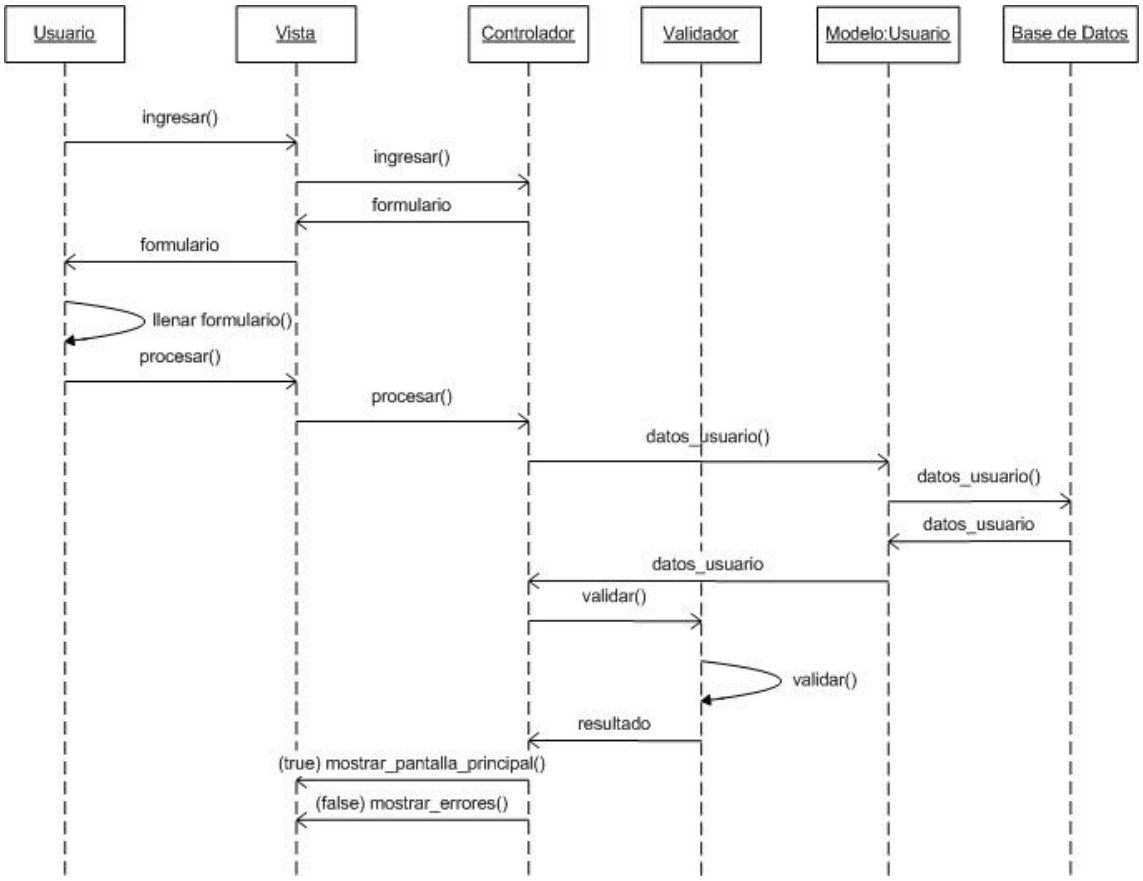


Figura D.7: Diagrama de secuencia para la función “elegir idioma”.

Función:	**Generación de listados**
Descripción:	Esta función permite la generación de listados en los que desplegará la información contenida en la Base de Datos hasta el momento con respecto a la opción seleccionada.
Entrada:	La entidad elegida para generar el listado (ej.: usuarios, proyectos, casos de testeo, etc.).
Fuente:	Líder de Proyectos, Tester, Administrador, Desarrollador.
Salida:	Listado.
Destino:	Gerencia, Equipo de testers, Equipo de desarrolladores, Líder del proyecto.
Precondición:	Que el criterio dado cuente con información en la Base de Datos.

Función:	**Generación de reportes**
Descripción:	Esta función permite la generación de reportes en los que desplegará la información contenida en la Base de Datos hasta el momento con respecto a la entidad seleccionada.
Entrada:	La entidad elegida y los criterios que intervendrán en la generación del reporte.
Fuente:	Líder de Proyectos, Administrador, Tester o Desarrollador.
Salida:	Reporte.
Destino:	Gerencia, Equipo de testers, Equipo de desarrolladores, Líder del proyecto.
Precondición:	Que los criterios dados sean coherentes y se cuente con información en la Base de Datos.

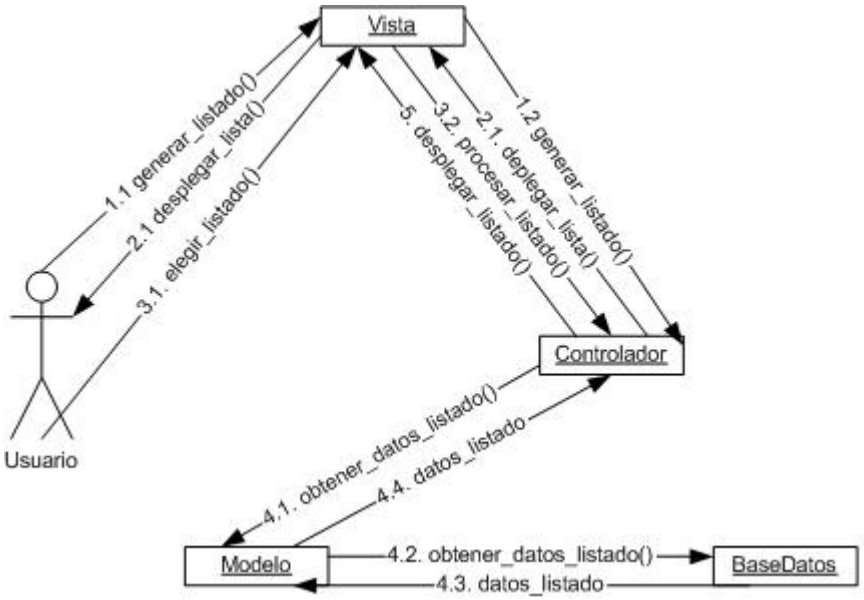


Figura D.8: Diagrama de colaboración para las funciones "generación de listados" y "generación de reportes".

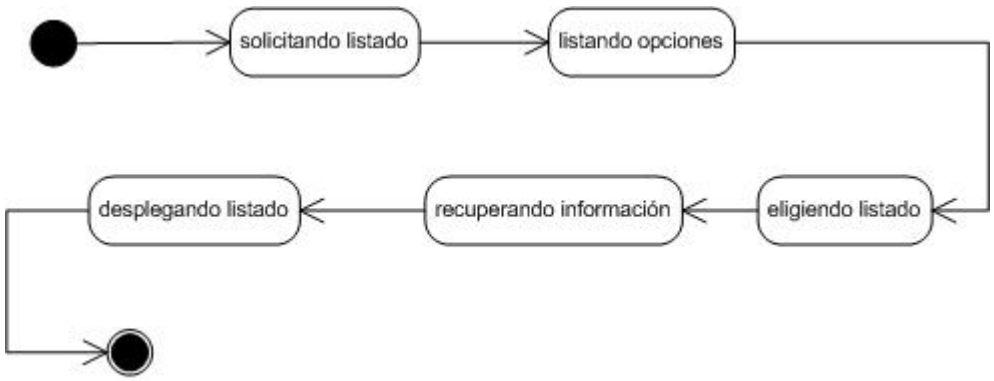


Figura D.9: Diagrama de estados para las funciones "generación de listados" y "generación de reportes".

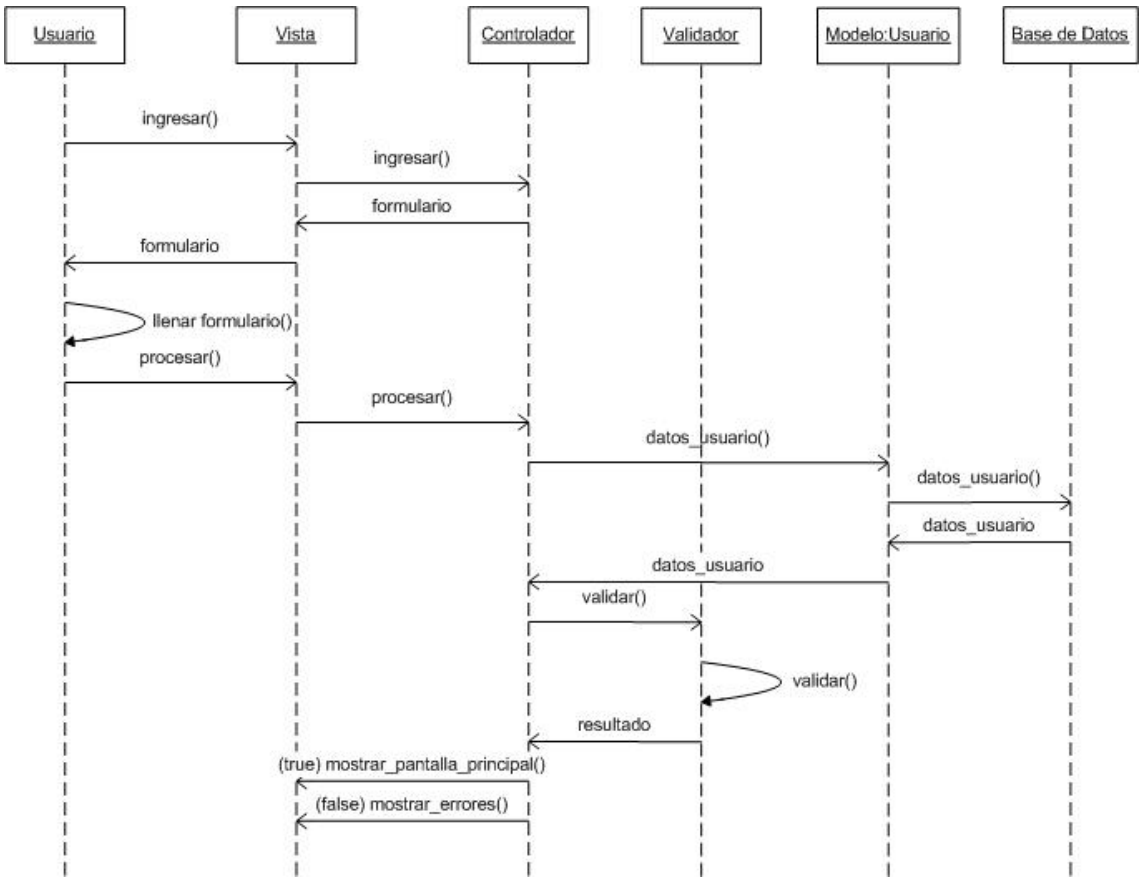


Figura D.10: Diagrama de secuencia para las funciones “generación de listados” y “generación de reportes”.

Función:	**Cambio de datos personales**
Descripción:	Esta función permite al usuario modificar algunos de sus datos personales.
Entrada:	Nuevos datos.
Fuente:	Usuario (sin importar el tipo).
Salida:	Modificación de datos en la Base de Datos.
Destino:	Base de Datos, usuario que generó la acción.
Precondición:	Que los nuevos datos sean válidos.

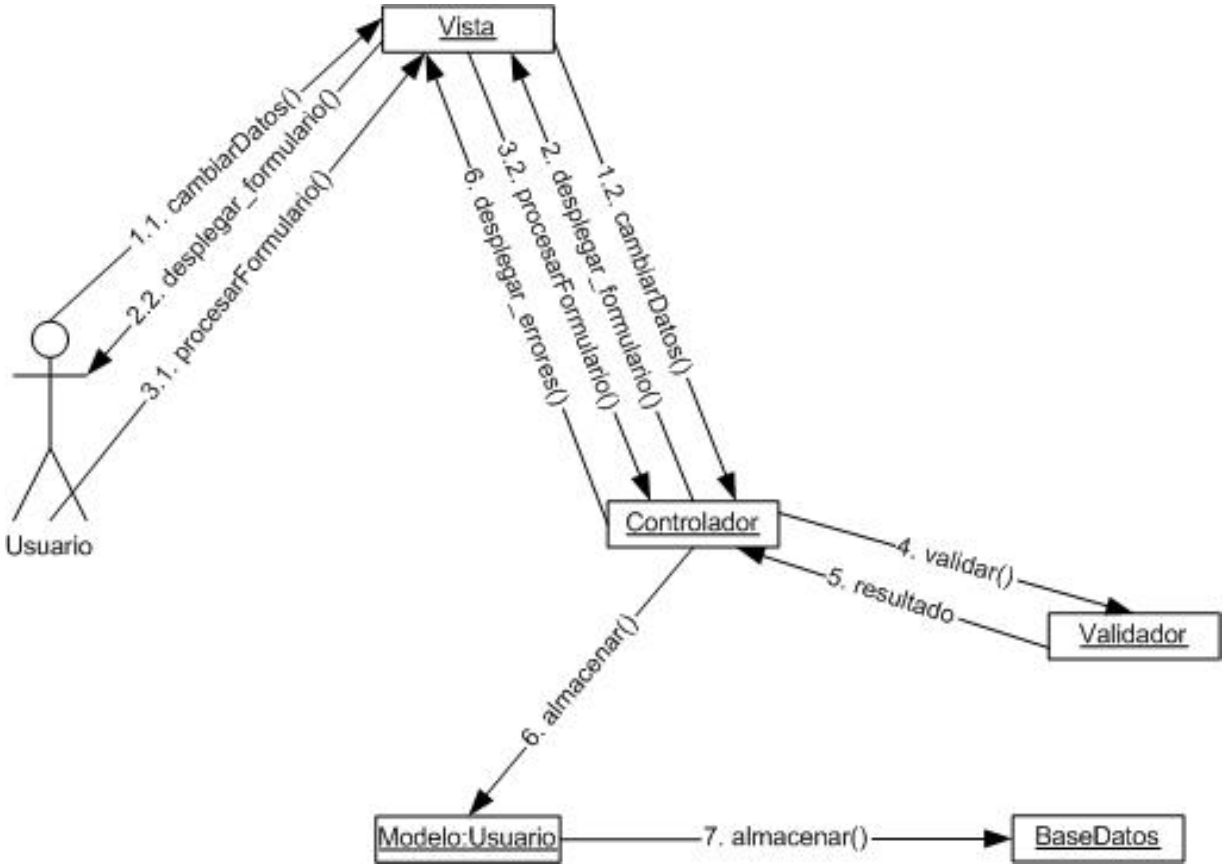


Figura D.11: Diagrama de colaboración para la función “cambio de datos personales”.

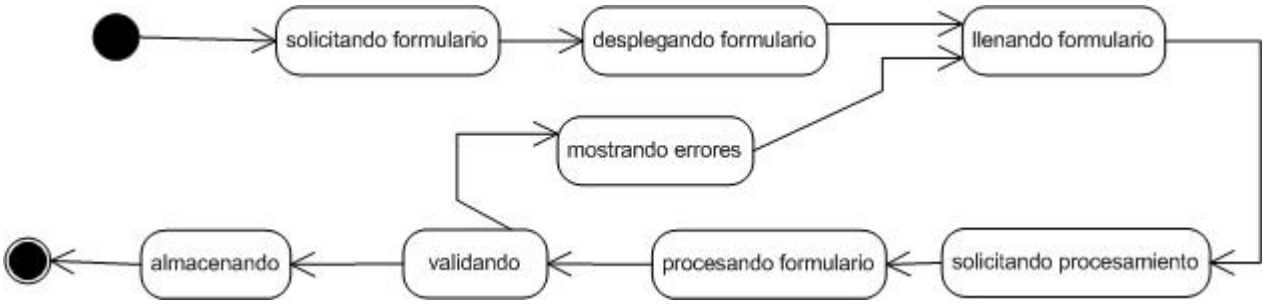


Figura D.12: Diagrama de estados para la función "cambio de datos personales".

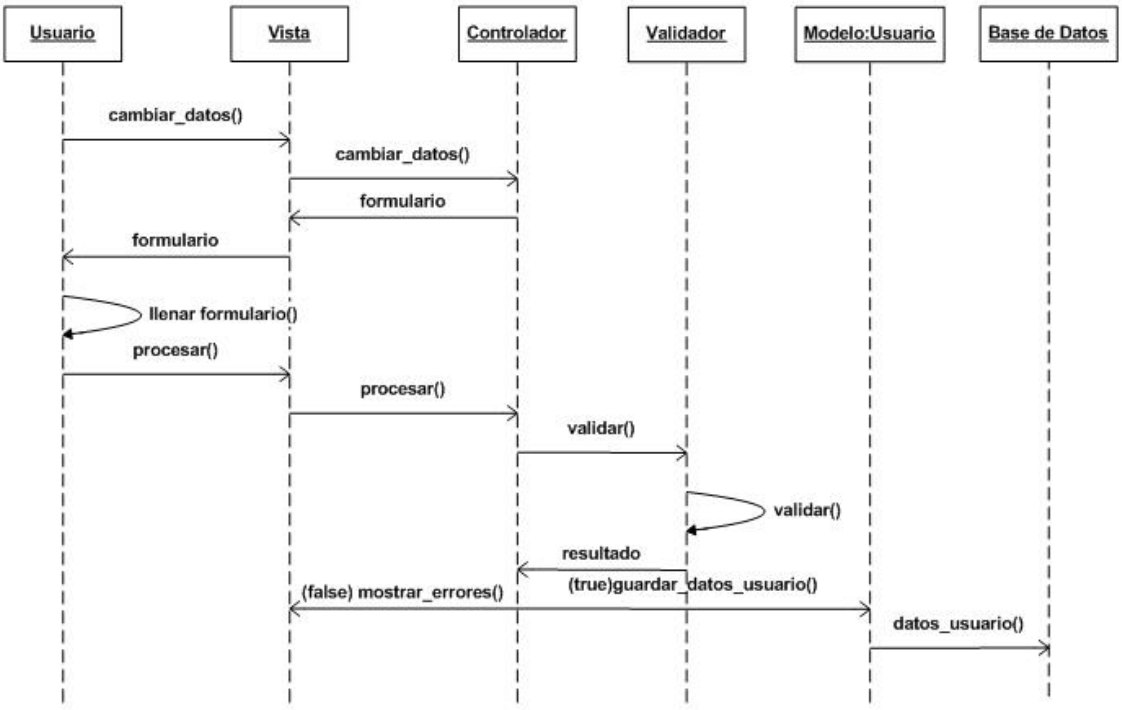


Figura D.13: Diagrama de secuencia para la función "cambio de datos personales".

Función:	**Cierre de la sesión**
Descripción:	Función que permite al usuario cerrar su sesión y salir de la aplicación.
Entrada:	Acción del usuario.
Fuente:	Usuario (sin importar el tipo).
Salida:	Eliminación de las variables de sesión y cierre de la aplicación.
Destino:	Usuario que originó la acción.
Precondición:	Ninguna.

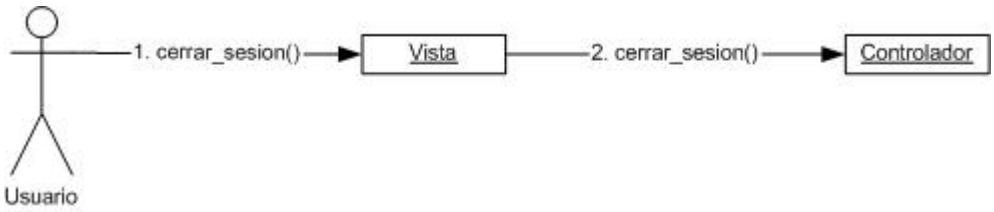


Figura D.14: Diagrama de colaboración para la función "cierre de sesión".

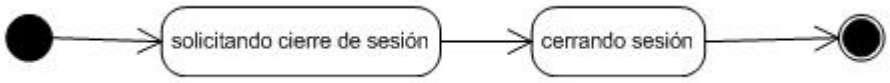


Figura D.15: Diagrama de estados para la función "cierre de sesión".

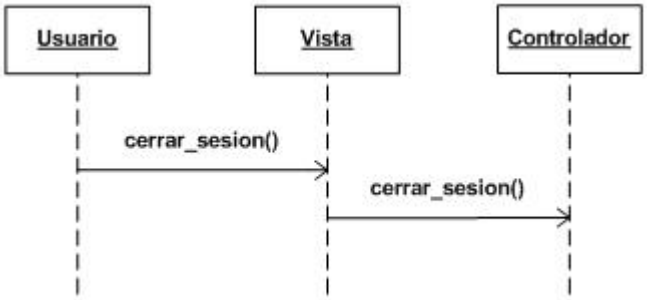


Figura D.16: Diagrama de secuencia para la función "cierre de sesión".

2.- **Usuario Administrador**

2.1.- Diagrama de casos de uso

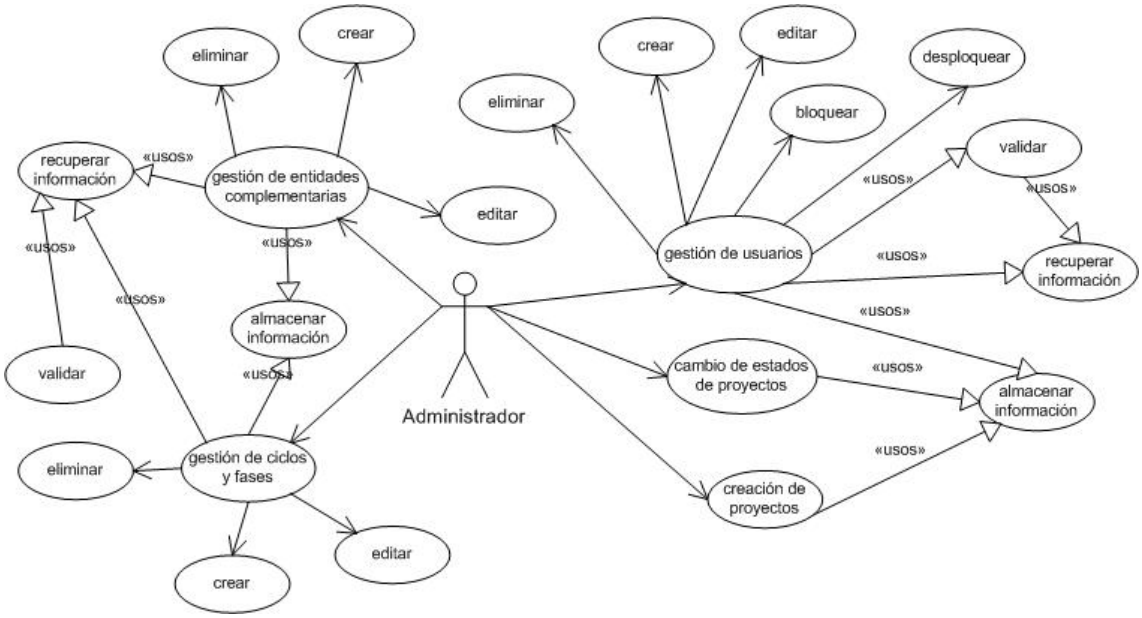


Figura D.17: Diagrama de casos de uso detallado para un usuario de tipo Administrador.

2.2.- Funciones y diagramas de colaboración, estado y secuencia

Función:	**Gestión de entidades complementarias**
Descripción:	Esta función hace referencia a las acciones de inserción, modificación y eliminación de todas las entidades complementarias del sistema (ej: prioridades, entornos, estados, etc.).
Entrada:	Nombre de la entidad, acción requerida.
Fuente:	Usuario.
Salida:	Listado de los registros de la entidad, formulario de edición o mensaje de confirmación de eliminación (depende de la acción elegida).
Destino:	Base de Datos, Ventana del usuario.
Precondición:	Que el usuario sea de tipo Administrador o Tester.

Función:	**Gestión de usuarios**
Descripción:	Esta función hace referencia a las acciones de inserción, modificación, eliminación bloqueo y desploqueo de los usuarios del sistema.
Entrada:	Datos del usuario
Fuente:	Usuario.
Salida:	Registro en la Base de Datos del nuevo usuario. Ventana de detalle para el usuario que lo creó.

Destino:	Base de Datos, Ventana del usuario.
Precondición:	Que el usuario sea de tipo Administrador.

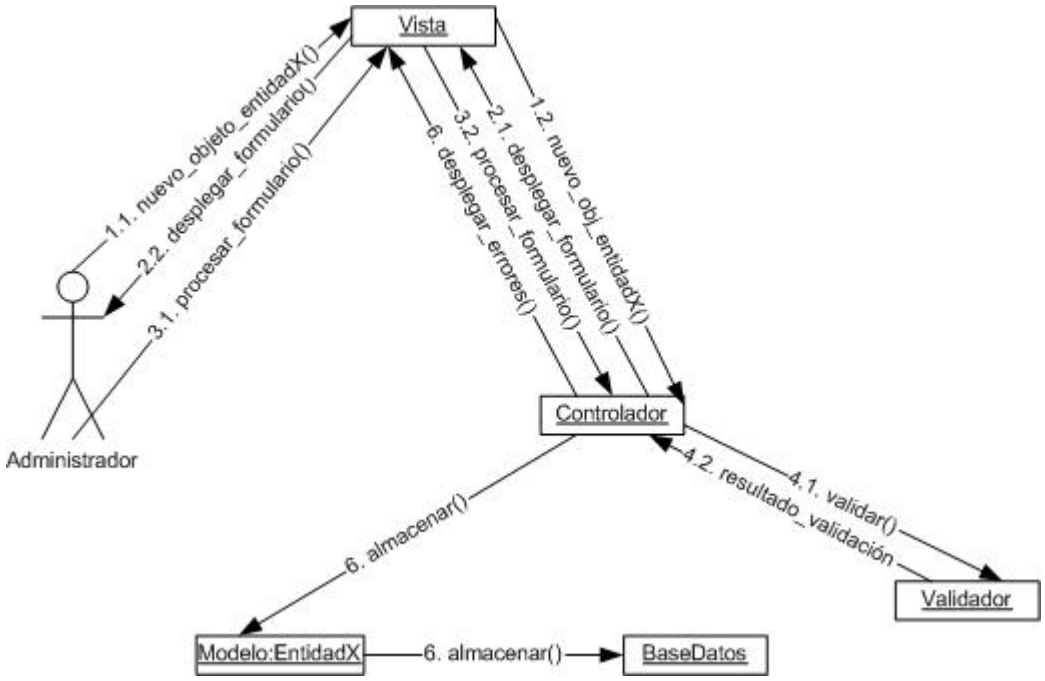


Figura D.18: Diagrama de colaboración para las funciones "gestionar entidades complementarias" y "gestionar usuarios". Caso: inserción de nuevo objeto.

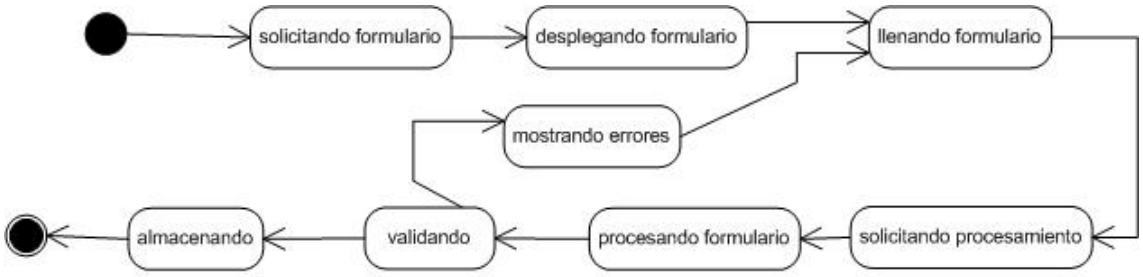


Figura D.19: Diagrama de estado para las funciones "gestionar entidades complementarias" y "gestionar usuarios". Caso: inserción de nuevo objeto.

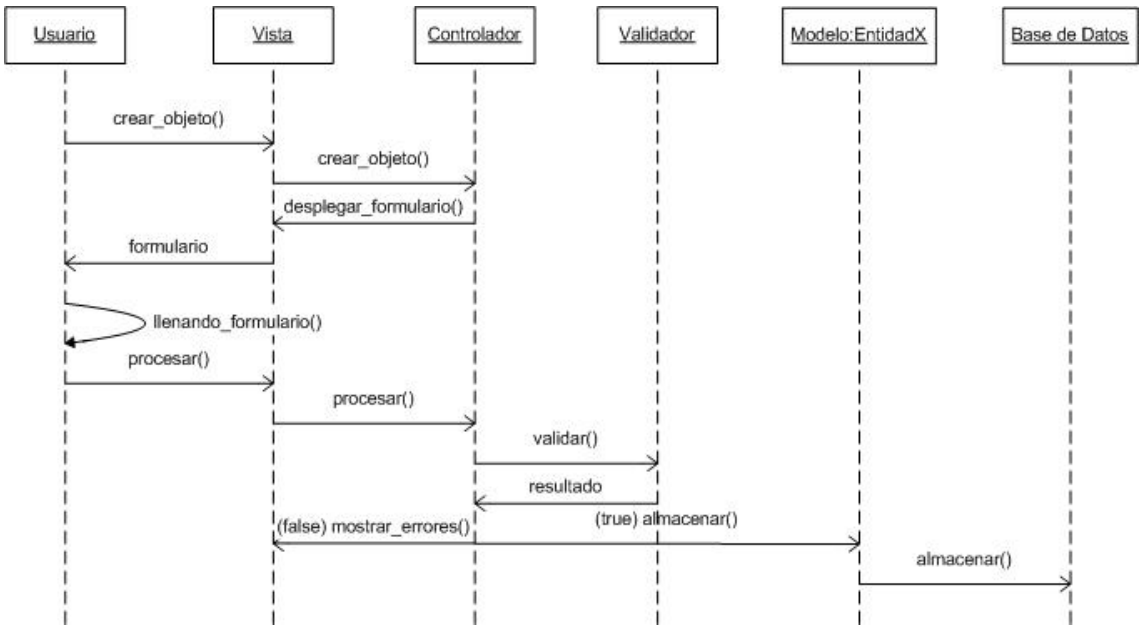


Figura D.20: Diagrama de secuencia para las funciones "gestionar entidades complementarias" y "Gestionar usuarios". Caso: inserción de nuevo objeto.

Función:	**Gestión de clientes**
Descripción:	Esta función hace referencia a las acciones de inserción, modificación y eliminación de clientes en el sistema.
Entrada:	Datos del cliente.
Fuente:	Usuario.
Salida:	Modificación del Cliente (inserción, edición ó eliminación) en la Base de Datos.
Destino:	Base de Datos, Ventana del usuario.
Precondición:	Que el usuario sea de tipo Administrador.

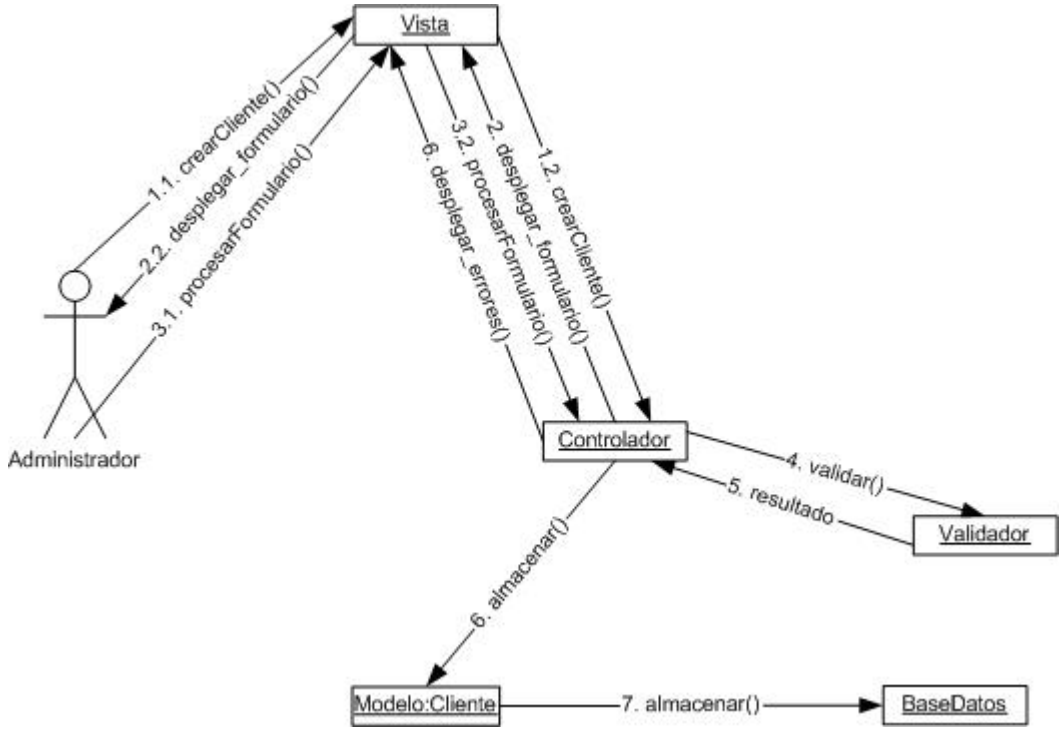


Figura D.21: Diagrama de colaboración para las funciones de gestión de clientes.

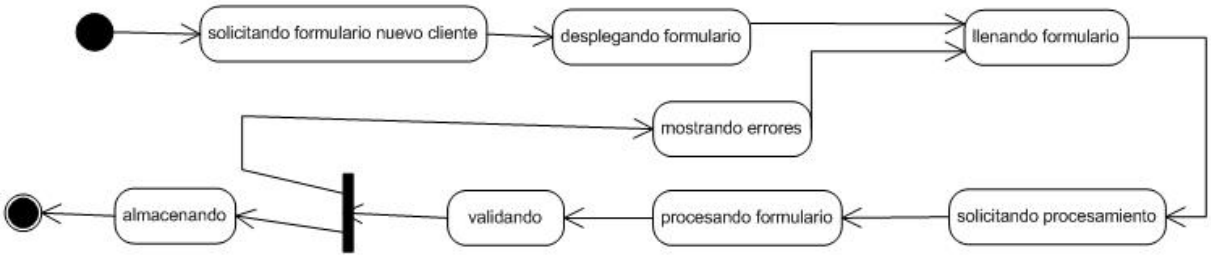


Figura D.22: Diagrama de estados para las funciones de gestión de clientes. Caso: crear cliente.

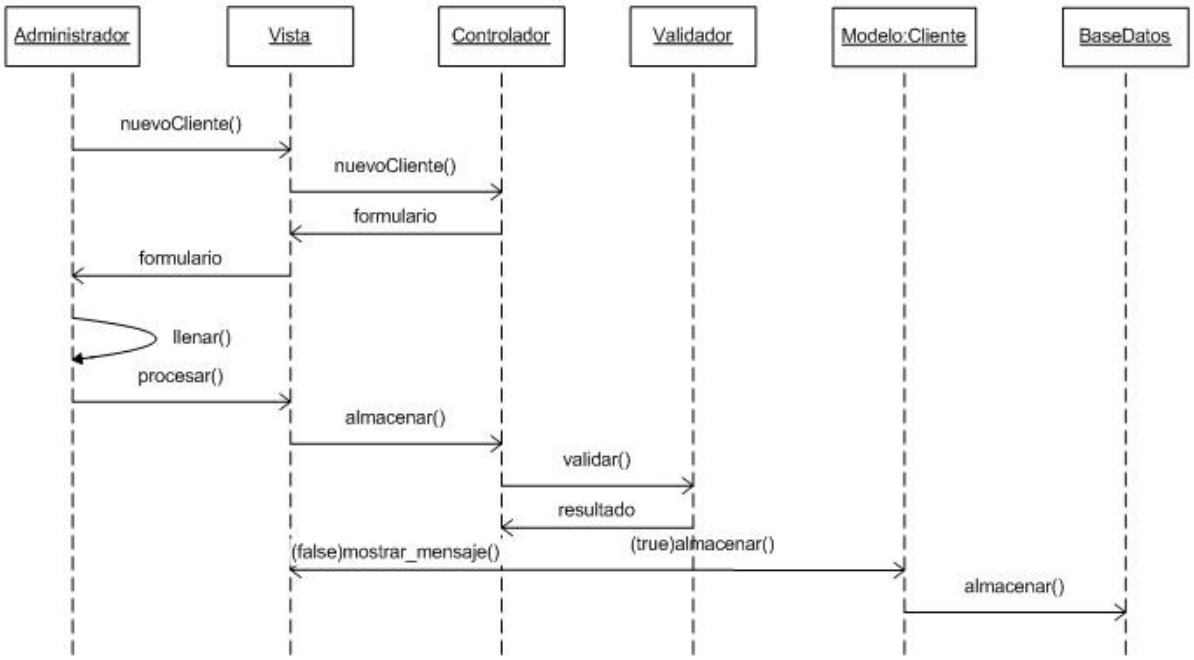


Figura D.23: Diagrama de secuencia para las funciones de gestión de clientes. Caso: crear cliente.

Función:	**Creación de proyectos**
Descripción:	Función que permite al usuario la creación de nuevos proyectos.
Entrada:	Datos del proyecto.
Fuente:	Usuario.
Salida:	Inserción en la Base de Datos del nuevo proyecto.
Destino:	Base de Datos, Ventana del usuario.
Precondición:	Que el usuario sea de tipo Administrador.

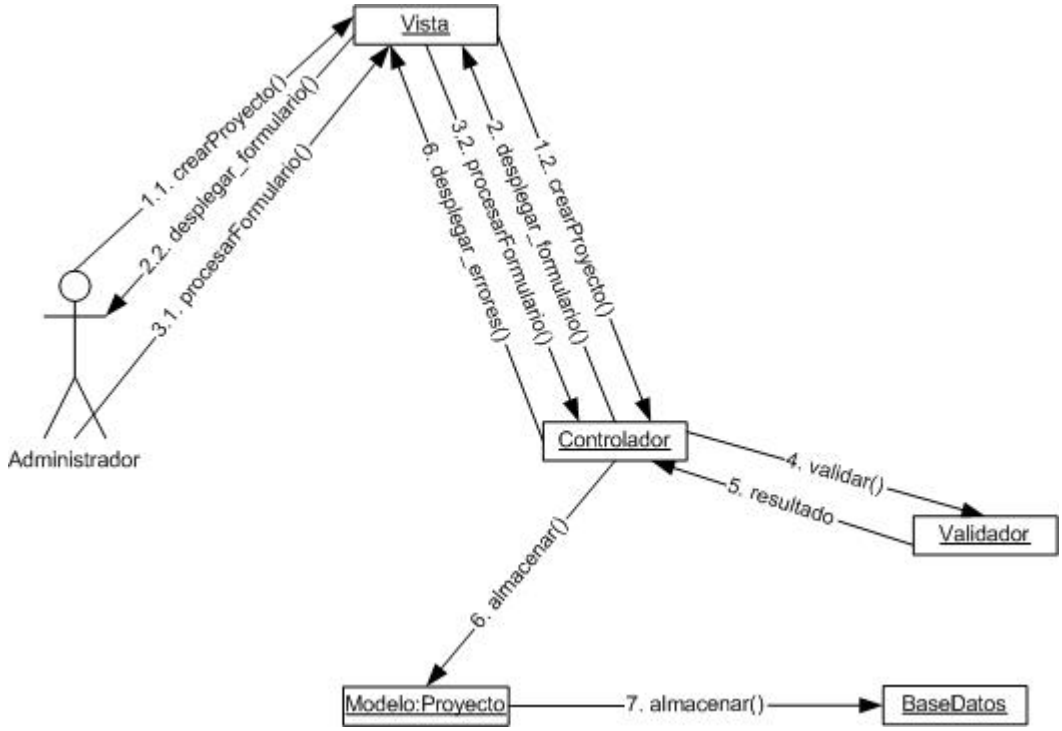


Figura D.24: Diagrama de colaboración para la función “crear proyecto”.

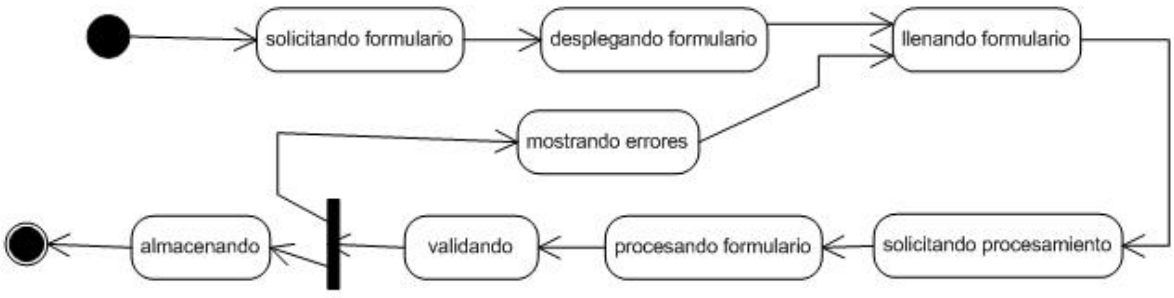


Figura D.25: Diagrama de estados para la función “crear proyecto”.

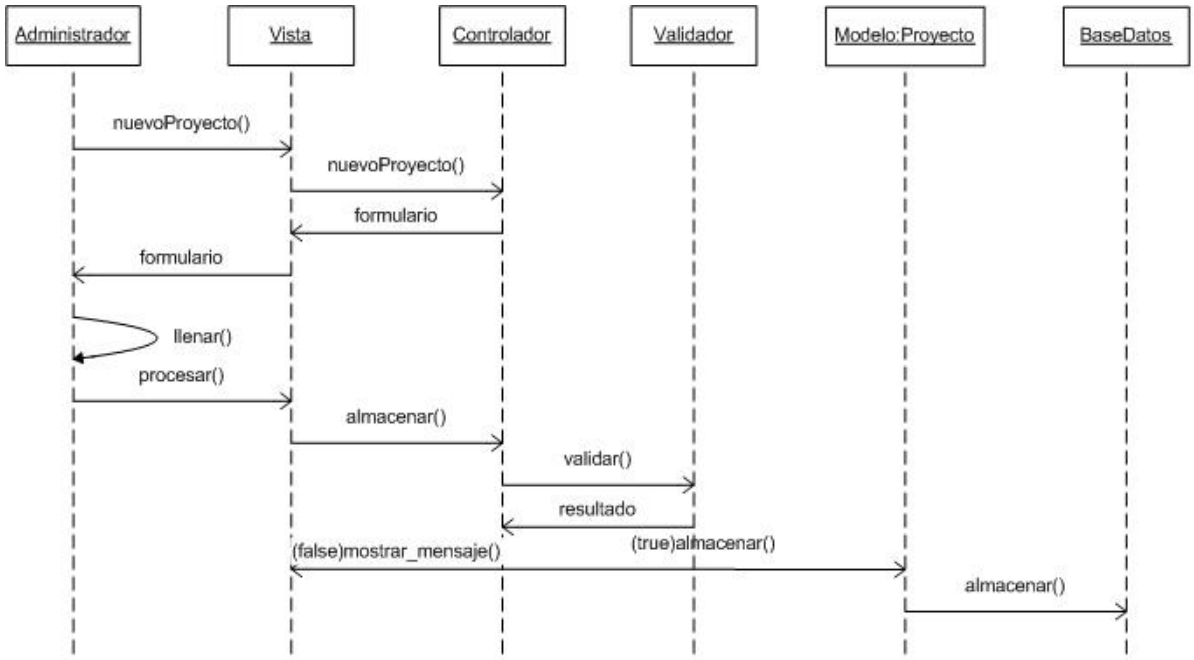


Figura D.26: Diagrama de secuencia para la función “crear proyecto”.

Función:	**Cambio de estado de un proyecto**
Descripción:	Esta función permite al usuario cambiar el estado de un proyecto en el sistema.
Entrada:	Nuevo estado del proyecto.
Fuente:	Usuario.
Salida:	Modificación del estado del proyecto en la Base de Datos.
Destino:	Base de Datos, Ventana del usuario.
Precondición:	Que el usuario sea de tipo Administrador.

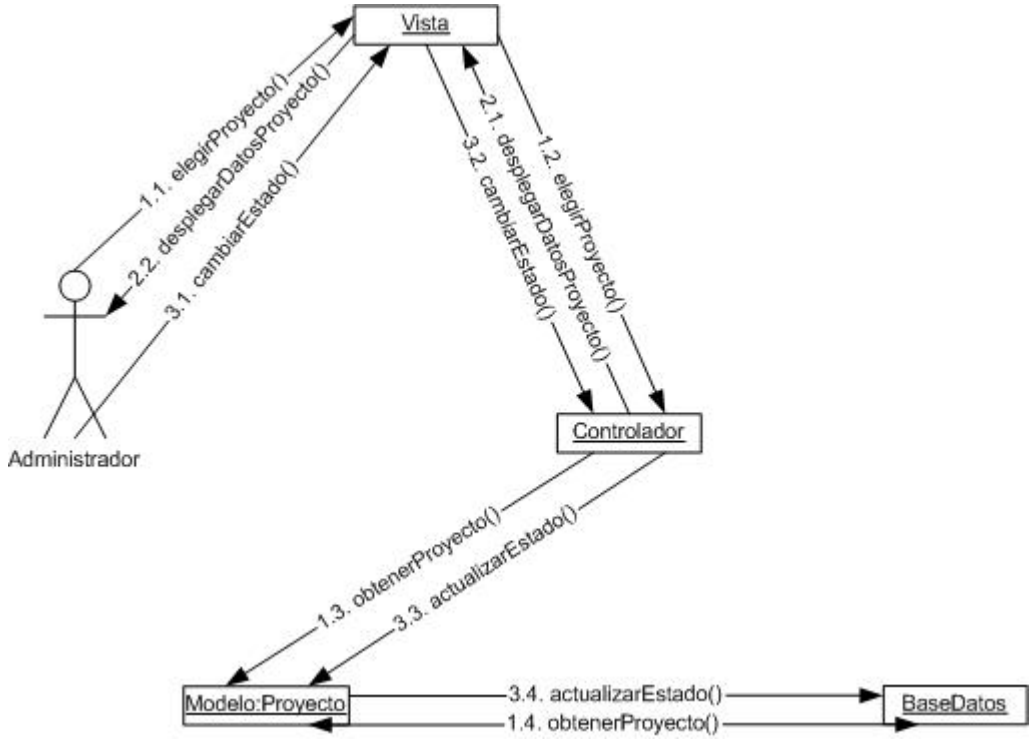


Figura D.27: Diagrama de colaboración para la función “crear proyecto”.

Figura D.28: Diagrama de estados para la función “crear proyecto”.

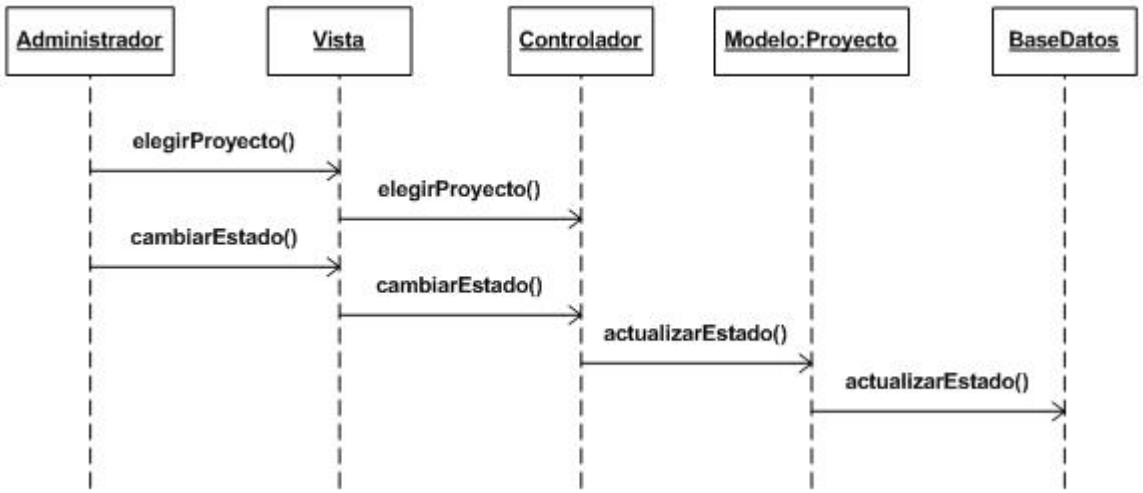


Figura D.29: Diagrama de secuencia para la función “crear proyecto”.

3.- Usuario Líder de Proyectos

3.1.- Diagrama de casos de uso

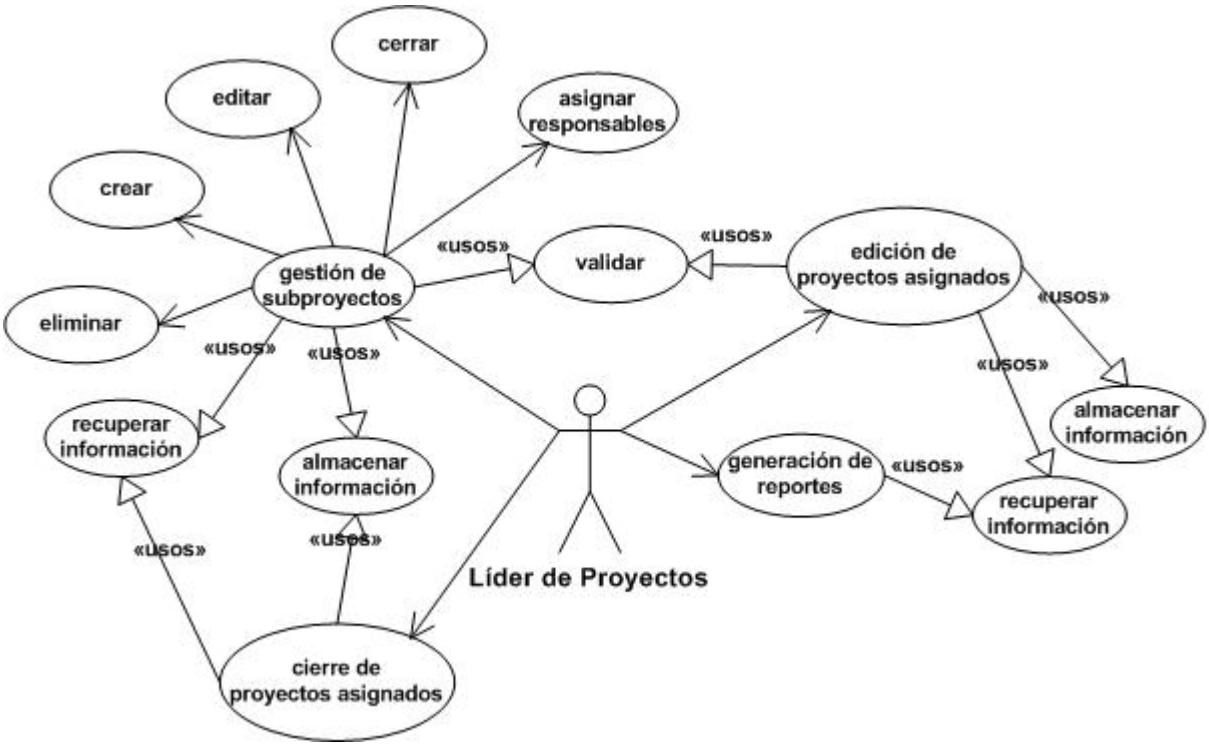


Figura D.30: Diagrama de casos de uso detallado para un usuario de tipo Líder de Proyectos.

3.2.- Funciones y diagramas de colaboración, estado y secuencia

Función:	**Gestión de proyectos asignados**
Descripción:	Función que permite al usuario elegir, modificar y eliminar proyectos. Durante su uso, el usuario puede hacer uso de cualquiera de las particularidades del proyecto.
Entrada:	Nombre del proyecto.
Fuente:	Usuario.
Salida:	Ventana de la aplicación donde se desplieguen todas las propiedades del proyecto.
Destino:	Usuario.
Precondición:	Que el usuario sea de tipo “Líder de Proyectos”.

Función:	**Gestión de subproyectos**
Descripción:	Función que permite al usuario elegir, crear, modificar y eliminar subproyectos.
Entrada:	Nombre del proyecto y datos del subproyecto.

Fuente:	Usuario.
Salida:	Ventana de la aplicación donde se desplieguen todas las propiedades del subproyecto.
Destino:	Usuario
Precondición:	Que el tipo del usuario sea: “Líder del proyecto”.

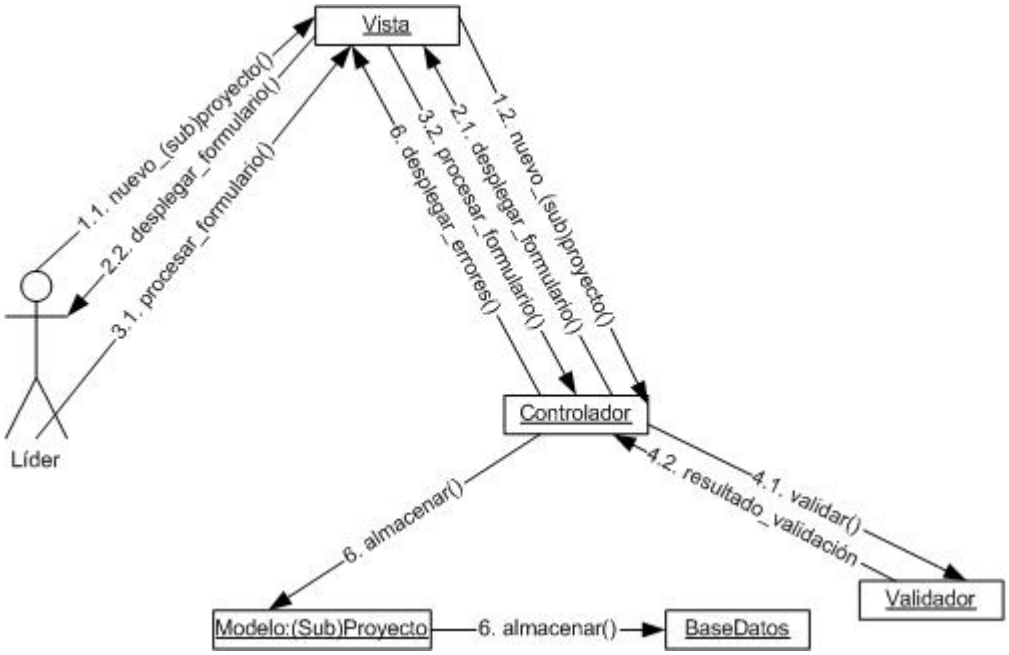


Figura D.31: Diagrama de colaboración para las funciones “gestionar proyectos” y “gestionar subproyectos”. Caso: Inserción de nuevo (sub)proyecto.

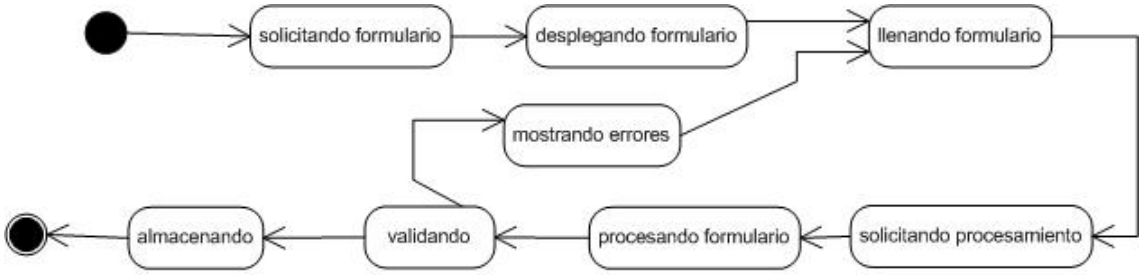


Figura D.32: Diagrama de estados para las funciones “gestionar proyectos” y “gestionar subproyectos”. Caso: Inserción de nuevo (sub)proyecto.

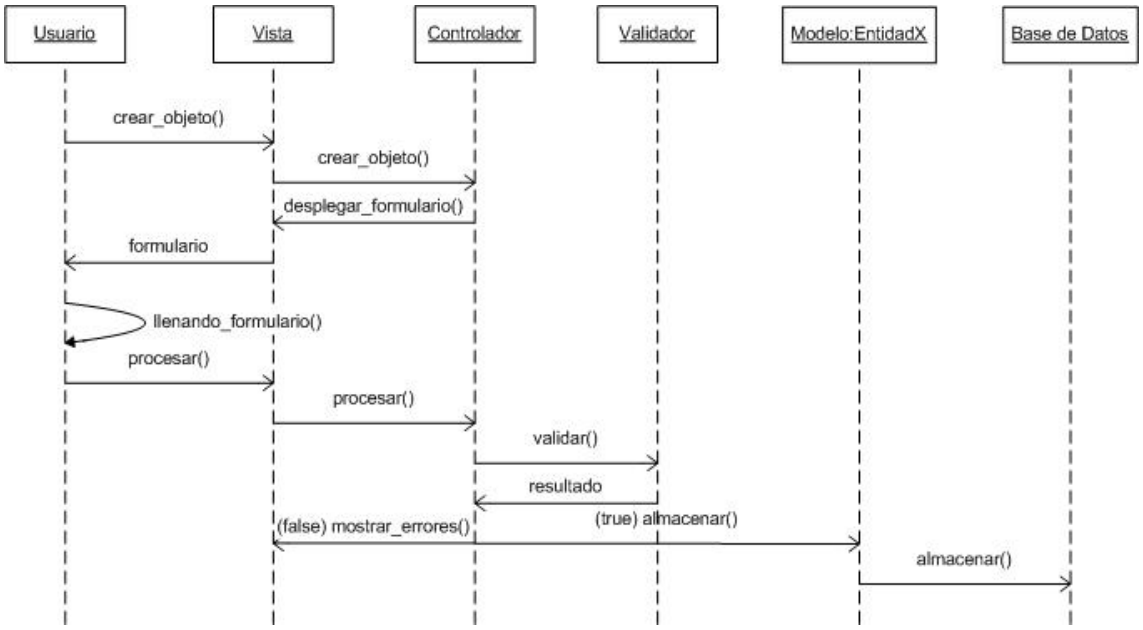


Figura D.33: Diagrama de secuencia para las funciones “gestionar proyectos” y “gestionar subproyectos”. Caso: Inserción de nuevo (sub)proyecto.

4.- **Usuario Tester**

4.1.- Diagrama de casos de uso

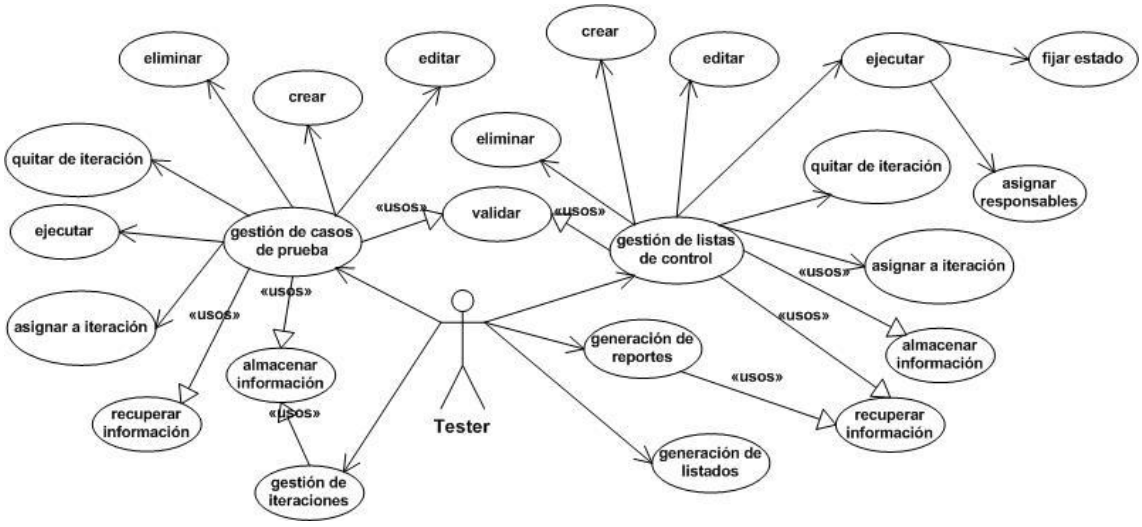


Figura D.34: Diagrama de casos de uso detallado para un usuario de tipo Tester.

4.2.- Funciones y diagramas de colaboración, estado y secuencia

Función:	**Gestión de Casos de Prueba**
Descripción:	Funciones para la administración de Casos de Prueba (creación, modificación, eliminación)).
Entrada:	Datos del Caso de Prueba. Para la creación de uno nuevo: nombre, objetivo, descripción, pasos de configuración y ejecución, resultado esperado y fases a las que aplica.
Fuente:	Tester.
Salida:	Actualizaciones en la Base de Datos.
Destino:	Base de Datos, Testers.
Precondición:	Ninguna.

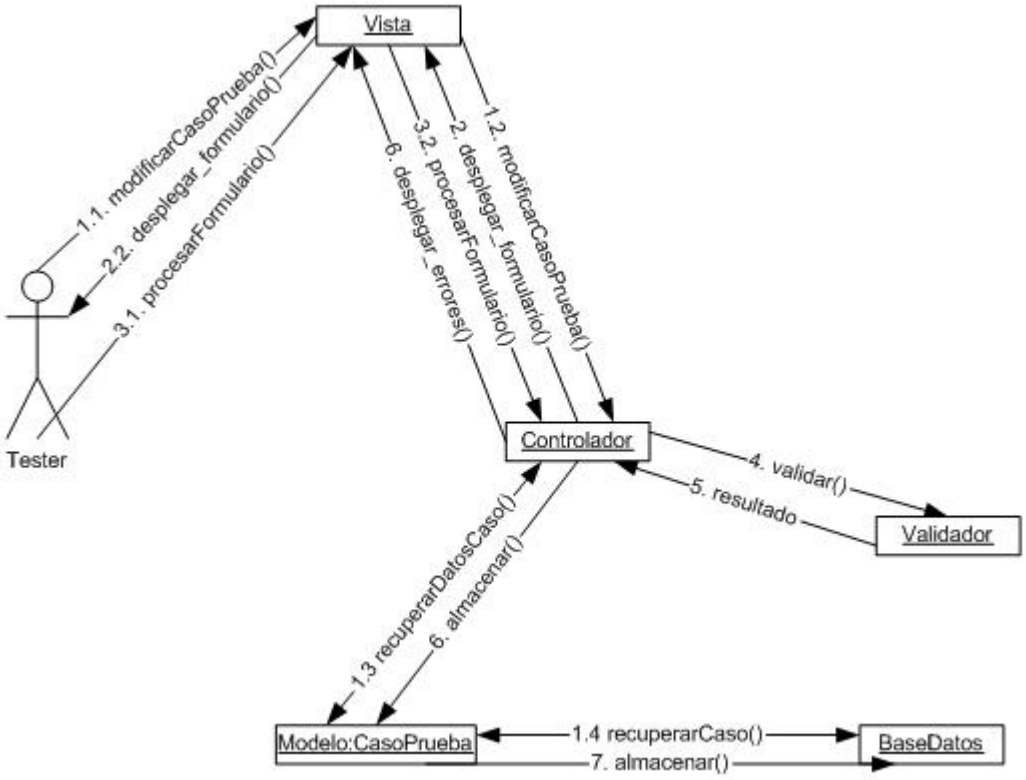


Figura D.35: Diagrama de colaboración para las funciones de gestión de casos de prueba. Caso: modificación de un testcase existente.

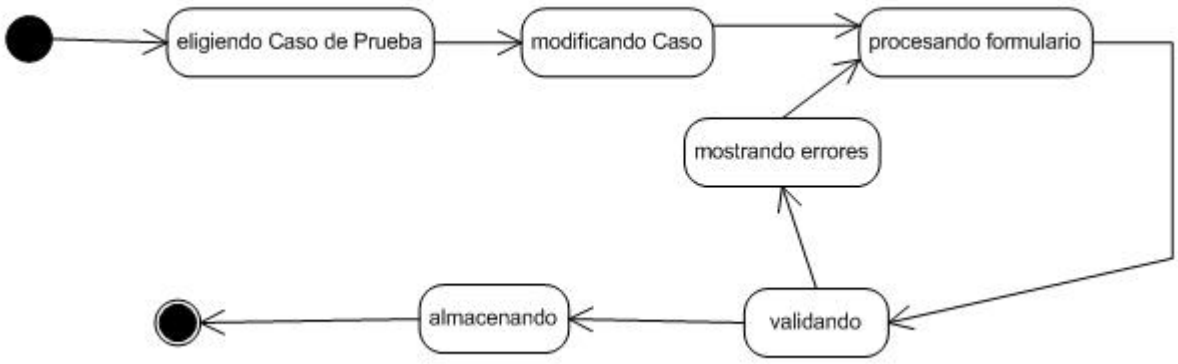


Figura D.36: Diagrama de Estados para las funciones de gestión de casos de prueba. Caso: modificación de un testcase existente.

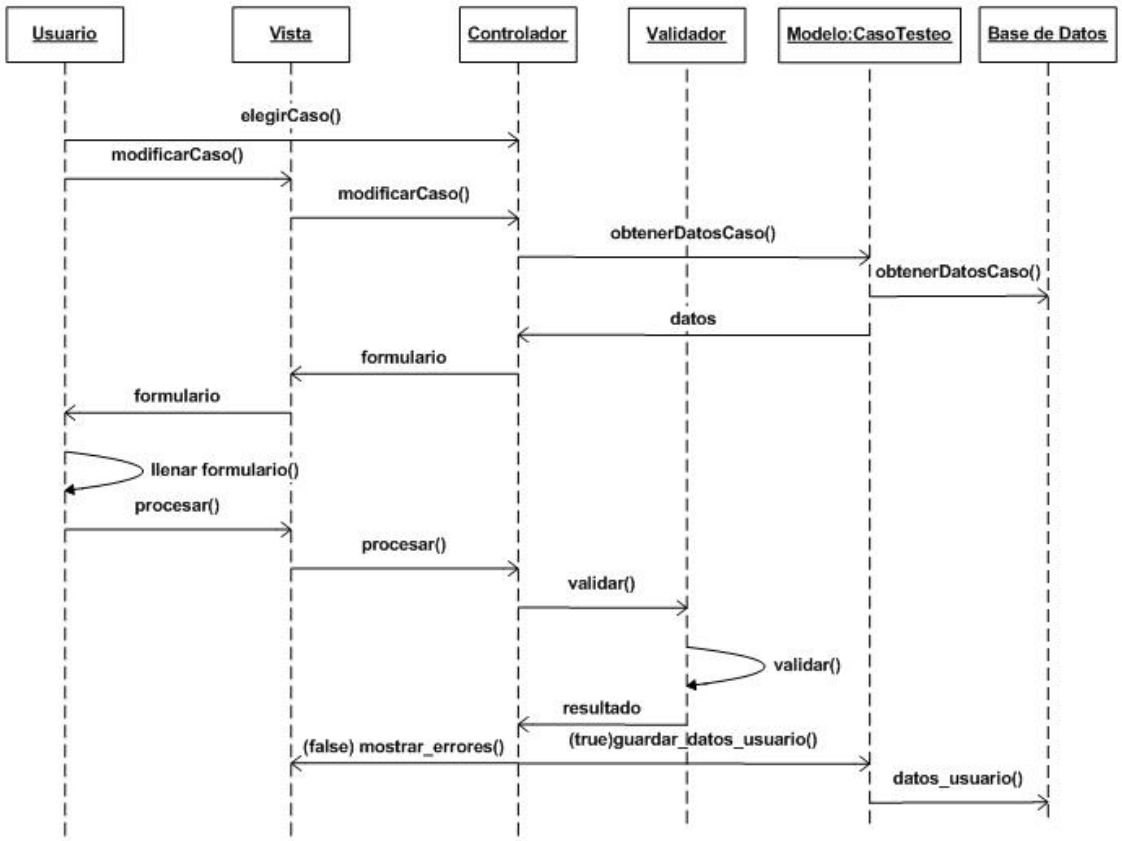


Figura D.37: Diagrama de secuencia para las funciones de gestión de casos de prueba. Caso: modificación de un testcase existente.

Función:	**Gestión de Listas de Control**
Descripción:	Funciones para la administración de Listas de Control (creación, modificación, eliminación)).
Entrada:	Datos de la lista. Para la creación de una nueva: nombre, objetivo, descripción, elementoso y fases a las que aplica.
Fuente:	Tester.
Salida:	Actualizaciones en la Base de Datos.
Destino:	Base de Datos, Testers.
Precondición:	Ninguna.

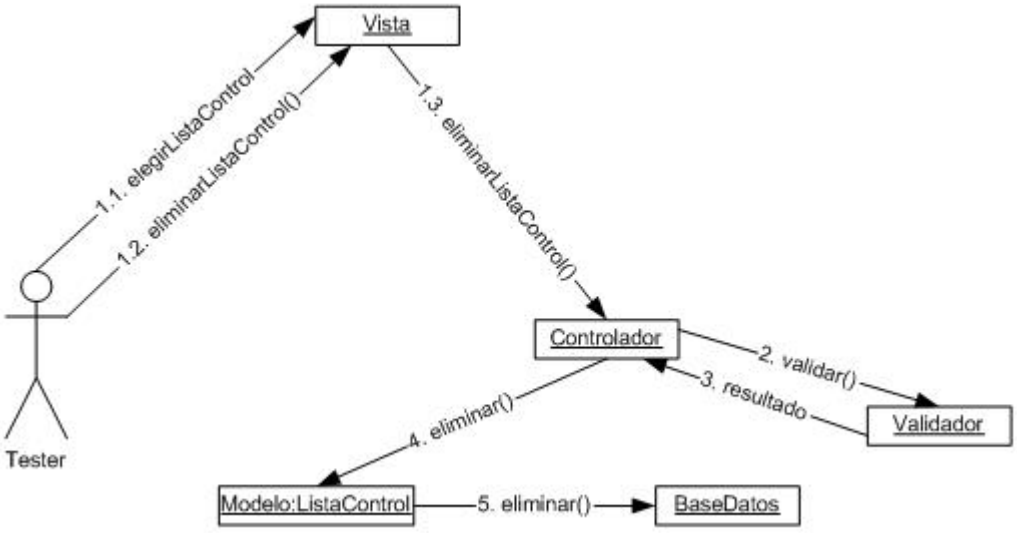


Figura D.38: Diagrama de colaboración para la función de gestión de iteraciones. Caso: eliminación de checklists.

Figura D.39: Diagrama de estados para la función de gestión de iteraciones. Caso: eliminación de checklists.

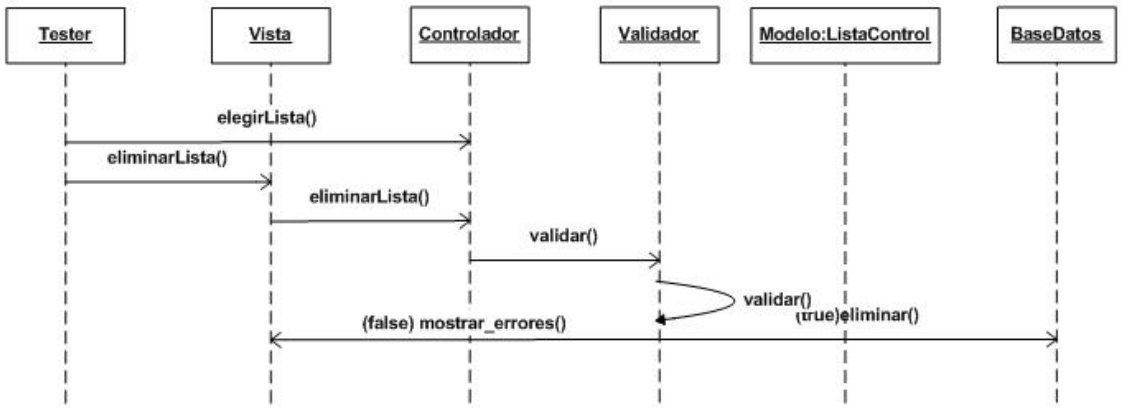


Figura D.40: Diagrama de secuencia para la función de gestión de iteraciones. Caso: eliminación de checklists.

Función:	**Gestión de iteraciones**
Descripción:	Funciones para la administración de iteraciones (creación, modificación, eliminación, cierre)).
Entrada:	Datos de la iteración. Para la creación de una nueva: nombre, fase, subproyecto, Casos de Prueba y Listas de Control que utilizará.
Fuente:	Tester.
Salida:	Actualizaciones en la Base de Datos.
Destino:	Base de Datos, Testers.
Precondición:	Que si la tupla (fase, subproyecto) dados tiene iteraciones previas, todas estén cerradas.

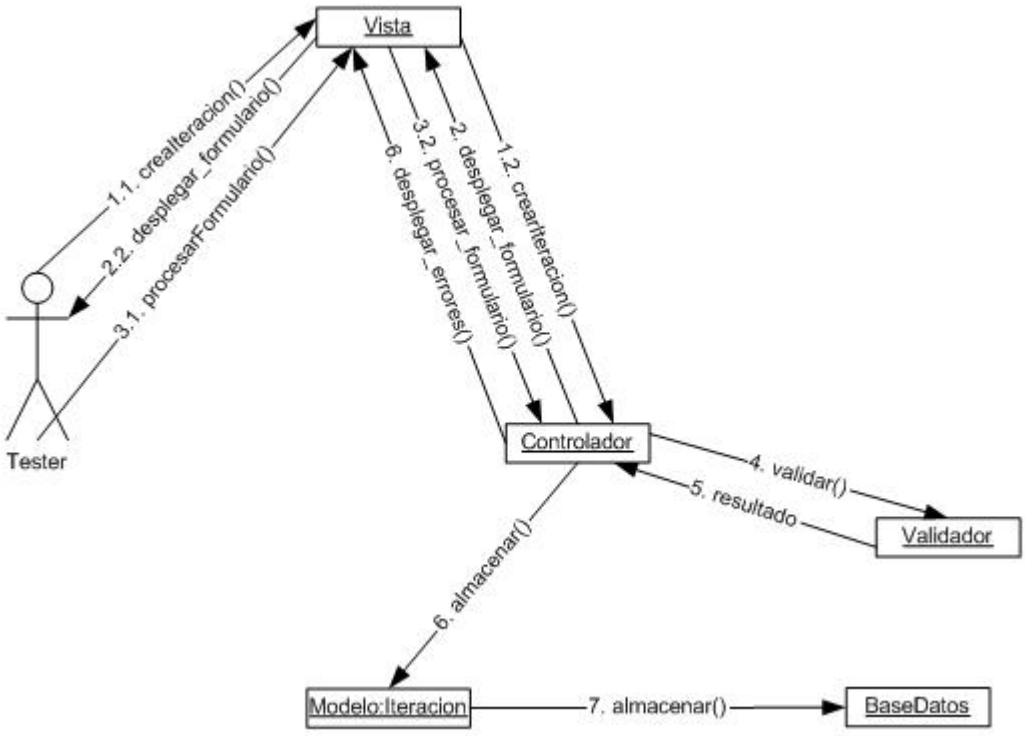


Figura D.41: Diagrama de colaboración para la función de gestión de iteraciones. Caso: creación de iteración.

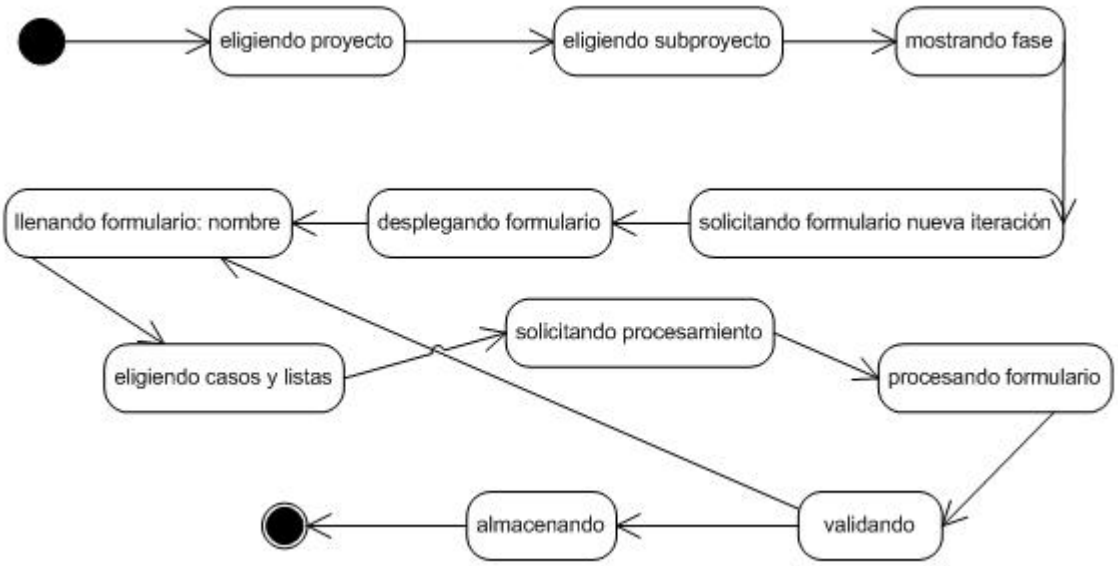


Figura D.42: Diagrama de estados para la función de gestión de iteraciones. Caso: creación de iteración.

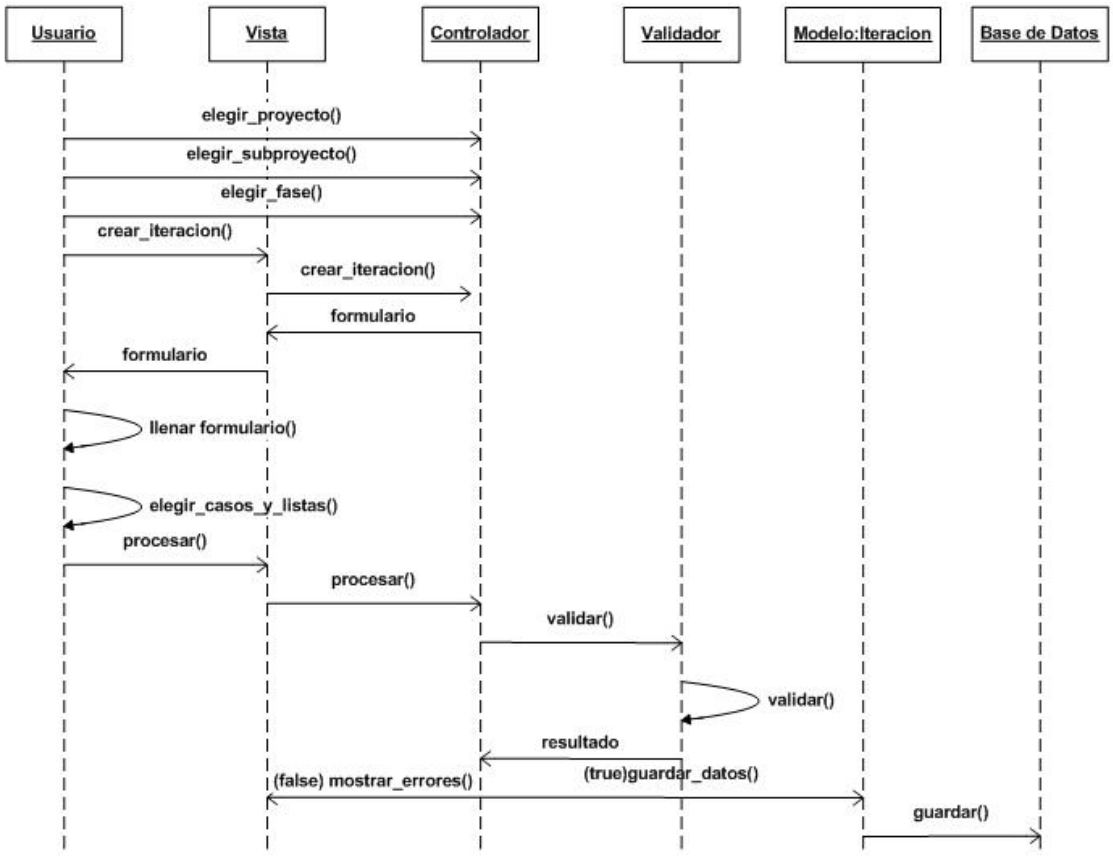


Figura D.43: Diagrama de secuencia para la función de gestión de iteraciones. Caso: creación de iteración.

Función:	**Ejecución del caso de testeo/listas de control**
Descripción:	Ejecución de un caso de testeo tantas veces como sea necesario (hasta que éste y todos los casos pertenecientes a su conjunto pasen).
Entrada:	Resultado de la ejecución (pasó/falló).
Fuente:	Tester.
Salida:	Almacenamiento en la Base de Datos.
Destino:	Equipo de Testers, Desarrolladores y Managers.
Precondición:	Que el caso haya sido elegido para el proyecto en curso.

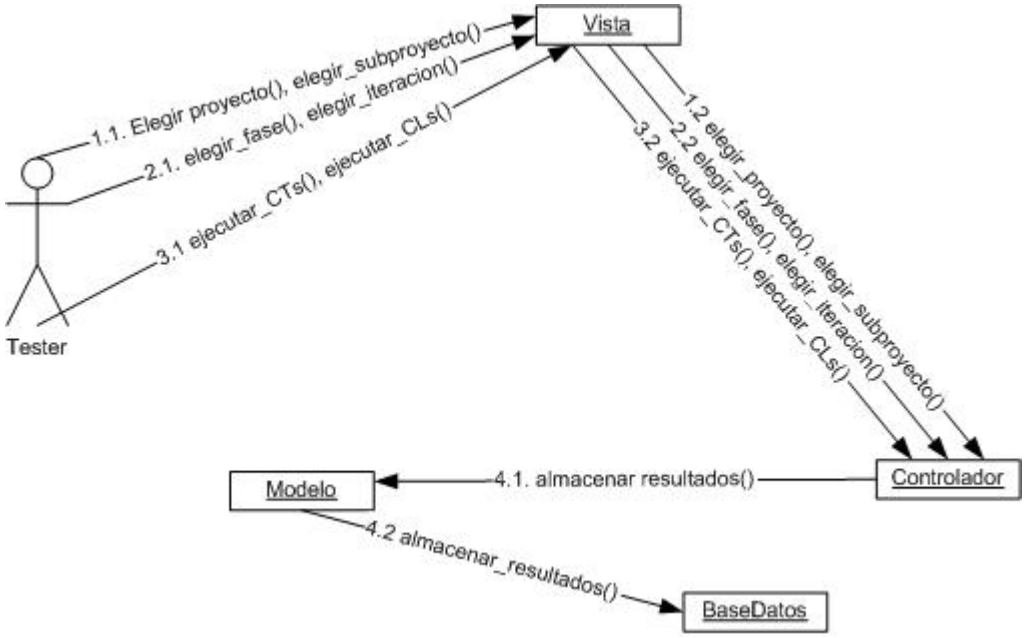


Figura D.44: Diagrama de colaboración para las funciones “ejecutar CT” y “ejecutar CL”.

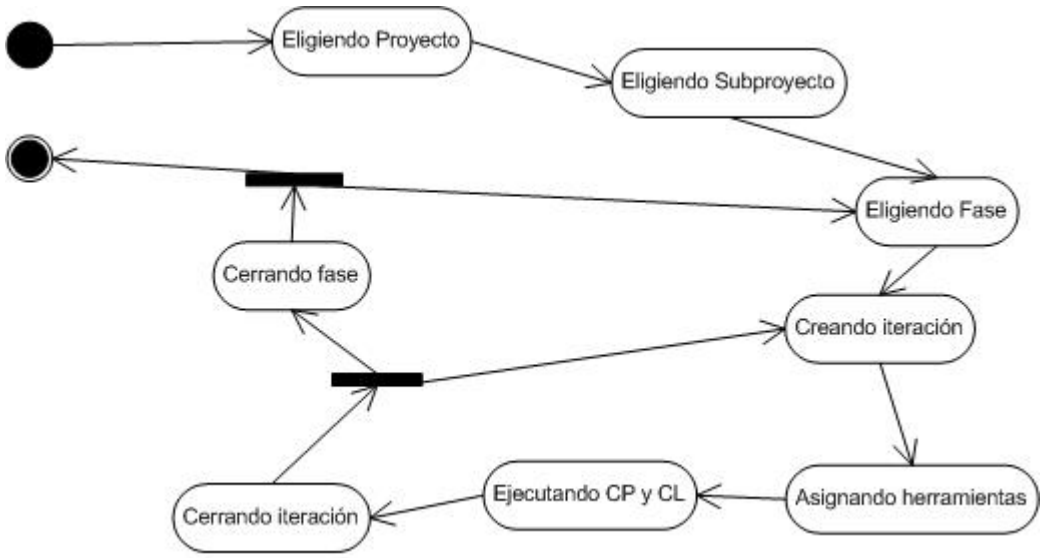


Figura D.45: Diagrama de Estados para las funciones “ejecutar CT” y “ejecutar CL”.

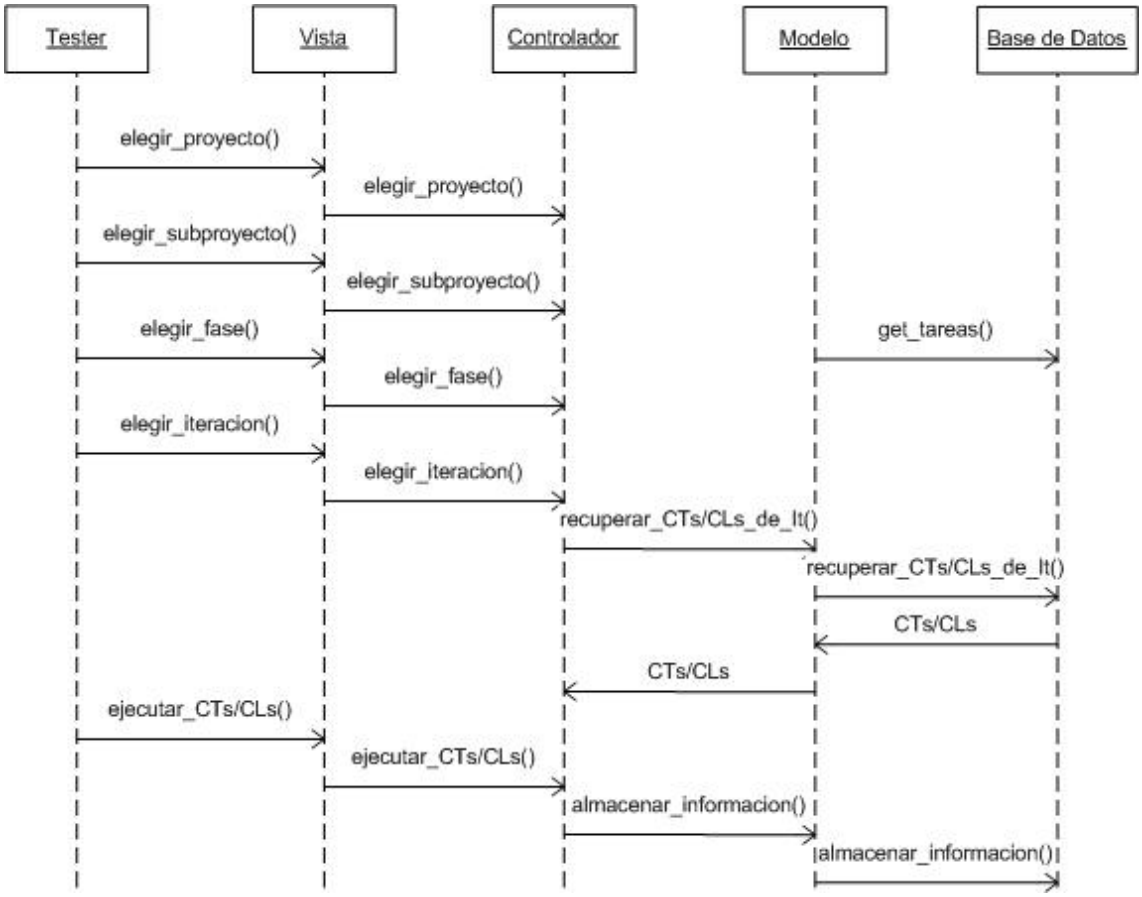


Figura D.46: Diagrama de secuencia para las funciones “ejecutar CT” y “ejecutar CL”.

Función:	**Asignación de tareas a desarrolladores**
Descripción:	Función para la asignación de tareas a los distintos desarrolladores cuando la ejecución de un Caso de Prueba o Lista de Control arroja resultados negativos.
Entrada:	nombre del desarrollador, descripción de la tarea, iteración a la que pertenece la tarea.
Fuente:	Tester.
Salida:	Actualizaciones en la Base de Datos (creación de una nueva tarea).
Destino:	Base de Datos, Desarrolladores.
Precondición:	Que la iteración a la que pertenece la tarea no esté cerrada y que el Caso de Prueba o Lista de Control haya arrojado resultados negativos.

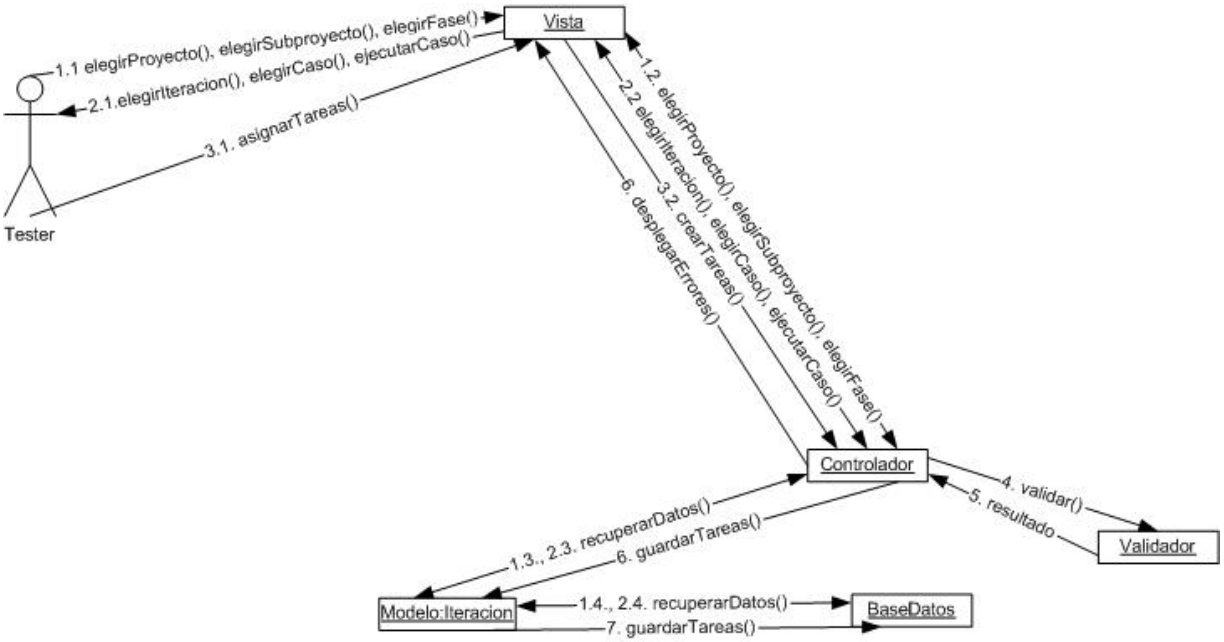


Figura D.47: Diagrama de colaboración para la función de asignación de tareas a desarrolladores.

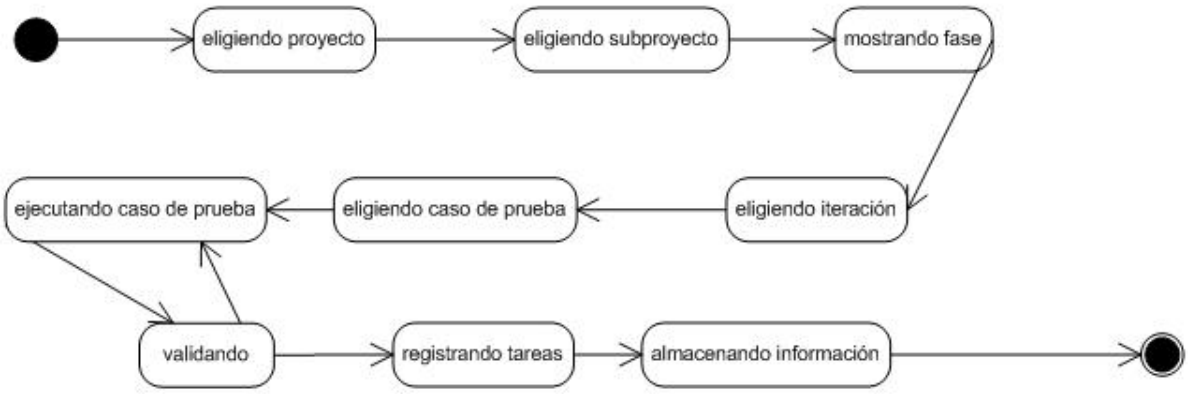


Figura D.48: Diagrama de estados para la función de asignación de tareas a desarrolladores.

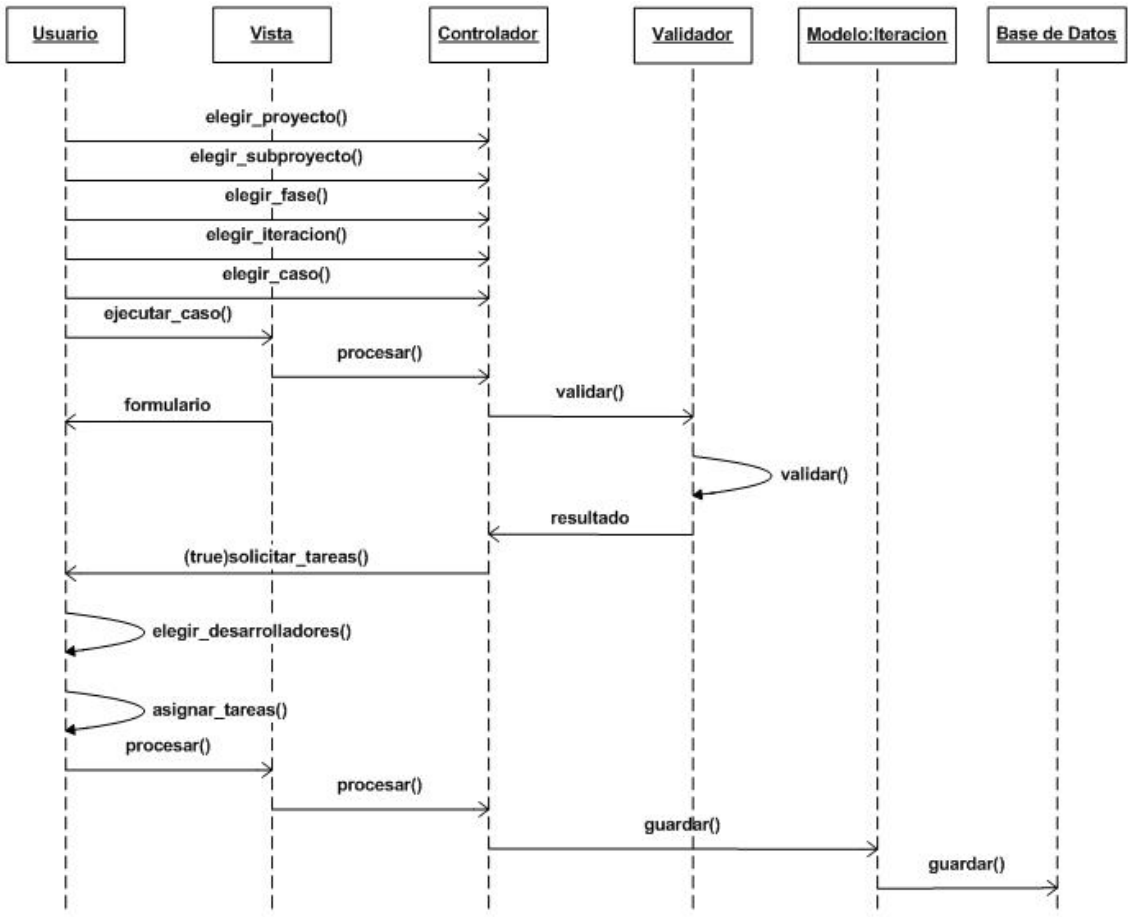


Figura D.49: Diagrama de secuencia para la función de asignación de tareas a desarrolladores.

5.- Usuario Desarrollador

5.1.- Diagrama de casos de uso

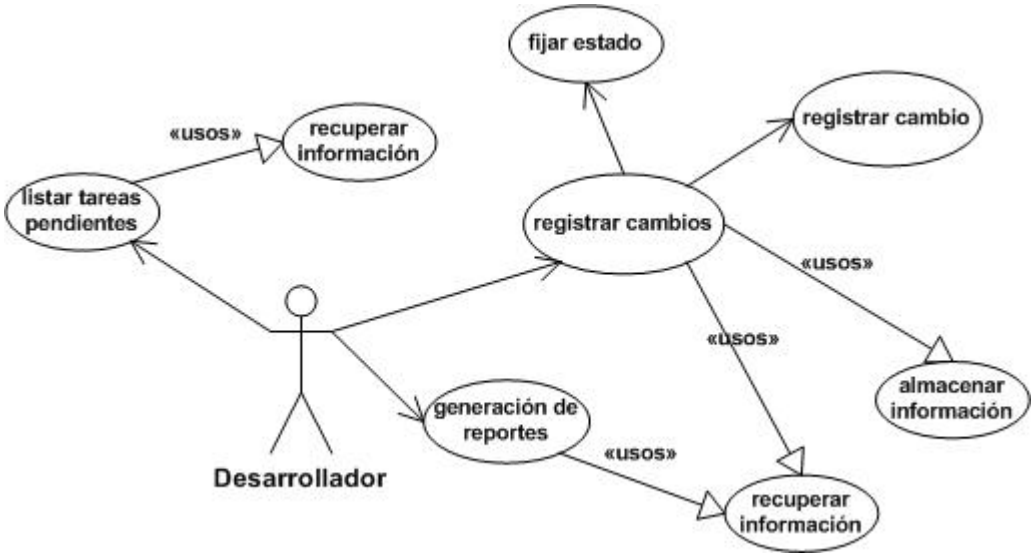


Figura D.50: Diagrama de casos de uso detallado para un usuario de tipo Desarrollador.

5.2.- Funciones y diagramas de colaboración, estado y secuencia

Función:	**Listado de tareas**
Descripción:	Permite al usuario listar todas sus tareas pendientes.
Entrada:	Nombre del usuario.
Fuente:	Usuario.
Salida:	Lista de tareas.
Destino:	Usuario.
Precondición:	Ninguna.

Función:	**Actualización de estados de las tareas**
Descripción:	Esta acción permitirá al usuario cambiar el estado de sus tareas pendientes y añadir comentarios.
Entrada:	Nombre del usuario, tarea elegida.
Fuente:	Usuario.
Salida:	Tarea con el estado cambiado.
Destino:	Base de Datos.
Precondición:	Que la tarea pertenezca a una iteración que tiene el estado “abierto”.

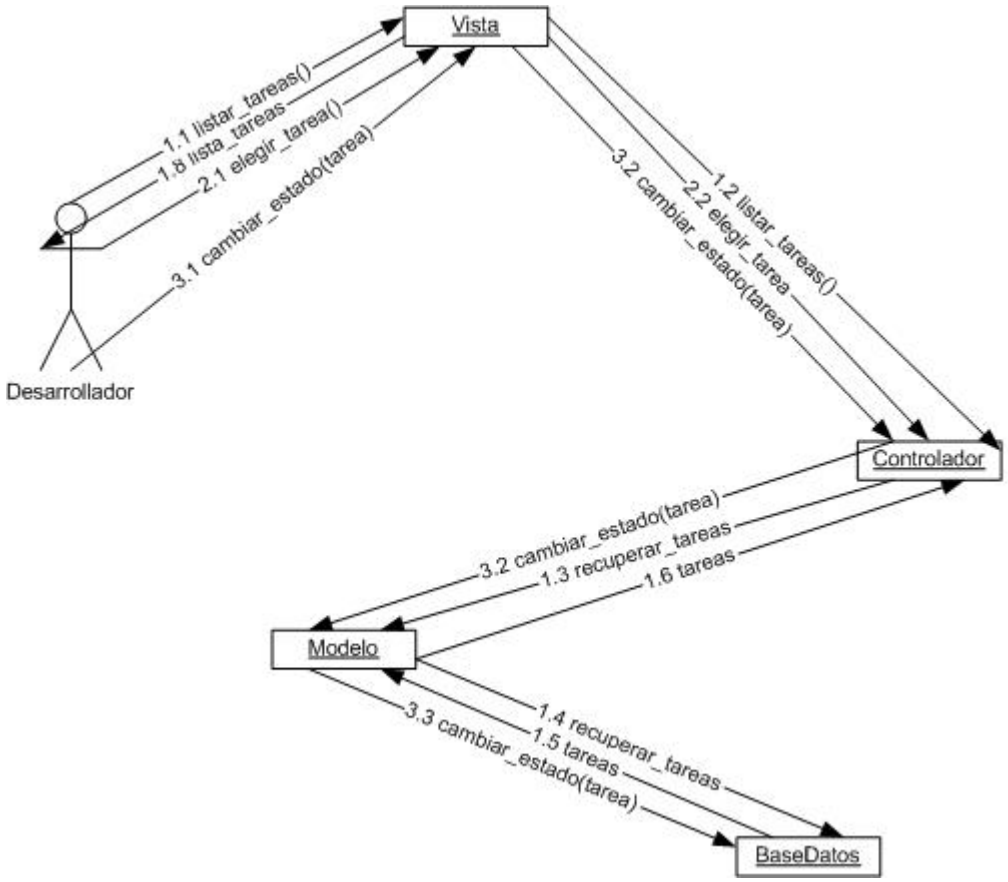


Figura D.51: Diagrama de colaboración para las funciones “listar tareas” y “cambiar estado tarea”.

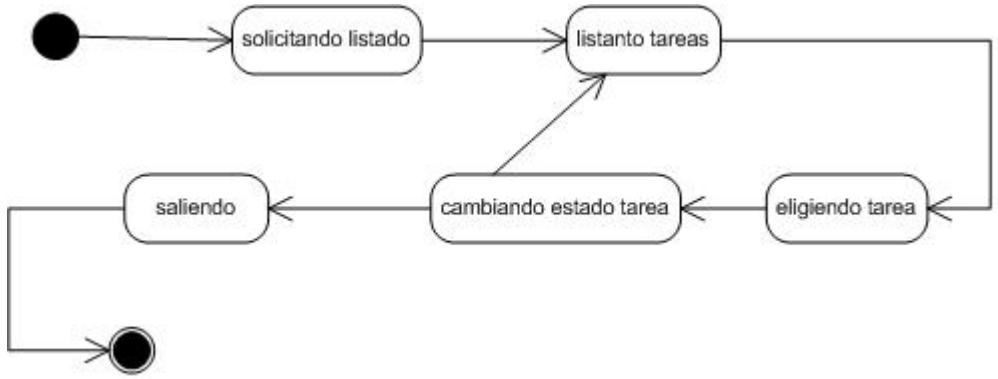


Figura D.52: Diagrama de estados para las funciones “listar tareas” y “cambiar estado tarea”.

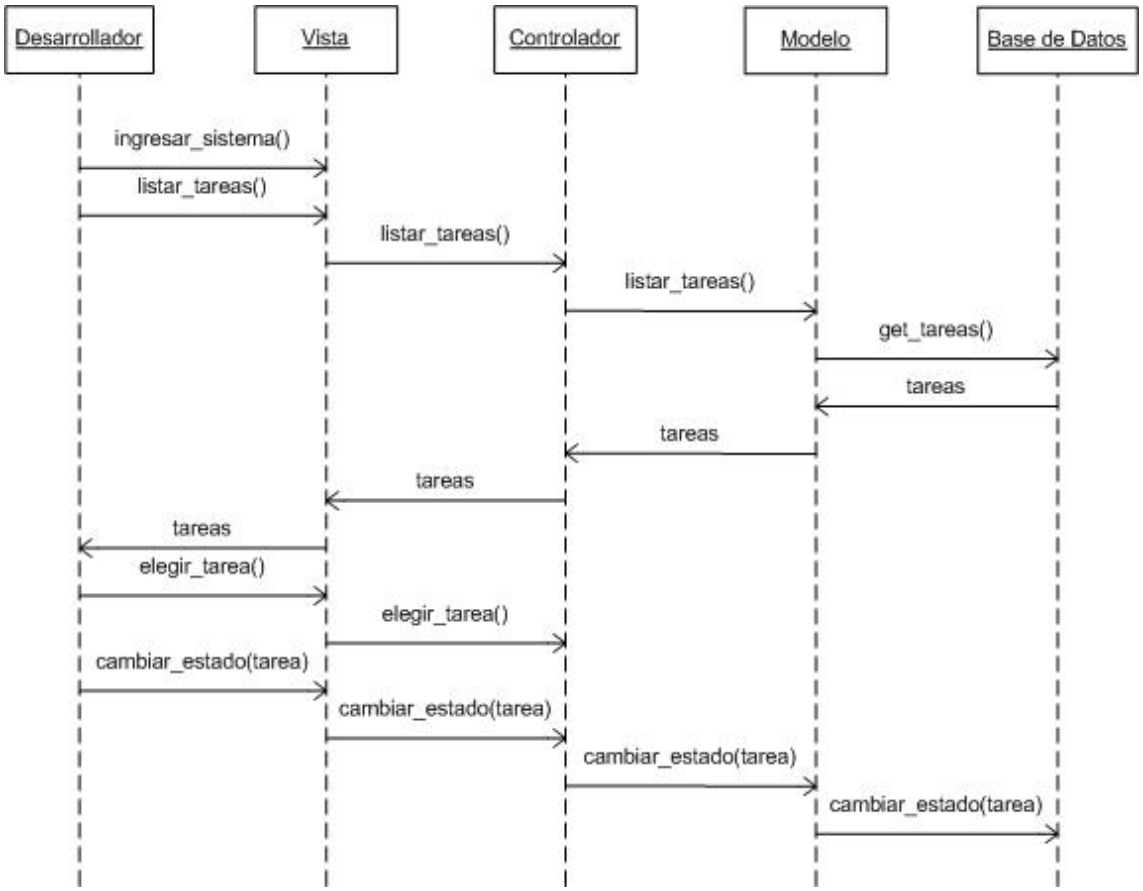


Figura D.53: Diagrama de secuencia para las funciones “listar tareas” y “cambiar estado tarea”.

6.- **Usuario Cliente**

6.1.- Diagrama de casos de uso

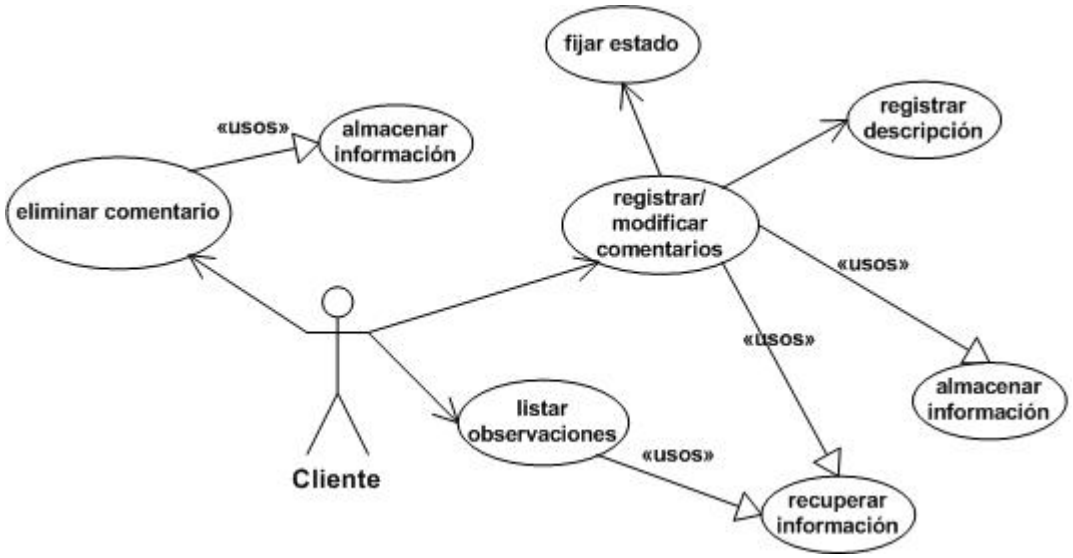


Figura D.54: Diagrama de casos de uso detallado para un usuario de tipo Cliente.

6.2.- Funciones y diagramas de colaboración, estado y secuencia

Función:	**Registro de comentarios**
Descripción:	Permite al cliente (dueño de uno o más proyectos) registrar sus observaciones.
Entrada:	Comentario.
Fuente:	Cliente.
Salida:	Registro en la Base de Datos.
Destino:	Tester responsable del control de calidad del subproyecto en el que se registró el comentario.
Precondición:	Elegir el proyecto, subproyecto y fase.

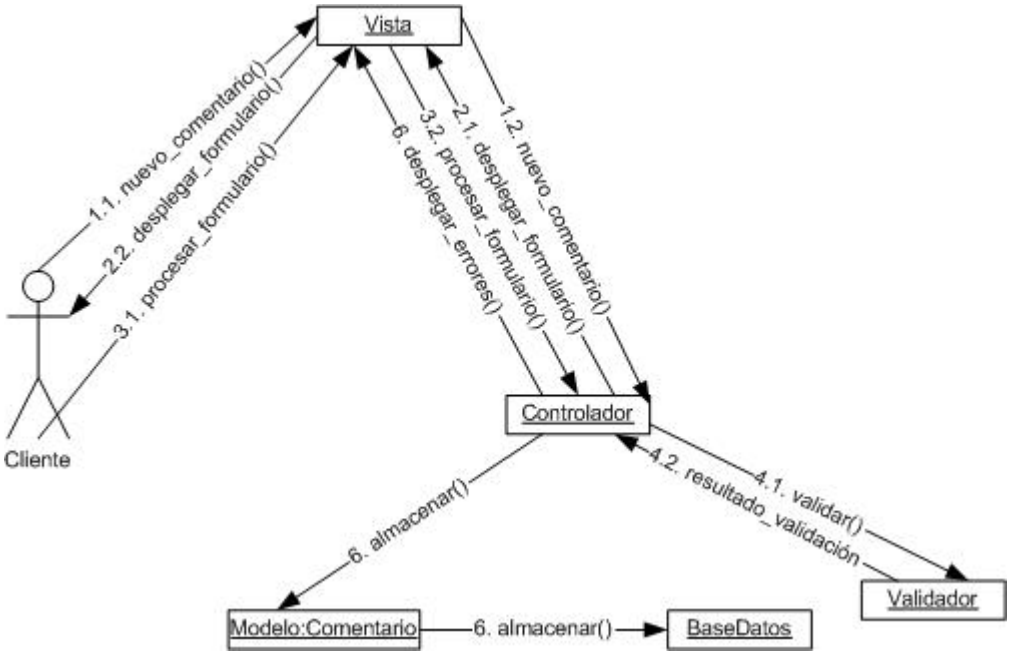


Figura D.55: Diagrama de colaboración para las funciones "listar comentarios" y "cambiar estado comentario".

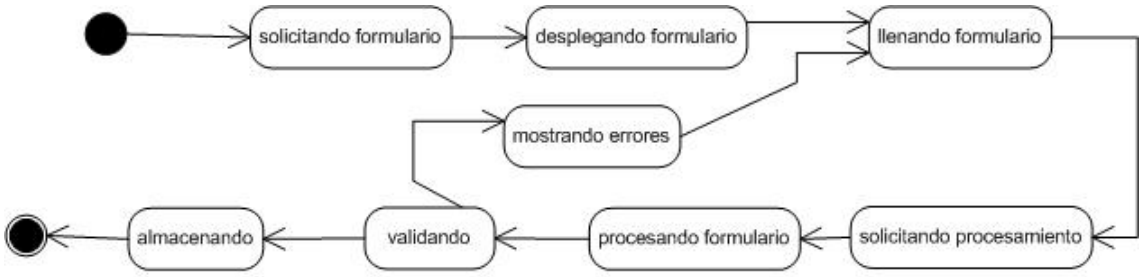


Figura D.56: Diagrama de estados para las funciones "listar comentarios" y "cambiar estado comentario".

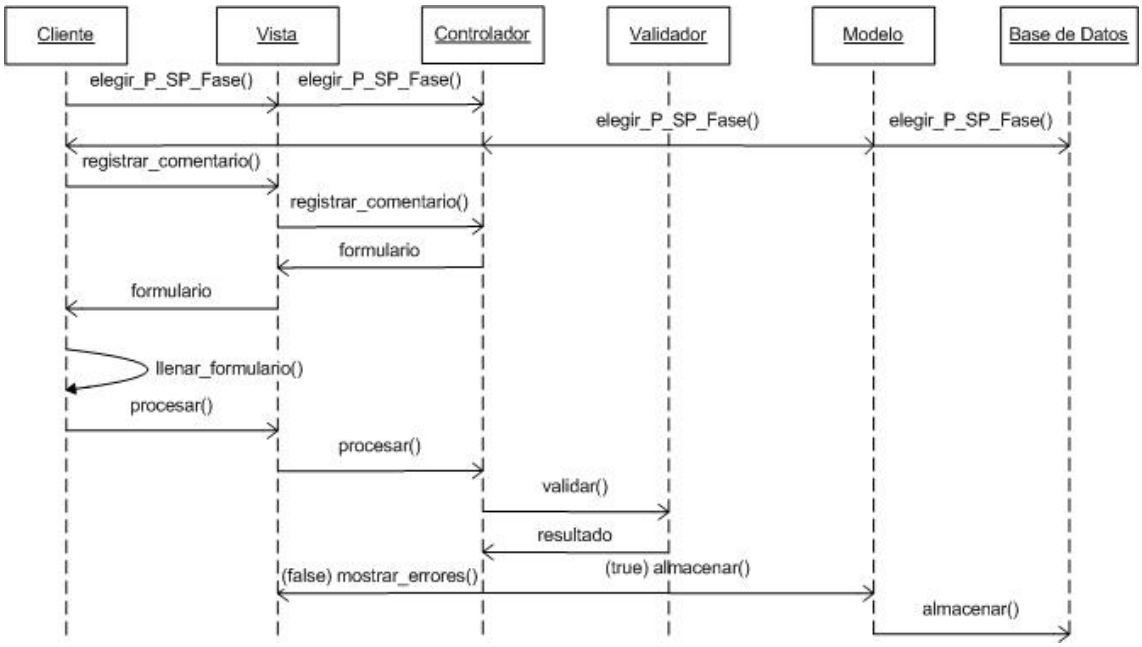


Figura D.57: Diagrama de secuencia para las funciones “listar comentarios” y “cambiar estado comentario”.

Función:	**Eliminación de comentarios**
Descripción:	Permite al cliente eliminar una observación que haya sido previamente puesta por él.
Entrada:	Código del comentario a eliminar.
Fuente:	Cliente.
Salida:	Eliminación del registro en la Base de Datos.
Destino:	Cliente.
Precondición:	Que la fase a la que pertenece la observación no esté cerrada.

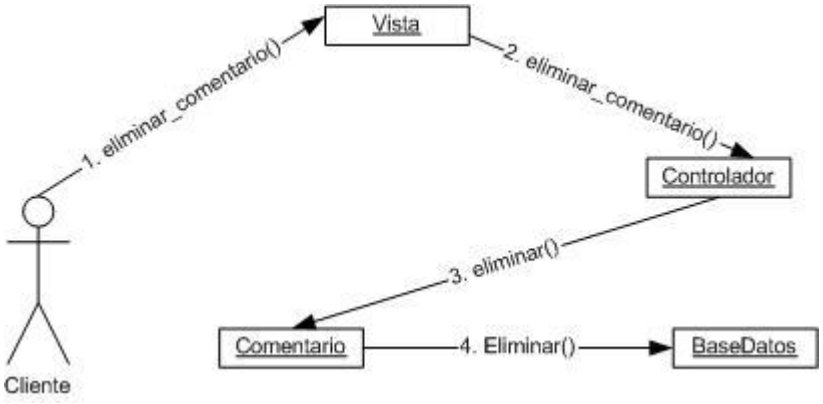


Figura D.58: Diagrama de colaboración para la función de eliminación de comentarios.

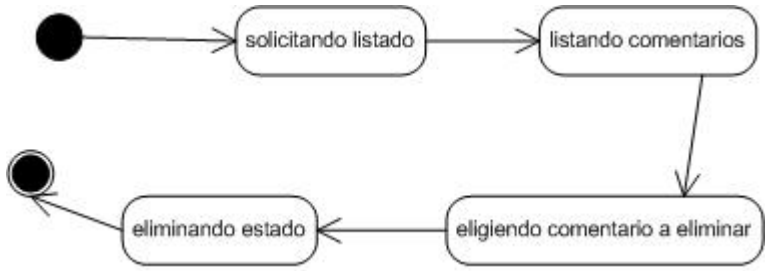


Figura D.59: Diagrama de estados para la función de eliminación de comentarios.

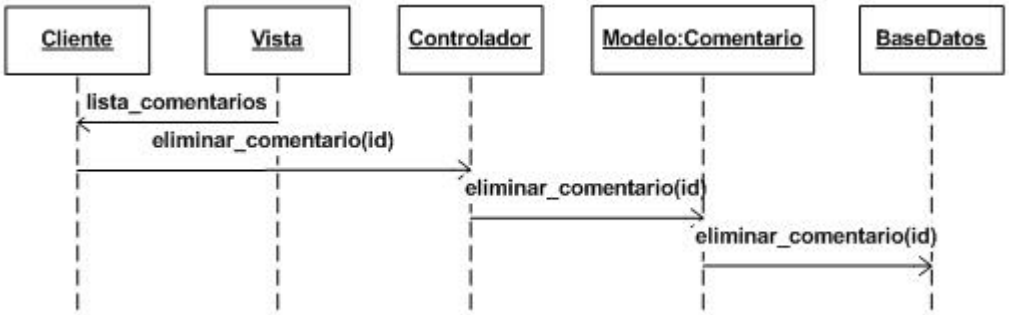


Figura D.60: Diagrama de secuencia para la función de eliminación de comentarios.

7.- **Sistema**

7.1.- Funciones

Función:	**Almacenar información**
Descripción:	Esta función almacena en la Base de Datos la información que reciba del usuario.
Entrada:	Datos a ser almacenados.
Fuente:	Tester, Manager.
Salida:	Nuevos registros en la Base de Datos.
Destino:	Base de Datos.
Precondición:	Que la información a almacenar haya sido validada previamente por el sistema.

Función:	**Recuperar información**
Descripción:	Esta función permite recuperar información de la Base de Datos de acuerdo a las solicitudes hechas por el usuario al sistema.
Entrada:	a) Solicitud del usuario o de otra función del sistema. b) Condiciones para el filtrado de la información a recuperar.
Fuente:	Tester, Sistema.
Salida:	Información de la Base de Datos.
Destino:	Tester, Manager, Gerencia, Sistema.
Precondición:	Ninguna.

Función:	**Validar**
Descripción:	Función que permite la validación de los datos entrantes al sistema para su posterior procesamiento y/o almacenamiento en la Base de Datos.
Entrada:	Datos a validar.
Fuente:	Tester, Manager.
Salida:	Resultado de la validación mediante un mensaje.
Destino:	Tester, Manager, Gerencia, Base de Datos.
Precondición:	Ninguna.

Función:	**Generación y envío de alarmas**
Descripción:	Función que permite al sistema crear y enviar las alarmas a los usuarios.
Entrada:	Señal del sistema.
Fuente:	Sistema.

Salida:	Mensajes de alerta.
Destino:	Todos los usuarios.
Precondición:	Ninguna.

Función:	**Generación de mensajes**
Descripción:	Función que permite la emisión de mensajes del sistema a los usuarios cuando sea necesario (por ejemplo cuando ocurra un error).
Entrada:	Acción del usuario.
Fuente:	Cualquier usuario.
Salida:	Mensaje de información (ejemplo: cuando el usuario modifica su contraseña), advertencia (cuando se desea eliminar un registro) o error (cuando el usuario comete algún error).
Destino:	Usuario que originó la acción.
Precondición:	Ninguna.

ANEXO E

DICCIONARIO DE CLASES

En este Diccionario se detallan los atributos y métodos de las principales clases del sistema (mostrados en orden alfabético).

Clases

1.- CasoTesteo

1.1.- Atributos

Nombre	Tipo	Descripción
nombre	String(100)	Nombre del Caso de Testeo
propósito	String(250)	Objetivo del caso
descripción	String(250)	Breve descripción
pasos_configuracion	String(250)	Pasos previos a la ejecución del caso
pasos_ejecucion	String(250)	Pasos para la ejecución del caso de testeo
resultado_esperado	String(250)	Resultado que se espera obtener

1.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearCaso	nombre, propósito, descripción, pasos_config, pasos_ejec, result_esperado		Crea un nuevo caso de Prueba (constructor de la clase)
obtenerCaso		Atributos del Caso de testeo	Obtiene los datos de un caso en particular.
modificarCcaso	nuevo_nombre, nuevo_propósito, nueva_descripción, nuevos_pasos_config, nuevos_pasos_ejec, nuevo_result_esperado		Modifica los datos del caso de testeo

eliminarCaso			Elimina el caso de testeo (destructor de la clase)

2.- **Checklist**

2.1.- Atributos

Nombre	Tipo	Descripción
nombre	String(100)	Nombre de la lista
propósito	String(250)	Objetivo de la lista
descripción	String(250)	Breve descripción

2.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearChecklist	nombre, propósito, descripción		Crea una nueva lista (constructor de la clase)
obtenerLista		Atributos del Checklist	Obtiene los atributos de una lista en particular.
modificarChecklist	nuevo_nombre, nuevo_propósito, nueva_descripción		Modifica los datos del checklist
eliminarCaso			Elimina el checklist (destructor de la clase)

3.- **Ciclo**

3.1.- Atributos

Nombre	Tipo	Descripción
nombre	String(50)	Nombre del ciclo
descripción	String(250)	Breve descripción

3.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearCiclo	nombre, descripción		Crea un nuevo ciclo (constructor de la clase)
obtenerCiclo		Nombre, descripción	Obtiene los atributos del ciclo
modificarCiclo	nuevo_nombre, nueva_descripción		Modifica los datos del ciclo
eliminarCiclo			Elimina el ciclo (destructor de la clase)

4.- **Cliente**

4.1.- Atributos

Nombre	Tipo	Descripción
nombre	String(100)	Nombre del cliente
descripción	String(250)	Breve descripción
empresa	String(100)	Empresa a la que pertenece el cliente
ciudad	String(50)	Ciudad donde reside el cliente
país	String(50)	País donde reside el cliente
teléfono	String(100)	Teléfono(s) del cliente
dirección	String(250)	Dirección física del cliente
email	String(50)	Correo electrónico
WWW	String(250)	Página Web del cliente
sector	String(50)	Sector de trabajo (Rubro)

4.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearCliente	nombre, descripcion, empresa, ciudad, pais, telefono, direccion, email, www, sector		Crea un nuevo cliente (constructor de la clase)

obtenerCliente		Nombre, descripción, empresa, ciudad, país, teléfono, dirección, email, www, sector	Obtiene los atributos del cliente
modificarCliente	nuevo_nombre, nueva_descripción, nueva_empresa, nueva_ciudad, nuevo_país, nuevo_teléfono, nueva_dirección, nuevo_email, nuevo_www, nuevo_sector		Modifica los datos del cliente
eliminarCliente			Elimina el cliente (destructor de la clase)

5.- **ElementoCL**

5.1.- Atributos

Nombre	Tipo	Descripción
descripción	String(250)	Sentencia

5.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearElemento	descripción		Crea un nuevo elemento (constructor de la clase)
obtenerElemento		Descripción	Obtiene los atributos del elemento
modificarCiclo	nueva_descripción		Modifica el elemento
eliminarElemento			Elimina el elemento (destructor de la clase)

6.- **Entorno**

6.1.- Atributos

Nombre	Tipo	Descripción
nombre	String(50)	Nombre del entorno
descripción_hardware	String(250)	Descripción del entorno de hardware
descripción_software	String(250)	Descripción del entorno de software
otros	String(250)	Descripciones adicionales

6.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearEntorno	nombre, descr_hw, descr_sw, otros		Crea un nuevo entorno (constructor de la clase)
obtenerEntorno		nombre, descr_hw, descr_sw, otros	Obtiene los atriburos del entorno
modificarEntorno	nuevo_nombre, nueva_descripción_hw, nueva_descripción_sw, nuevo_otros		Modifica los datos del entorno
eliminarEntorno			Elimina el entorno (destructor de la clase)

7.- **Fase**

7.1.- Atributos

Nombre	Tipo	Descripción
nombre	String(50)	Nombre de la fase
descripcion	String(250)	Breve descripción

7.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearFase	nombre, descripcion		Crea una nueva fase (constructor de la clase)
obtenerFase		Nombre, descripcion	Obtiene los atributos de la fase
modificarFase	nuevo_nombre, nueva_descripción		Modifica los datos de la fase
eliminarFase			Elimina la fase (destructor de la clase)

8.- **Iteración**

8.1.- Atributos

Nombre	Tipo	Descripción
nombre	String(50)	Nombre de la iteración

8.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearIteración	nombre		Crea una nueva iteración (constructor de la clase)
obtenerIteración		Nombre	Obtiene los atributos de la iteración
modificarIteración	nuevo_nombre		Modifica los datos de la iteración
eliminarIteración			Elimina la iteración (destructor de la clase)

9.- **Proyecto**

9.1.- Atributos

Nombre	Tipo	Descripción
nombre	String(50)	Nombre de la fase

objetivo	String(250)	Objetivo del proyecto
descripción	String(250)	Breve descripción
fecha_inicial	Date	Fecha de apertura del proyecto
fecha_final	Date	Fecha de cierre del proyecto
estado	String(15)	Estado en el que se encuentra del proyecto
avance	Int(3)	Porcentaje de avance del proyecto

9.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearProyecto	nombre, objetivo, descripcion, fecha_i, fecha_f, estado, avance		Crea un nuevo proyecto (constructor de la clase)
obtenerProyecto		Nombre, objetivo, descripcion, fecha_i, fecha_f, estado, avance	Obtiene los atributos del proyecto
modificarProyecto	nuevo_nombre, nueva_descripción		Modifica los datos del proyecto
cerrarProyecto			Cierra el proyecto (cambio de estado)
eliminarProyecto			Elimina el proyecto (destructor de la clase)

10.- **ResponsableCL**

10.1.- Atributos

Nombre	Tipo	Descripción
descripcion_tarea	String(50)	Descripción de la tarea (de un checklist)
fecha_registro	Date	Fecha de registro de la tarea
estado	String(250)	Estado de la tarea
comentarios	String(250)	Comentarios del responsable

10.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearTarea	descripcion_tarea, fecha_registro, estado, comentarios		Crea una nueva tarea (constructor de la clase)
obtenerTarea		descripción_tarea, fecha_registro, estado, comentarios	Obtiene los atributos de la tarea
modificarTarea	nueva_descripción_tarea, nueva_fecha_registro, nuevo_estado, nuevos_comentarios		Modifica los datos de la tarea
eliminarTarea			Elimina la tarea (destructor de la clase)

11.- **ResponsableCT**

11.1.- Atributos

Nombre	Tipo	Descripción
descripcion_tarea	String(50)	Descripción de la tarea (de un caso de testeo)
fecha_registro	Date	Fecha de registro de la tarea
estado	String(250)	Estado de la tarea
comentarios	String(250)	Comentarios del responsable

11.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearTarea	descripcion_tarea, fecha_registro, estado, comentarios		Crea una nueva tarea (constructor de la clase)
obtenerTarea		descripcion_tarea, fecha_registro, estado, comentarios	Obtiene los atributos de la tarea
modificarTarea	nueva_descripcion_tarea, nueva_fecha_registro, nuevo_estado, nuevos_comentarios		Modifica los datos de la tarea

eliminarTarea			Elimina la tarea (destructor de la clase)

12.- **ResultadoCL**

12.1.- Atributos

Nombre	Tipo	Descripción
resultado	Int(1)	Resultado de la ejecución del checklist
comentario	String	Comentario del ejecutor
fecha_ejecucion	Date	Fecha de registro del resultado

12.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearResultado	resultado, fecha_ejecucion		Crea un nuevo resultado (constructor de la clase)
obtenerResultado		resultado, fecha_ejecucion	Obtiene los atributos del resultado
modificarResultado	nuevo_resultado, nueva_fecha_ejecucion		Modifica el resultado
eliminarResultado			Elimina el resultado

13.- **ResultadoCT**

13.1.- Atributos

Nombre	Tipo	Descripción
resultado	Int(1)	Resultado de la ejecución del caso de testeo
resultado_obtenido	String(250)	Descripción del resultado obtenido
fecha_ejecucion	Date	Fecha de registro del resultado

13.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearResultado	resultado, resultado_obtenido, fecha_ejecucion		Crea un nuevo resultado (constructor de la clase)
obtenerResultado		resultado, resultado_obtenido, fecha_ejecucion	Obtiene los atributos del resultado
modificarResultado	nuevo_resultado, nuevo_resultado_obtenido, nueva_fecha_ejecución		Modifica el resultado
eliminarResultado			Elimina el resultado (destructor de la clase)

14.- **Subproyecto**

14.1.- Atributos

Nombre	Tipo	Descripción
nombre	String(100)	Nombre del subproyecto
descripcion	String(250)	Descripción del subproyecto
fecha_inicio	Date	Fecha de inicio del subproyecto
fecha_cierre	Date	Fecha de cierre del subproyecto
avance	Int(3)	Porcentaje de avance del subproyecto
estado	String(15)	Estado del subproyecto

14.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearSubproyecto	nombre, descripcion, fecha_inicio, fecha_cierre, avance		Crea un nuevo subproyecto (constructor de la clase)
obtenerSubproyecto		nombre, descripcion, fecha_inicio, fecha_cierre, avance	Obtiene los atributos del subproyecto

modificarSubproyecto	nuevo_nombre, nueva_descripcion, nueva_fecha_inicio, nueva_fecha_cierre, nuevo_avance		Modifica los datos del subproyecto
cerrarSubproyecto			Cierra el subproyecto (cambio de estado)
eliminarSubproyecto			Elimina el subproyecto (destructor de la clase)

15.- **Usuario**

15.1.- Atributos

Nombre	Tipo	Descripción
login	String(20)	Nombre de la cuenta del usuario
contrasena	String(35)	Contraseña del usuario (encriptada)
nombre	String(30)	Nombre del usuario
apellido	String(30)	Apellido(s) del usuario
tipo	String(1)	Tipo de usuario (administrador, tester, desarrollador, etc).
email	String(50)	Correo electrónico
telefono	String(30)	Teléfono(s) del usuario
estado	Int(1)	Estado del usuario (activo=1, bloqueado=0)
idioma	String(15)	Idioma por defecto del usuario
alerta_email	Int(1)	Define si el usuario recibirá las alertas por email o no
alerta_sms	Int(1)	Define si el usuario recibirá las alertas por SMS o no

15.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearUsuario	login, contrasena, nombre, apellido, tipo, email, telefono, idioma, alerta_email alerta_sms		Crea un nuevo usuario con el estado=1 por defecto (constructor de la clase)

obtenerUsuario		login, contrasena, nombre, apellido, tipo, email, telefono, idioma, alerta_email alerta_sms, estado	Obtiene los atributos del usuario
modificarUsuario	nuevo_login, nueva_contrasena, nuevo_nombre, nuevo_apellido, nuevo_tipo, nuevo_email, nuevo_telefono, nuevo_idioma, nueva_alerta_email nueva_alerta_sms		Modifica los datos del usuario
bloquearUsuario			Cambia el estado del usuario a 0 (bloqueado)
desbloquarUsuario			Cambia el estado del usuario a 1 (activo)
eliminarUsuario			Elimina el usuario (destructor de la clase)

16.- **Versión**

16.1.- Atributos

Nombre	Tipo	Descripción
nombre	Int(1)	Nombre de la versión
fecha_liberacion	String(250)	Fecha de liberación de la versión
diferencias_anterior	String(250)	Diferencias de la versión anterior con la actual
pendiente_proxima	String(250)	Diferencias de la versión actual con la próxima
comentarios	String(250)	Comentarios

16.2.- Métodos

Nombre	Parámetros	Retorno	Descripción
crearVersion	nombre, fecha, diferencias, pendiente, comentarios		Crea una nueva versión (constructor de la clase)
obtenerVersion		nombre, fecha, diferencias, pendiente, comentarios	Obtiene los atributos de la versión
modificarVersion	nuevo_nombre, nueva_fecha, nuevas_diferencias, nuevos_pendientes, nuevos_comentarios		Modifica la versión
eliminarVersion			Elimina la versión (destructor de la clase)

ANEXO F

DICCIONARIO DE DATOS

1.- Base de Datos

- Nombre: castordb
- DBMS: MySQL.

2.- Tablas

La clave principal de cada tabla está subrayada.

2.1.- Alerta

Tabla que almacena todas las alertas emitidas por el sistema a los usuarios.

2.1.1.- Campos

Nombre	Tipo	Descripción
id_alerta	Int(15) Auto	Identificador de la alerta
usuario	Varchar(20)	Usuario que recibió el mensaje
mensaje	Text	Mensaje enviado
fecha	Int(11)	Fecha de envío
tipo	Char(1)	Medio de envío de la alerta. H=Herramienta. M=Mail. S=SMS
resultado	Int(1)	Resultado del envío de la alerta. 0=error. 1=correcto

2.2.- **AlertaTarea**

Tabla que almacena la relación alerta-tarea de las alertas enviadas.

2.2.1.- Campos

Nombre	Tipo	Descripción
id_alerta_tarea	Int(15) Auto	Identificador de la alerta-tarea
id_alerta	Int(15)	Identificador de la alerta. Clave foránea de la tabla Alerta
id_tarea	Int(8)	Identificador de la tarea. Clave foránea de la tabla IteracionTCTarea ó IteracionCLTarea (dependiendo del tipo)
tipo	Varchar(2)	Tipo de tarea (tc=Caso de Prueba; cl=Lista de Control)

2.3.- **Checklist**

Tabla que almacena los datos de los checklists.

2.3.1.- Campos

Nombre	Tipo	Descripción
id_checklist	Int(5) Auto	Identificador del checklist
nombre	Varchar(100)	Nombre de la lista
objetivo	Text	Objetivo de la lista
fecha_creacion	Int(11)	Fecha de creación de la lista en segundos
usuario_creador	Varchar(20)	Usuario que creó la lista
id_proyecto	Int(11)	Proyecto para el que se creó el checklist (el valor de 0 en este campo indica que el checklist puede ser utilizado en más de un proyecto). Clave foránea de la tabla Proyecto

2.4.- **Ciclo**

Tabla que almacena los datos de los ciclos de vida.

2.4.1.- Campos

Nombre	Tipo	Descripción
id_ciclo	Int(5) Auto	Identificador del ciclo
nombre	Varchar(50)	Nombre del ciclo
descripcion	Text	Descripción del ciclo
fecha_creacion	Int(11)	Fecha de creación del ciclo en segundos
usuario_creador	Varchar(20)	Usuario que creó el ciclo

2.5.- **CicloFase**

Tabla que guarda la relación ciclo-fase (es decir, qué fases pertenecen a qué ciclo).

2.5.1.- Campos

Nombre	Tipo	Descripción
id_ci_fa	Int(10) Auto	Identificador de la tabla
id_ciclo	Int(5)	Identificador del ciclo. Clave foránea de la tabla Ciclo
id_fase	Int(5)	Identificador de la fase. Clave foránea de la tabla Fase

2.6.- **Cliente**

Tabla que almacena los datos de los clientes.

2.6.1.- Campos

Nombre	Tipo	Descripción
id_cliente	Int(10) Auto	Identificador del cliente
nombre	Varchar(100)	Nombre del cliente
descripcion	Varchar(250)	Descripción del cliente
empresa	Varchar(250)	Empresa del cliente
ciudad	Varchar(100)	Ciudad a la que pertenece el cliente
id_pais	Varchar(2)	Identificador del país al que pertenece el cliente. Clave foránea de la tabla País
telefono	Varchar(100)	Teléfono(s) de contacto del cliente
direccion	Varchar(250)	Dirección física del cliente
email	Varchar(50)	Dirección de correo electrónico del cliente
www	Varchar(250)	Página web del cliente
id_sector	Int(3)	Identificador del rubro en el que trabaja el cliente. Clave foránea de la tabla Sector
fecha_creacion	Int(11)	Fecha de creación del cliente en segundos
usuario_creador	Varchar(20)	Usuario que insertó los datos del cliente en la Base de Datos

2.7.- **CLElemento**

Tabla que guarda la relación checklist-elementoChecklist y los datos de los elementos.

2.7.1.- Campos

Nombre	Tipo	Descripción
id_cle	Int(10) Auto	Identificador del elemento
id_cl	Int(5)	Identificador de la lista a la que pertenece el elemento. Clave foránea de la tabla Checklist
descripcion	Text	Descripción (sentencia)

2.8.- **CLFase**

Tabla que guarda la relación checklist-fase (es decir, para qué fases es aplicable la lista).

2.8.1.- Campos

Nombre	Tipo	Descripción
id_ch_fa	Int(10) Auto	Identificador del registro de la tabla
id_checklist	Int(5)	Identificador de la lista. Clave foránea de la tabla Checklist
id_fase	Int(4)	Identificador de la fase. Clave foránea de la tabla Fase

2.9.- **Comentario**

Tabla que almacena las observaciones de los clientes.

2.9.1.- Campos

Nombre	Tipo	Descripción
id_comentario	Int(11) Auto	Identificador
id_sp	Int(11)	Identificador del subproyecto. Clave foránea de la tabla Subproyecto
id_fase	Int(3)	Identificador de la fase. Clave foránea de la tabla Fase
comentario	Varchar(250)	Comentario del cliente
fecha_comentario	Int(11)	Fecha en la que el cliente dejó el comentario
cliente	Varchar(20)	Cliente que dejó el comentario
comentario_revision	Varchar(250)	Comentario del usuario que atendió la solicitud del cliente

fecha_revision	Int(11)	Fecha en que se atendió el comentario
usuario	Varchar(20)	Usuario que atiende el comentario del cliente
Estado	Int(1)	Estado del comentario. 0=Pendiente. 1=Atendido

2.10.- **Config**

Tabla que almacena algunos de los valores de configuración del sistema.

2.10.1.- Campos

Nombre	Tipo	Descripción
id_config	Int(3) Auto	Identificador
nombre	Varchar(20)	Nombre
descripcion	Varchar(250)	Descripción
valor	Int(3)	Valor

2.11.- **Documento**

Tabla que almacena los datos de los documentos de los proyectos.

2.11.1.- Campos

Nombre	Tipo	Descripción
id_documento	Int(10) Auto	Identificador del documento
id_proyecto	Int(10)	Identificador del proyecto al que pertenece el documento. Clave foránea de la tabla Proyecto
descripcion	Varchar(250)	Descripción del documento
ubicación	Varchar(250)	Ubicación en el servidor del documento
fecha	Int(11)	Fecha en que se copió el documento al servidor

2.12.- **Entorno**

Tabla que almacena los datos de los entornos de ejecución.

2.12.1.- Campos

Nombre	Tipo	Descripción
id_entorno	Int(3) Auto	Identificador del entorno
nombre	Varchar(50)	Nombre del entorno
descripcion_hw	Varchar(250)	Descripción del entorno de hardware
descripcion_sw	Varchar(250)	Descripción del entorno de software
otros	Varchar(250)	Descripciones adicionales

2.13.- **Estado**

Tabla que almacena los datos de los entornos de ejecución.

2.13.1.- Campos

Nombre	Tipo	Descripción
id_estado	Int(2) Auto	Identificador del estado
nombre	Varchar(50)	Nombre del estado
descripcion	Text	Descripción del estado
color	Varchar(6)	Color que identifica al estado
puede_eliminarse	Int(1)	Indica si el estado puede ser eliminado o no
tester	Int(1)	Indica si el estado es accesible por un usuario de tipo tester
desarrollador	Int(1)	Indica si el estado es accesible por un usuario de tipo administrador

2.14.- **Fase**

Tabla que almacena los datos de las fases.

2.14.1.- Campos

Nombre	Tipo	Descripción
id_fase	Int(4) Auto	Identificador del entorno
nombre	Varchar(50)	Nombre de la fase
descripcion	Varchar(250)	Descripción de la fase
fecha_creacion	Int(11)	Fecha de creación de la fase en segundos
usuario_creador	Varchar(20)	Usuario que creó la fase
accesible_cliente	Int(1)	Indica si la fase puede ser vista por el cliente. 0=No. 1=Si

2.15.- **Idioma**

Tabla que almacena los datos de los idiomas disponibles en el sistema.

2.15.1.- Campos

Nombre	Tipo	Descripción
id_idioma	Int(2) Auto	Identificador del idioma
Nombre	Varchar(100)	Nombre del idioma

2.16.- **Iteración**

Tabla que almacena los datos de los entornos de ejecución.

2.16.1.- Campos

Nombre	Tipo	Descripción
id_iteracion	Int(15) Auto	Identificador de la iteración
id_sp	Int(10)	Identificador del subproyecto al que pertenece la iteración. Clave foránea de la tabla Subproyecto
id_fase	Int(4)	Identificador de la fase a la que pertenece la iteración. Clave foránea de la tabla Fase
nombre	Varchar(250)	Nombre de la iteración
fecha_creacion	Varchar(250)	Fecha de creación de la iteración
usuario_creador	Varchar(20)	Usuario que creó la iteración
id_estado	Int(3)	Estado de la iteración
fecha_cierre	Int(11)	Fecha de cierre de la iteración en segundos
usuario_cierre	Varchar(20)	Usuario que cerró la iteración
avance	Int(3)	Porcentaje de avance de la iteración

2.17.- **IteraciónCL**

Tabla que guarda la relación iteración-checklist.

2.17.1.- Campos

Nombre	Tipo	Descripción
id_it_cl	Int(15) Auto	Identificador de la tabla
id_it	Int(10)	Identificador de la iteración. Clave foránea de la tabla Iteración
id_cl	Int(4)	Identificador del checklist. Clave foránea de la tabla Checklist
id_result	Varchar(250)	Resultado de la lista
id_estado	Varchar(20)	Estado de la lista. Clave foránea de la tabla Estado
id_version	Int(3)	Versión en la que se ejecutó la lista. Clave foránea de la tabla Versión
id_entorno	Int(11)	Entorno de ejecución de la lista. Clave foránea de la tabla Entorno
id_nivel	Varchar(20)	Nivel asignado a la lista en la actual iteración. Clave foránea de la tabla Nivel
id_prioridad	Int(3)	Prioridad dada a la lista en la actual iteración. Clave foránea de la tabla Prioridad

Avance	Int(3)	Porcentaje de avance de ejecución de la lista (depende de la cantidad de elementos ejecutados)
Fecha_ejecucion	Int(11)	Fecha de la última ejecución de la lista en la iteración actual en segundos
Usuario_ejecutor	Varchar(20)	Usuario que ejecutó la lista

2.18.- **IteraciónCLTarea**

Tabla que guarda las tareas asignadas a los desarrolladores y testers para un checklist ejecutado en una determinada iteración.

2.18.1.- Campos

Nombre	Tipo	Descripción
id_it_cl_ta	Int(8) Auto	Identificador de la tabla
id_it_cl	Int(10)	Identificador de la iteración-checklist. Clave foránea de la tabla IteraciónCL
Usuario	Varchar(20)	Login del usuario que tiene la tarea
descripcion_tarea	Text	Descripción de la tarea del tarea
id_estado	Int(2)	Estado de la lista. Clave foránea de la tabla Estado
fecha_reporte	Int(11)	Fecha de reporte de la tarea en segundos
fecha_revision	Int(11)	Fecha de revisión del desarrollador en segundos
comentarios	Text	Comentarios puestos por el desarrollador luego de la revisión

2.19.- **IteraciónCLE**

Tabla que guarda la relación iteración-checklist-elemento y los resultados que los elementos tuvieron durante la ejecución del checklist en la iteración dada.

2.19.1.- Campos

Nombre	Tipo	Descripción
id_it_cle	Int(8) Auto	Identificador de la tabla
id_it_cl	Int(10)	Identificador de la iteración-checklist. Clave foránea de la tabla IteraciónCL
id_elemento	Int(11)	Identificador del elemento de la lista
resultado	Int(1)	Resultado obtenido durante la ejecución (0=no ejecutado aún; 1=pasó; -1=falló; -2=fue omitido)
comentario	Varchar(250)	Comentario del ejecutor

2.20.- **IteraciónTC**

Tabla que guarda la relación iteración-testcase.

2.20.1.- Campos

Nombre	Tipo	Descripción
id_it_tc	Int(15) Auto	Identificador de la tabla
id_it	Int(10)	Identificador de la iteración. Clave foránea de la tabla Iteración
id_tc	Int(10)	Identificador del testcase. Clave foránea de la tabla Testcase
resultado_actual	Text	Resultado obtenido de la ejecución
fecha_ejecucion	Int(11)	Fecha de la última ejecución del caso en la iteración actual en segundos
usuario_ejecutor	Varchar(20)	Usuario que ejecutó el caso
id_entorno	Int(11)	Entorno de ejecución del caso. Clave foránea de la tabla Entorno
id_nivel	Varchar(20)	Nivel asignado al caso en la actual iteración. Clave foránea de la tabla Nivel
id_prioridad	Int(3)	Prioridad dada al caso en la actual iteración. Clave foránea de la tabla Prioridad
id_resultado	Int(2)	Resultado general de la ejecución
id_estado	Int(2)	Estado del caso. Clave foránea de la tabla Estado
Usuario_ejecutor	Varchar(20)	Usuario que ejecutó el caso

2.21.- **IteraciónTCTarea**

Tabla que guarda las tareas asignadas a los desarrolladores y testers para un caso de testeo ejecutado en una determinada iteración.

2.21.1.- Campos

Nombre	Tipo	Descripción
id_it_tc_ta	Int(8) Auto	Identificador de la tabla
id_it_tc	Int(10)	Identificador de la iteración-casotesteo. Clave foránea de la tabla IteraciónTC
Usuario	Varchar(20)	Login del usuario que tiene la tarea
descripcion_tarea	Text	Descripción de la tarea del desarrollador
id_estado	Int(2)	Estado de la lista. Clave foránea de la tabla Estado
fecha_reporte	Int(11)	Fecha de reporte de la tarea en segundos
fecha_revision	Int(11)	Fecha de revisión del desarrollador en segundos

comentarios	Text	Comentarios puestos por el desarrollador luego de la revisión

2.22.- **Nivel**

Tabla que almacena la información sobre los distintos tipos de niveles.

2.22.1.- Campos

Nombre	Tipo	Descripción
id_nivel	Int(2) Auto	Identificador del nivel
nombre	Varchar(20)	Nombre del nivel
descripcion	Text	Descripción del nivel
valor	Int(2)	Valor que toma el nivel
puede_eliminarse	Int(1)	Indica si el nivel puede o no ser eliminado

2.23.- **País**

Tabla que guarda los datos de los países.

2.23.1.- Campos

Nombre	Tipo	Descripción
id_pais	Varchar(2)	Identificador del país
nombre	Varchar(50)	Nombre del país

2.24.- **Prioridad**

Tabla que guarda los datos de las distintas prioridades.

2.24.1.- Campos

Nombre	Tipo	Descripción
id_prioridad	Int(2) Auto	Identificador de la prioridad
nombre	Varchar(20)	Nombre de la prioridad
descripcion	Text	Descripción de la prioridad
valor	Int(2)	Valor que toma la prioridad
puede_eliminarse	Int(1)	Indica si la prioridad puede o no ser eliminado

2.25.- **Proyecto**

Tabla que guarda los proyectos.

2.25.1.- Campos

Nombre	Tipo	Descripción
id_proyecto	Int(10) Auto	Identificador del proyecto
nombre	Varchar(100)	Nombre del proyecto
objetivo	Text	Objetivo del proyecto
descripcion	Text	Descripción del proyecto
id_cliente	Int(5)	Identificador del cliente al que pertenece el proyecto. Clave foránea de la tabla Cliente
lider	Varchar(20)	Usuario responsable del proyecto
desarrollo	Varchar(20)	Usuario responsable del desarrollo del proyecto
calidad	Varchar(20)	Usuario responsable del control de calidad del proyecto
usuario_creador	Varchar(20)	Usuario que registró el subproyecto
fecha_creacion	Int(11)	Fecha de registro del subproyecto
fecha_inicio	Int(11)	Fecha de inicio del proyecto (dada por el usuario)
fecha_cierre	Int(11)	Fecha de cierre del proyecto (dada por el usuario)
fecha_actualizacion	Int(11)	Fecha de última modificaicón del proyecto
usuario_actualizacion	Int(11)	Login del usuario responsable de la última modificación
id_estado	Int(3)	Identificador del estado en el que se encuentra el proyecto. Clave foránea de la tabla Estado
id_ciclo	Int(3)	Identificador del ciclo de vida asignado al proyecto. Clave foránea de la tabla Ciclo
avance	Int(3)	Porcentaje de avance del proyecto

2.26.- **Sector**

Tabla que guarda los datos de los distintos sectores (rubros).

2.26.1.- Campos

Nombre	Tipo	Descripción
id_sector	Int(3) Auto	Identificador del sector
nombre	Varchar(50)	Nombre del sector
descripcion	Varchar(250)	Descripción del sector

2.27.- **Subproyecto**

Tabla que guarda los subproyectos.

2.27.1.- Campos

Nombre	Tipo	Descripción
id_subproyecto	Int(10) Auto	Identificador del subproyecto
id_proyecto	Int(10)	Identificador del proyecto al que pertenece el subproyecto. Clave foránea de la tabla Proyecto
nombre	Varchar(100)	Nombre del subproyecto
descripcion	Text	Descripción del subproyecto
Id_estado	Int(3)	Identificador del estado en el que se encuentra el subproyecto. Clave foránea de la tabla Estado
fecha_inicio	Int(11)	Fecha de inicio del subproyecto
fecha_cierre	Int(11)	Fecha de cierre del subproyecto
fecha_creacion	Int(11)	Fecha de registro del subproyecto
usuario_creador	Varchar(20)	Usuario que registró el subproyecto
fecha_actualizacion	Int(11)	Fecha de última modificaicón del proyecto
usuario_actualizacion	Int(11)	Login del usuario responsable de la última modificación
desarrollo	Varchar(20)	Usuario responsable del desarrollo del subproyecto
calidad	Varchar(20)	Usuario responsable del control de calidad del subproyecto
avance	Int(3)	Porcentaje de avance del subproyecto

2.28.- **SubproyectoFase**

Tabla que guarda la relación subproyecto-fase.

2.28.1.- Campos

Nombre	Tipo	Descripción
id_sp_fa	Int(11) Auto	Identificador único
Id_sp	Int(10)	Identificador del subproyecto. Clave foránea de la tabla Subproyecto
Id_fase	Int(3)	Identificador de la fase. Clave foránea de la tabla Fase
Avance	Int(3)	Porcentaje de avance de la fase del subproyecto
Id_estado	Int(3)	Identificador del estado en el que se encuentra la fase. Clave foránea de la tabla Estado

Fecha_cierre	Int(11)	Fecha de cierre de la fase en segundos
Usuario_cierre	Varchar(20)	Usuario responsable del cierre de la fase

2.29.- **TCFase**

Tabla que guarda la relación fase-testcase, es decir, a qué fases es aplicable cada caso de testeo.

2.29.1.- Campos

Nombre	Tipo	Descripción
id_tcfa	Int(10) Auto	Identificador único
Id_tc	Varchar(100)	Identificador del caso de testeo. Clave foránea de la tabla Testcase
Id_fase	Int(3)	Identificador de la fase. Clave foránea de la tabla Fase

2.30.- **Testcase**

Tabla que guarda los casos de testeo.

2.30.1.- Campos

Nombre	Tipo	Descripción
id_testcase	Int(10) Auto	Identificador del caso de testeo
Nombre	Varchar(100)	Nombre del caso de testeo
Objetivo	Text	Objetivo del caso
Descripcion	Text	Descripción del caso
Pasos_configuracion	Text	Pasos de configuración
Pasos_ejecucion	Text	Pasos de ejecución
Resultado_esperado	Text	Resultado que se espera obtener
Id_proyecto	Int(10)	Identificador del proyecto para el que fue creado el caso. Si el valor es de 0, entonces el caso es aplicable a más de un proyecto
Fecha_creacion	Int(11)	Fecha de creación del caso
Usuario_creador	Varchar(20)	Usuario que creó el caso de testeo

2.31.- **Usuario**

Tabla que guarda información sobre los usuarios.

2.31.1.- Campos

Nombre	Tipo	Descripción
login	Varchar(20)	Nombre de usuario
contrasena	Varchar(35)	Contraseña del usuario encriptada
nombre	Varchar(30)	Nombre(s) del usuario
apellido	Varchar(30)	Apellido(s) del usuario
tipo	Varchar(5)	Tipo de usuario (Administrador, Líder, Tester, Desarrollador o Cliente)
email	Varchar(50)	Correo electrónico del usuario
telefono	Varchar(30)	Teléfono móvil del usuario (para recibir las alertas por SMS)
estado	Int(1)	Estado del usuario (0=bloqueado; 1=activo)
alerta_email	Int(1)	Define si el usuario recibirá o no las alertas por email
alerta_sms	Int(1)	Define si el usuario recibirá o no las alertas por SMS
idioma_por_defecto	Varchar(2)	Idioma que el usuario utiliza por defecto
fecha_creacion	Int(11)	Fecha de creación del usuario
usuario_creador	Varchar(20)	Usuario que creó la cuenta

2.32.- **Versión**

Tabla que guarda los datos de las versiones de los proyectos.

2.32.1.- Campos

Nombre	Tipo	Descripción
id_version	Int(10) Auto	Identificador del sector
id_proyecto	Int(10)	Identificador del proyecto al que pertenece la versión. Clave foránea de la tabla Proyecto
fecha_liberacion	Int(11)	Fecha de lanzamiento de la versión en segundos
nombre	Varchar(20)	Nombre de la versión
diferencias_version_previa	Varchar(250)	Diferencias de la versión con la anterior (del mismo proyecto)
diferencias_proxima_version	Varchar(250)	Puntos pendientes de la versión para una próxima
comentarios	Varchar(250)	Comentarios