UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 9

TÍTULO: Herramienta para la automatización del proceso de pruebas de Caja Negra en los Proyectos Productivos de la Facultad 9.

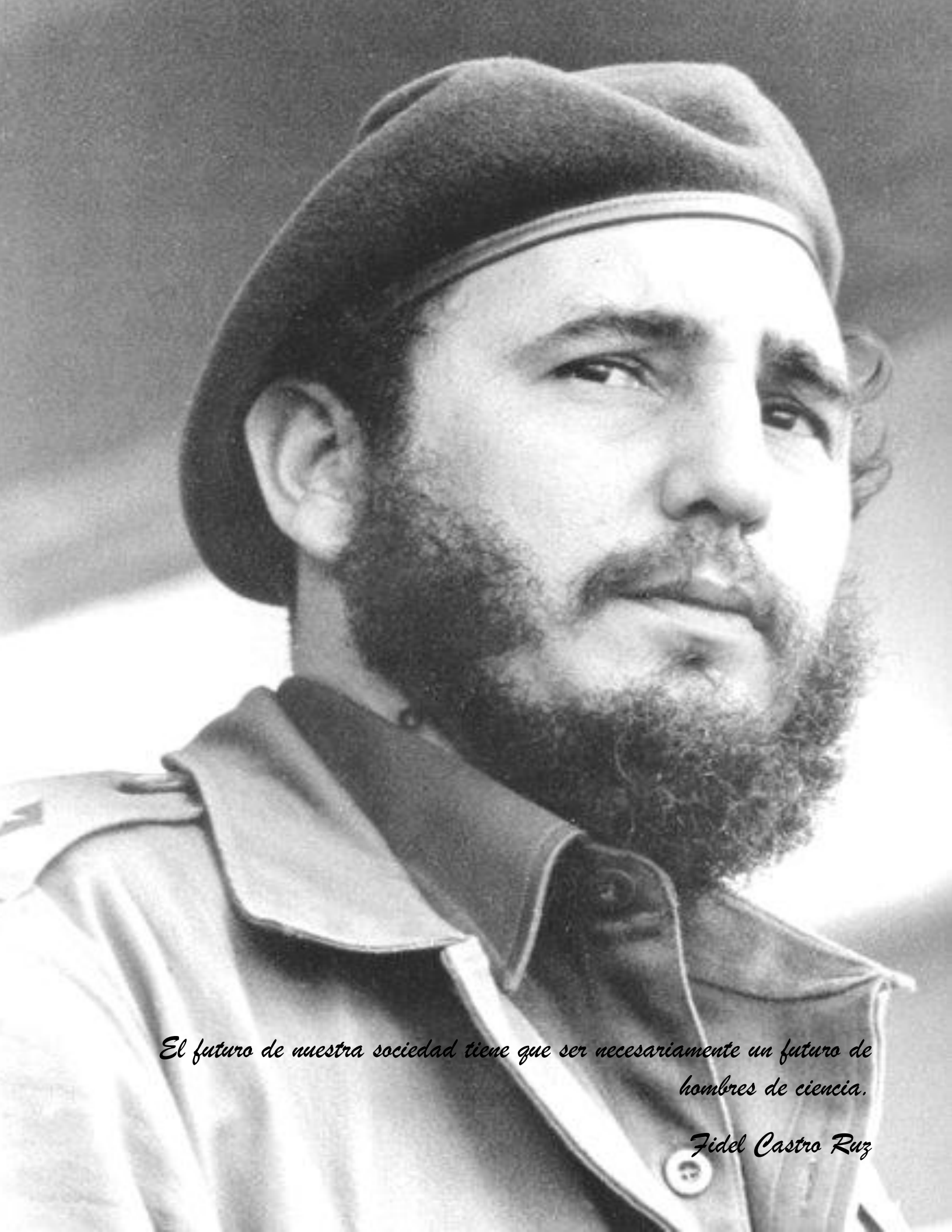
TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor: Maikel Alejo Hernández

Tutor: Ing. Yaquelín Cintra Almaguer

Ciudad de La Habana Junio 2009

AÑO DEL 50 ANIVERSARIO DEL TRIUNFO DE LA REVOLUCIÓN



El futuro de nuestra sociedad tiene que ser necesariamente un futuro de hombres de ciencia.

Fidel Castro Ruz

DEDICATORIA

A 5 personas que considero imprescindibles en mi vida, en primer lugar a mi mamita querida que siempre ha sabido guiarme por el buen camino, y siempre ha estado conmigo en los momentos más difíciles de mi vida, siempre me ha apoyado en todos mis proyectos; en segundo lugar a mi padre que ha sido el ejemplo en el cual me he inspirado para seguir adelante cada día, a su forma ha sabido guiarme siempre por el buen camino, por trabajar tan duro día a día para que pudiera ser lo que soy, eso no lo hacen todos los padres, pero tú no eres un padre cualquiera, tu eres sencillamente "extraclase". Quiero que sepan que si la vida me da otra oportunidad de elegir, volverían a ser mis padres.

A dos mujeres que la vida me dio el privilegio de tenerlas a mi lado desde que tengo 4 años, dos mujeres, que para mí son como mis otras madres, y por eso agradezco a la vida por haberme dado la posibilidad de tenerlas. Ustedes han sido como el faro que ha guiado mi vida, me han enseñado el valor de la honradez, de la modestia, a ser mejor persona cada día, si pudiera darle algún calificativo al igual que a mi madre las considerarías mis diosas, mi razón de ser y el espejo en el que día a día me miro y me dan ganas de seguir triunfando en la vida. Esas mujeres son mi querida abuela Ana luisa y mi tía del alma María Elena.

A mi hermanita, tata te quiero con la vida, aunque la vida nos separó durante casi toda nuestras vidas, eso sólo sirvió para fortalecer nuestro cariño de hermanos, no todo tiene la dicha de tener una hermana como tú, tan preocupada siempre por mí, que siente mis problemas como si fueran tuyos, y eso se agradece mucho, de veras que si.

A mi tío Güicho, que más que un tío es para mí como un padre, también me aconsejó siempre y me supo guiar, siempre preocupado por mis estudios, de verdad siempre te tendré presente donde quiera que yo esté.

A mi abuelo Humberto que a pesar de ser como es, me supo acoger como un hijo, sé que me quiere mucho al igual que yo a él.

A mis primas del alma que más que primas son también mis hermanas Anitica y Adianez, las quiero mucho.

A mi novia Susana, que siempre estuvo a mi lado en los momentos difíciles, me dio cariño y apoyo cuando más lo necesité.

Y por último a una persona que representa mucho para mí, y que por cosas de la vida no puede estar a mi lado en este momento, también esta dedicatoria es para ti.

A mi primo Frank y a su novia Arialis, por haber pasado muchos de los buenos momentos en la universidad a su lado, a mi primo Tico y a su mujer Yuneikis.

AGRADECIMIENTOS

Quiero agradecer en primer lugar a la Revolución y a nuestro Comandante en Jefe por haberme dado la oportunidad de estudiar en un centro como este, y formarme como profesional y revolucionario, a mis compañeros de toda la carrera, pero en especial a uno que me ayudó mucho a lo largo del desarrollo de mi tesis, a mi amigo Rolando Toledo, al “Villa”, a Héctor (el gordo), como le decimos cariñosamente y en fin a todos los que de una forma u otra me ayudaron en este trabajo.

GENERALES DEL TUTOR

Tutor: Ing. Yaquelín Cintra Almaguer

E-mail: ycintra@uci.cu.

- Ingeniera en Ciencias Informáticas, Universidad de las Ciencias Informáticas, 2008.
- Profesor del Departamento de Ingeniería y Gestión de Software- Práctica Profesional, facultad 9.

RESUMEN

En los proyectos productivos de la facultad 9, de la Universidad de las Ciencias Informáticas no existe hoy en día una herramienta que automatice el proceso de pruebas de caja negra; precisamente el presente trabajo constituye una investigación que pretende suplir tal necesidad. En el mismo se describen las tecnologías actuales más utilizadas para el desarrollo de software en el mundo y las escogidas para desarrollar el sistema que se propone. Se incluyen además las etapas de Modelación del Negocio, Análisis, Diseño e Implementación sobre la base del análisis de la arquitectura que soporta el sistema. Al finalizar se realiza el modelo de implementación, en el que se definen los principales paquetes de componentes y subsistemas de implementación que conforman la aplicación; quedando los mismos evidenciados en el diagrama de componentes. Además de la distribución de los dispositivos necesarios para que se ejecute correctamente el sistema representados en el diagrama de despliegue propuesto.

PALABRAS CLAVES

Prueba, Pruebas de Caja negra, Implementación, Análisis, Diseño, Arquitectura.

INDICE

INDICE DE FIGURAS

INTRODUCCIÓN

En los últimos años el software ha experimentado un desarrollo sin precedentes, cada vez es mayor la complejidad y la cantidad de software que se demanda en el mercado. En la actualidad existe una tendencia a que los usuarios necesitan el software en un tiempo bastante corto y a un costo menor, esto trae consigo que no siempre los software que se presentan en el mercado tengan la calidad que el usuario espera de los mismos y es en donde aparecen muchas veces las desavenencias entre el cliente y la empresa productora de software, esto pudiera estar provocado debido a que de manera general los métodos de desarrollo de software que se siguen son, básicamente, aquellos que los propios individuos "artesanalmente" siguen, o sea no se rigen en cuenta ninguna metodología de desarrollo de software.

El tema de la calidad es un tema que preocupa a muchos productores de software, ya desde la década de los 70 aproximadamente muchos productores de software e investigadores han dedicado sus investigaciones a cómo lograr un producto con calidad y cómo medir la calidad del software.

La calidad del software puede parecer un concepto alejado de la vida diaria de la mayoría de las personas. Cuántas veces en nuestro ordenador ha aparecido un mensaje de error o una pantalla azul, se está ante un problema de calidad del software; cuando un fallo en el sistema de gestión inutiliza pantallas de información, es un problema de calidad, cuando en un mercado se bloquean los ordenadores de la caja de cobro y anotación de ventas, tiene que ver la calidad, un ejemplo bien documentado se produjo con el lanzamiento en 1996 del primer cohete Arianne 5 de la Agencia Espacial Europea.

Este ingenio que costó 10 años y 7.000 millones de euros desarrollar, explotó por un defecto en el software de control interno antes de que pasara un minuto de vuelo, estos son solo algunos ejemplos de fallos debido a la baja calidad del software y en los cuales como queda bien claro las pérdidas han sido considerables. Es muy probable que se haya sufrido los efectos de estos problemas de calidad en forma de retrasos, pérdidas de tiempo o dinero, etc. como simples ciudadanos. Lamentablemente, estos problemas pueden ser mucho más graves si afectan a sistemas críticos; es decir, aquellos cuyo fallo puede provocar graves pérdidas económicas o problemas ambientales o sociales e, incluso, la pérdida de vidas humanas que es más triste.

Las actividades de aseguramiento de la calidad del software son sin duda una parte muy importante en la producción de un software. Debido a la competencia existente a nivel mundial se hace difícil insertarse en el mercado, por lo que se requiere un producto cada vez mejor, aunque no siempre estos mecanismos se aplican de la forma correcta.

Estas actividades se realizan con el objetivo de detectar errores antes de la culminación del software ya que si se detectan problemas luego de elaborado el producto, no solo traería pérdidas monetarias, sino pérdidas de tiempo y esfuerzo, por lo que es imprescindible tener en cuenta durante todas las etapas del ciclo de vida del software es necesario realizar pruebas con el objetivo de ir detectando y eliminando errores en cada iteración y para que una vez culminado el desarrollo de del sistema los posibles errores no impliquen grandes cambios, lo que traería como resultado retrasos en la entrega del producto. Además se puede tener una idea de cómo va avanzando la construcción del producto y cuáles son las características del producto que se va obteniendo.

No todas las empresas dedicadas a la producción de software ponen su empeño en lograr un producto con mayor calidad, que no tenga problemas en cuanto a tiempo de entrega, al rendimiento del mismo, sólo se preocupan por producir para cumplir con un cliente que le hizo una petición de algún producto por lo que llevan a cabo el proceso de desarrollo de la aplicación de una forma artesanal y violan un grupo de acciones para lograr finalmente su objetivo y es ahí donde radica principalmente el problema de la calidad.

En Cuba la industria del software aún pudiera decirse que es pionera en esta rama. La Universidad de las Ciencias Informáticas (UCI) como uno de los máximos exponentes del desarrollo de esta industria, tiene como misión principal la informatización de la sociedad cubana, y además ir construyendo poco a poco en nuestro país las bases para que en un futuro Cuba se convierta en productora de software que contribuirá al desarrollo económico del país.

Pero como es evidente la UCI no está fuera del dominio del grupo de los problemas mencionados anteriormente, por lo que en la misma de llevas a cabo un conjunto de acciones para lograr que los productos que en esta se realizan tengan la mayor calidad posible.

Para esto existe en la universidad un grupo de aseguramiento de la calidad del software, que es el encargado a nivel central de asesorar a los grupos de calidad que existen en las 10 facultades que

componen esta nuestra universidad. En cada unas de las facultades existen grupos de aseguramientos de la calidad los cuales son los máximos responsables de asegurar la calidad de los productos que por estos laboratorios pasan antes de ser liberados a la dirección de calidad a nivel central que es quien libera el producto finalmente.

Con el fin de lograr un producto cada vez mejor se llevan cabo un conjunto de acciones entre las que destacan “la pruebas de software” que son sin duda una parte importante del aseguramiento de la calidad del software.

Aún así, con la aplicación de estas pruebas, en el proyecto de calidad de la facultad 9 no se logra la calidad óptima en los productos debido a que estas pruebas se realizan de forma manual lo cual no permite que se encuentren la mayor cantidad de errores en cada revisión, muchas veces no se pueden entregar los resultados de las revisiones en tiempo, debido a la demora de los diseñadores de casos de prueba en confeccionar los mismos, y luego proceder a la realización de las pruebas. Teniendo en cuenta la **Situación problemática** descrita anteriormente se identificó el siguiente **Problema Científico:** ¿Cómo lograr que el proceso de realización de pruebas de caja negra en los proyectos productivos de la facultad 9 se realice de una forma óptima?, para lo cual se plantea como **Objetivo General:** Implementar una herramienta de software que permita automatizar las pruebas de caja negra, partiendo de la siguiente **Idea a Defender:** Si se implementa una herramienta de software que permita automatizar el desarrollo de las pruebas de caja negra en los proyectos productivos de la facultad 9 se logrará optimizar dicho proceso. Por lo que el **Objeto de Estudio** estará centrado en el estudio de los diferentes métodos de pruebas de caja negra que existen a nivel mundial, enfocando nuestro **Campo de Acción** en la automatización del proceso de pruebas de caja negra en los proyectos productivos de la facultad 9.

Para dar cumplimiento al objetivo general planteado anteriormente se definen a continuación algunas tareas con cada una de las subtareas que contribuyen a dar cumplimiento a nuestro principal objetivo.

Tareas de la investigación:

1. Caracterizar los diferentes criterios de pruebas de caja negra.

1.1 Investigar sobre los diferentes criterios usados para la realización de pruebas de caja negra.

1.2 Seleccionar el criterio de pruebas de caja negra más apropiado para la automatización del proceso de pruebas.

2. Definir la herramienta para la implementación del software que permita la automatización de las pruebas de caja negra.

2.1 Realizar un estudio sobre los diferentes lenguajes de programación y definir el más factible para la implementación de la herramienta.

3. Modelar una herramienta de software que permita automatizar las pruebas de caja negra.

3.1 Definir el modelo de negocio de la herramienta a automatizar.

3.2 Modelar el flujo de trabajo de requerimientos.

3.3 Modelar el flujo de trabajo de diseño.

3.4 Modelar el flujo de trabajo de implementación.

3.5 Modelar el flujo de trabajo de pruebas.

4. Implementar una herramienta de software que permita automatizar el proceso de pruebas de caja negra.

A continuación se citan los métodos científicos empleados para realizar la investigación:

Métodos Teóricos:

Modelación: Se seleccionó este método debido a que se debe establecer un modelo para guiar la investigación, además en esta investigación se va a llevar a cabo un análisis profundo de cada uno de los modelos de prueba de caja negra que existen con el objetivo de determinar por cual estará regido la herramienta que se propone.

Histórico-Lógico: Se va a realizar un estudio profundo de las diferentes modelos de prueba de caja negra, así como la evolución de dichos métodos para determinar el más adecuado para la implementación de la herramienta que el presente trabajo se propone.

Métodos Empíricos

Observación:

Mediante la observación se perciben los principales problemas que existen en el proyecto para realizar las pruebas de caja negra; pues en la actualidad no se recogen todos los errores que pudiesen ser detectados luego de implementar uno de los métodos de prueba.

Al culminar este trabajo se espera obtener los siguientes resultados:

1. Generación de la documentación y artefactos de ingeniería obtenidos en el proceso de desarrollo de la herramienta de software que permita automatizar las pruebas de caja negra a los proyectos productivos de los polos de la facultad 9

2. Implementación de una herramienta de software que permita automatizar las pruebas de caja negra.

El trabajo estará estructurado en 3 capítulos:

Capitulo 1: Fundamentación teórica.

En este capítulo se exponen los principales conceptos asociados al dominio del problema, así como las principales herramientas y metodologías que serán usadas en el desarrollo de la aplicación.

Capítulo 2: Modelar una herramienta de software que permita automatizar las pruebas de caja negra.

En este capítulo se realizará la modelación de la herramienta, la cual incluye la realización de todos los diagramas así como la generación de los diferentes artefactos y su documentación correspondiente.

Capítulo 3: Implementar la herramienta que permita automatizar el proceso de pruebas de caja negra.

En este capítulo se abordará la implementación de la herramienta así como toda la documentación asociada a este flujo de trabajo. La misma tendrá como resultado el producto terminado.

CAPITULO 1: FUNDAMENTACION TEORICA

1.1 Introducción

En el presente capítulo se exponen los principales conceptos asociados al dominio del problema y que serán utilizados a lo largo de la investigación. Además se realiza un análisis y comparación de las diferentes herramientas y metodologías existentes en la actualidad, lo que quedará reflejado con la selección de las herramientas a utilizar.

1.2 Conceptos asociados al dominio del problema

1.2.1 Pruebas de Caja Negra

Las pruebas de caja negra, también denominadas pruebas de comportamiento, se centran en los requisitos funcionales del software, o sea, la prueba de caja negra permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de caja negra no es una alternativa a las técnicas de prueba de caja blanca. Más bien se trata de un enfoque complementario que intenta descubrir diferentes tipos de errores que los métodos de caja blanca no detectan. **(1)**

1.2.2 Calidad

"Calidad es adecuación al uso del cliente". **(2)**

La Organización Internacional de Estándares (ISO, por sus siglas en inglés) ha publicado hasta hoy varios estándares relacionados con la calidad en general y también de manera particular con la calidad de software, de todas las normas ISO publicadas, ISO 9000 corresponde al estándar de gestión de calidad ampliamente aceptado y que ha sido la base para otras normas más específicas, ISO 9000 [ISO9000, 2000] conceptualiza la calidad como "*el grado en el que un conjunto de características inherentes cumple con los requisitos*" **(3)**

1.2.3 Calidad de software

La Calidad del Software es "la concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo documentados y con las características implícitas que se esperan de todo software desarrollado profesionalmente". **(1)**

Teniendo en cuenta lo anteriormente expuesto se puede decir que la calidad de un software está dada por el grado de aceptación que tenga el cliente con el producto o servicio brindado, en otras palabras, en la medida que se sea capaz de satisfacer las necesidades planteadas por el usuario se puede entonces tener una idea de cuan eficiente es nuestro producto.

1.3 Prueba de software

Las pruebas de software son elementos críticos para determinar la calidad del software. Las pruebas permiten validar y verificar el software, entendiendo como validación del software el proceso que determina si el software satisface los requisitos, y verificación como el proceso que determina si los productos de una fase satisfacen las condiciones de la misma. **(4)**

El proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, es decir, realizar las pruebas para la detección de errores y asegurar que la entrada definida produce resultados reales de acuerdo con los resultados requeridos. **(1)**

Las pruebas de software son una parte esencial en el proceso de desarrollo de un software. Existen varios tipos de pruebas (Usabilidad, Especificación, Unidad, Integración, Regresión) las cuales actúan como un todo y están encaminadas a garantizar la calidad del mismo a través de la detección de los errores existentes en cada una de las fases desarrollo.

1.4 Metodología:

A continuación se mencionan dos definiciones de metodología **(5)**:

1. Ciencia del método.

2. Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal.

1.5 Conceptos Generales

1.5.1 Requerimientos Funcionales

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir, son la base para tener una idea clara sobre lo que debe hacer el sistema. **(1)**

1.5.2 Modelo de Dominio

Un modelo de dominio es un modelo captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las "cosas" que existen o los eventos que suceden en el entorno en el que trabaja el sistema. **(8)**

1.6 Metodologías de desarrollo:

1.6.1 Programación extrema (XP)

La metodología XP (Extreme Programming) por sus siglas en inglés es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. **(6)**

Roles XP (6)

Los roles de acuerdo con la propuesta original de Beck[1] son:

- **Programador:** El programador escribe las pruebas unitarias y produce el código del sistema.

- **Cliente:** Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.

[1] **K.Beck: Autor del libro Exreme Programming Expained del año 2000**

- Encargado de pruebas (Tester): Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

- Encargado de seguimiento (Tracker): Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.

- Entrenador (Coach): Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

- Consultor. Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

- Gestor (Big boss): Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

Proceso XP (6)

El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye el valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración. **(6)**

Fases XP (6)

Fase I: Exploración

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

Fase II: Planificación de la Entrega

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días.

Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la "velocidad" de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración.

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

Fase III: Iteraciones

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

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

Fase IV: Producción

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento).

Fase V: Mantenimiento

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la

puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

Fase VI: Muerte del Proyecto

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

1.6.2 SCRUM

Scrum divide un proyecto en iteraciones (a las que se les llaman carreras cortas) de 30 días. Antes de que comience una carrera se define la funcionalidad requerida para esa carrera y entonces se deja al equipo para que la entregue. El punto es estabilizar los requisitos durante la carrera. Sin embargo la gerencia no se desentiende durante la carrera corta, todos los días el equipo sostiene una junta corta (quince minutos), llamada Scrum, dónde el equipo discute lo que hará al día siguiente. En particular muestran a los bloques de la gerencia: los impedimentos para progresar que se atraviesan y que la gerencia debe resolver. También informan lo que se ha hecho para que la gerencia tenga una actualización diaria del estado en que se encuentra el proyecto. **(7)**

El centro de Scrum es el acercamiento a la creencia de que la mayoría de los sistemas de desarrollo tienen una mala filosofía básica. La falla de los proyectos, sistemas inapropiados y herramientas de productividad poco efectivas son pruebas de que el proceso de desarrollo necesita más rigor, si se logra que todos los desarrolladores se relacionen entre si, los problemas se irán. Este declara que el proceso de desarrollo de los sistemas es un proceso impredecible y complicado que sólo se puede describir en forma global. Además define que el proceso del desarrollo de los sistemas es un conjunto de actividades separadas de las cuales se tiene cierto conocimiento, las herramientas de uso fácil y las técnicas ayudan a obtener un mejor desarrollo en equipo para construir los sistemas. Desde el momento en que se separan las actividades el control de cada una adopta un riesgo inherente. **(7)**

El ritmo de Scrum es la clave de su éxito. La administración determina la prioridad para cada carrera, su determinación está influenciada por la prioridad de las entregas y los requerimientos. **(7)**

Un corto gran esfuerzo de trabajo duradero aproximadamente durante 30 días durante el cual un ejecutable y otras características son construidos por un equipo de ingeniería, el cual es definido por el Backlog asignado. **(7)**

Las características de las carreras son:

- ✓ No debe de durar más de 30 días.
- ✓ La carrera es emprender por un cruce funcional de equipo comprendido de no más de 9 miembros.
- ✓ Cada carrera tiene una meta específica. Un ejecutable demostrando la meta que será completada por el equipo durante la carrera.
- ✓ Si alguna fuerza externa determina que la carrera está trabajando erróneamente, la carrera es detenida y reiniciada con un nuevo Backlog y propósito.
- ✓ Una vez que inicia la carrera, un nuevo Backlog no puede ser agregado a la carrera excepto si, el administrador del proyecto determina que el nuevo Backlog refuerza la viabilidad del producto.

1.6.3 Feature Driven Development (FDD):

Se enfoca en iteraciones cortas que entregan funcionalidad tangible. En el caso del FDD las iteraciones duran dos semanas como máximo. Se definen dos tipos de programadores: dueños de clases y programadores jefe. **(7)**

FDD es extremadamente efectivo en proyectos largos con una lógica del negocio compleja. Y no es efectiva en proyectos pequeños y con una simple lógica del negocio. Esto no significa que no se deba usar FDD para ayudar a la administración para hacer un proyecto de pequeña escala pero es mejor dando un soporte fuerte al código y a las revisiones de diseño **(7)**

El FDD tiene cinco procesos. Los primeros tres se hacen al principio del proyecto. Los últimos dos se hacen en cada iteración. Cada proceso se divide en tareas y se da un criterio de comprobación. Cada uno de estos procesos se compone de: criterio de entrada, tareas, verificación y criterio de salida, esto ayuda a tener el proyecto bajo control con un proceso perfectamente sistematizado. **(7)**

Cada tarea lleva un conjunto de personas que desarrollan distintas actividades dependiendo del rol que tengan dentro del proyecto. La mayoría de estas actividades son obligatorias ya que ayudan a llevar el control del desarrollo del proyecto. **(7)**

1.6.4 La Metodología RUP

El proceso unificado de desarrollo (RUP) es una metodología que presenta una infraestructura flexible de desarrollo de software que proporciona prácticas recomendadas probadas y una arquitectura configurable. Es un proceso práctico.

Es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de amplitud y diferentes tamaños de proyecto. Está basado en componentes, lo que significa que el sistema software está formado por componentes software interconectados por interfaces. **(6)**

Los principales aspectos de RUP se definen en las siguientes características **(6)**

-Dirigido por casos de usos: Los casos de uso (CU) son fragmentos de funcionalidades que el sistema debe cumplir para dar respuesta a una petición del usuario, y son estos CU los que guían el diseño, implementación y prueba de un sistema. Basándose en los modelos de CU los desarrolladores crean una serie de modelos de diseño e implementación que se llevan a cabo en el desarrollo de software.

-Iterativo e incremental: El desarrollo de un proyecto se lleva a veces un gran esfuerzo, que puede durar varios meses incluso hasta un año o más. Es aconsejable dividir el trabajo en partes más pequeñas. Cada una de estas partes se considera una iteración que resulta en un incremento. Las iteraciones se refieren a pasos dentro del flujo de trabajo y los incrementos al crecimiento del producto. Estas iteraciones deben ser controladas: deben seleccionarse y ejecutarse de forma planificada. En cada una de las iteraciones se tratan los riesgos más importantes.

Un proceso iterativo controlado presenta varios beneficios tales como:

- Reducción del coste del riesgo a los de un solo incremento.

- Reducción del riesgo de no sacar el producto al mercado en el tiempo previsto.

-Acelera el ritmo de desarrollo en su totalidad debido a que los desarrolladores trabajan de manera eficiente para obtener un resultado.

-Centrado en la Arquitectura: El concepto de Arquitectura incluye los aspectos estáticos y dinámicos más significativos del sistema. Surge de las necesidades de la empresa como la perciben los usuarios y los inversores y se refleja en los CU.

La arquitectura es la encargada de darle la forma al software (SW). Para darle esta forma al SW los arquitectos trabajan sobre los CU más significativos con el objetivo de encontrar precisamente dicha forma. Es además el arquitecto el encargado de seleccionar la plataforma sobre la cual va a funcionar el SW, así como, el Sistema gestor de Base de Datos y el protocolo de comunicación. Es la arquitectura una parte esencial en la construcción del software debido a que permite mirar el SW desde varios puntos de vista: es la que va a dar una idea de cómo va a quedar el software.

En RUP se identifican 4 fases: Inicio, Elaboración, Construcción y Transición. Cada una de estas fases termina con un hito. Además existen 6 flujos de trabajo esenciales (modelo de negocio, requerimiento, análisis y diseño, implementación, prueba, despliegue).

En RUP existe además el término “persona” que son los que diseñan guían y llevan a cabo un “proyecto” que es un elemento organizativo a través del cual se gestiona el desarrollo de un software. El resultado de un proyecto es la versión de un producto. El producto son los artefactos que se crean durante la vida del proyecto como los modelos, código fuente, ejecutables, y documentación

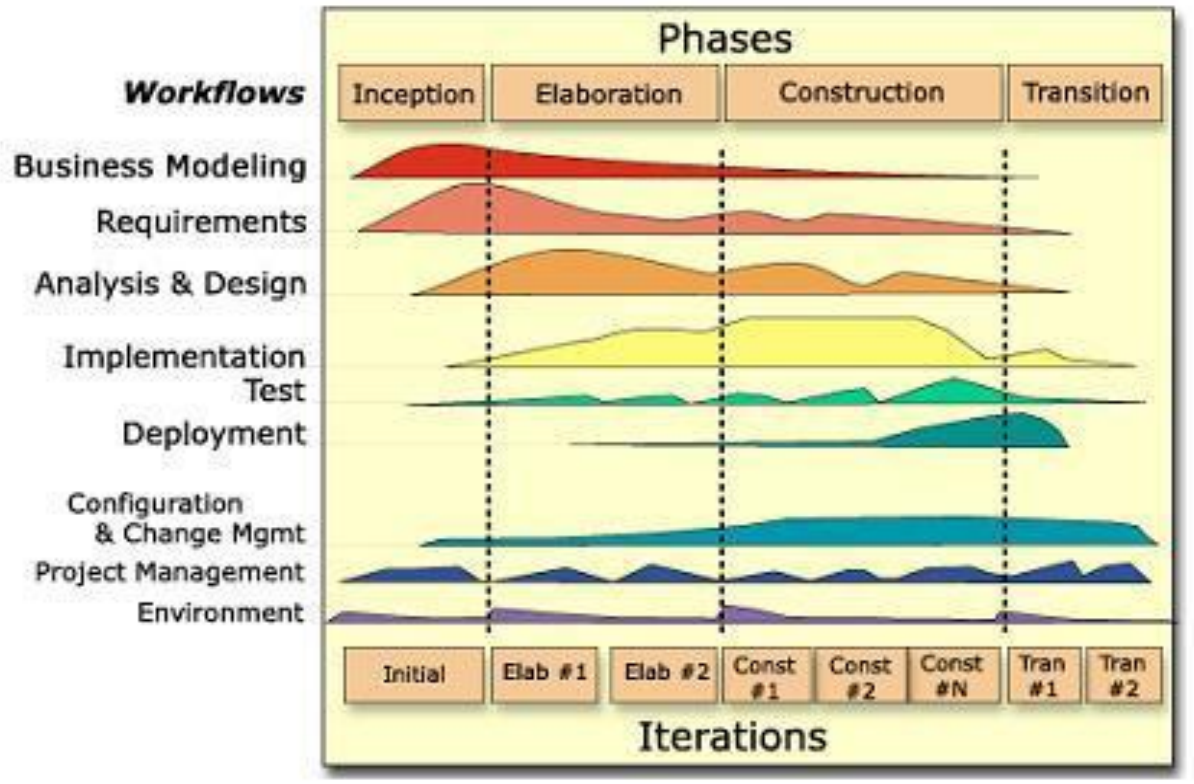


Figura 1: Fases e Iteraciones de la Metodología RUP.

1.7 Selección de la Metodología

El desarrollo de esta herramienta estará guiado por la metodología RUP, debido a que la misma constituye una metodología muy potente para llevar a cabo el desarrollo de un software, es una metodología flexible, o sea, que permite adaptarla según el proyecto que s e está realizando. Tiene además una estructura organizativa muy buena, divide el ciclo de vida del proyecto en fases y flujos de trabajo y en cada iteración se realizan todos los flujos de trabajo, lo que permite al terminar cada fase tener una versión del producto y poder ir mejorándolo a medida que avanza el ciclo de desarrollo del mismo.

1.8 Lenguajes de Programación

1.8.1 C#

C# combina las mejores ideas de lenguajes como C, C++ y Java con las mejoras de productividad de .NET, Framework de Microsoft brinda una experiencia de codificación muy productiva tanto para los nuevos programadores como para los veteranos. **(10)**

C# es un lenguaje orientado a objetos que combina las ventajas de C, C++, y java para lograr un punto intermedio entre el lenguaje de bajo nivel y el lenguaje de alto nivel, esto brinda la posibilidad de crear aplicaciones de escritorio que aunque no tan robustas como las creadas en C++ son aplicaciones que se realizan en un tiempo mucho menor a las que se realizaban anteriormente en C o C++ y con un nivel de seguridad mayor, debido que el C# incorpora un fuerte trabajo en cuanto a la seguridad de sus aplicaciones.

Estas medidas de seguridad pueden determinar si una aplicación puede escribir o leer un archivo de disco. También permiten insertar firmas digitales en la aplicación para asegurarse de que la aplicación fue escrita por una fuente de confianza. .NET Framework también permite incluir información de componentes, y de versión, dentro del código real. Esto hace posible que el software se instale cuando se lo pidan, automáticamente o sin la intervención del usuario. Juntas, todas estas funciones reducen los costes de asistencia para la empresa. **(10)**

C# incorpora además las interfaces que son un conjunto de clases, métodos y propiedades y eventos que especifican un conjunto de funcionalidades. Las clases en C# pueden implementar interfaces y servirse de las funcionalidades que brindan estas sin necesidad de interferir en su código.

1.8.2 C++

C++ es un lenguaje de Programación Orientado a Objetos (OO), el mismo tiene como antecesor al lenguaje C. Es un híbrido entre la programación OO y la programación estructurada por lo que se dice que es un lenguaje multiparadigma[2]. Tiene muchas ventajas en cuanto al trabajo a bajo nivel, es muy potente es por eso que es muy usado a la hora de construir compiladores y otros programas que

[2] Multiparadigma: Es un lenguaje de programación donde se puede utilizar varios paradigmas de programación como son orientado a objetos y estructurado.

necesitan de una programación a bajo nivel muy fuerte. Las aplicaciones que se construyen sobre este lenguaje son muy robustas. Soporta varios tipos de datos así como herencia: herencia simple que es que una clase hereda de una clase padre y la herencia múltiple que es que una clase puede heredar de varias clases padres. Esto ofrece gran ventaja para resolver problemas de la vida real y que se presentan a diario. A pesar de todo esto este lenguaje requiere de mucho ingenio para su aprendizaje.

1.8.3 Java

Java es en un punto intermedio, su modelo de objetos es sencillo e incluye algunos tipos de datos (como los arreglos) que no únicamente se utilizan en el modelo de programaci6n orientada a objetos, acercándose más al modelo de lenguaje estructurado. Las características de Java son, esencialmente, las de C++, con algunas mejoras en el manejo de los objetos y ayuda al programador, como el hecho de incorporar un recolector automático de basura que impide incurrir en un error por inexperiencia en el manejo directo de memoria **(11)**

El lenguaje de programación Java es un lenguaje orientado a objetos que surgió bajo la filosofía de la Programación Orientada a Objetos (POO) esto significa que no tuvo que evolucionar hacia ella a diferencia de otros lenguajes de programación como el C++ por ejemplo, que tuvieron que arrastrar estructuras de versiones anteriores, en java no está presente la herencia múltiple, o sea, que una clase no puede heredar de dos padres. Además de ser un lenguaje multiplataforma, o sea, los programas realizados en Java corren sobre cualquier Sistema operativo (SO), ya sea Windows, Linux (incluyendo todas sus versiones y distribuciones), o Macintosh (Mac), lo que le concede una ventaja extraordinaria sobre otros lenguajes ya que lo puede usar cualquier desarrollador sin importar el SO que utilice el mismo de ahí que Java en la actualidad esté entre los lenguajes más utilizados a nivel mundial, aunque el mismo requiere de un hardware para poder correr.

Selección del Lenguaje de Programación.

C# es un lenguaje de programación que brinda múltiples ventajas para el desarrollo de aplicaciones de escritorio, es orientado a objetos completamente lo que posibilita crear aplicaciones en tiempo real y muy cómodos a la hora de escribir código. No necesita los archivos cabecera como los “punto h” de C++, no importa el orden en que fueron definidas las clases, incluye una serie de librerías que facilitan el trabajo de

los programadores, ganando mucho tiempo en la parte de la codificación, tiene además un recolector de basura automático lo que implica que el programador no debe preocuparse por eliminar objetos cuando acabe su vida útil, además de la compatibilidad que tiene con diversos sistemas gestores de Bases de Datos como Postgres SQL y SQL Server.

1.9 Sistema Gestor de Base de Datos (SGBD).

Conjunto de datos interrelacionados+ conjunto de programas para acceder a esos datos. Diseñados para gestionar grandes bloques de datos e información. Deben mantener seguridad en la información almacenada (ante caídas y accesos no autorizados).

1.9.1 PostgresSql

PostgresSql es una base de datos relacional con código fuente disponible libremente. Es el motor de bases de datos de código abierto más potente del momento y en sus últimas versiones empieza a no tener que envidiarle nada a otras bases de datos comerciales. Entre las características más importantes de PostgresSql están que soporta: **(12)**

- Llaves ajenas (foreign keys).
- Joins.
- Vistas (views).
- Disparadores (triggers).
- Reglas (Rules).
- Funciones/procedimientos almacenados en numerosos lenguajes de programación,
- Numerosos tipos de datos, posibilidades de definir nuevos tipos
- Soporta el almacenamiento de objetos binarios grandes (gráficos, videos, sonido, ...)
- Herencia de tablas.

- PITR - point in time recovery
- Tablespaces
- Replicación asíncrona

1.10 Herramientas de diseño.

1.10.1 Rational Rose Enterprise Edition.

Rational Rose Enterprise es una herramienta de diseño que soporta la generación de código a partir de modelos en Ada, ANSI C++, C++, CORBA, Java/J2EE, Visual C++ y Visual Basic. Como todos los demás productos Rational Rose, proporciona un lenguaje común de modelado para el equipo, que facilita la creación de software de calidad más rápidamente. **(7)**

Características adicionales incluidas:

- Característica de control por separado de componentes modelo que permite una administración más granular y el uso de modelos
- Soporte de ingeniería Forward y/o reversa.
- La generación de código Ada, ANSI C ++, C++, CORBA, Java y Visual Basic, con capacidad de sincronización modelo- código configurables **(12)**
- Soporte Enterprise Java Beans™ 2.0
- Capacidad de análisis de calidad de código **(12)**
- El Add-In para modelado Web provee visualización, modelado y las herramientas para desarrollar aplicaciones de Web **(12)**
- **Modelado UML** para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos
- Capacidad de crear definiciones de tipo de documento XML (DTD) para el uso en la aplicación **(12)**
- Integración con otras herramientas de desarrollo de Rational **(12)**

Rational Rose fue diseñado para trabajar sobre las siguientes plataformas y sistemas operativos: Windows 2000, Windows NT y Windows XP. En el mismo se pueden construir diferentes diagramas como los de clases, componentes, despliegue, secuencia, colaboración, caso de uso y modelo físico de datos. Este se puede integrar además con diferentes IDE tales como Borland JBuilder versiones 7.0 a 10.0, Sun Forte for Java Community y Enterprise Edition 3.0, Microsoft Visual Studio 6, Microsoft Visual Studio 2003 y, Microsoft Visual Studio 2005, entre otros.

1.10.2 Visual Paradigm.

Visual Paradigm o Paradigma Visual es una herramienta de diseño, que facilita entre otras funcionalidades la visualización de UML en su última notación 2.1, además en el mismo se pueden modelar 13 tipos de diagramas diferentes. VP-UML le permite diseñar nuevas notaciones o incorporar formas personalizadas mediante el uso de símbolos o icono de imagen de importación. A continuación se mencionan algunos de los diagramas que se pueden modelar con visual paradigm: **(8)**

1. Diagrama de clases.
2. Diagrama caso de uso.
3. Diagrama de secuencia.
4. Diagrama de la comunicación.
5. Diagrama de máquina de estados.
6. Diagrama de actividad.
7. Diagrama de componentes.
8. Diagrama de despliegue.
9. Diagrama de paquetes.
10. Objeto diagrama.
11. Diagrama de estructura compuesta.
12. Calendario diagrama.
13. Diagrama de visión general interacción.
14. Caso de uso corriente de los acontecimientos lista.
15. Generar diagramas de flujo de secuencia de los acontecimientos listas.
16. Modelo de negocio caso de uso de apoyo.

VP es una herramienta de modelado fácil y muy potente que permite la creación de diagramas en un tiempo bastante corto a los desarrolladores. Además de ser una herramienta multiplataforma.

1.11 Selección de la herramienta de diseño

VP es una herramienta de diseño de las más potentes que existen, es una herramienta multiplataforma, que tiene versiones libre, en la que se pueden construir diferentes tipos de diagramas y además permite la generación de código a partir de un diseño determinado, lo que le confiere un alto grado de usabilidad.

1.12 Lenguaje de Modelado

El lenguaje de modelado que se utiliza para la confección de la herramienta es UML 2.0 debido a las múltiples ventajas que presenta el mismo.

UML es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Ayuda a la captura de decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. **(6)**

UML está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. Incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. **(6)**

1.13 Métodos de Prueba de Caja Negra.

1.13.1 Métodos de prueba basados en grafos (1)

En este método se debe entender los objetos (objetos de datos, objetos de programa tales como módulos o colecciones de sentencias del lenguaje de programación) que se modelan en el software y las relaciones que conectan a estos objetos. Una vez que se ha llevado a cabo esto, el siguiente paso es definir una serie de pruebas que verifiquen que todos los objetos tienen entre ellos las relaciones esperadas. En este método:

-Se crea un grafo de objetos importantes y sus relaciones.

-Se diseña una serie de pruebas que cubran el grafo de manera que se ejerciten todos los objetos y sus relaciones para descubrir errores.

Las pruebas basadas en grafos empiezan con la definición de todos los nodos y pesos de nodos. O sea, se identifican los objetos y los atributos. El modelo de datos puede usarse como punto de partida, pero es importante tener en cuenta que muchos nodos pueden ser objetos de programa (no representados explícitamente en el modelo de datos). Para proporcionar una indicación de los puntos de inicio y final del grafo, es útil definir unos nodos de entrada y salida.

Una vez que se han identificado los nodos, se deberían establecer los enlaces y los pesos de enlaces. En general, conviene nombrar los enlaces, aunque los enlaces que representan el flujo de control entre los objetos de programa no es necesario nombrarlos. En muchos casos, el modelo de grafo puede tener bucles (por ejemplo, un camino a través del grafo en el que se encuentra uno o más nodos más de una vez). El grafo ayudara a identificar aquellos bucles que hay que probar. Cada relación es estudiada separadamente, de manera que se puedan obtener casos de prueba. La *transitividad* de relaciones secuenciales es estudiada para determinar como se propaga el impacto de las relaciones a través de objetos definidos en el grafo. La transitividad puede ilustrarse considerando tres objetos X, Y y Z. Se consideran las siguientes relaciones: *X es necesaria para calcular Y, Y es necesaria para calcular Z*(1)

Por tanto, se ha establecido una relación transitiva entre X y Z:

X es necesaria para calcular Z

Basándose en esta relación transitiva, las pruebas para encontrar errores en el cálculo de Z deben considerar una variedad de valores para X e Y. **(1)**

La *simetría* de una relación (enlace de grafo) es también una importante directriz para diseñar casos de prueba. Si un enlace es bidireccional (simétrico), es importante probar esta característica. La característica UNDO [BE1951 (deshacer) en muchas aplicaciones para ordenadores personales implementa una limitada simetría. Es decir, UNDO permite deshacer una acción después de haberse completado. Esto deberá probarse minuciosamente y todas las excepciones **(1)**

1.13.2 Partición equivalente (1):

La partición equivalente es un método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada.

Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada. Típicamente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

Las clases de equivalencia se pueden definir de acuerdo con las siguientes directrices:

-Si una condición de entrada especifica un *rango,* se define una clase de equivalencia valida y dos no validas.

-Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos no válidas.

-Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una no válida.

-Si una condición de entrada es lógica, se define una clase de equivalencia válida y una no válida.

Análisis de valores límite (1):

El análisis de valores límite (AVL) es una técnica de diseño de casos de prueba que complementa a la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL lleva a la elección de casos de prueba en los “extremos “de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida.

Las directrices de (AVL) son similares en muchos aspectos a las que proporciona la partición equivalente, por ejemplo:

Si una condición de entrada especifica un rango delimitado por los valores a y b, se deben diseñar casos de prueba para los valores a y b, y para los valores justo por debajo y justo por encima de *a* y b, respectivamente.

Si una condición de entrada especifica un número de valores, se deben desarrollar casos de prueba que ejerciten los valores máximo y mínimo. También se deben probar los valores justo por encima y justo por debajo del máximo y del mínimo.

Aplicar las directrices 1 y 2 a las condiciones de salida. Por ejemplo, si se supone que se requiere una tabla de temperatura / presión como salida de un programa de análisis de ingeniería. Se deben diseñar casos de prueba que creen un informe de salida que produzca el máximo (y el mínimo) número permitido de entradas en la tabla.

Si las estructuras de datos internas tienen límites preestablecidos (por ejemplo, una matriz que tenga un límite definido de 100 entradas), hay que asegurarse de diseñar un caso de prueba que ejercite la estructura de datos en sus límites. La mayoría de los ingenieros del software llevan a cabo de forma intuitiva alguna forma de AVL. Aplicando las directrices que se acaban de exponer, la prueba de límites será más completa y, por tanto, tendrá una mayor probabilidad de detectar errores.

Pruebas de comparación (1)

Cuando se desarrolla un software en el cual la fiabilidad es absolutamente crítica en ese tipo de aplicaciones se utiliza hardware y software redundante para minimizar la posibilidad de error. Cuando se desarrolla software redundante varios equipos de ingeniería del software desarrollan versiones independientes de una aplicación usando las mismas especificaciones. Cuando ocurre esto se deben probar todas las versiones con los mismos juegos de datos para comprobar si las salidas son las mismas. Luego se ejecutan las versiones en paralelo para hacer una comparación en tiempo real de los resultados para garantizar la consistencia del software

Con las lecciones aprendidas de las aplicaciones redundantes, los investigadores sugieren que para aplicaciones críticas se deben desarrollar versiones independientes, aunque al final solo se valla a distribuir una versión de este software.

Estas versiones independientes son las bases del este tipo de prueba de caja negra, que plantea hacerles pruebas a las diferentes versiones realizadas del software, a partir del diseño de casos de prueba realizados con otra técnica como partición equivalente por ejemplo. Si las salidas de todas las versiones es idéntica se asume que todas las implementaciones son correctas en caso contrario se revisan todas las versiones a ver cuál es la que tiene problemas.

Prueba de la tabla ortogonal: (1)

Hay aplicaciones donde el número de parámetros de entrada es pequeño y los valores de cada uno de los parámetros están claramente delimitados. Cuando estos números son muy pequeños (por ejemplo, 3 parámetros de entrada tomando 3 valores diferentes), es posible considerar cada permutación de entrada y comprobar exhaustivamente el proceso del dominio de entrada. En cualquier caso, cuando el número de valores de entrada crece y el número de valores diferentes para cada elemento de dato se incrementa, la prueba exhaustiva se hace impracticable.

La prueba de la tabla ortogonal puede aplicarse a problemas en que el dominio de entrada es relativamente pequeño pero demasiado grande para posibilitar pruebas exhaustivas. El método de prueba de la tabla ortogonal es particularmente útil al encontrar errores asociados con fallos localizados -una categoría de error asociada con defectos de la lógica dentro de un componente software.

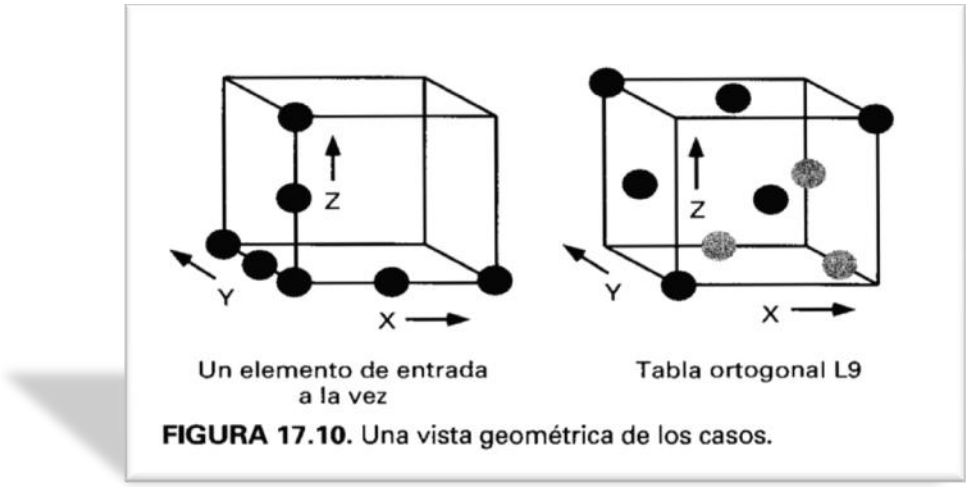


Figura 2: Tabla ortogonal de casos de prueba

Cuando se realiza la prueba de la tabla ortogonal, se crea una tabla ortogonal de casos de prueba. La tabla ortogonal tiene una propiedad de equilibrio. Es decir, los casos de prueba representados por punto negros en la figura están uniformemente dispersos en el dominio de prueba. El alcance de la prueba por todo el dominio de entrada es más completo.

La prueba de la tabla ortogonal permite proporcionar una buena cobertura de prueba con bastantes menos casos de prueba que en la estrategia exhaustiva.

Selección del tipo de Prueba de Caja Negra por el cual se regirá la implementación de la Herramienta para la automatización del proceso de pruebas de caja negra en los proyectos productivos de la facultad 9.

Luego de haber realizado un estudio de los diferentes métodos de prueba de caja negra teniendo en cuenta sus principales características, ventaja y desventajas, se llega a la conclusión de que el método por el cual se regirá la implementación de la herramienta para la automatización del proceso de pruebas de caja negra en los proyectos productivos de la facultad 9, es el método de la *partición equivalente*, debido a que este método propone para el diseño de los casos de prueba dividir los datos de entrada de un programa en clases de datos, con el objetivos de detectar clases de errores y no errores específicos, lo que trae como consecuencia que el número de casos de prueba que hay que diseñar es mucho menor. Esta técnica se basa en la creación de clases de equivalencia, que son un conjunto de estados válidos y

no válidos que el software debe ser capaz de aceptar o no en dependencia de la consistencia del mismo. La principal ventaja de esta técnica a parte de su eficacia en la detección de errores, es el tiempo en que se pueden detectar estas clases de errores por parte de los probadores.

Conclusiones Parciales

En este capítulo se trataron los diferentes conceptos asociados al dominio del problema y conceptos generales que se manejan a lo largo de la investigación. También se exponen las principales herramientas que serán utilizadas en el desarrollo de la aplicación.

CAPÍTULO 2: MODELACION DE LA HERRAMIENTA

2.1 Introducción

En este capítulo se realizará la modelación de la herramienta para la automatización del proceso de caja negra en los proyectos productivos de la facultad 9, se expondrán además los principales diagramas que modelan la herramienta entre los que se encuentra el modelo de dominio, diagramas de casos de uso del sistema, diagramas de clases del análisis y del diseño, así como la descripción detallada de cada uno de los casos de uso que componen la misma y se muestran además los requisitos funcionales que componen la aplicación.

2.2 Modelo de Dominio

Debido a la relativa sencillez de la aplicación y el entorno en donde está enmarcado el sistema se decidió que no era necesario realizar un modelo de negocio, por lo que se procedió a la confección de un modelo de dominio para capturar los principales elementos involucrados en la problemática que la aplicación resuelve.

En la figura 3 se muestra el modelo de dominio realizado teniendo en cuenta el problema planteado anteriormente.

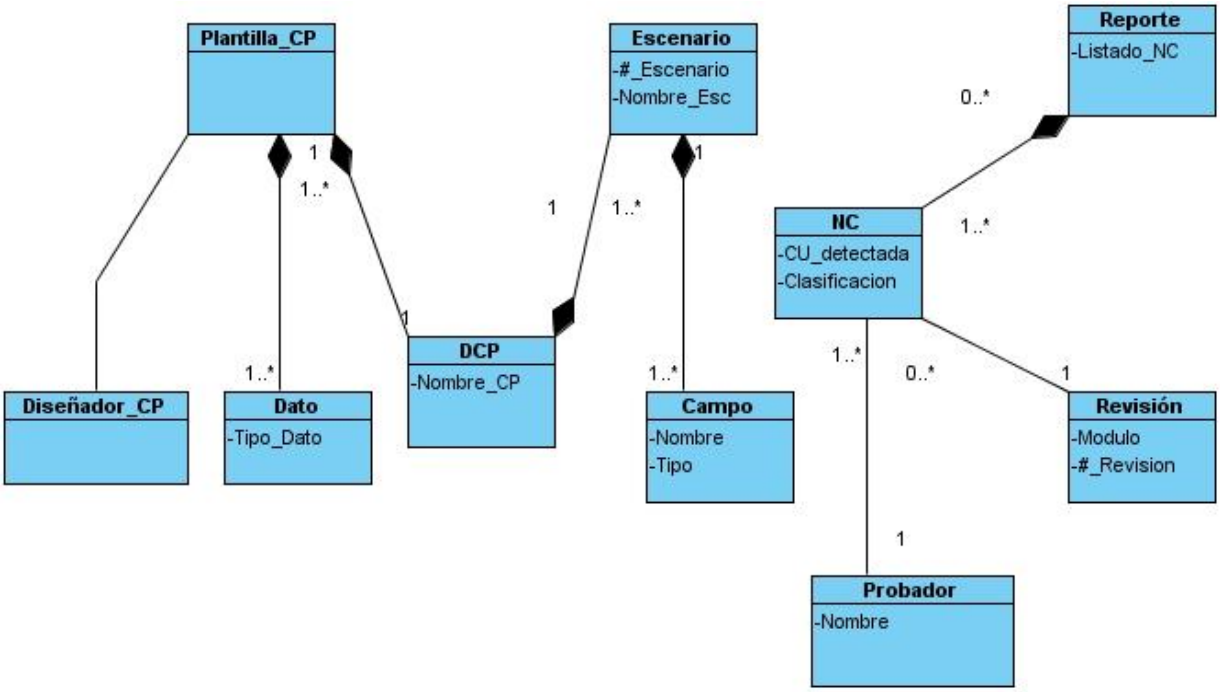


Figura 3: Modelo de Dominio del Problema.

2.3 Requerimientos

El flujo de trabajo de requerimiento constituye uno de los flujos de trabajo más importantes debido a que en este es donde se define que es lo que tiene que hacer exactamente el sistema que se pretende construir, de esta manera se establece un contrato entre los usuarios y los desarrolladores, los usuarios deben comprender lo requisitos definidos por los desarrolladores, y los desarrolladores son los máximos responsables de hacerlos cumplir. Estos requerimientos se clasifican en funcionales y no funcionales.

2.4 Requerimientos Funcionales.

- ✓ Permitir la gestión de casos de prueba (inserción, modificación, eliminación).
- ✓ Generar plantilla de diseño de caso de prueba a partir de los campos y escenarios que el diseñador defina.
- ✓ Generar un conjunto de juegos de datos de acuerdo al dominio definido para cada campo.

- ✓ Permitir registrar las no conformidades detectadas en la ejecución de las pruebas usando los casos de prueba generados. (registrar las diferentes revisiones para un caso de prueba).

Brindar reportes:

- Cantidad de no conformidades para una revisión de un producto.
- Productos con mayor cantidad de no conformidades.
- Listar las no conformidades de un producto para una revisión.
- Clasificar las no conformidades (filtrar por clasificaciones)

2.5 Requerimientos No Funcionales

Usabilidad:

Para hacer uso del sistema es necesario poseer conocimientos básicos de computación y sobre las aplicaciones de escritorio en sentido general.

Rendimiento:

El sistema debe ser capaz de responder de una forma inmediata a las peticiones realizadas por los usuarios que interactúan con la misma. Tener una base de datos que esté normalizada correctamente para garantizar la integridad de los datos.

Software:

La PC donde se vaya a utilizar la herramienta debe tener instalado el framework de .NET para garantizar que se ejecute correctamente.

2.6 Actores del sistema.

Actor	Descripción
Diseñador de Caso de Prueba	Es el encargado de diseñar los casos de pruebas que serán utilizados a la hora de probar un caso de uso. Además realiza la gestión de casos de pruebas (inserción, modificación y eliminación) de casos de prueba.
Probador	Es el encargado de realizar las pruebas, así como de registrar las diferentes *no conformidades* detectadas y de clasificar las mismas.
Jefe de Proyecto	Es el que tiene permisos para realizar los reportes, que le permitan determinar en que estado está uno o varios proyectos.

2.6.1 Diagrama de Casos de Uso del Sistema.

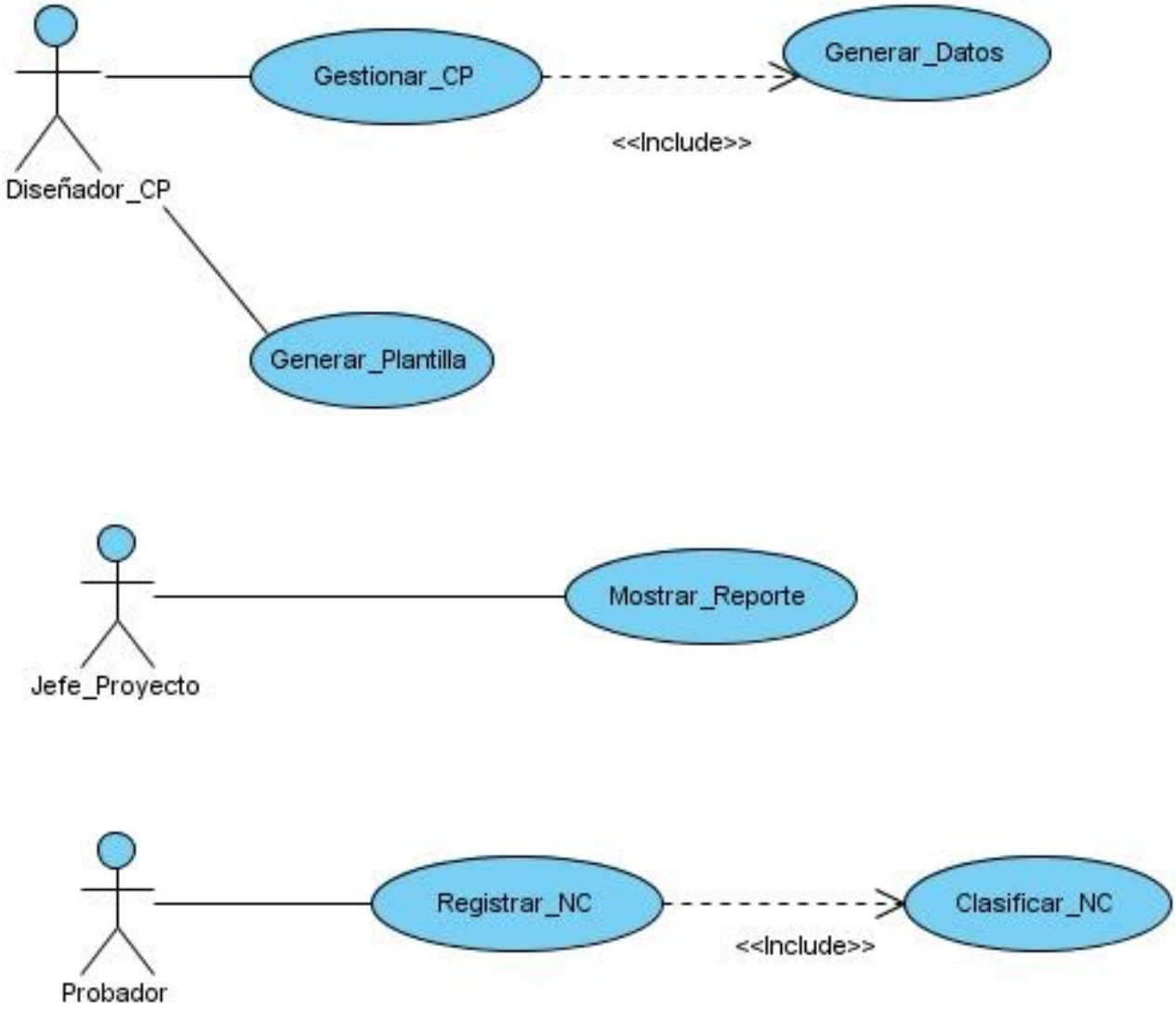


Figura 4: Diagrama de Casos de Uso del Sistema

2.7 Descripción de Casos de Uso.

CU_Gestionar_CP.

Caso de Uso:	Gestionar_CP.
Actor:	Diseñador Caso de Prueba.
Resumen:	El CU se inicia cuando el Diseñador de caso de prueba selecciona la opción de Gestionar caso de prueba. A partir de ahí el Diseñador de caso de prueba tiene la opción de insertar, modificar o eliminar un caso de prueba
Precondiciones:	El Diseñador de caso de prueba debe seleccionar la opción gestionar caso de prueba.
Referencias	RF_1
Prioridad	Alta

Flujo normal de eventos Sección Gestionar_CP.	
Acción del Actor	**Respuesta del Sistema**
1. Se inicia cuando el diseñador de caso de prueba selecciona la opción gestionar caso de prueba.	2. El sistema brinda la siguientes opciones: a) Insertar caso de Prueba. b) Modificar caso de Prueba. c) Eliminar caso de prueba.
3. El usuario procede a seleccionar una de las opciones siguientes:	

a) Insertar caso de Prueba. b) Modificar caso de Prueba. c) Eliminar caso de prueba.	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
2.1 El usuario selecciona la opción Insertar_CP y no selecciona la cantidad de secciones ni de escenario que va a tener el caso de prueba.	2.1.1 El sistema muestra un mensaje indicando que debe seleccionar los mismos.
5.1 El diseñador de caso dejó algún campo vacío.	5.1.1 El sistema muestra un mensaje “Debe llenar el o los campos en blanco”.
Sección “Insertar Caso de Prueba”	
Acción del Actor	**Respuesta del Sistema**
	El sistema muestra un formulario para seleccionar los datos que conformarán el nuevo caso de prueba.
El diseñador de CP introduce los datos para diseñar el caso de prueba.	
El diseñador de CP selecciona la opción guardar datos del caso de prueba.	El sistema muestra un mensaje preguntando si desea generar la planilla de CP.
El diseñador de CP selecciona la opción aceptar.	El sistema genera la planilla con los datos guardados anteriormente.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
En caso de que falle la conexión a la base de	El sistema muestra un mensaje de error

datos.	indicando que falló la conexión.
El diseñador de caso de prueba no llena los campos necesarios para diseñar el caso de prueba.	El sistema muestra un mensaje indicando que debe llenar los campos en blanco.
Flujo Normal de Eventos	
Sección " Modificar Caso de Prueba"	
Acción del Actor	**Respuesta del Sistema**
	El sistema muestra un formulario para seleccionar el CP que se desea modificar.
El diseñador de CP selecciona el CP que desea modificar.	El sistema muestra los datos del CP seleccionado para ser modificados.
El diseñador de CP modificar los datos y los vuelve a guardar.	El sistema guarda los datos modificados.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
En caso de que cuando el diseñador de CP valla a guardar los datos modificados falle la conexión a la base de datos.	El sistema muestra un mensaje de error indicando que falló la conexión.
Sección " Eliminar Caso de Prueba"	
Acción del Actor	**Respuesta del Sistema**
	El sistema muestra un formulario para seleccionar el CP que se desea eliminar.
El diseñador de CP selecciona el CP que desea eliminar.	El sistema elimina el caso de prueba seleccionado por el diseñador de CP.
	El sistema muestra un mensaje indicando que se elimina satisfactoriamente el caso de

	prueba.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
En caso que el diseñador de CP valla a eliminar un caso de prueba y el sistema no lo elimina por falló en la conexión a la BD	El sistema muestra un mensaje de error indicando que falló la conexión.

CU_Registrar_NC.

Caso de Uso:	Registrar_NC.
Actor:	Probador.
Resumen:	El caso de uso comienza cuando el probador selecciona la opción de registrar una no_conformidad (NC) determinada, procede a hacer el registro y luego el sistema la registra satisfactoriamente.
Precondiciones:	El probador debe seleccionar la opción Registrar_NC.
Referencias	RF_3.
Prioridad	Media.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El probador selecciona la opción Registrar_NC.	2. El sistema el sistema muestra una ventana para realizar el registro de la NC.
3. El probador llena los datos de la NC y la inserta en la BD.	4. El sistema se conecta a la BD y registra la NC.

	El sistema muestra un mensaje confirmando que se registró satisfactoriamente la NC.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
3.1 El probador no llena los campos correspondientes a la NC, para ser registrada.	3.1.1 El sistema muestra un mensaje indicando que debe llenar los campos correspondientes.
4.1 En caso de que el sistema no se pueda conectar a la BD para registra la NC.	4.1.1 El sistema muestra un mensaje indicando que ha fallado la conexión a la BD.

CU_Clasificar_NC.

Caso de Uso:	Clasificar_NC.
Actor:	Probador.
Resumen:	El caso de uso comienza cuando el probador selecciona la opción de clasificar no_conformidad (NC), procede a hacer el registro y luego el sistema la registra satisfactoriamente.
Precondiciones:	El probador debe seleccionar la opción Registrar_NC.
Referencias	RF_3.
Prioridad	Media.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

	1. El sistema el sistema muestra una tres radiobutton para realizar la clasificación de la NC.
2. El probador pincha una de estas opciones.	3. El sistema se registra esta clasificación.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
2.1 El probador no selecciona ningún criterio de clasificación.	2.1.1 El sistema muestra un mensaje indicando que debe seleccionar un criterio de clasificación.
3.1 el probador vuelve a la opción 1.	

CU_Gestionar_Reporte.

Caso de Uso:	Gestionar_Reporte.
Actor:	Jefe de Proyecto.
Resumen:	El caso de uso comienza cuando el jefe de proyecto selecciona la opción “Gestionar Reportes”. A partir de ahí el jefe de proyecto tiene la opciones de “mostrar_cant_NC_revisión” o “mostrar_prod_mayor_cant_NC”.
Precondiciones:	El Jefe de Proyecto debe seleccionar la opción “Gestionar Reportes”
Referencias	RF_4.
Prioridad	Alta.
Flujo Normal de Eventos	

Acción del Actor	Respuesta del Sistema
1. El jefe de proyecto selecciona la opción "Gestionar_Reporte".	2. El sistema muestra brinda las siguientes opciones: 1. Mostrar_Prod_Mayor_Cant_NC. 2. Mostrar_cant_NC_revisión.
3. El jefe de proyecto selecciona una de las siguientes opciones. a) Mostrar_Prod_Mayor_Cant_NC. b) Mostrar_cant_NC_revisión.	

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
1. En caso de que el sistema no se puede conectar a la BD para mostrar el reporte.	2. El sistema muestra un mensaje indicando el error.
3. El jefe de proyecto acepta el mensaje.	4. El sistema cierra esta ventana, permitiendo las opciones anteriores.

Flujo Normal de Eventos	
Sección "Mostrar_cant_NC_Revision"	
Acción del Actor	**Respuesta del Sistema**

El jefe de proyecto selecciona la opción Mostrar Cant_NC_Revisión.	El sistema muestra un formulario para que el jefe de proyecto seleccione la revisión y el proyecto del los cuales se quiere conocer el reporte.
El jefe de proyecto selecciona el proyecto y de este la revisión a la cual se le quiere conocer la cantidad de NC.	El sistema muestra los datos solicitados por el jefe de proyecto.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
El jefe de proyecto no selecciona el proyecto del cual se quiere conocer el reporte.	El sistema muestra un mensaje indicando que debe seleccionar un proyecto.
El jefe de proyecto no selecciona la revisión de la que se quiere conocer el reporte.	El sistema muestra un mensaje indicando que debe seleccionar una revisión.
En caso de que falle la conexión a la BD.	El sistema muestra un mensaje de error "Fallo en la conexión".

CU_Generar_Planilla.

Caso de Uso:	Generar Planilla.
Actor:	Probador.
Resumen:	El caso de uso comienza cuando el probador selecciona la opción "Generar Planilla", luego el sistema genera la planilla en un documento Word con los datos definidos por el probador.
Precondiciones:	El Jefe de Proyecto debe seleccionar la opción "Gestionar Reportes"
Referencias	RF_4.
Prioridad	Alta.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

El CU se inicia cuando el probador selecciona la opción generar planilla.	El sistema genera la planilla de caso de prueba en un documento Word.

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
El probador no selecciona los datos con que será conformada la planilla de caso de prueba.	El sistema muestra un mensaje diciendo que debe llenar los campos correspondientes para poder ser generada la planilla.

2.8 Análisis.

El modelo de análisis contiene las clases del análisis y todos aquellos artefactos asociados. El modelo de análisis puede ser un artefacto temporal. Tal es el caso donde el mismo evoluciona hacia el modelo de diseño o cuando continúa viviendo a través del proyecto, o quizás más allá, sirviendo como un modelo conceptual general del Sistema **(6)**

Encontrar un conjunto candidato de clases del análisis es el primer paso en la transformación del sistema desde el mero estado de comportamiento requerido a una descripción de cómo el sistema trabajará. En este esfuerzo, las clases del análisis son usadas para representar los roles de los elementos del modelo, los cuales proveerán el comportamiento necesario para satisfacer los requerimientos funcionales especificados por los casos de uso y los no funcionales especificados por los requerimientos adicionales **(6)**

A continuación mostramos los diagramas de clases del análisis correspondiente al flujo de trabajo de análisis y diseño.

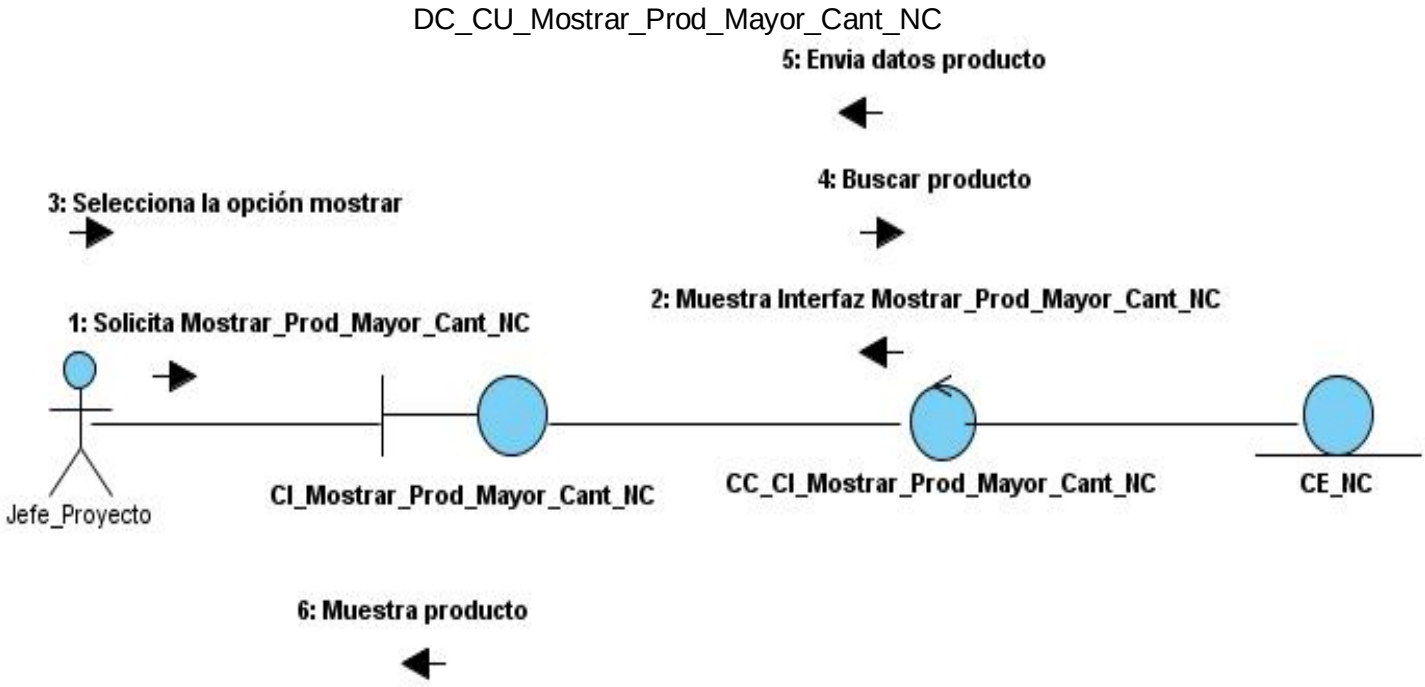


Figura 5: DC_CU_Mostrar_Prod_Mayor_Cant_NC

DC_CU_Eliminar_CP.

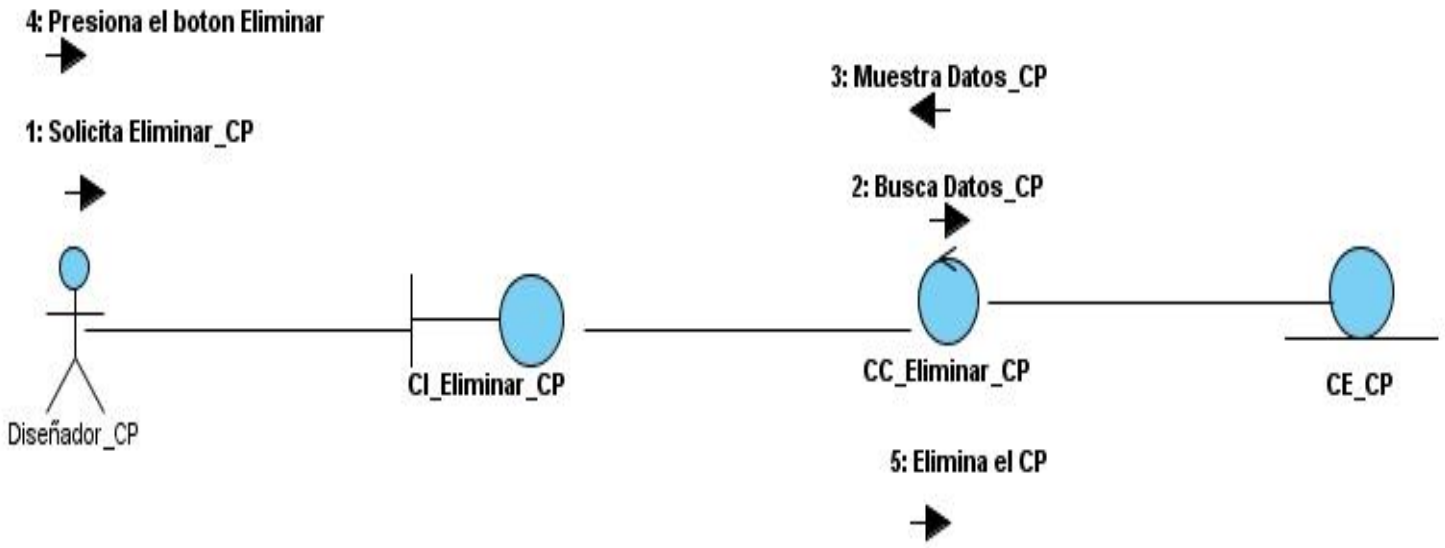


Figura 6: DC_CU_Eliminar_CP.

DC_CU_ Insertar_NC.

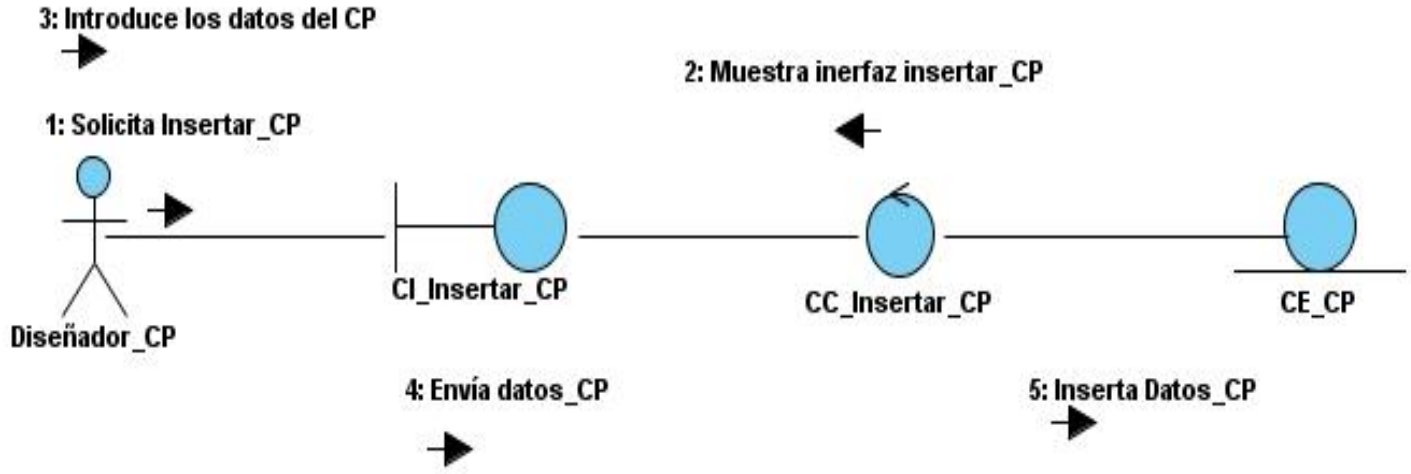


Figura 7: DC_CU_ Insertar_NC.

DC_CU_ Modificar_CP.

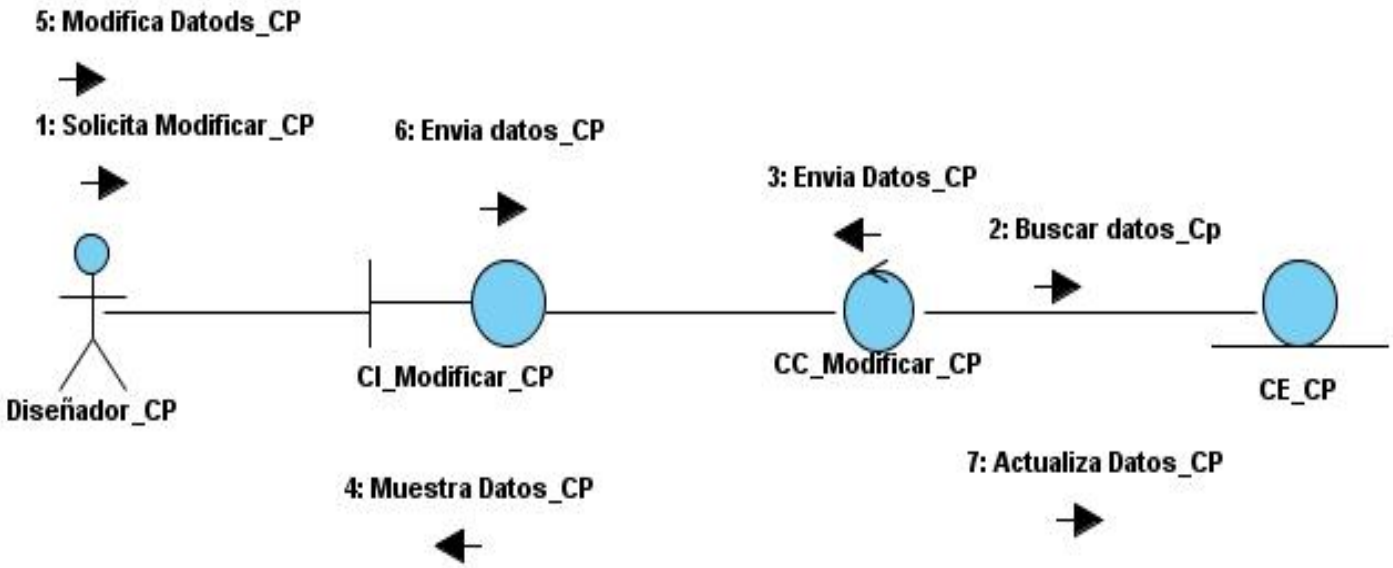


Figura 8: DC_CU_ Modificar_CP.

DC_CU_Registrar_NC.

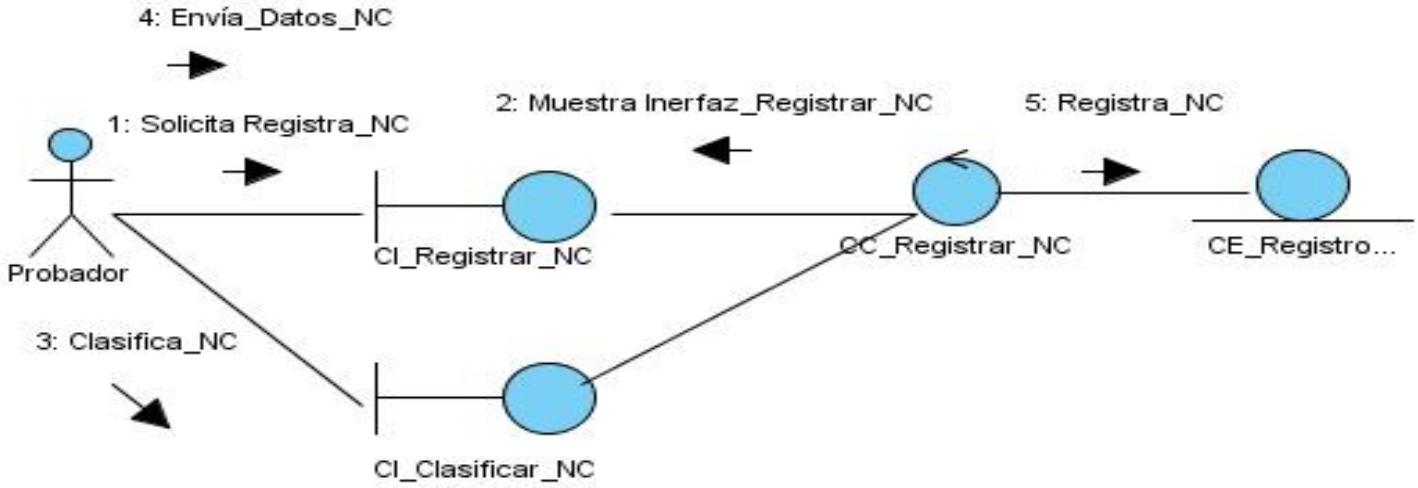


Figura 9: DC_CU_Registrar_NC.

2.9 Diseño.

En el diseño se modela el sistema y se encuentra su forma (incluida la arquitectura) para que soporte todos los requisitos incluyendo los requisitos no funcionales y otras restricciones que se le suponen. Una entrada esencial en el diseño es el resultado del análisis, esto proporciona una comprensión detallada de los requisitos e impone una estructura del sistema que el equipo de desarrollo de esforzarse por conservar lo más fielmente posible. **(6)**

Los principales propósitos del diseño son:

- Comprender en profundidad los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos etc.
- Crear una entrada apropiada y punto de partida para las actividades de implementación capturando los requisitos o subsistemas individuales, interfaces y clases.

- Ser capaces de descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia.

Patrones:

Los patrones son una herramienta de resolución de problemas en la ingeniería de software aunque pueden ser aplicados a otros ámbitos de la informática y la ciencia en general. El surgimiento de los patrones está basado en la experiencia de varios desarrolladores que tratan de resolver algún problema en específico. Estos capturan las experiencias positivas para promover buenas prácticas.

> Un patrón es una unidad de información nombrada, instructiva e intuitiva que captura la esencia de una familia exitosa de soluciones probadas a un problema recurrente dentro de un cierto contexto. **(9)**
>
> El objetivo de los patrones es crear un lenguaje común a una comunidad de desarrolladores para comunicar experiencia sobre los problemas y sus soluciones.

Pueden referirse a distintos niveles de abstracción, desde un proceso de desarrollo hasta la utilización eficiente de un lenguaje de programación. **(9)**

Un buen patrón debería:

-Solucionar un problema

-Ser un concepto probado

-La solución no es obvia

-Describe participantes y relaciones entre ellos

-Tiene un componente humano alto: estética y utilidad

Clasificación de los patrones:

Los patrones utilizados en el diseño de la herramienta son los siguientes:

Patrones de CU.

Estos patrones son utilizados a la hora de modelar los diferentes casos de uso que son identificados o que conformaran la aplicación.

La utilización de patrones de caso de uso queda evidenciada en el caso de uso Gestionar_CP

Patrones de Asignación de responsabilidades (GRASP).

Es un acrónimo que significa General Responsibility Asignment Software Patterns (patrones generales de software para asignar responsabilidades) El nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a objetos. **(10)**

Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades.

Patrón Experto: Es el encargado de asignarle la responsabilidad a la clase que contenga la información necesaria para realizar un acción determinada. Es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen. **(10)**

Entre los principales beneficios de este patrón es que conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases "sencillas" y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión.

Patrón Creador: Este patrón es muy sencillo y sin duda toda persona que ha trabajado un poco con la POO lo ha utilizado. Lo que define este patrón es que una instancia de un objeto la tiene que crear el

objeto que tiene la información para ello. ¿Qué significa esto?, pues que si un objeto A utiliza específicamente otro B, o si B forma parte de A, o si A almacena o contiene B, o si simplemente Atiene la información necesaria para crear B, entonces A es el perfecto creador de B. **(11)**

Principales beneficios de este patrón.

Se brinda soporte a un **bajo acoplamiento**, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase ***creada*** tiende a ser visible a la clase ***creador,*** debido a las asociaciones actuales permite elegirla como el parámetro adecuado.

Patrón Bajo Acoplamiento: El Bajo Acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad. No puede considerarse en forma independiente de otros patrones como Experto o Alta Cohesión, sino que más bien ha de incluirse como uno de los principios del diseño que influyen en la decisión de asignar responsabilidades. **(11)**

Entre los Principales beneficios que brinda este patón se encuentran:

-No se afectan por cambios de otros componentes.

- Fáciles de entender por separado.

- Fáciles de reutilizar.

Patrón Alta Cohesión: Una clase con mucha cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y 1os mejoramientos. La ventaja que significa una gran funcionalidad también soporta un aumento de la capacidad de reutilización. En otras palabras la alta cohesión pretende que una clase que esté relacionada con otras clases no realice un gran número de funcionalidades.

Entre los principales beneficios de este patrón se encuentran:

- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo se genera un bajo acoplamiento.
- La ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

Patrón Controlador: Este patrón ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan. Un controlador de casos de uso constituye una buena alternativa cuando hay muchos eventos de sistema entre varios procesos: asigna su manejo a clases individuales controlables, además de que ofrece una base para reflexionar sobre el estado del proceso actual. La principal ventaja de este patrón es controlar todos los eventos asociados al caso de uso para el cual fue diseñado. Garantiza que la empresa o los procesos de dominio sean manejados por la capa de los objetos del dominio y no por la de la interfaz.

A continuación se muestra una tabla donde se resumen las principales características de los patrones de asignación de responsabilidades utilizados. (Esta información es del libro UML y Patrones)

Patrón	Descripción
Patrón Experto	¿Quién asumirá la responsabilidad en el caso general? Asignar una responsabilidad al experto en información: la clase que posee la información necesaria para cumplir con la responsabilidad.
Patrón Creador	¿Quién crea? Asignar a la clase B la responsabilidad de crear una instancia de clase A, si se cumple una de las siguientes condiciones: 1. B contiene. A 4. B registra A 2. B agrega A 5. B utiliza A muy de cerca

	3. B tiene los datos de inicialización de A
Patrón Bajo Acoplamiento	Cómo dar soporte a poca dependencia y a una mayor reutilización? Asignar las responsabilidades de modo que se mantenga bajo acoplamiento.
Patrón Alta Cohesión	¿Cómo mantener controlable la complejidad? Asignar las responsabilidades de modo que se mantenga una alta cohesión.

Modelo 3 Capas.

Una arquitectura común de los sistemas de información que abarca una interfaz para el usuario y el almacenamiento persistente de datos se conoce con el nombre de arquitectura de tres capas. He aquí una descripción clásica de las capas verticales (En Arquitectura y patrones es esta información)

1. Presentación: ventanas, reportes, etc.

2. Lógica de aplicaciones: tareas y reglas que rigen el proceso.

3. Almacenamiento: mecanismo de almacenamiento persistente.

La calidad tan especial de la arquitectura de tres capas consiste en aislar la lógica de la aplicación y en convertirla en una capa intermedia bien definida y lógica del software. En la capa de presentación se realiza relativamente poco procesamiento de la aplicación; las ventanas envían a la capa intermedia peticiones de trabajo. Y este se comunica con la capa de almacenamiento del extremo posterior.

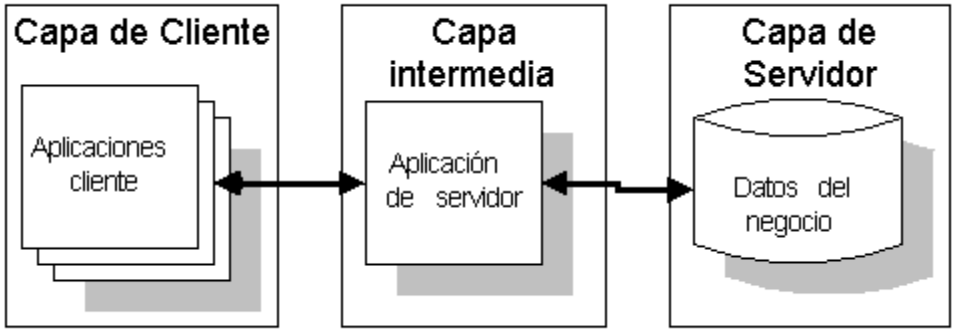


Figura 10: Distribución de las tres capas del modelo.

A continuación se enumeran algunas ventajas de las aplicaciones de 3 capas (del documento de 3 capas):

- Los componentes de la aplicación pueden ser desarrollados en cualquier lenguaje general lo que posibilita que el grupo de desarrolladores no se centre en el uso de un solo lenguaje.
- Los componentes están centralizados lo que posibilita su fácil desarrollo, mantenimiento y uso.
- Los componentes de la aplicación pueden estar esparcidos en múltiples servidores permitiendo una mayor escalabilidad.
- Los problemas de limitación para las conexiones a las bases de datos se minimizan ya que la base de datos solo es vista desde la capa intermedia y no desde todos los clientes. Además que las conexiones y los drivers de las bases de datos no tienen que estar en los clientes.

Los componentes de aplicación de la capa intermedia pueden ser asegurados centralmente usando una infraestructura común. Se pueden conceder o denegar los permisos componente a componente simplificando la administración.

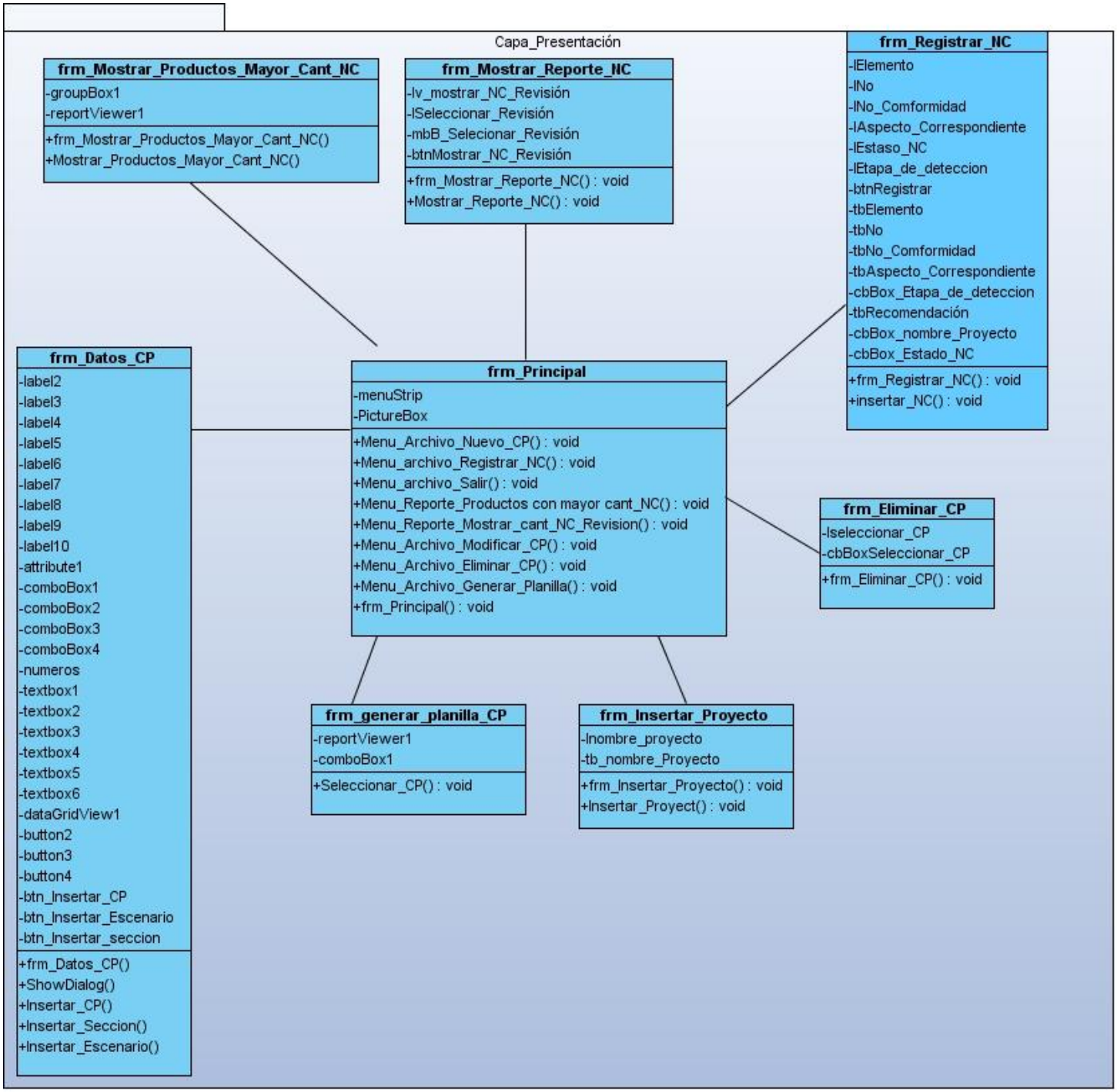


Figura 11: Capa presentación.

La capa de Lógica de Negocio: Es donde se reciben las peticiones del usuario y se envían las respuestas a los mismos. Es además la encargada de comunicar la capa de presentación y la capa de acceso a datos.

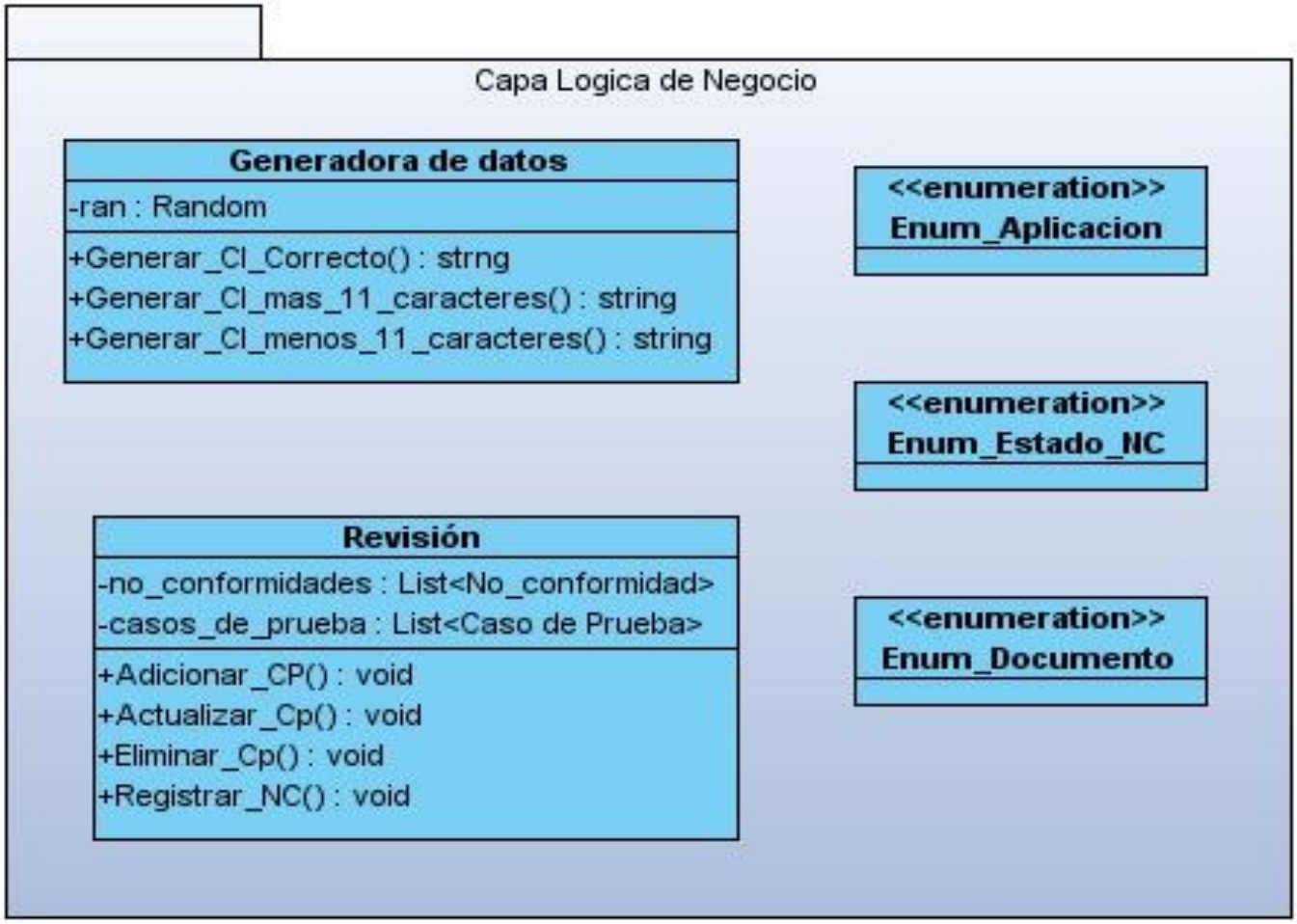


Figura 12: Capa Lógica de Negocio

La capa de acceso a datos: Es la que contiene las clases que interactúan con la BD Son las clases que utilizan los procedimientos almacenados. Recibe solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

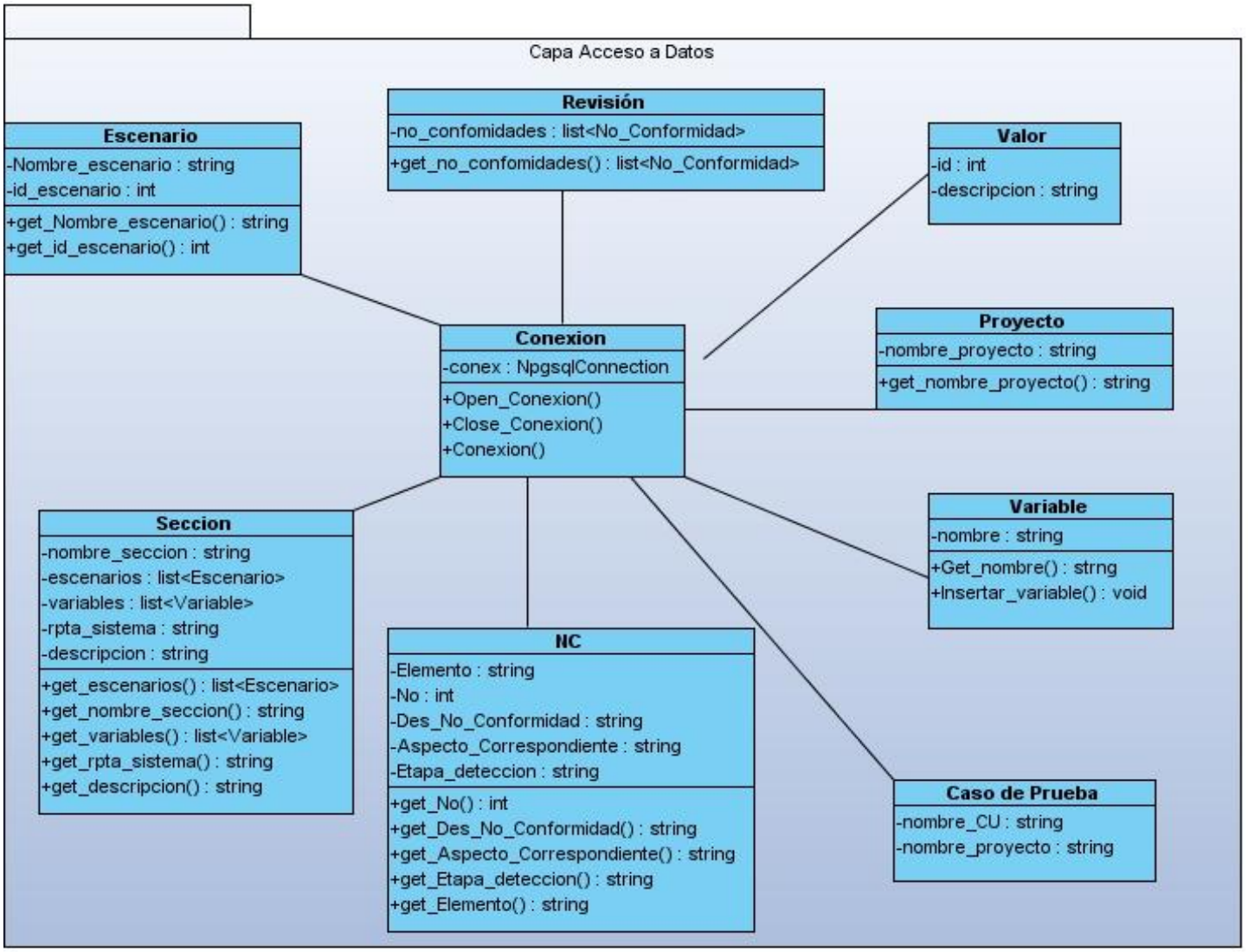


Figura 13: Capa Acceso a Datos

Conclusiones parciales

En el presente capítulo se realizó la modelación de la herramienta, lo que incluye el flujo de trabajo de análisis y diseño, propuesto por la metodología de desarrollo RUP, con sus respectivos artefactos, los

cuales incluyen los diagramas de clases del análisis y del diseño respectivamente. Además se realizó una descripción de los patrones propuestos para realizar el diseño de la herramienta, quedando de esta manera el sistema listo para pasar a la fase de implementación.

CAPITULO 3: IMPLEMENTACION DE LA HERRAMIENTA

3.1 Introducción

El flujo de trabajo de Implementación se hace con el objetivo de definir la organización del código teniendo en cuenta los subsistemas de implementación organizadas por capas, la implementación de los elementos de diseño en términos de ficheros fuentes, binarios, ejecutables y para poder integrar los diferentes componentes de desarrolladores o equipos y generar un ejecutable entregable o producto final. Al finalizar, en la implementación deben quedar plasmados todos los requisitos recogidos en la fase de Requerimientos. Vale destacar que este flujo de trabajo está fuertemente regido por el flujo de Análisis y Diseño. En este capítulo se expondrá el artefacto Modelo de Implementación, que incluye la especificación de cada uno de sus subsistemas definidos.

3.2 Modelo de Implementación

El objetivo del modelo de implementación es identificar las partes físicas de la implementación de tal manera que puedan ser mejor entendidas y manejadas. Este modelo define la manera en que los diferentes módulos son organizados para poder controlar sus versiones, eliminarlos o distribuirlos. Además logra organizar las personas o equipos de trabajo.

3.3 Modelo de Despliegue

Un diagrama de despliegue muestra las relaciones físicas entre los componentes hardware y software en el sistema final, es decir, la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes software (procesos y objetos que se ejecutan en ellos). Es un grafo de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes software, objetos, procesos (caso particular de un objeto). En general un nodo será una unidad de computación de algún tipo, desde un sensor a un mainframe. Las instancias de componentes software pueden estar unidas por relaciones de dependencia, posiblemente a interfaces (ya que un componente puede tener más de una interfaz).

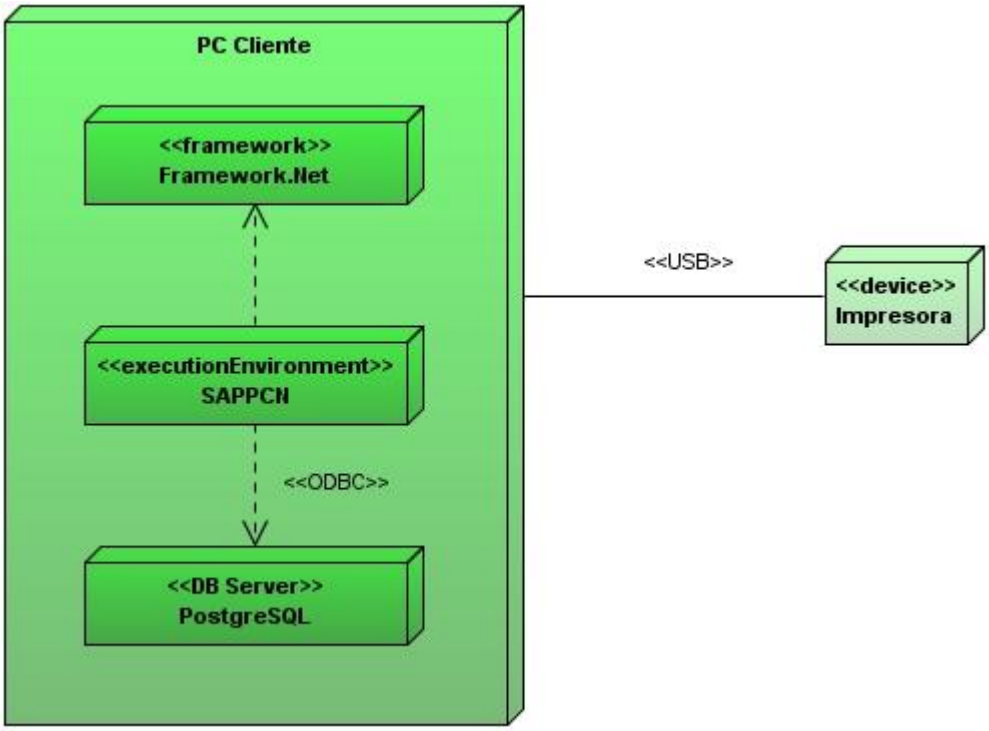


Figura 14 Diagrama de Despliegue

3.4 Diagrama de Componentes

Componente: Parte modular de un sistema, desplegable y reemplazable que encapsula implementación y un conjunto de interfaces y proporciona la realización de los mismos. Un componente software puede ser desde una subrutina de una librería matemática, hasta una clase, una base de datos, un archivo DLL, un JavaBeans, o incluso una aplicación que pueda ser usada por otra aplicación por medio de una interfaz especificada. **(24)**

El Diagrama de Componentes propuesto para la solución del software, los componentes no son más que el empaquetamiento físico de los elementos de un modelo, como el Modelo de Diseño, donde presentan varios estereotipos como ficheros ejecutables, ficheros de código fuente, entre otros.

<<executable>> es un programa que puede ser ejecutado en un nodo.

<<file>> es un fichero que contiene código fuente o datos.

<<library>> es una librería estática o dinámica.

<<table>> es una tabla de base de datos.

<<document>> es un documento.

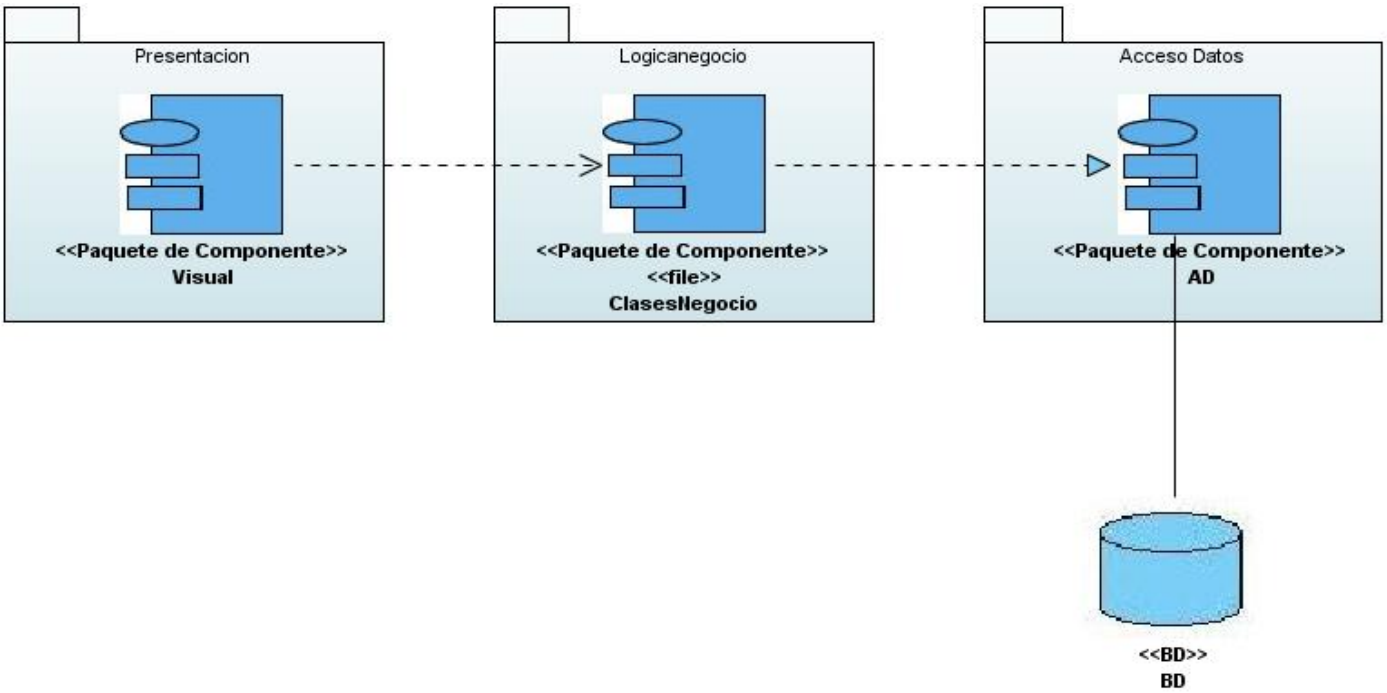


Figura 15 Diagrama de Componentes

Conclusiones parciales

En este capítulo se definieron los principales paquetes de componentes, archivos ejecutables y ficheros binarios que conforman el diagrama de componentes propuesto. Además quedan reflejados los principales subsistemas, así como el diagrama de despliegue correspondiente de la aplicación a desarrollar.

CONCLUSIONES GENERALES

- ✓ En el presente trabajo se describieron las principales características del proceso de pruebas de caja negra, llevadas a cabo para garantizar la fiabilidad de cualquier aplicación de software.

- ✓ Con el uso de la metodología de desarrollo RUP, se desarrollaron 3 de los flujos de trabajo que propone: Modelamiento del Negocio, Análisis y diseño e implementación, lo que permitió llevar a cabo la recopilación de la información necesaria para un mejor entendimiento del negocio por parte del desarrollador.

- ✓ Como resultado general se obtuvo el diseño e implementación de un sistema informático que permite automatizar el proceso de pruebas de caja negra.

RECOMENDACIONES

- ✓ Realizar el proceso de prueba de la herramienta propuesta, con el fin de verificar sus funcionalidades y factibilidad de su uso en los proyectos productivos de la facultad 9.
- ✓ Añadir nuevas funcionalidades a la herramienta implementada con el objetivo de enriquecer su alcance y su utilidad.
- ✓ Seguir profundizando en el estudio de elementos que enriquezcan el conocimiento acerca de la herramienta propuesta.
- ✓ Hacer un estudio y un análisis de alguna plataforma o lenguaje en software libre que soporte la migración de la aplicación propuesta, sin que tenga que emplearse un gran tiempo de ejecución para esta migración.

REFERENCIAS BIBLIOGRAFICAS

1. Pressman, Roger S. ***Ingeniería del Software. Un enfoque práctico.*** **Ciudad Habana : Felix Varela, 2005.**

2. J.M., Juran. ***Juran ya la Planificación para la calidad.*** **s.l. : Ediciones Díaz de Santos, 1990.**

3. Espinoza, Daniel Rolando Valdivia. ***Estándares de Calidad para Pruebas de Software.*** **San Marcos : s.n., 2005.**

4. Centro de Innovación de TI-Puebla. [En línea] [Citado el: 23 de febrero de 2009.] http://www.citip.org.mx/SERVICIOS/SOFTWARE/PRUEBASLAB/Pages/PruebasSoftware.aspx.

5. Real Academia Española. [En línea] [Citado el: 25 de Febrero de 2009.] http://www.rae.es/rae.html.

6. Ivar jacobson, Grady Booch, James Rumbaugh. ***El proceso Unificado de Desarrollo de Software.*** **Madrid : Pearson Education, 2000. ISBN-84-7829-036-2.**

7. GSInnova. [En línea] [Citado el: 24 de febrero de 2009.] http://www.rational.com.ar/herramientas/roseenterprise.html.

8. Visual Paradigm. [En línea] [Citado el: 24 de febrero de 2009.] http://www.visual-paradigm.com/product/vpuml/communityedition.jsp.

9. Alexander, Christopher. ***A Pattern Language: Towns, Buildings, Construction.*** **1977.**

10. Larman, Craig. ***UML yPatrones.*** **PRENTICE HALL, México : s.n., 1999. 970-17-0261-1.**

11. Departamento de Ingenieria y Gestion de software, Universidad de las Ciencias Informáticas. **Patrones de diseño. *Patrones de diseño.*** **La Habana : s.n., 2006.**

12. Olivencia., José Luis Leiva. ***Manejo de bases de datos.*** **Málaga : Departamento Lenguajes y Ciencias de la Computación., 2005.**

13. ***Extreme Programming Expained:Embrace Change.*** K.Beck. **s.l. : Addison Wesley, 2000. ISBN 201-61641-6.**

14. Jeff Ferguson, Brian Patterson,jason beres. ***La Biblia de C#.*** **Madrid : EDICIONES ANAYA MULTIMEDIA, 2003. ISBN: 84-415-1484-4.**

15. Informáticas, Comunidad de PHP-Universidad de Ciencias. **Comunidad PHP-Universidad de las Ciencias Informáticas. [En línea] UCI. [Citado el: 20 de febrero de 2008.] http://php.uci.cu.**

16. Informática Milenium, SA. de CV. **Milenium. [En línea] [Citado el: 01 de 03 de 2009.] http://www.informaticamilenium.com,mx/paginas/español/preguntas/concepto.html.**

17. ***Metodologías Ágiles en el Desarrollo del Software.*** Información), Grupo ISSI(Ingeniería del Software y Sistemas de. **Alicante : s.n., 2003.**

18. G., Francisco Bacerril. ***Java a su Alcance.*** **Cuauhtemoc,México : LitogrAfica Ingrarnex, 1998. ISBN 970-10-1774-9.**

19. Espinoza, Daniel Rolando Valdivia. ***Estándares de Calidad Para Pruebas de Software.*** **San Marcos : s.n., 2005.**

20. ARRIOLA, HÉCTOR ALBERTO HEBER MENDÍA. ***METODOLOGÍAS ÁGILES INCORPORADAS A LAS NECESIDADES DE LAS.*** **Guatemala : s.n., 2006.**

21. El rincón de linux para Hispanohablantes. [En línea] El rincón de Linux. [Citado el: 20 de febrero de 2009.] http://www.linux-es.org/node/536.

22. GSInnova. [En línea] [Citado el: 5 de marzo de 2009.] http://www.rational.com.ar/herramientas/roseenterprise.html..

23. Teleformacion.uci.cu. (2008). ***Sitio de la Asignatura Ingeniería de Software 2. Conferencia4. FT Implementación.*** **Obtenido de http://teleformacion.uci.cu/mod/resource/view.php?id=22199**

Bibliografía.

1. Pressman, Roger S. ***Ingeniería del Software. Un enfoque práctico.*** **Ciudad Habana : Felix Varela, 2005.**
2. Ivar jacobson, Grady Booch, James Rumbaugh. ***El proceso Unificado de Desarrollo de Software.*** **Madrid : Pearson Education, 2000. ISBN-84-7829-036-2.**
3. Larman, Craig. ***UML yPatrones.*** **PRENTICE HALL, México : s.n., 1999. 970-17-0261-1.**

GLOSARIO

Abreviaturas

CU: Caso de uso.

NC: No conformidad.

CP: Caso de prueba.

FDD: Feature Driven Development.

PostgresSql: Gestor de Base de datos.

RUP: Proceso unificado de desarrollo.

UML: Lenguaje unificado de modelado

XP: Extreme Programming o Programación Extrema.

Términos

Calidad de software: Término asociado al nivel de satisfación del cliente con el software.

Metodologías de desarrollo de software: Término usado para nombrar un conjunto de metodologías que guían el desarrollo de un software.

Prueba: Artefacto que se genera durante la fase de desarrollo de un software.

Partición Equivalente: Criterio de prueba de caja negra.

Requerimientos Funcionales: Capacidades o Condiciones que el sistema debe cumplir.

Requerimientos No Funcionales: Cualidad o Propiedad que el sistema debe tener.

Rendimiento: Tipo de requerimiento no funcional

Software: Tipo de requerimiento no funcional

Usabilidad: Tipo de requerimiento no funcional