Universidad de las Ciencias Informáticas

Facultad 5

Título: "Prototipo de Aplicación para la Gestión del Proceso de Pruebas de Software en el Polo de Hardware y Automática."

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

Autores: Anniester Alfredo González González.
Eriberto Vanegas Lago.

Tutores: Ing. Gerandys Hernández Casanova.
Ing. Junior Muñoz Treto.

Mayo de 2009.
"Año del 50 aniversario del triunfo de la Revolución."

"¿Por qué esta magnífica tecnología científica, que ahorra trabajo y nos hace la vida más fácil, nos aporta tan poca felicidad? La repuesta es esta: simplemente porque aún no hemos aprendido a usarla con tino."

(Albert Einstein)

DATOS DE CONTACTOS:

Tutor: Ing. Gerandys Hernández Casanova.
Universidad de las ciencias Informáticas, Habana, Cuba.
Correo electrónico: ghernandez@uci.cu.

Tutor: Ing. Junior Muñoz Treto.
Universidad de las ciencias Informáticas, Habana, Cuba.
Correo electrónico: jmunoz@uci.cu.

AGRADECIMIENTOS:

Agradecemos profundamente a la Revolución Cubana, por habernos dado la oportunidad de instruirnos y hacernos hombres de bien, sin importar procedencia, ni color, ni creencias religiosas.

A nuestro Comandante Fidel que ha confiado siempre en la Juventud y ha luchado tanto por convertir en realidad los sueños de las nuevas generaciones.

Agradecemos a nuestros padres y demás familiares por su apoyo incondicional en todo momento, por habernos inculcado los mejores valores y habernos guiado siempre por el buen camino.

A nuestros tutores por su dedicación y empeño en que el trabajo saliera bien.

Agradecemos a nuestros amigos, a esos que nos han acompañado estos 5 años y a los que no pudieron llegar hasta aquí, pero que nunca nos han dejado de apoyar.

A todas las personas que de una forma u otra nos han brindado su ayuda y han hecho posible la culminación satisfactoria de este trabajo.

DEDICATORIA:

"A mi mamá, mi papá y mi hermanita por ser esas personas en las cuales siempre he confiado y me he apoyado para lograr lo imposible. Por quererme tanto y apoyarme demasiado, por enseñarme que la familia es lo más importante que puede existir en el mundo."

"A esa tía enorme que no se ha separado de mi ni en el más mínimo instante. Como no podría ser ella parte de esta hazaña."

"A esas personas que hoy no se encuentran pero me brindaron tanto apoyo que haría falta otra tesis para mostrarlo."

Anniester

Dedico este trabajo primeramente a mis padres, que siempre han confiado en mí y me han apoyado incondicionalmente en cada uno de mis pasos en la vida. A mi hermana, que es la niña de mis ojos y la quiero con la vida. A mis abuelos, que son mis otros padres, que me criaron e hicieron de mí un hombre del que hoy, sin dudas, pueden estar orgullosos. A todos los demás miembros de mi familia que de una forma u otra siempre se han preocupado por mi futuro.

No podría dejar de mencionar a mis amigos, a mis queridos amigos que siempre han estado conmigo en las buenas y malas; a esos que me acompañaron estos últimos 5 años y a los otros que tomaron otro camino, pero que ven sus triunfos reflejados en los míos y viceversa. Le dedico este trabajo especialmente a Dayana, mi apoyo emocional y motor impulsor de toda obra que nace de mí, gracias por existir, por darme fuerzas para seguir solo hacia adelante. Agradezco además a su familia, que también es la mía.
En fin, le dedico este trabajo a todos los que me quieren, y a los que no también, por haberme ayudado a ser como soy.

Eriberto

RESUMEN:

Las tecnologías de la información actualmente son elementos fundamentales para el desarrollo de la economía de un país. Es por esto que los países desarrollados, y en vías de desarrollo, basan su crecimiento económico en la aplicación y la programación estratégica de las herramientas computacionales, definiendo políticas que los inducirán a su permanencia en el dinamismo mundial de los próximos años.

A esta ardua e importante tarea también se ha sumado Cuba. Ya es posible ver resultados prometedores, pero el objetivo es seguir evolucionando y convertir la producción de software en uno de los principales renglones económicos del país. Se han firmado contratos y se han comercializado productos con otros países, la mayoría de estos proyectos se llevan a cabo principalmente en la Universidad de las Ciencias Informáticas **(UCI)**.

En la actualidad se ha convertido en una regla inviolable el hecho de que todo proyecto vinculado a la producción de software necesita establecer un Sistema de Gestión de la Calidad **(SGC)** desde sus inicios. El mismo estará enfocado tanto a los procesos como a los productos. En la UCI este es un tema que se toma muy en cuenta. Ya se han dado los primeros pasos, principalmente en el área de Control de la Calidad **(CC)**, que forma parte de este SGC, con la realización de una Estrategia de Pruebas desarrollada por el Grupo de Calidad **(GC)** interno de la Facultad 5. Sin embargo este fue solo un pequeño avance, el siguiente consiste en crear un prototipo de aplicación informática que permita gestionar el proceso de pruebas de software.

PALABRAS CLAVES: Sistema de Gestión de la Calidad **(SGC)**, Control de la Calidad **(CC)**, Prototipo, Gestionar.

TABLA DE CONTENIDOS:

INDICE DE TABLAS:

INDICE DE ILUSTRACIONES:

INTRODUCCIÓN:

Las Tecnologías de la Información y las Comunicaciones **(TIC)** van desarrollándose a grandes pasos. Debido a esto los problemas son cada vez más complejos, por lo que ha sido necesario buscar soluciones, caminos y paradigmas nuevos para resolverlos. Estas soluciones, en la mayoría de los casos, incluyen la utilización de software por la gran cantidad de información y complejidad de los mismos.

El desarrollo progresivo del software, como consecuencia del desarrollo de las TIC, también se ha convertido en una tarea bastante compleja en estos días, que ha sobrepasado, sin dudas y en gran medida, la habilidad para el mantenimiento del mismo en algunas empresas que los producen o los utilizan.

Se hace cada vez más necesario para la empresa cubana del software, así como las de todo el mundo, encontrar las mejores alternativas para incrementar la producción y a la vez garantizar la calidad de los productos, además de lograr la difícil satisfacción de los clientes y usuarios.

Cuba dedica una cantidad considerable de recursos humanos y materiales al desarrollo de la Informática como parte del crecimiento económico necesario de la isla. En estos momentos se cuenta con profesionales de mucho conocimiento y creatividad. Desde el Triunfo de la Revolución, la política para con los demás países del mundo ha estado basada en el respeto mutuo y en la cooperación. Uno de los países con los que se han firmado varios acuerdos importantes es la hermana República Bolivariana de Venezuela.

Actualmente se desarrollan varios de estos proyectos en la UCI, que es uno de los pilares fundamentales en cuanto al desarrollo del software en Cuba, y la misma tiene como misión formar profesionales comprometidos con su Patria y calificados en la rama de la Informática, logrando así que se produzca software con toda la calidad requerida para competir en el mercado internacional.

Dicha universidad abre sus puertas en el año 2002 y su estructura actual cuenta con 10 facultades que corresponden a perfiles diferentes y en los que se trata de abarcar las diferentes vertientes de la informática. En la facultad número 5 se encuentra el Polo de Hardware y Automática **(PHA)**, el cual surge

con el objetivo de brindar solución a diferentes problemas dentro del sector de automatización dentro y fuera del país. Se cuenta con un equipo multidisciplinario en el que interactúan estudiantes, profesores, asesores, ingenieros y especialistas dedicados a las ciencias de automatización.

En el PHA se mezclan soluciones de software y hardware a temas que afectan el ambiente industrial y corporativo de empresas nacionales e internacionales. Su desarrollo va encaminado al uso y explotación de metodologías y técnicas consideradas libres, con fines de ampliar los productos de software así como la documentación y conocimiento que se generen en su desarrollo.

La dirección del PHA define las políticas y reglas generales que deberán cumplir estrictamente los proyectos que forman parte de él. Entre ellas está la implementación de un SGC desde los inicios de fundado el proyecto, ya que este mecanismo, de vital importancia hoy en día, más que una simple necesidad estratégica, se ha convertido en un arma imprescindible para sobrevivir en el altamente competitivo mercado internacional.

Dentro de este SGC juega un papel de suma importancia el CC, que funciona durante todo el proceso de desarrollo, incluye: supervisar y mejorar el proceso, comprobar que se siguen los procedimientos acordados, que se alcanza el nivel de calidad deseado y que se localizan y resuelven los problemas. Dentro del mismo juegan un importante papel las Pruebas de Software que son las que permiten identificar posibles fallos de implementación, calidad, o usabilidad del software implementado.

Los profesionales que desarrollan su actividad laboral entorno a las tecnologías informáticas, muchas veces consideran que las pruebas de software, consideradas de gran importancia por los expertos, son una tarea poco creativa, de escasa consideración dentro de las organizaciones y de poca proyección profesional. A todo esto se le tiene que sumar la falta de formación específica y la poca importancia que las empresas dan a estos procesos así como el poco tiempo que se les dedica. Son escasas las empresas que utilizan alguna metodología de pruebas o aplicaciones informáticas que les permitan automatizar y administrar el proceso.

Dependiendo de la complejidad del software, el proceso de pruebas puede llegar a ser el más largo y costoso. Si no se realiza a tiempo puede causar grandes pérdidas de tiempo y recursos, por lo que se hace necesario adoptar las metodologías, normas, estándares, y escoger las herramientas propicias tempranamente.

En estos momentos el PHA ha mostrado avances significativos con respecto al tema de la calidad. Esta tarea ha recaído sobre el GC interno del polo, el cual propuso y desarrolló satisfactoriamente, como una de sus primeras y más importantes tareas, una Estrategia de Prueba que permite la rápida detección y corrección de errores en los productos. Esta tarea tuvo gran significación y fue un avance en el proceso de control de la calidad de los entregables. Sin embargo la estrategia se realizó únicamente para el Proyecto: SCADA Nacional y no se tuvieron en cuenta los demás proyectos. La misma se lleva a cabo de forma manual y es difícil mantener una buena organización y control del trabajo realizado, además exige tiempo y recursos con los que en ocasiones no se cuenta. En fin, la actual estrategia no resuelve los problemas que vienen afectando al polo en cuanto a las pruebas de software.

La dirección del polo, en coordinación con la dirección del GC, se plantea la necesidad de perfeccionar el área de CC mediante la gestión del proceso de pruebas de software, haciendo uso de las estrategias, herramientas y los artefactos que reduzcan la necesidad de la implicación o de la interacción manual o humana en dicho proceso.

Surge entonces el siguiente **problema científico**: ¿Cómo gestionar el proceso de pruebas de software en el PHA?

Como **Objeto de Estudio** se plantea: el **Control de la Calidad de Software** en el PHA, y el **Campo de Acción**: el proceso de **Pruebas de Software** en el PHA.

El **Objetivo General** establecido es: desarrollar un prototipo de aplicación Informática que permita gestionar el proceso de pruebas de software en el PHA.

Como **Objetivo específico** se plantea: definir un proceso de pruebas que se adapte a las necesidades del PHA, y le de solución a los problemas actuales.

Las **Tareas de la Investigación** son las siguientes:

1. Elaboración del marco teórico y metodológico de la investigación.
2. Selección de la metodología y las herramientas a utilizar en el proyecto.
3. Realización del levantamiento de requisitos.
4. Diseño de la Base de Datos para gestionar la información obtenida en el proceso de pruebas de software.
5. Diseño e implementación del prototipo de aplicación Web para la gestión del proceso de pruebas de software.

Idea a Defender:

- La introducción de un sistema informático para la gestión de un proceso de pruebas centralizado, en el PHA, controlará el cumplimiento estricto de las estrategias de pruebas establecidas y contribuirá con la mejora de la calidad, tanto de los productos realizados como de los proceso de desarrollo de software.

Viabilidad:

Debido a los conocimientos y experiencias adquiridos hasta el momento por el GC de la Facultad 5, se hace posible el desarrollo de un prototipo de aplicación web para la gestión del proceso de pruebas del PHA.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción:

El presente capítulo muestra un estudio acerca de las tendencias actuales en cuanto al tema de la Calidad de Software en las Ciencias Informáticas. Se tratan temas específicos como los SGC y el CC, dentro de este último las Pruebas de software, definiéndose los distintos métodos, tipos y niveles de las mismas, así como las estrategias y otros conceptos relacionados.

Se tratan además otros temas como las Normas y Estándares de calidad más utilizados a nivel mundial, además de las Metodologías de Desarrollo de Software. Por último se trata el tema de las nuevas tecnologías informáticas y las herramientas más utilizadas para la gestión del proceso de pruebas de software.

1.1- Calidad de Software.

Existen muchas definiciones para este término debido a que cada persona tiene una idea diferente del mismo, en dependencia de su punto de vista y del entorno donde se desarrolle. Es sumamente difícil establecer un concepto abarcador que satisfaga todas las variantes de lo que se entiende por calidad. El significado de esta palabra puede adquirir múltiples interpretaciones, ya que también dependerá del nivel de satisfacción o conformidad del cliente.

El término calidad es ambiguamente definido y pocas veces comprendido, esto se debe a que la calidad no es una sola idea, es un concepto multidimensional. La dimensión de la calidad incluye el interés de la entidad, el punto de vista de la entidad y los atributos de la entidad. Por cada concepto existen diferentes niveles de abstracción. Varía para cada persona particular. (Jacobson, et al., 2000)

Algunas de las definiciones más importantes:

Calidad: Propiedad o conjunto de propiedades inherentes a algo, que permite juzgar su valor. (Real Academia Española, 2001)

Calidad: Conjunto de propiedades y características de un producto o servicio que le confieren su aptitud para satisfacer necesidades explícitas o implícitas. (ISO 8402)

Calidad: Grado en el que un conjunto de características inherentes cumple con los requisitos. (ISO 9001)

Es posible calificar la calidad desde dos puntos de vista: **usual** y **profesional.**

Punto de vista usual: La calidad contiene características intangibles, términos como: alta, baja, y buena calidad son utilizados sin intentar definirlos.

Punto de vista profesional: Juran (1970) definió la calidad como "Adaptabilidad de Uso", esto implica dos parámetros: calidad de diseño y calidad de conformidad. Es decir, adaptable a la necesidad de los usuarios. (Jacobson, et al., 2000)

El avance de las TICs pudiera ser más rápido si no estuviera frenado por los pocos recursos que se destinan a la gestión de calidad de los procesos y productos software. Ante la interrogante de porqué muchos de los proyectos fracasan en su intento por satisfacer los requerimientos de los usuarios, se encuentra el desconocimiento de técnicas y metodologías que permiten garantizar la calidad.

Es importante diferenciar entre la calidad del producto software y la calidad del proceso de desarrollo. No obstante, las metas que se establezcan para la calidad del producto van a determinar las metas a establecer para la calidad del proceso de desarrollo, ya que la calidad del producto va a estar en función de la calidad del proceso de desarrollo. Sin un buen proceso de desarrollo es casi imposible obtener un buen producto. (De Antonio Jiménez)

"**La calidad del software** es el grado con el que un sistema, componente o proceso cumple con los requerimientos especificados y las necesidades o expectativas del cliente o usuario". (IEEE 1990)

"**La calidad de software** es la concordancia del software producido con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y

con las características implícitas que se espera de todo software desarrollado profesionalmente." (S.Pressman, 1998)

El tipo y número de actividades de garantía de calidad que es necesario adoptar en un proyecto, o en una organización, depende del tamaño y complejidad de los productos software que se estén desarrollando. Existen otros factores influyentes como son: los tipos de procesos de desarrollo de software, los métodos y herramientas utilizados, la estructura organizativa escogida y la motivación del personal.

La calidad de un proceso contribuye a mejorar la calidad del producto, y, a su vez, la calidad del producto contribuye a mejorar la calidad en uso. La finalidad de la calidad en uso es medir la efectividad, productividad, seguridad y la satisfacción de los usuarios (pertenecientes a perfiles determinados) que interactúan con el producto en escenarios específicos de uso. La finalidad de la calidad en uso es medir la efectividad, productividad, seguridad y la satisfacción de los usuarios (pertenecientes a perfiles determinados) que interactúan con el producto en escenarios específicos de uso. (ISO 9126)

Lograr el éxito en la producción de software es hacerlo con calidad y demostrar su buena calidad. Esto sólo es posible con la implantación de los mecanismos necesarios, que estén directamente relacionados con la política establecida para la elaboración del software y que estén en correspondencia con la definición internacional ISO de calidad y por los estándares del grupo ISO 9000 y 9001 del 2000.

1.1.1- Modelos de Calidad de Software.

La calidad de un producto software debe evaluarse usando un modelo que tenga en cuenta los criterios para satisfacer las necesidades de los desarrolladores, mantenedores y usuarios finales.

Estos modelos vienen a ayudar en la puesta en práctica del concepto general de calidad, ofreciendo una definición más operacional. Son herramientas que guían a las organizaciones a la Mejora Continua y la Competitividad, dándoles especificaciones de los tipos de requisitos que deben implementarse para poder brindar productos y servicios de alto nivel.

En los modelos de calidad, esta se define de forma jerárquica. Es un concepto que se deriva de un conjunto de sub-conceptos, cada uno de los cuales se van a evaluar a través de un conjunto de indicadores o métricas. (De Antonio Jiménez)

Cuando se decide implantar un modelo de calidad en una empresa, generalmente el propósito del mismo es que se puedan desarrollar productos, bienes y servicios que tengan una mejor calidad y que además cumplan con las necesidades y expectativas de los clientes.

La misión general de estos modelos es unir todos los esfuerzos, con el objetivo de obtener resultados positivos cada vez mejores y avanzar hacia la competitividad y la calidad total de la empresa. La base para diseñar e implantar un buen modelo de calidad es conocer las características y las necesidades de la empresa que lo aplicará, además de los deseos y pretensiones de los clientes actuales o potenciales.

Se debe entender que un modelo de calidad no es una metodología que resuelva todos los problemas de forma sencilla y clara, estos modelos dicen “que” hacer, no “como” hacerlo. Los mismos requieren una planificación detallada y una cuidadosa recolección de medidas. Pueden ser muy costosos incluso para un número reducido de factores, por lo que requieren recursos extras y, como consecuencia de ello, se usan con poca frecuencia.

Entre las ventajas de contar con uno de estos modelos es que la calidad se convierte en algo concreto, que se puede definir, se puede medir y, sobre todo, se puede planificar. Los modelos de calidad ayudan también a comprender las relaciones que existen entre diferentes características de un producto software.

Actualmente existen muchos modelos, pero una desventaja es que aún no ha sido demostrada la validez absoluta de ninguno de ellos. Las conexiones que establecen entre características, atributos y métricas se derivan de la experiencia, a esto se debe que existan varios.

A continuación se presentan algunos de los modelos para la GCS:

- **CMMI:** Diseñado por el Carnegie Mellon Software Engineering Institute **(SEI)**. Es un modelo de calidad del software que clasifica las empresas en niveles de madurez. Estos niveles sirven para conocer la madurez de los procesos que se realizan para producir software. Está orientado a mejora de procesos en diferentes niveles de madurez, más hacia proyectos específicos.

- **Norma ISO/IEC 12007:** Diseñada por la International Organization for Standardization **(ISO)** (Organización Internacional para la Estandarización). Orientado al proceso del ciclo de vida del software.

- **SPICE** (Software Process Improvement and Capability Determination). Modelo para la mejora y evaluación de los procesos de desarrollo y mantenimiento de sistemas y productos de software.

- **Modelo EFQM de Excelencia:** Se trata de un modelo no normativo, cuyo concepto fundamental es la autoevaluación basada en un análisis detallado del funcionamiento del sistema de gestión de la organización usando como guía los criterios del modelo.

Reconocer la importancia y la necesidad de la correcta implantación de un modelo de calidad en las empresas, es el primer paso hacia el aseguramiento y mantenimiento de la competitividad de la misma en el mercado de los próximos años. Este proceso no es para nada fácil y deberá tener sus características propias en dependencia de la empresa donde se desarrolle.

1.1.2- Sistema de Calidad.

En el Capítulo 5 de la Norma UNE 66.904, coincidente con la ISO 9004, se recogen las directrices del Sistema de Calidad **(SC)** *(Ver Anexo1)* indicando la necesidad de que afecte todas las actividades de la empresa, y todas las fases del proceso, así como que estas actividades estén expresadas por escrito.

Un SC es un método planificado y sistemático de medios y acciones, encaminado a asegurar suficiente confianza en que los productos y servicios, se ajusten a las especificaciones. Es un marco en el que se establecen las diferentes estrategias, actividades y herramientas de garantía de calidad que se van a utilizar.

Sistema de calidad: Estructura de organización, de responsabilidades, de actividades, de recursos y de procedimientos que se establecen para llevar a cabo la gestión de calidad. (ISO 9000)

Un SC consta de dos partes:

- **Documentación** en la que se describe el sistema, procedimientos, etc. ajustándose a una norma.
- **Parte práctica**, que tiene dos vertientes:
    - Aspectos físicos.
    - Aspectos humanos.

En general, el SC está condicionado por:

- Organización con la que se cuenta.
- Tipo y naturaleza del producto o servicio.
- Medios materiales y humanos.
- Exigencias de mercado o clientes.

Como consideración básica, un SC tiene que ser inteligente, capaz de adaptarse a la realidad de cada ambiente en el que se desarrolle, eficaz en el control de los procesos, entendido como la capacidad para reconstruir la historia de los procesos aplicados mediante registros para llegar al resultado final, y capaz de regenerarse en la búsqueda de una mejora continua.

La mayoría del software nace y vive para obtener beneficios, el SC a implantar debe ser aquel a través del cual se obtengan los beneficios máximos. La implantación de cualquier sistema, es beneficioso, y en la mayoría de los casos, es solo la imposición del cliente, y no el propio convencimiento, lo que obliga a su introducción.

Una vez diseñado, y antes de su lanzamiento, si se pretende llevarlo a buen fin, se requiere siempre una formación y mentalización de todo el personal. Debido a esto, es conveniente separar el lanzamiento del SC de cualquier otra acción, como el lanzamiento de un nuevo producto. Elegido el sistema de implementación, sea éste global, por áreas, por procesos, etc., es necesario arbitrar los sistemas para su

mejora permanente, midiendo resultados y a través de la realización de auditorías, cuantificar su grado de implementación, los progresos y mejoras obtenidos que pongan de manifiesto la eficacia del sistema.

1.1.3- Gestión de la Calidad del Software.

La gestión de la calidad dentro de la Ingeniería de Software va encaminada al aseguramiento de la calidad del software a lo largo del proceso de desarrollo. Incluye métodos y herramientas de análisis, diseño, codificación y prueba, el control de la documentación y de los cambios, los procedimientos para asegurar el ajuste a los estándares, y los mecanismos de medida (métricas) e informes.

Gestionar la calidad a nivel de empresa u organización consiste en crear una estructura organizativa apropiada para fomentar la calidad del trabajo de todas las personas y departamentos de la empresa.

Se entiende por gestión de la calidad el conjunto de actividades coordinadas para dirigir y controlar una organización en lo relativo a la calidad. Generalmente incluye el establecimiento de la política de la calidad y los objetivos de la calidad, así como la planificación, el control, el aseguramiento y la mejora de la calidad. (ISO 9000)

- **Planificación de la Calidad de Software.** (Encargada de realizar el proceso administrativo de desarrollar y mantener una relación entre los objetivos y recursos de la organización; y las oportunidades cambiantes del mercado.)

- **Control de la Calidad de Software.** (Son las técnicas y actividades de carácter operativo, utilizadas para satisfacer y evaluar los requisitos relativos a la calidad de los productos software desarrollados.)

- **Aseguramiento de la Calidad de Software.** (Actividades planificadas y sistemáticas necesarias para aportar la confianza de que el software satisfará los requisitos de calidad.)

- **Mejora de la Calidad de Software.** (Contribuye, por medio de las mediciones, los análisis de los datos y las auditorias, a efectuar mejoras en la calidad del software.)

La gestión de la calidad, a nivel de organización en entidades de software, ha seguido dos líneas que pueden perfectamente complementarse entre sí. Por una parte, se ha seguido la línea marcada por las entidades internacionales de estandarización para todos los productos y servicios, a través de las normas ISO 9000, y por otra, el mundo del software ha creado su propia línea de trabajo en la gestión de la calidad, trabajando sobre los procesos de producción de software como medio para asegurar la calidad del producto final.

1.1.4- Sistema de Gestión de la Calidad del Software.

Para conducir y operar una organización en forma exitosa se requiere que ésta se dirija y controle en forma sistemática y transparente. Se puede lograr el éxito implementando y manteniendo un Sistema de Gestión que esté diseñado para mejorar continuamente su desempeño mediante la consideración de las necesidades de todas las partes interesadas. La gestión de una organización comprende la gestión de la calidad entre otras disciplinas de gestión. (ISO 9000)

La adopción de un Sistema de Gestión de la Calidad debería ser una decisión estratégica que tome la alta dirección de la organización. El diseño y la implementación de un sistema de gestión de la calidad de una organización están influenciados por diferentes necesidades, objetivos particulares, los productos que proporciona, los procesos que emplea y el tamaño y estructura de la organización. (ISO 9004)

El SGC es una herramienta para el cambio, pero no realiza el cambio, la única que puede realizar el cambio de cultura en la organización, es la dirección. Las organizaciones pueden, de acuerdo a sus necesidades, certificar o no su SGC, pero la certificación sólo tiene verdadero valor cuando refleja una organización centrada en el cliente, flexible pero rigurosa y capaz de desenvolverse eficientemente en un entorno económico y tecnológico constantemente cambiante.

Los SGC están formados por la estructura organizativa, las responsabilidades, los procedimientos, los procesos y los recursos necesarios para llevar a cabo la gestión de la calidad. Se aplica en todas las actividades realizadas en una empresa y afecta todas las fases, desde el estudio de las necesidades del

consumidor hasta el servicio posventa, los beneficios que reporta a la organización y las etapas de su implantación.

Estos sistemas son significativos para cualquier proyecto, grande o pequeño. Puede aplicarse tanto a un solo departamento como a todos los demás que formen parte del proyecto. Sin embargo, los mejores resultados los obtienen las organizaciones preparadas para implementarlo completamente.

1.2- Control de la Calidad del Software.

El CC consiste en una serie de inspecciones, revisiones y pruebas utilizadas a lo largo del proceso de desarrollo de software para comprobar que cada producto cumple con los requisitos pactados por ambas partes. Incluye un bucle de realimentación (feedback) del proceso que creó el producto. La combinación de medición y realimentación permite mejorar el proceso cuando los productos de trabajo creados fallan al cumplir sus especificaciones. Este enfoque ve el CC como parte del proceso de fabricación.

Existen muchas maneras de llevar a cabo el CC, las actividades de control pueden ser manuales, completamente automáticas o una combinación de ambas. Lo que si no puede faltar es la definición de todos los productos así como las especificaciones de los mismos, en las que se puedan comparar los resultados de cada proceso. El bucle de realimentación es esencial para reducir los defectos producidos.

Controlar la calidad es mantener una observación constante al cumplimiento de las tareas que pueden ofrecer una calidad objetiva a la forma en la que se está desarrollando un proyecto. Permite realizar las rectificaciones pertinentes al desarrollo en cuanto este empieza a desviarse de sus objetivos, alejando la inclusión de la calidad al trabajo. Esto permite realizar el mejoramiento de los procesos débiles, lo que definitivamente desembocará en un aseguramiento de la calidad en los procesos ejecutados por la organización.

No es posible desarrollar un software sin antes haber creado una unidad organizativa que guie el proceso, que sea la encargada de definir y planificar las actividades. Sin embargo todos estos mecanismos por si solos no garantizan que se cumplan las orientaciones al pie de la letra, para ello es imprescindible implantar todos los mecanismos de control que sean necesarios.

Control de la Calidad del Software (CCS): Técnicas y actividades de carácter operativo, utilizadas para verificar los requisitos relativos a la calidad, se centran en mantener bajo control el proceso de desarrollo, y eliminar las causas de los defectos en las diferentes fases del ciclo de vida del software.

El CCS está centrado en dos objetivos fundamentales:

- Mantener bajo control un proceso.
- Eliminar las causas de los defectos en las diferentes fases del ciclo de vida.

El CCS debe asegurar que el producto acabado cumple con los requerimientos. Debe comprobar que las funciones se ejecutan de forma correcta. El objetivo principal es descubrir todos los fallos del sistema antes de que llegue al usuario final. De esta forma será posible detectar aquellas situaciones anómalas del sistema que deberán ser controladas, si todavía no lo están.

El CC puede implantarse como un sistema independiente, en este caso sería un Sistema de Control de la Calidad **(SCC),** o puede seguir siendo una de las tareas o subsistemas dentro del SGC.

Es posible definir al **Sistema de Control de Calidad de Software (SCCS)** como la estructura encargada de organizar las evaluaciones, inspecciones, auditorías y revisiones que aseguren que se cumplan las responsabilidades asignadas, se utilicen eficientemente los recursos y se logre el cumplimiento de los objetivos del producto. Tiene la intención de mantener bajo control un proceso y eliminar las causas de los defectos en las diferentes fases del ciclo de vida de un producto software.

Es importante tener en cuenta los costos que involucra el SCCS. Si se piensa en las tareas que se debe realizar en este control, puede observase que es necesario llevar a cabo tareas de búsqueda de problemas, testeo, realimentación, rectificación, elaboración, modificación y estudio de la documentación; entre otras actividades. Todas ellas tienen costos involucrados.

1.2.1- Pruebas de Software.

Cuando aparecieron los primeros sistemas informáticos se incluyó, a nivel metódico e imprescindible, un hasta entonces, nuevo proceso en la confección de los mismos: el proceso de pruebas.

Se calcula que la fase o proceso de pruebas puede representar hasta más de la mitad del coste y tiempo de desarrollo de un programa. Se requiere muchas veces un tiempo similar al de la programación, lo que obviamente provoca un alto costo económico aun cuando estos no involucran vidas humanas, puesto que en este último caso el costo suele ser bastante elevado, siendo esta etapa más cara que el propio desarrollo y diseño de los distintos programas que conforman el sistema.

El proceso de pruebas se enfoca en la lógica interna del software y las funciones externas. Es un proceso de ejecución de un programa con la intención de descubrir un error, un buen caso de prueba es aquel que tiene alta probabilidad de mostrar un error no descubierto hasta entonces.

Las pruebas de software son un proceso centrado en el objetivo de encontrar defectos a un software; puede ser por razones de depuración o de aceptación del mismo. Tratar de encontrar defectos es una parte esencial en toda prueba, para ello los probadores evalúan el producto desde un punto de vista crítico sometiéndolo a una serie de acciones inquisitivas y el producto responde con su comportamiento como reacción.

Es necesario probar el producto primeramente en sus escenarios básicos de funcionamiento, comprobando que cumple con todos los requisitos funcionales definidos por los clientes, luego se prueban los escenarios extremos para verificar su robustez, el estrés que soportan, el rendimiento y otros factores importantes a tener en cuenta.

El objetivo de la etapa de pruebas es mejorar la calidad del producto desarrollado. Además, implica:

- Verificar la interacción de componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar los defectos, y que los mismos sean corregidos antes de entregar el software al cliente.

- Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

El objetivo final de las pruebas, es mantener bien informada a la organización respecto a los defectos observados, que estén relacionados con los requerimientos del sistema, ya sean estos explícitamente definidos o implícitamente asumidos.

Las pruebas no mejoran directamente la calidad del sistema, pero sí lo hacen indirectamente previendo un panorama claro de las debilidades observadas y de los riegos asociados a la organización. Esto permite tomar decisiones respecto a la asignación de recursos para contribuir al mejoramiento de la calidad del sistema. Se deben realizar durante todo el ciclo de vida: probar la funcionalidad de los primeros prototipos; probar la estabilidad, cobertura y rendimiento de la arquitectura; probar el producto final, etc. Lo que conduce al principal beneficio de la prueba: proporcionar un feedback mientras hay todavía tiempo y recursos para hacer algo.

Es importante tener presente que realizar pruebas exhaustivas a programas grandes o complejos, muchas veces resulta complicado, y puede llegar a ser imposible teniendo en cuenta factores como el tiempo, los gastos y los factores matemáticos y humanos. El hecho de que un proceso de pruebas no descubra ningún error no es suficiente para asegurar que el producto probado no los tiene. Esto no es motivo para desacreditar o restarle importancia al proceso de pruebas. Ante la imposibilidad de la perfección es necesario buscar formas humanamente abordables y económicamente aceptables de encontrar errores.

A pesar de que la mayoría de los desarrolladores están de acuerdo que es mejor prevenir defectos que encontrarlos y corregirlos, la realidad es que actualmente las empresas y proyectos son incapaces de producir sistemas libres de defectos.

1.2.2- Métodos de Pruebas.

Una buena prueba debe tener 3 atributos:

1- Debe tener una alta probabilidad de encontrar un error.
2- No debe ser redundante.

3- No debería ser ni demasiado sencilla ni demasiado compleja.

Los Métodos de Prueba de Software tienen el objetivo de diseñar pruebas que descubran diferentes tipos de errores con menor tiempo y esfuerzo.

Los dos métodos de prueba más utilizados son el **Método de Prueba de Caja Blanca (CB)**, Transparente o de cristal y el **Método de Prueba de Caja Negra (CN)**. La diferencia entre ambos radica en el hecho de que con la CB es necesario conocer el código y estar bastante relacionado con él para saber exactamente cuál es la lógica interna de lo que se va a probar, sin embargo en la CN solo basta conocer las posibles entradas y salidas del programa.

Ambos métodos son importantes a la hora de probar un software por muy pequeño que sea, un método complementa al otro. Pueden realizarse juntos o separados en dependencia de las necesidades y las características de la organización, pero la unión de ambos es la que garantiza la rápida detección y corrección de los errores que contiene el software, y que no han sido identificados aún por los desarrolladores.

Para cada método específico existe una variedad de Técnicas de Pruebas diferentes que permiten perfilar el trabajo hacia los objetivos concretos que se persigan con las pruebas. El alcance de estas técnicas está determinado por la complejidad y tamaño del programa. En dependencia de la estrategia que se plantee la organización, serán utilizadas cada una de ellas.

1.2.3- Método de Prueba: Caja Blanca.

A las pruebas de CB también se les conoce como pruebas estructurales o pruebas de caja transparente. Están encaminadas a comprobar que las operaciones internas del programa se ajusten a las especificaciones, y garantiza que todos los componentes internos se prueben adecuadamente.

Prueba de Caja Blanca: “es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba.” (S.Pressman, 1998)

El objetivo principal de estas pruebas consiste en garantizar que se ejerciten, por lo menos una vez, todos los caminos independientes de cada uno de los módulos, todas las decisiones lógicas en sus ambas variantes (verdadero y falso), todos los bucles en sus límites y con sus límites operacionales, y finalmente las estructuras internas de los datos para comprobar su validez.

Utilizando este método se pueden determinar cuáles son los casos de prueba, a partir del código fuente del software, utilizándose las especificaciones para determinar el resultado esperado de cada caso. Los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como la integridad de la información externa. La prueba de CB del software se basa en el minucioso examen de los detalles procedimentales.

1.2.4- Técnicas de Prueba de Caja Blanca.

Entre las Técnicas de Prueba de CB hay dos vertientes: la que se basa en la **Cobertura del Código** y la que se basa en la **Estructura del Código**.

La cobertura es una medida del alcance que tendrá cada una de las técnicas de prueba que se apliquen en cada caso. Estas técnicas dependerán completamente de los factores tiempo y tamaño del código que se desea probar, además de la estrategia y los planes que se haya trazado la dirección del proyecto. Entre las técnicas de cobertura están las que siguen a continuación:

- **Cobertura de segmentos.** (Se encarga de ejecutar todos los segmentos o sentencias del programa, es poco utilizado en la práctica debido a la complejidad y tamaño del software.)

- **Cobertura de ramas.** (Refinación de la cobertura de segmentos, recorre todos los puntos de decisión en caso de que existan variantes en los segmentos. Tampoco es muy utilizado en la práctica debido a las limitaciones en cuanto al tiempo, recursos y personal disponible.)

- **Cobertura de condición/decisión.** (Complementa la cobertura de ramas, se utiliza cuando las expresiones booleanas, usadas para decidir las ramas a seguir, son muy complejas.)

- **Cobertura de bucles.** (Mejora la cobertura de condición/decisión, cuando los segmentos están controlados por decisiones.)

La importancia de las técnicas de cobertura radica en que se puede ir barriendo una gran parte del código del programa en busca de errores, sin embargo en la actualidad estas técnicas son raramente utilizadas porque los gastos que implican superan los beneficios que suponen las pruebas.

Estas técnicas tienen limitaciones en cuanto a la posibilidad de lograr una buena cobertura cuando el software es complejo. Otra de las limitaciones es que las pruebas de CB prueban solamente que el software funciona correctamente, sin embargo no pueden garantizar de ninguna manera que el programa haga lo que realmente debería hacer.

Las **Técnicas de Prueba de Estructura** también comprueban que la operación interna del programa se ajusta a las especificaciones y que todos los componentes internos se hayan probado adecuadamente. Usan la estructura de control para obtener los casos de prueba e Intentan garantizar que todos los caminos de ejecución del programa quedan probados. Entre estas técnicas de pruebas tenemos:

- **Pruebas del Camino Básico.** (Define un conjunto básico de caminos de ejecución. Genera un caso de prueba para cada camino de ejecución.)

- **Pruebas de condición.** (Consisten en diseñar casos de prueba para que todas las condiciones del programa se evalúen a cierto/falso.)

- **Pruebas de Bucles.** (Diseñar casos de prueba para que se intente ejecutar un bucle.)

Las técnicas de pruebas estructurales son las más utilizadas cuando se decide utilizar el método de CB debido a que las mismas necesitan menos tiempo y recursos para realizarse, además abarcan todo el código y muchas veces no es necesario probar todas y cada una de las sentencias del código.

1.2.5- Método de Prueba: Caja Negra.

Las pruebas de caja negra, también denominada prueba 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. (S.Pressman, 1998)

Con este método, los casos de prueba y los resultados se determinan a partir de la especificación funcional de los métodos de una clase. Es decir, la prueba de CN se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. Una prueba de CN examina algunos aspectos del modelo fundamental del sistema sin tener mucho en cuenta la estructura lógica interna del software.

Intentan encontrar errores de las siguientes categorías:

1- Funciones incorrectas o ausentes.
2- Errores de interfaz.
3- Errores en estructuras de datos o en accesos a bases de datos externas.
4- Errores de rendimiento.
5- Errores de inicialización y de terminación.

A la mayoría de los usuarios de programas extensos no les interesan los detalles de su funcionamiento; lo único que desean es una respuesta acorde con la entrada realizada, por lo que se puede aseverar que, al menos en los programas grandes y complejos, las pruebas de CN son las más utilizadas y las preferidas por usuarios y desarrolladores.

Los métodos de prueba de CN también cuentan con diferentes técnicas de prueba que pueden ser utilizadas en dependencia del alcance y de los objetivos específicos de las mismas.

1.2.6- Técnicas de Prueba de Caja Negra.

Mediante estas técnicas de prueba de CN se obtiene un conjunto de casos de prueba que satisfacen los siguientes criterios:

1- Casos de prueba que reducen, en un coeficiente mayor que uno, el número de casos de prueba adicionales que se deben diseñar para alcanzar una prueba razonable.
2- Casos de prueba que determinan la presencia o ausencia de clases de errores en lugar de errores asociados solamente con la prueba que se está realizando.

Entre las técnicas más utilizadas están las siguientes:

- **Particiones o Clases de Equivalencia:** Técnica algebraica que trata a cada parámetro como un modelo donde unos datos son equivalentes a otros. Se realiza dividiendo un rango excesivamente amplio de posibles valores reales, a un conjunto reducido de clases de equivalencia, entonces es suficiente probar un caso de cada clase, pues los demás datos de la misma clase son equivalentes. Las clases más comunes son dos: clases válidas y clases inválidas

- **Análisis de Valores Límites:** Esta técnica es utilizada muchas veces como complemento de la partición de equivalencia. Su fundamento se basa en que la mayor cantidad de errores se encuentra en los límites del rango de valores de entrada de un programa, se prueban los valores limites del rango incluyendo los valores fuera del rango que estén más cercanos a él.

- **Conjetura de Errores:** Es una técnica que se basa mayormente en la intuición y la experiencia adquirida por el que diseña las pruebas. No tiene un procedimiento estándar definido sin embargo goza de gran popularidad y actualmente es muy utilizada por las personas dedicadas a realizar pruebas al software.

Para obtener resultados óptimos utilizando estas técnicas de prueba, es necesario combinarlas adecuadamente. Van encaminadas a probar que el software responde correctamente a las entradas de los usuarios, mostrando de esta forma que su respuesta se corresponde a los requisitos funcionales.

Las limitaciones potenciales de estas técnicas consisten en que se torna un poco complicado lograr una buena cobertura de los valores de entrada cuando los rangos de los mismos son muy amplios. El programa puede tener errores internos que pueden surgir en el momento menos oportuno y bajo condiciones específicas, no se puede garantizar que el software no contenga errores, sino que responde adecuadamente a lo que debería hacer.

1.3- Niveles y Tipos de Pruebas.

Una vez que se ha generado el código comienzan las pruebas del programa. El proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, y en los procesos externos funcionales; 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. (Berzal Galiano)

El tamaño y complejidad del proceso de pruebas depende en gran medida del tamaño y complejidad del producto que se está desarrollando. Con el objetivo de simplificar el proceso y orientarlo a cada etapa del desarrollo del software, las pruebas se han dividido en varios niveles.

1.3.1- Pruebas Unitarias.

Cuando se implementa software, resulta recomendable comprobar que el código que hemos escrito funciona correctamente. Para ello implementamos pruebas que verifican que nuestro programa genera los resultados que de él esperamos. Conforme vamos añadiéndole nueva funcionalidad a nuestras aplicaciones, creamos nuevas pruebas con los que podemos medir nuestros progresos y comprobar que lo que antes funcionaba sigue funcionando (prueba de regresión). Las pruebas de unidad también son de vital importancia: aunque no añadimos nueva funcionalidad, estamos modificando la estructura interna de nuestro programa y debemos comprobar que no introducimos errores. Las pruebas de unidad son, por tanto, muy importantes en el desarrollo de software. Para agilizar las pruebas resulta recomendable que un test sea completamente automático y compruebe los resultados esperados. (Berzal Galiano)

Las pruebas unitarias son consideradas de menor escala, consisten en probar cada uno de los módulos que conforman el programa, cuando estos módulos son extensos o complejos se dividen para probar

objetivamente partes más pequeñas, este nivel de pruebas es el más común y el que primero debe utilizarse.

Inicialmente, la prueba se centra en cada módulo individualmente, asegurando que funcionan adecuadamente como una unidad. De ahí el nombre de prueba de unidad. La prueba de unidad hace un uso intensivo de las técnicas de prueba de Caja Blanca, ejercitando caminos específicos de la estructura de control del módulo para asegurar un alcance completo y una detección máxima de errores. A continuación, se deben ensamblar o integrar los módulos para formar el paquete de software completo. (S.Pressman, 1998)

Para que una prueba unitaria sea óptima se deben cumplir los siguientes requisitos:

- **Automatizable**: no debería requerirse la intervención manual.
- **Completas**: deben cubrir la mayor cantidad de código.
- **Repetibles o Reutilizables**: no se deben crear pruebas que sólo puedan ser ejecutadas una sola vez.
- **Independientes**: la ejecución de una prueba no debe afectar la ejecución de otra.
- **Profesionales**: las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc.

El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el código debe satisfacer. Estas pruebas aisladas proporcionan cinco ventajas básicas:

1. **Fomentan el cambio**: Las pruebas unitarias facilitan que se cambie el código para mejorar su estructura (lo que se ha llamado **refactorización**), permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
2. **Simplifican la integración**: Permiten llegar a la fase de integración con un alto grado de seguridad de que el código está funcionando correctamente. De esta manera se facilitan las pruebas de integración.

3. **Documentan el código**: Las propias pruebas son documentación del código debido que ahí se puede ver cómo utilizarlo.
4. **Separación de la interfaz y la implementación**.
5. **Los errores están más acotados y son más fáciles de localizar**: Debido a que las pruebas unitarias pueden desenmascararlos.

Es importante tener en cuenta que las pruebas unitarias no descubrirán todos los errores del código. Por definición, sólo prueban las unidades por sí solas. Por lo tanto, no descubrirán errores de integración, problemas de rendimiento y otros problemas que afectan a todo el sistema en su conjunto. Además puede no ser trivial anticipar todos los casos especiales de entradas que puede recibir en realidad la unidad de programa bajo estudio. Las pruebas unitarias sólo son efectivas si se usan en conjunto con otras pruebas de software.

1.3.2- Pruebas de Integración.

La prueba de Integración se dirige a todos los aspectos asociados con el doble problema de verificación y de construcción del programa. Durante la integración, las técnicas que más prevalecen son las de diseño de casos de prueba de caja negra, aunque se pueden llevar a cabo algunas pruebas de caja blanca con el fin de asegurar que se cubren los principales caminos de control. (Berzal Galiano)

La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. (S.Pressman, 1998)

Se comprueba la compatibilidad y funcionalidad de los interfaces entre las distintas partes que componen un sistema, estas partes pueden ser módulos, aplicaciones individuales, aplicaciones cliente/servidor, etc. Este tipo de pruebas es especialmente relevante en aplicaciones distribuidas. (Raja Prado, 2007)

Estas pruebas tienen por objetivo verificar el funcionamiento de dos o más módulos. Se deben poner en práctica desde la creación de dos módulos que interactúen entre sí. En el caso de que se necesiten más

de dos módulos para efectuar las pruebas, deberán generarse simples emuladores de módulos que entreguen los datos esperados para la prueba individual de cada uno.

Es posible plantear las pruebas de integración desde dos puntos de vista: **estructural** o **funcional**:

Las **pruebas estructurales de integración** son similares a las pruebas de CB; pero trabajan a un nivel conceptual superior. En lugar de referirse a sentencias del lenguaje, se refrieren a llamadas entre módulos. Se tratan de identificar todos los posibles esquemas de llamadas y ejercitarlos para lograr una buena cobertura de segmentos o de ramas.

Las **pruebas funcionales de integración** son similares a las pruebas de CN. Se trata de encontrar fallos en la respuesta de un módulo cuando su operación depende de los servicios prestados por otro(s) módulo(s). Según se van acercando al sistema total, estas pruebas se van basando más y más en la especificación de requisitos del usuario.

Las pruebas finales de integración cubren todo el sistema y pretenden cubrir plenamente la especificación de requisitos del usuario. Además, a estas alturas ya suele estar disponible el manual de usuario, que también se utiliza para realizar las pruebas hasta lograr una cobertura aceptable.

1.3.3- Pruebas de Sistema.

Las Pruebas de Sistema verifican que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. Está constituida por varios tipos de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema. Algunas de estos tipos de pruebas son:

- **Prueba de validación:** Proporciona una seguridad final de que el software satisface los requerimientos funcionales y de rendimiento. Además, valida los requerimientos establecidos comparándolos con el sistema que ha sido construido. Durante la validación se usan exclusivamente técnicas de prueba de CN.
- **Prueba de recuperación:** Fuerza un fallo del software y verifica que la recuperación se lleva a cabo apropiadamente.

- **Prueba de seguridad:** Se determinan los niveles de permisos de usuarios, las operaciones de acceso al sistema y el acceso a datos.
- **Prueba de resistencia:** Determinan hasta donde puede soportar el programa determinadas condiciones extremas.
- **Prueba de Rendimiento:** Determinan los tiempos de respuesta, el espacio que ocupa el módulo en disco o en memoria, el flujo de datos que genera a través de un canal de comunicaciones, etc.
- **Prueba de Robustez:** Determinan la capacidad del programa para soportar entradas incorrectas.
- **Prueba de Usabilidad:** Se determina la calidad de la experiencia de un usuario en la forma en la que éste interactúa con el sistema, se considera la facilidad de uso y el grado de satisfacción del usuario.
- **Prueba de instalación:** Se centra en comprobar que el sistema desarrollado se puede instalar en diferentes configuraciones hardware y software y bajo condiciones excepcionales, por ejemplo con espacio de disco insuficiente o con continuas interrupciones.

1.3.4- Pruebas de Aceptación.

Son las que hará el cliente, se determina que el sistema cumple con lo deseado y se obtiene la conformidad del cliente. (Raja Prado, 2007)

Las pruebas de aceptación son realizadas principalmente por los usuarios, con el apoyo del equipo de desarrollo. El propósito es confirmar que el sistema está terminado, que desarrolla puntualmente las necesidades de la organización y que es aceptado por los usuarios.

Estas son básicamente pruebas funcionales, sobre el sistema completo, y buscan una cobertura de la especificación de requisitos y del manual del usuario. No se realizan durante el desarrollo, pues sería impresentable al cliente; sino que se realizan sobre el producto terminado y entregado, o pudiera ser una versión del producto o una iteración funcional pactada previamente con el cliente.

La experiencia muestra que aún después del más cuidadoso proceso de pruebas por parte del desarrollador, quedan una serie de errores que sólo aparecen cuando el cliente comienza a usarlo.

Una prueba de aceptación puede ir desde un informal caso de prueba hasta la ejecución sistemática de una serie de pruebas bien planificadas. De hecho, las pruebas de aceptación pueden tener lugar a lo largo de semanas o meses, descubriendo así errores latentes o escondidos que pueden ir degradando el funcionamiento del sistema. Estas pruebas son muy importantes, ya que definen el paso nuevas fases del proyecto como el despliegue y mantenimiento. (S.Pressman, 1998)

Si el software se desarrolla como un producto que va a ser usado por muchos clientes, no es práctico realizar pruebas de aceptación formales para cada uno de ellos. La mayoría de los desarrolladores de productos de software llevan a cabo un proceso denominado prueba **alfa** y **beta** para descubrir errores que solo el usuario final puede descubrir.

La prueba alfa se lleva a cabo por un cliente en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y los problemas de uso. Las pruebas alfa se llevan a cabo en un entorno controlado.

La prueba beta se lleva a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador. El cliente registra todos los problemas (reales o imaginarios) que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador. Como resultado de los problemas informados durante la prueba beta, el desarrollador del software lleva a cabo modificaciones y así prepara una versión del producto de software para toda la clase de clientes.

Este nivel de prueba es el último que se le aplica al software, si hasta el momento se ha llevado a cabo el proceso de pruebas cabalmente y como debe ser, este nivel no debe presentar dificultades demasiado grandes para los desarrolladores, en caso contrario existe la posibilidad de que el producto no pueda ser liberado y haya que corregir los errores o inconformidades encontrados a última hora.

El beneficio mayor que suponen estas pruebas recae en los clientes porque las mismas les dan la oportunidad de probar el programa en escenarios críticos antes de aceptarlo y comercializarlo o realizar con él la actividad para la cual se haya diseñado.

1.4- Estrategia de Pruebas del Software.

Toda prueba a realizarse debe ser planificada y definida detalladamente, esta es la única manera de contribuir a que la misma se realice de manera correcta y se obtengan resultados reales en el proceso de pruebas.

Una estrategia de prueba de software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que dan como resultado una correcta construcción del software. La estrategia proporciona un mapa que describe los pasos que hay que llevar a cabo como parte de la prueba, cuando se deben planificar y realizar esos pasos, y cuanto esfuerzo, tiempo y recursos se van a requerir. Por tanto cualquier estrategia de prueba debe incorporar la planificación de la prueba, el diseño de casos de prueba, la ejecución de las pruebas y la agrupación y evaluación de los datos resultante. (S.Pressman, 1998)

Cuando se realiza la estrategia de pruebas es importante tener en cuenta en todo momento las características propias del producto a probar. La misma debe ser flexible, promover la creatividad y la adaptabilidad necesaria para que se familiarice fácilmente a las condiciones del sistema. Al mismo tiempo dicha estrategia debe ser suficientemente fuerte, permitiendo así que se le pueda dar seguimiento a la planificación y la gestión a medida que avanza el proyecto.

Entre las principales características de las estrategias de prueba están:

- La prueba comienza en el nivel de módulo y trabaja “hacia afuera”.
- En diferentes puntos son adecuadas a la vez distintas técnicas de prueba.
- La prueba la realiza la persona que desarrolla el software y (para grandes proyectos) un grupo de pruebas independiente.
- La prueba y la depuración son actividades diferentes.

La estrategia de prueba del software debe incluir pruebas de bajo nivel que verifiquen que todos los pequeños segmentos de código fuente se han implementado correctamente, así como pruebas de alto nivel que validen las principales funciones del sistema frente a los requisitos del cliente.

Entre los principales objetivos de crear esta guía está: planificar las pruebas necesarias en cada iteración, diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, cómo realizar las pruebas y creando, si es posible, componentes de prueba ejecutables para automatizar las pruebas. Otro de los objetivos es realizar diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Los productos de desarrollo de software en los que se detectan defectos son probados de nuevo y posiblemente devueltos a otra etapa, de forma que los defectos puedan ser arreglados.

IEEE ofrece el estándar 829. Esta es una plantilla que sirve de guía para la realización de los artefactos que se generan a lo largo del proceso de pruebas. Contiene una serie de puntos que permiten diseñar una estrategia propia, que se acomode a las necesidades específicas.

Actualmente es una ventaja estratégica para cualquier negocio contar con una estrategia que guie todo su proceso de pruebas; este método, en muchos casos, proporciona a las diferentes organizaciones una posibilidad real de mejorar el rendimiento y la calidad del producto producido.

1.5- Normas y Estándares de Calidad.

Las normas son reglas metodológicas y modelos adoptados por organizaciones nacionales o internacionales que se dedican al establecimiento de estándares. Estas organizaciones se apoyan en foros de expertos que se suelen constituir en comités muy calificados. (M. Minguet Melián, et al., 2003)

El sistema que conforma la normalización está integrado por una conceptuación, una diferenciación y un ordenamiento de una serie de principios y procedimientos para establecer unidades y definiciones, métodos de toma y conservación de muestras, métodos de ensayo, además de especificaciones cualitativas y códigos de práctica.

Según la Organización Internacional de Normalización **(ISO)**, la **Normalización** es la actividad que tiene por objeto establecer, ante problemas reales o potenciales, disposiciones destinadas a usos comunes y repetidos, con el fin de obtener un nivel de ordenamiento óptimo en un contexto dado, que puede ser tecnológico, político o económico.

La **Normalización** o E**standarización** es la redacción y aprobación de normas que se establecen para garantizar el acoplamiento de elementos construidos independientemente, así como garantizar el repuesto en caso de ser necesario, garantizar la calidad de los elementos fabricados y la seguridad de funcionamiento.

Las normas propuestas por las diferentes organizaciones, son en todos los casos recomendaciones, no poseen para nada el rango de ley. Las empresas interesadas en seguir determinada norma suelen demostrarlo mediante el correspondiente certificado de la organización generadora del estándar, y así evitarse cuantiosos gastos en demostrar que cumple las normativas en todos los productos o en propuestas de futuros clientes.

La normalización o estandarización persigue fundamentalmente tres objetivos:

- **Simplificación:** Se trata de reducir los modelos quedándose únicamente con los más necesarios.
- **Unificación:** Para permitir el intercambio a nivel internacional.
- **Especificación:** Se persigue evitar errores de identificación creando un lenguaje claro y preciso.

Entre las diferentes organizaciones más prestigiosas que dictan estas normas se encuentra el Comité Europeo de Normalización **(CEN)**, el Instituto de Ingenieros Eléctricos y Electrónicos **(IEEE)**, la Unión Internacional de Telecomunicaciones **(ITU)** y la más seguida en la informática, la Organización Internacional para la Estandarización **(ISO)**, que ha sido adoptada por más de 130 países para su uso y se está convirtiendo en el medio principal con el que los clientes pueden juzgar la competencia de un desarrollador de software.

La sociedad actual es la principal promotora e impulsora del tema relacionado con la promoción de estándares y procesos de normalización en la calidad de software. En correspondencia con esto las

empresas que producen software muestran más atención y dedicación al tema de la certificación de los productos, la cual se ha convertido en un sello de calidad que favorece al mercado y brinda numerosas ventajas frente a la competencia que existe en el mercado.

1.6- Metodologías de Desarrollo de Software.

El diseño del software, al igual que los enfoques de diseño de ingeniería en otras disciplinas, va cambiando continuamente a medida que se desarrollan métodos nuevos, análisis mejores y se amplía el conocimiento. Las metodologías de diseño del software carecen de la profundidad, flexibilidad y naturaleza cuantitativa que se asocian normalmente a las disciplinas de diseño de ingeniería más clásicas. Sin embargo, sí existen métodos para el diseño del software; también se dispone de calidad de diseño y se pueden aplicar notaciones de diseño. (S.Pressman, 1998)

Todo desarrollo de software es riesgoso y difícil de controlar, pero si no se aplica una metodología, al final, lo más probable a obtener son clientes insatisfechos con el resultado y desarrolladores aún más insatisfechos. La dificultad propia del desarrollo de software, y su impacto en el negocio, han puesto de manifiesto las ventajas, y en muchos casos la necesidad, de aplicar una metodología formal para llevar a cabo los proyectos de este tipo. El objetivo es convertir el desarrollo de software en un proceso formal, con resultados predecibles, que permitan obtener un producto final de alta calidad, que satisfaga las necesidades y expectativas del cliente.

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software. En ellas se va indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado. Se divide el proyecto en etapas, en caso de que sea necesario. Se definen las tareas específicas a realizarse en cada fase y las personas que deben participar en el desarrollo de estas las actividades, además del rol que deben desempeñar. Se determinan las restricciones que deben aplicarse así como las técnicas y herramientas posibles a emplear.

Las metodologías se clasifican de la siguiente forma:

- **Estructuradas**.

- Orientadas a procesos.
- Orientadas a datos.
- Mixtas.

- **No estructuradas**.
  - Orientadas a objetos.
  - Sistemas de tiempo real.

Una metodología de desarrollo de software se refiere, además, a un framework (estructura de soporte) que es usado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información. A lo largo del tiempo, una gran cantidad de métodos han sido desarrollados, diferenciándose por su fortaleza y debilidad. Estos frameworks son a menudo vinculados a algún tipo de organización, que además desarrolla, apoya su uso y promueve la metodología. Las metodologías en general imponen un proceso coordinado y disciplinado sobre el desarrollo de software con el fin de hacerlo más eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar, inspirado por otras disciplinas de la ingeniería.

Hasta hace poco el proceso de desarrollo llevaba asociado un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso. (H. Canós, et al.)

1.6.1- Metodologías Tradicionales.

Históricamente, las metodologías tradicionales han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes. (H. Canós, et al.)

Actualmente las metodologías tradicionales o ingenieriles no se distinguen por ser muy exitosas, y en muchos casos carecen de popularidad. Las críticas más frecuentes coinciden en son burócratas y no

permiten adaptarse al tipo de proyecto que se esté desarrollando, además, si los pasos definidos en las mismas son seguidos, la mayoría de las veces el desarrollo se retarda.

Estas propuestas más tradicionales se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. (Jacobson, et al., 2000)

Algunas de las metodologías pesadas más conocidas son: **Waterfall y Microsoft Solutions Framework for CMMI.**

Sin embargo han surgido propuestas de metodologías tradicionales que inciden en distintas dimensiones del proceso de desarrollo. Entre ellas, propuestas centradas específicamente en el control del proceso. Estas han demostrados ser bastante efectivas y necesarias en gran número de proyectos, sobre todo en aquellos de gran tamaño respecto a tiempo y recursos.

En sentido general la experiencia ha demostrado que las metodologías tradicionales no ofrecen una buena solución para proyectos donde el entorno es volátil y donde los requisitos no se conocen con exactitud, porque no están pensadas para trabajar con incertidumbre.

Aplicar metodologías tradicionales obliga a forzar al cliente a que tome la mayoría de las decisiones al principio. Luego, el coste del cambio de una decisión tomada puede llegar a ser muy elevado si se aplican metodologías tradicionales.

1.6.2- Metodologías Agiles.

Las metodologías ágiles surgen como una extensión a las metodologías tradicionales para mejorar el desarrollo de sistemas, según el tipo de proyecto y empresa, añadiendo y optimizando las practicas de desarrollo de software.

Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo se resignan a prescindir del "buen hacer" de la ingeniería del software, asumiendo el riesgo que ello conlleva. En este escenario, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las metodologías ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto. (H. Canós, et al.)

El tema de las metodologías agiles es bastante reciente dentro de la ingeniería de software, nace en febrero de 2001, y está muy de moda al gozar de la preferencia de muchos de los desarrolladores actualmente.

Estas metodologías, en general, se centran especialmente en el factor humano o el producto software, dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

Entre las metodologías agiles más exitosa están: **Extreme Programing (XP), SCRUM, Crystal Methodologies, Dynamic Systems Development Method** (DSDM), entre otras.

Actualmente uno de los temas más debatidos por las empresas productoras de software, en cuanto al uso de las metodologías, es si se adoptan las tradicionales o las ágiles, para ello se comparan ambas en cuanto a sus características y diferencias *(Ver Anexo2).*

Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo, o convencimiento, no las ven como alternativa para las metodologías tradicionales. Su objetivo es esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de desarrollo de

software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

1.6.3- RUP.

En primer lugar, el Proceso Unificado es un proceso de desarrollo de software. Un proceso de desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos de usuario en un sistema software. Sin embargo, el Proceso Unificado es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes aéreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de procesos. (Jacobson, et al., 2000)

RUP es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo, a través del **UML** (Lenguaje Unificado de Modelado), y trabajo de muchas metodologías utilizadas por los clientes. Entre sus características se puede destacar que unifica los mejores elementos de metodologías anteriores y está preparado para desarrollar grandes y complejos proyectos. Se rige por el paradigma Orientado a Objetos y utiliza el UML para representar todos los esquemas de un sistema software, y es además una parte fundamental de dicho proceso.

En su modelación define como sus principales elementos:

- **Trabajadores** (“quién”). Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.
- **Actividades** (“cómo”). Es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.
- **Artefactos** (”qué”). Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.
- **Flujo de actividades** (“Cuándo”). Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

Las tres características fundamentales de RUP son:

- **Dirigido por Casos de Uso (CU).** Un CU es un fragmento de la funcionalidad del sistema que proporciona al usuario un resultado importante. Representan los requisitos funcionales. Los CU reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los CU guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso.

- **Centrado en la Arquitectura.** La arquitectura incluye los aspectos estáticos y dinámicos más significativos del sistema. Es una vista del diseño completo con las características más importantes resaltadas, dejando el detalle al lado. La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los CU relevantes desde el punto de vista de la arquitectura.

- **Iterativo e Incremental.** Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos, al crecimiento del producto. RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo. Es práctico dividir el trabajo en partes más pequeñas o mini-proyectos. Cada mini-proyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en los flujos de trabajo, y los incrementos, al crecimiento del producto. Cada iteración se realiza de forma planificada es por eso que se dice que son mini-proyectos.

El Proceso Unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo constituye la versión de un producto para los clientes. Cada uno de estos ciclos consta de cuatro fases: **Inicio, Elaboración, Construcción, y Transición** *(Ver Anexo 3)***.** Cada fase se divide a su vez en iteraciones.

En el **Inicio** se define la visión, los objetivos y el alcance del proyecto, tanto desde el punto de vista funcional como del técnico, obteniéndose como uno de los principales resultados una lista de los CU y una lista de los factores de riesgo del proyecto. El principal esfuerzo está radicado en los flujos de **"Modelamiento del Negocio"** y el **"Análisis de Requerimientos"**. Es la única fase que no necesariamente culmina con una versión ejecutable, si bien muchas veces se desarrollan las interfaces con el usuario, o se prueban algunos aspectos técnicos críticos.

La fase de **Elaboración** tiene como principal finalidad completar el análisis de los CU y definir la arquitectura del sistema. En esta etapa se busca eliminar los principales riesgos técnicos.

La fase de **Construcción** está compuesta por un ciclo de varias iteraciones, en las cuales se van incorporando sucesivamente los casos de uso, de acuerdo a los factores de riesgo del proyecto. Este enfoque permite por ejemplo contar en forma temprana con versiones el sistema que satisfacen los principales Casos de Uso. Los cambios en los requerimientos no se incorporan hasta el inicio de la próxima iteración.

La última fase es la de **Transición,** su finalidad es poner el producto en manos de los usuarios finales, para lo que se requiere desarrollar nuevas versiones actualizadas del producto, completar la documentación, entrenar al usuario en el manejo del producto, y en general, tareas relacionadas con el ajuste, configuración, instalación y facilidad de uso del producto.

RUP además está dividido en 9 flujos de trabajo, 6 de ellos ingenieriles y 3 de apoyo. Los mismos son:

- **Modelar el negocio:** Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.

- **Requerimientos:** Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.

- **Análisis y diseño:** Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.

- **Implementación:** Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.

- **Prueba (Testeo):** Busca los defectos a los largo del ciclo de vida.

- **Instalación o despliegue:** Produce una liberación (release) del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.

- **Administración del proyecto:** Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.

- **Administración de configuración y cambios:** Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.

- **Ambiente:** Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

Cada proyecto y producto software es diferente. Todos tienen prioridades, requerimientos y tecnologías diferentes. Sin embargo, en todos los proyectos, se debe minimizar el riesgo, garantizar la predictibilidad de los resultados y entregar software de calidad superior a tiempo. RUP es una plataforma flexible de procesos de desarrollo de software que ayuda proveyendo guías consistentes y personalizadas de procesos para todo el equipo de proyecto.

Luego de haber realizado una investigación sobre el tema de las metodologías de desarrollo de software en la actualidad, y de haber estudiado las características de las metodologías existentes más usadas, se decidió utilizar RUP como metodología para el desarrollo del software propuesto, debido a que el mismo es una infraestructura flexible de desarrollo de software que proporciona prácticas recomendadas probadas y una arquitectura configurable. Es un Proceso práctico que permite seleccionar fácilmente el conjunto de componentes de proceso que se ajustan a las necesidades específicas del proyecto. Haciendo uso de él se podrán alcanzar resultados predecibles unificando el equipo con procesos comunes que optimicen la comunicación y creen un entendimiento común para todas las tareas, responsabilidades y artefactos. Además mantiene al equipo enfocado en producir incrementalmente software operativo a tiempo, con las características y la calidad requerida.

1.7- Arquitecturas de Software.

En los inicios de la informática, la programación se consideraba un arte y se desarrollaba como tal, debido a la dificultad que entrañaba para la mayoría de las personas. Con el tiempo se han ido descubriendo y desarrollando formas y guías generales, en base a las cuales se puedan resolver los problemas. A estas, se les ha denominado Arquitectura de Software, porque, a semejanza de los planos de un edificio o construcción, indican la estructura, funcionamiento e interacción entre las partes del software.

1.7.1- Arquitectura Cliente-Servidor.

La arquitectura Cliente-Servidor representa la combinación de sistemas que pueden colaborar entre sí para dar a los usuarios toda la información que ellos necesiten sin que tengan que saber donde está ubicada. Este tipo de organización se basa en que, entre todos los ordenadores que están en la red, unos ofrecen servicios (los llamados servidores) y otros usan esos servicios (los denominados clientes).

Las aplicaciones de gestión basadas en la arquitectura cliente-servidor permiten mayor acceso a la información. Los cambios realizados en las plataformas de los Clientes o de los Servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.

Entre las ventajas de esta arquitectura están:

- El cliente y el servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes.
- Las funciones de cliente y servidor pueden estar en plataformas separadas, o en la misma plataforma.
- Un servidor da servicio a múltiples clientes en forma concurrente. Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los clientes o de los servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.
- Un sistema de servidores realiza múltiples funciones al mismo tiempo que presenta una imagen de un solo sistema a las estaciones clientes. Esto se logra combinando los recursos de cómputo que se encuentran físicamente separados en un solo sistema lógico, proporcionando de esta manera el servicio más efectivo para el usuario final.

1.7.2- Arquitectura Tres Capas.

Las aplicaciones con arquitecturas en capas, constituyen uno de los estilos que aparecen con mayor frecuencia. De forma general, se define el estilo en capas como una organización jerárquica, tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. *(Ver Anexo4)*

Las diferentes capas son:

1.- Capa de presentación: es la que ve el usuario, también llamada "capa de usuario", presenta el sistema al usuario, le comunica la información y captura la información dando un mínimo de proceso, realiza un filtrado previo para comprobar que no hay errores de formato. Esta capa se comunica únicamente con la capa de negocio.

2.- Capa de negocio: es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio, e incluso de lógica del negocio, pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él.

3.- Capa de datos: es donde residen los datos y es la encargada de acceder a ellos. Está formada por uno o más gestores de bases de datos que realizan todo las operaciones de almacenamiento. Reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Las ventajas de esta arquitectura en capas son:

- El estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual, permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
- El estilo admite muy naturalmente optimizaciones y refinamientos.
- Proporciona amplia reutilización.

La ventaja principal de este estilo, es que el desarrollo se puede llevar a cabo en varios niveles. En caso de algún cambio solo se trabaja en el nivel requerido sin tener que revisar entre código mezclado.

1.8- Tecnologías Actuales.

Cuando se habla de producción de software no se pueden dejar de mencionar las tecnologías actuales más utilizadas para el desarrollo del mismo. Entre ellas están los Servidores Web, lenguajes de programación y Sistema Gestores de Bases de Datos.

1.8.1- Servidores WEB.

Un servidor Web es un programa que implementa el Protocolo de Transferencia de Hipertexto **(HTTP)** basado en arquitectura cliente-servidor. Este protocolo está diseñado para transferir hipertextos, páginas

Web o páginas de Lenguaje de Marcas de Hipertexto **(HTML)**, textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproductores de sonidos.

Uno de los servidores Web más populares en el mercado y el más utilizado actualmente, es **Apache**, de código abierto y gratuito, disponible para GNU/Linux y Windows, entre otros. Apache presenta entre otras características mensajes de error altamente configurables, bases de datos de autenticación y negociado de contenido. Tiene amplia aceptación en la red. En el 2005, se convirtió en el más usado, siendo el servidor HTTP del 70% de los sitios Web en el mundo y creciendo aún su cuota de mercado.

XAMPP es un servidor independiente de plataforma, software libre, que consiste principalmente en la base de datos MySQL, el servidor Web Apache y los intérpretes para lenguajes de script: PHP y Perl. El nombre proviene del acrónimo de **X** (para cualquiera de los diferentes sistemas operativos), **A**pache, **M**ySQL, **P**HP, **P**erl. El programa está liberado bajo la licencia GNU y actúa como un servidor Web libre, fácil de usar y capaz de interpretar páginas dinámicas. Actualmente XAMPP está disponible para Microsoft Windows, GNU/Linux, Solaris, y MacOS X.

Luego de un estudio detallado sobre los servidores web, se decidieron seleccionar los servidores Apache y XAMPP para la realización de la aplicación propuesta. Debido a que ambos son software libre y multiplataforma.

1.8.2- Lenguaje de Programación de la parte del Cliente.

En el mundo de Internet existen muchas tecnologías que se pueden emplear para programar los clientes web, como ActiveX, Applet, Flash, VRML, etc., pero sólo dos son las tecnologías más extendidas y se pueden considerar "el estándar": HTML, Java Script y CSS.

HTML: Es el lenguaje de marcado predominante para la construcción de páginas web. Usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. Este lenguaje está hecho de etiquetas y atributos que trabajan conjuntamente para dar alguna característica específica a la página web; el navegador interpreta dichas etiquetas y atributos y

las despliega en la pantalla, solo se limita a describir la estructura y el contenido de un documento y no el formato de la página y su apariencia.

Java Script: Lenguaje que permite interactuar con el navegador de manera dinámica y eficaz, proporcionando a las páginas web mayor dinamismo. El advenimiento de Java Script ha resuelto de manera fácil y elegante la mayoría de los problemas con los que se enfrenta el diseñador de páginas Web, referente a la programación. Sus requerimientos son relativamente sencillos, es un lenguaje cuyos códigos se interpretan en el navegador del cliente, sin tener que ir y venir del cliente al servidor actualizando la información.

CSS: Las hojas de estilo en cascada (*Cascading Style Sheets*, CSS) son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

Estos tres lenguajes serán los utilizados en la implementación de la parte del cliente de la aplicación a realizar.

1.8.3- Lenguaje de Programación para el Servidor.

En este lado se encuentran diferentes lenguajes como: PERL, ASP, JSP, PHP. Todos ellos permiten modelar el negocio de acuerdo a los requerimientos y niveles de manejo de datos logrando la correcta comunicación y seguridad entre los módulos del negocio.

PHP: Las siglas: PHP son un acrónimo recursivo que significa Pre-Procesadores de Hipertexto, es un lenguaje de programación que sirve principalmente para proporcionar características dinámicas a una página Web. Se interpreta y ejecuta directamente en el servidor en el que está albergada la página Web. El visitante únicamente recibe el resultado buscado por el código en el que está escrito.

Es uno de los lenguajes del lado del servidor más extendidos en la web. Se trata de un lenguaje de creación relativamente reciente que ha tenido una gran aceptación en la comunidad mundial debido a la

potencia y simplicidad que lo caracterizan. Permite embeber sus pequeños fragmentos de código dentro de la página HTML y realizar determinadas acciones de una forma fácil y eficaz. Está publicado bajo Licencia PHP, y la Fundación de Software Libre considera esta licencia como software libre.

Luego de un estudio detallado sobre los lenguajes del lado del servidor, se decidió seleccionar PHP para la realización de la aplicación propuesta.

1.8.4- Sistema Gestor de Base de Datos.

Los gestores de base de datos son sistemas formados por un conjunto de datos y un paquete de software para la gestión del mismo. Se controla el almacenamiento de datos redundantes. Los datos resultan independientes de los programas que los usan, se almacenan las relaciones entre los datos junto con éstos y se puede acceder a ellos de diversas formas.

Para la construcción de aplicaciones de gestión se destacan por su eficiencia gestores como: Oracle, considerado uno de los más potentes, MySQL, un sistema de gestión de base de datos relacional, multi-hilo y multiusuario con más de seis millones de instalaciones y PostgreSQL, considerado un Sistema de Gestión de Bases de Datos de código abierto (gratuito y con código fuente disponible) siendo uno de los más avanzados del mundo.

MySQL: es una base de datos rápida y fiable que se integra a la perfección con PHP y que resulta muy adecuada para aplicaciones dinámicas basadas en Internet.

Ventajas:

- Mayor rendimiento.
- Mejores utilidades de administración.
- Integración perfecta con PHP.
- Sin límites en los tamaños de los registros.
- Mejor control de acceso de usuarios.

MySQLAB desarrolla MySQL como software libre en un esquema de licenciamiento dual. Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero aquellas empresas que quieran incorporarlo en productos privativos deben comprar una licencia específica que les permita este uso.

La aplicación a desarrollar no será en ningún modo privativa, por tanto se decidió usar MySQL como gestor de base de datos.

1.9- Herramientas para el desarrollo.

Las fases de un desarrollo web, así como los lenguajes de programación usados, son muy extensos y variados, y por ello se necesitan herramientas específicas para cada una de ellas.

En el desarrollo web existen herramientas para el diseño, la maquetación, la programación, y para la depuración. Todas son muy importantes, desde el Sistema Operativo hasta el comando más insignificante, y por ello se deben elegir la más adecuada a las necesidades y capacidades del proyecto.

1.9.1- Visual Paradigm for UML. Community Edition. Suite 3.4.

Es una herramienta **CASE** (Ingeniería de Software Asistida por Ordenador) para modelar UML, muy potente y de fácil uso. Permite dibujar todo tipo de diagramas UML, revertir código fuente a modelos UML y generar código fuente desde los diagramas UML.

Incluye los objetos más recientes de UML además de diagramas de casos de uso, diagramas de clase, diagramas de componentes, reversa instantánea para Java, C++, DotNet Exe/dll, XML Schema, y Corba IDL, ofrece soporte para Rational Rose, integración con Microsoft Visio, además permite generar reportes y documentación en HTML/PDF. Una de sus características fundamentales reside en que es multiplataforma.

Esta herramienta acelera el desarrollo de aplicaciones, ya que sirve de puente visual entre arquitectos, analistas y diseñadores de sistemas de información, haciendo el trabajo más fácil y dinámico. (EA 7.0)

Permite hacer diagramas UML, generación automática de código, sincronización entre modelos y código entre otras posibilidades. Posee una licencia Comercial, con versión libre para la comunidad y versiones de evaluación del resto. (EA 7.0)

La herramienta también automatiza tareas tediosas que pueden distraer a diseñadores del desarrollo. Ofrece entre sus beneficios la navegación intuitiva entre el código y el modelo visual.

Por todas las características mencionadas, y luego de hacer un estudio sobre las herramientas CASE más utilizadas se decide utilizarla en el desarrollo del proyecto.

1.9.2- Quanta Plus.

Quanta Plus es una herramienta de desarrollo web para el entorno de escritorio K (KDE: **K D**esktop **E**nvironment). Está diseñado para el desarrollo web y rápidamente se está convirtiendo en un editor maduro y con gran número de características positivas.

Proporciona una interfaz de múltiples documentos **(MDI)** poderosa e intuitiva para los desarrolladores web. Permite a través del uso de acciones personalizadas, guiones y barras de herramientas, automatizar muchas de las tareas a realizar. Con el uso de otros programas, como el Kommander, es posible extender Quanta Plus al agregarle otras funcionalidades. Las funciones incluyen interfaz de multi-documento, edición y plantillas de WYSIWYG (***W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et:* "lo que ves es lo que obtienes").

La característica principal consiste en que es una herramientas de desarrollo web gratuita disponible en el mercado, su filosofía de código abierto y sus altamente personalizables secuencias de comandos (scripts) le hacen la herramienta perfecta para aquellos desarrolladores que prefieren tener control total sobre su ambiente de desarrollo. Quanta Plus es una alternativa de código abierto para editores web como el Adobe Dreamweaver CS3 y para editores HTML como el Microsoft Frontpage 2003.

Luego de realizar un estudio detallado de las herramientas existentes en estos momentos para el desarrollo web, se decidió utilizar Quanta Plus, debido a la filosofía de código abierto que sigue y a las demás características que facilitan el trabajo a los desarrolladores.

1.9.3- Aptana Studio Community Edition.

Es un **IDE**, por sus siglas en inglés (**I**ntegrated **D**evelopment **E**nvironment**:** Ambiente de desarrollo Integrado) de desarrollo para aplicaciones de la web 2.0, gratuito, código abierto, con soporte Ajax, PHP, Ruby on Rails, Adobe Air, iPhone, etc. Con Aptana se facilita en desarrollo integrado de Ajax con las tecnologías emergentes.

Está basado en el conocido entorno de desarrollo Eclipse, también Código Abierto. Pero mientras que Eclipse está enfocado al desarrollo para Java, Aptana Studio es una distribución enfocada en el desarrollo web, con soporte a HTML, CSS y Javascript, así como opcionalmente a otras tecnologías mencionadas como PHP, Adobe Air o Ruby on Rails. Aptana Studio está disponible como una aplicación independiente o se puede adicionar a Eclipse.

Aptana Studio Community Edition: Es la versión gratuita, que contiene la mayoría de las funcionalidades del IDE, como edición, sincronización y administración de proyectos. Con soporte para todas las tecnologías antes mencionadas.

Luego de realizar una búsqueda, estudio y análisis de los IDEs de desarrollo web más utilizados a nivel mundial, y de compararlos en cuanto a aspectos como licencias de uso y tipos de software, se decidió utilizar Aptana en el desarrollo del presente trabajo.

1.10- Herramientas para la Gestión del Proceso de Pruebas.

El proceso de pruebas dentro de los flujos de trabajo establecidos por RUP está presente en las tres últimas fases del proceso de desarrollo del software. Con el aumento de la complejidad del software este proceso también ha evolucionado hasta convertirse en uno de los más importantes e imprescindibles.

Actualmente se hace cada vez más necesario contar con herramientas para el control y la gestión del proceso de pruebas de software. Llega un momento en el cual a los encargados de esta tarea se les hace difícil tener el control necesario sobre los procedimientos y documentos generados a lo largo de las iteraciones que se realizan.

A continuación se hace referencia a algunas de las herramientas más utilizadas para la gestión del proceso de pruebas:

QaTraq Professional: Herramienta de gestión de casos de prueba, perteneciente a la organización Testmanagement. Proporciona una ubicación central para todas las pruebas. Los documentos están vinculados a través del proceso, por lo tanto, sus secuencias de comandos de prueba están relacionadas con el plan de pruebas pertinentes, casos de prueba a las pruebas adecuadas, scripts, etc. Puede ser instalado en las plataformas Linux, Unix y Windows.

Esta herramienta está caracterizada por:

1. Mejora de la coordinación entre las pruebas, los jefes de equipo y probadores.
2. Una base de conocimientos de la información a compartir entre el equipo de prueba.
3. Un cauce formal de las pruebas a seguir por desarrolladores y probadores.
4. Instantáneos informes basados en casos de prueba creados y ejecutados.
5. Informes y estadísticas de las listas de las pruebas más eficaces.

Sin embargo QaTraq Professional es software propietario. Tiene una licencia de usuario concurrente, es posible comprar licencias adicionales en cualquier momento y agregarlas a la instalación. También se facilita una licencia de 30 días de prueba.

Otra de las herramientas utilizadas es **TestDirector,** de Mercury Interactive. Es una herramienta de gestión que permite coordinar los esfuerzos de evaluación, desde la planificación o ejecución hasta la organización y gestión del completo proceso de testeo. Facilita la colaboración entre los distintos grupos de trabajo involucrados con el proceso de calidad, permitiendo el acceso a la información desde múltiples unidades de negocio o ubicaciones geográficas. Esta herramienta también forma parte del software propietario, cuenta con una licencia “Free to try” (libre para probar) de 30 días y además con 5 licencias de usuarios disponibles.

En el mercado existen además otras herramientas libre y de código abierto para la automatización y gestión del proceso de pruebas, sin embargo su número es bastante reducido y no se les dedican los recursos necesarios para el desarrollo de las mismas. Por otra parte, ninguna de ellas se adapta a las necesidades actuales del PHA.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

En este capítulo se abordan temas relacionados con las características del sistema, se realiza la presentación formal del mismo y la caracterización de la situación problemática. También es objetivo brindar una propuesta del sistema y realizar la descripción del negocio, que incluye los artefactos más importantes generados en dicho proceso. Por último se especifican los requerimientos del sistema, los casos de uso y demás artefactos.

2.1- Situación Problemática.

Actualmente, a nivel mundial, ha aumentado considerablemente la importancia que se le confiere a las pruebas de software y a todo el proceso en general. A pesar de que las mismas no garantizan la calidad directamente, se puede decir que lo hacen de forma indirecta, por lo que se les han comenzado a dedicar más recursos y esfuerzos en consecuencia con la complejidad del software que se esté desarrollando.

La UCI, al ser el pilar fundamental de producción de software de Cuba, no está exenta de este cambio. El PHA, perteneciente a la Facultad 5, se ha dado a la tarea de perfeccionar el área de CC debido a que actualmente el proceso de pruebas dentro del polo no se realiza de forma manual completamente. Esto trae como consecuencias que el mismo no esté controlado y por tanto sea más difícil de gestionar.

Actualmente no existe uniformidad entre los proyectos en cuanto al proceso de pruebas, el mismo se realiza dependiendo de las necesidades y prioridades del momento. No se ha definido hasta un método formal para realizar las solicitudes del proceso, por lo que en la mayoría de los casos el mismo comienza sin previa planificación y administración, sin contar con los requerimientos mínimos que se necesitan en cuanto a documentación y otros recursos.

Por otra parte, la documentación es un artefacto imprescindible en la mayoría de los procesos, y más cuando se trata de las pruebas que, por regla general, deben estar completa y formalmente reflejadas en documentos. Las afectaciones con respecto a esto residen en que, al no utilizarse ninguna herramienta para el control de versiones, se hace difícil y engorroso mantener organizada y actualizada toda la información que se genera y muchas veces la misma está duplicada, es innecesaria o simplemente se pierde.

Otro de los aspectos es la seguridad de los documentos, los mismos se guardan generalmente en computadoras o dispositivos de almacenamiento a los que otras personas también tienen libre acceso y pueden alterarla de alguna forma, existiendo la posibilidad de pérdida de los datos al no realizarse salvas de seguridad.

Los resultados de las pruebas, así como los demás documentos deben estar al alcance de todas las personas que los necesiten en el momento que los necesiten, sin embargo todo lo antes expuesto con respecto a la ubicación física de las mismas impide que esto se realice de manera correcta. Una de las características principales de las pruebas, es que sean repetibles, pero dada estas condiciones no es posible mantener la regla. Las pruebas de regresión también dependen en gran medida del acceso y la integridad de los documentos, por lo que estas se ven afectadas y comprometidas cuando aparece algún problema.

Todo lo antes expuesto perjudica en gran medida tanto a desarrolladores como probadores, los primeros no podrán determinar las debilidades del proceso de desarrollo y tomar las medidas pertinentes, los segundos no podrán reutilizar ni mejorar el trabajo realizado.

Por otra parte, la aplicación y adaptación de metodologías, estándares, modelos y estrategias en el proceso de pruebas no brindan los resultados óptimos, ni aún los esperados. Esta situación se mantendrá hasta tanto no se resuelvan los principales problemas que afectan directamente al proceso de pruebas. Otros recursos como el tiempo, las computadoras y las personas que se dedican al proceso también se ven afectados de alguna forma, y en muchos casos no son suficientes debido a la desorganización y falta de control.

Todos los problemas antes mencionados inciden directamente tanto en la calidad del proceso de desarrollo del software como en la calidad del producto final, al afectarse factores claves como el tiempo de terminación de los entregables, cumplimientos con los acuerdos pactados y mal manejo de los recursos con los que se cuenta. Si no se garantiza un proceso de pruebas con calidad entonces no será

posible garantizar que el producto cumpla con los requisitos acordados, satisfaga las expectativas del cliente y pueda competir en el mercado.

2.2- Propuesta del Sistema.

Ante las necesidades existentes, el equipo de desarrollo se propuso desarrollar una herramienta que cumplirá con las necesidades siguientes:

1. Debe ser multiplataforma.
2. Desarrollada bajo paradigmas y herramientas libres.
3. Estandarizar el proceso de pruebas de software en el PHA.
4. Ser flexible y poder adaptarse a otros procesos de pruebas diferentes a las del polo.
5. Gestionar y controlar el proceso de pruebas en los proyectos del PHA.
6. Gestionar eficientemente las Solicitudes de los proyectos que necesiten realizar el proceso de pruebas.
7. Crear y asignar un equipo de pruebas para cada proyecto solicitante.
8. Cumplir estrictamente con la estrategia establecida para el proceso de pruebas de software.
9. Gestionar eficientemente los artefactos necesarios en el proceso de pruebas.
10. Facilitar el trabajo con la documentación, definir un óptimo control de versiones y almacenarlas en un lugar seguro al que tenga acceso todo el que esté interesado en la misma.
11. Facilitar el acceso, mediante la red, a la documentación del proceso de pruebas que se esté realizando, además de otros documentos de consulta.
12. Mantener un registro de los resultados diarios de las pruebas que se estén realizando.
13. Realizar, cuando se desee, un registro general del proceso.

Esta herramienta, entre sus primeras tareas, establecerá una metodología estándar para todo el proceso de pruebas en la que se podrá definir la estrategia a seguir, en dependencia de los niveles, tipos, métodos y técnicas de pruebas necesarios en cada caso, garantizando así que todos los proyectos cumplan con las reglas establecidas y exista uniformidad, lo que significará un avance con respecto al proceso de normalización y estandarización necesario en el PHA.

Para llevar a cabo tales propósitos, inicialmente se ha propuesto el desarrollo de un sistema que, sin tener todas las funcionalidades con las que debe disponer en su versión final, será lo suficientemente estable como para dar a conocer de forma simple los requisitos esenciales. Este prototipo de aplicación web, llamado CCALPHA (Control de Calidad del PHA), será un gran paso de avance, con el se estará mejorando sustancialmente el área de CC, lo que contribuirá con el aseguramiento de la calidad de los productos y de los procesos de desarrollo de software. En próximas versiones se deberán incluir módulos para los demás procesos que incluye el CC.

2.3- Modelo de Negocio.

El flujo de trabajo de modelo del negocio se realiza en la fase de Inicio, definida por RUP. Da una visión de qué es necesario hacer para dar respuesta a las solicitudes del usuario, lo cual se logra definiendo los procesos, roles y responsabilidades de la organización en los modelos de casos de uso del negocio y de objetos. Este modelo del negocio brinda una vía natural para determinar los requerimientos del sistema de información.

Este flujo de trabajo tiene como objetivos principales comprender la estructura y la dinámica de la organización en la cual se va a implantar el sistema, comprender los problemas actuales de la organización e identificar las mejoras potenciales, derivar los requerimientos del sistema que va a soportar la organización y lograr una comunicación efectiva entre los usuarios y el equipo de proyecto con el objetivo de llegar a un entendimiento de lo que hay que hacer; es la clave del éxito en la producción de un software.

2.3.1- Actores del Negocio.

Un Actor del Negocio representa un individuo, grupo, entidad, organización, máquina o sistema de información externos con los que interactúa el negocio. Lo que se modela como actor es el Rol que se juega cuando se interactúa con el negocio para beneficiarse de sus resultados. (Callejas Cuervo, et al.)

Tabla 1: Actores del Negocio

Actor del Negocio	Descripción
Líder de Proyecto	Individuo que se encuentra al frente de un proyecto productivo, es el encargado de solicitar que se realice el proceso de pruebas en su proyecto. Solicita los reportes y resultados finales del proceso de pruebas.

2.3.2- Trabajadores del Negocio.

Un trabajador del negocio es quien define el comportamiento y las responsabilidades de un individuo que interactúa en el negocio realizando una o varias actividades, interactuando con otros trabajadores del negocio y manipulando las entidades del negocio. (Callejas Cuervo, et al.)

Tabla 2: Trabajadores del Negocio

Trabajadores del Negocio	Descripción
Administrador de Pruebas	Es el encargado de aprobar o no la solicitud de realización del proceso de pruebas. Selecciona los miembros del equipo de trabajo que realizarán el proceso de pruebas. Es el responsable del éxito de las pruebas. Este rol involucra el defensor de prueba y calidad, planificación y administración de recursos y resolución de problemas que impidan las pruebas.
Analista de Pruebas	Es el responsable de identificar y definir los niveles y tipos de pruebas requeridas, monitorear el progreso de la prueba, el resultado en cada ciclo de prueba y evaluar la calidad total experimentada, como un resultado de las actividades de prueba. Este rol lleva la responsabilidad para representar apropiadamente las necesidades de los stakeholder (grupos o individuos que son el público interesado) que no tienen representación regular y directa en el proyecto.

Diseñador de Casos de Pruebas.	Es el responsable de definir el método de prueba y asegurar su implementación exitosa. Identifica las técnicas apropiadas, herramientas e instrucciones para implementar las pruebas necesarias y encauzar los recursos correspondientes para las pruebas. Además es el responsable de realizar dos de los artefactos más importantes del proceso, el Plan de Pruebas y el Diseño de Casos de Pruebas
Probador	Es el responsable durante las actividades principales de las pruebas, lo cual incluye la conducción de las pruebas necesarias y el registro del resultados de la pruebas y de las no conformidades encontradas, así como un reporte diario en forma de resumen del trabajo realizado.

2.3.3- Modelo de Casos de Uso del Negocio.

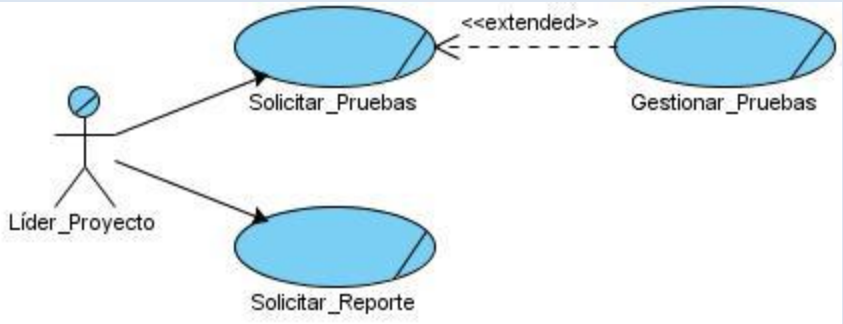
El Modelo de Casos de Uso del Negocio describe los procesos de negocio de una empresa en términos de casos de uso y actores del negocio, que se corresponden con los procesos del negocio y los clientes, respectivamente.

A continuación se describen los procesos de negocio en términos de CU y actores del negocio:

Diagrama de Casos de Uso del Negocio.

El diagrama de CU del negocio se construye para tener una visión general de los diferentes procesos de negocio de la organización. En este, aparece cada proceso del negocio como un CU. Este diagrama permite mostrar los límites y el entorno de la organización bajo estudio. Sólo se mostrarán los actores del negocio correspondientes a los roles externos al sistema, de forma que los procesos de negocio en los que sólo tomen parte roles internos a la organización no estarán conectados a ningún actor.

Ilustración 1: Diagrama de CU del Negocio.



2.3.4- Realización de Casos de Uso del Negocio.

A continuación se documenta la realización de los CU, donde se describen textualmente los CU del negocio.

Tabla 3: CUN Solicitar Pruebas.

1. Nombre del Caso de Uso	**Solicitar Pruebas**
Actor	**Líder de Proyecto**
Trabajador	Administrador de Pruebas
Descripción	El CU se inicia cuando un Líder de Proyecto solicita, mediante un documento formal, la realización del proceso de pruebas en su proyecto. En el documento de Solicitud se registran los datos del proyecto, tales como el nombre, un identificador, tiempo destinado para el proceso, recursos disponibles, orden de prioridad del proyecto, fase en la que se encuentra y lo que se quiere probar (módulo, interfaz, etc.), además él Líder del Proyecto debe facilitar toda la documentación necesaria para realizar las pruebas. Si el proyecto ha sido creado anteriormente no es necesario introducir los datos. En dependencia del análisis de los

	datos recogidos, el Administrador de Pruebas, determina aprobar o no la Solicitud y luego se le informa la decisión al Líder de Proyecto solicitante.

Tabla 4: CUN Solicitar Reportes

2. Nombre del Caso de Uso	**Solicitar Reportes**
Actor	**Líder de Proyecto**
Trabajadores	Administrador de Pruebas, Probador
Descripción	El CU inicia cuando el Líder de Proyecto solicita al Administrador de pruebas un Reporte del estado del proceso de pruebas. El Administrador de Pruebas solicita el reporte del estado al Probador, que es el responsable del registro de los resultados de cada prueba realizada, así como de la confección del Documento de No Conformidades. El Probador realiza el Reporte del trabajo hecho hasta el momento y lo entrega al Administrador de Pruebas, quien a su vez lo hace llegar hasta el Líder de Proyecto solicitante.

Tabla 5: CUN Gestionar Pruebas

3. Nombre del Caso de Uso	**Gestionar Pruebas**
Actor	**Líder de Proyecto.**
Trabajadores	Administrador de Pruebas, Analista de Pruebas, Diseñador de Casos de Pruebas, Probador
Descripción	El CU se inicia cuando el Administrador de Pruebas, luego de haber aprobado la solicitud del proceso de pruebas, procede a la creación del equipo de trabajo que llevará a cabo el proceso. Se seleccionan los recursos humanos y

	se les informa sobre los detalles de la tarea a desarrollar, así como el rol que va a desempeñar cada uno y sus responsabilidades, además se les informa la ubicación de la información necesaria para realizar el proceso. Acto seguido el Analista de Pruebas define el Nivel de pruebas y los Tipos de pruebas necesarias, además deberá monitorear todo el proceso de pruebas y los resultados de cada ciclo, evaluando en todo momento la calidad total experimentada. El Diseñador de Casos de Pruebas define los métodos de pruebas a realizarse y debe asegurar su implementación exitosa, identifica además las técnicas de prueba a utilizar, las herramientas idóneas para el trabajo y procede a realizar el Plan de Pruebas y el Diseño de Casos de Pruebas. Finalmente, el Probador realiza los Casos de Pruebas propuestos y al mismo tiempo va llenando el Documento de No Conformidades, en la medida que estas van apareciendo. El Probador debe realizar un resumen de la jornada diaria y llevar un resumen del trabajo realizado hasta el momento.

2.3.5- Diagramas de Actividades.

Ilustración 2: Diagrama de Actividades CU Solicitar Pruebas

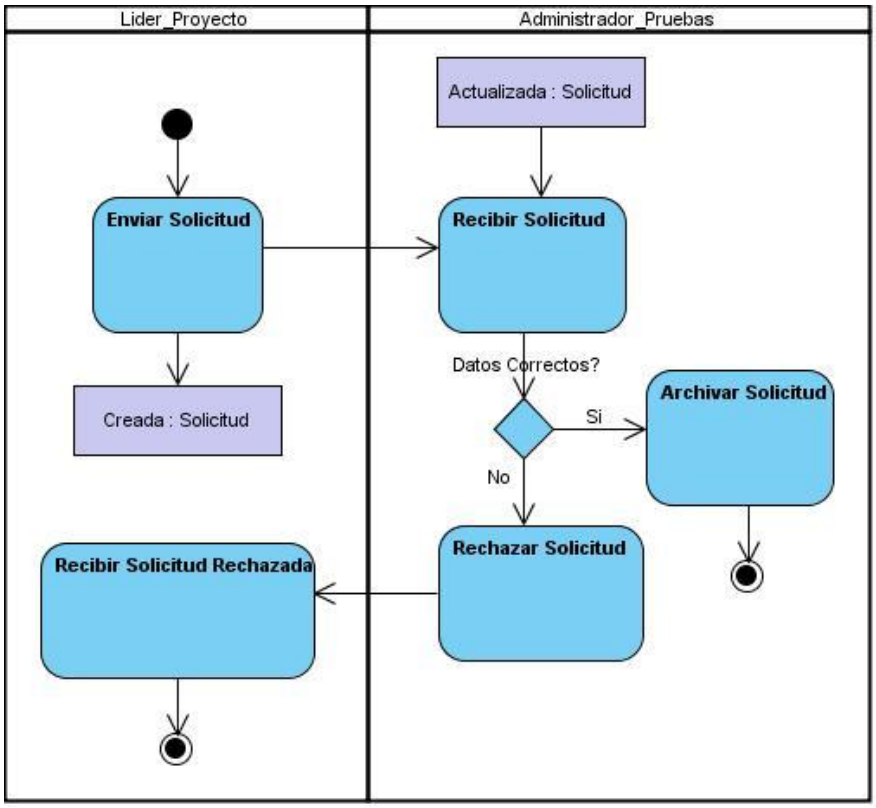


Ilustración 3: Diagrama de Actividades CU Solicitar Reporte

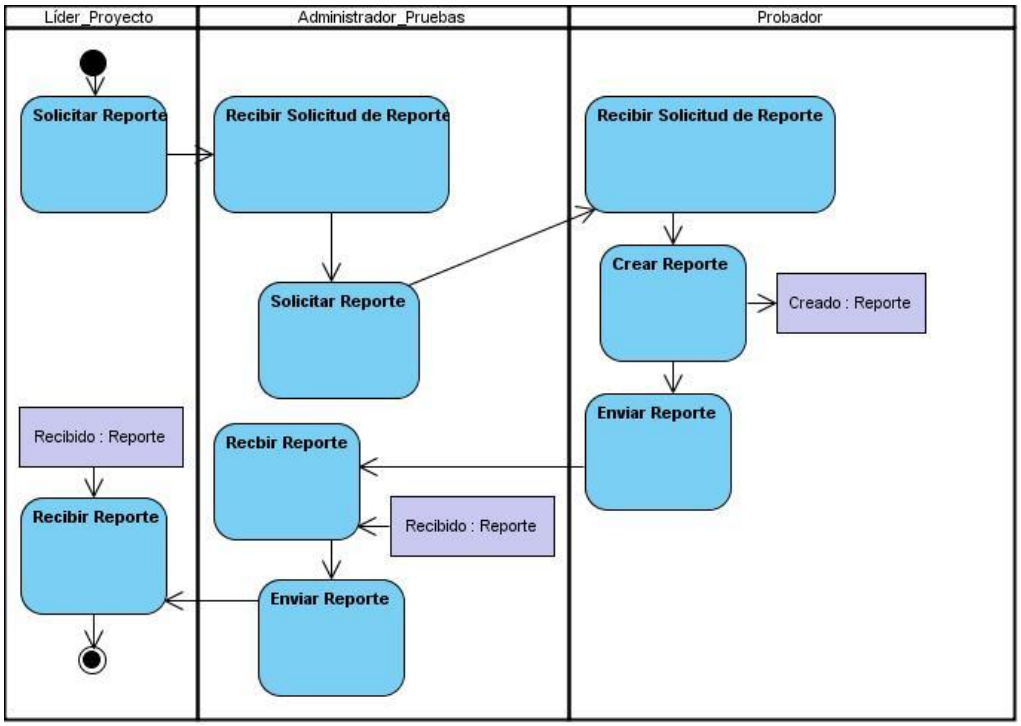
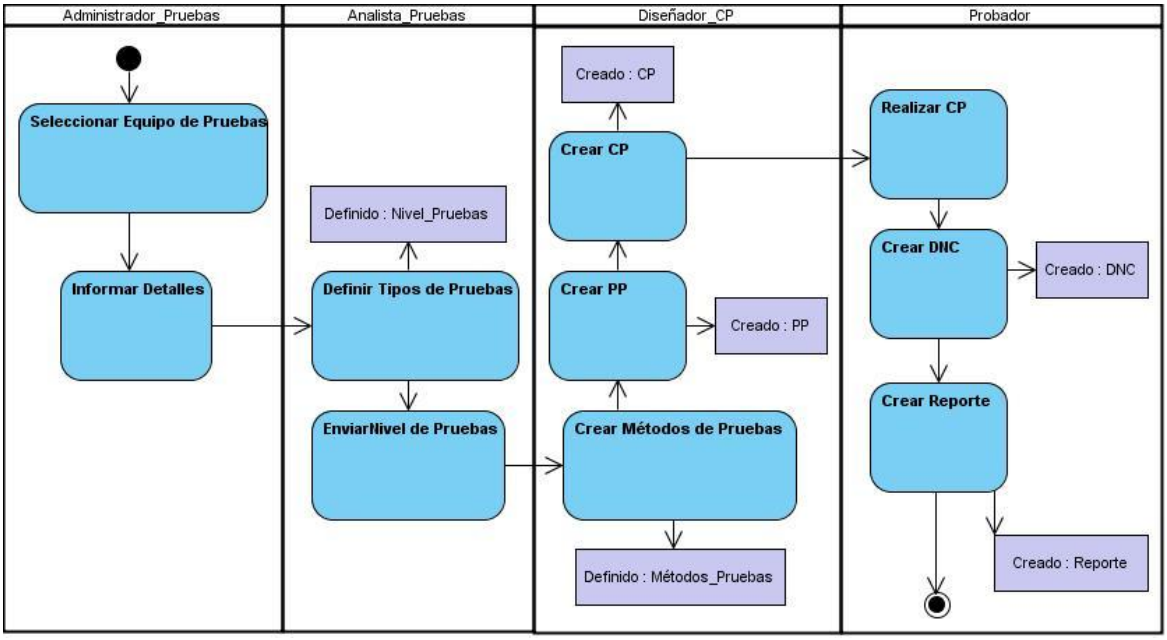


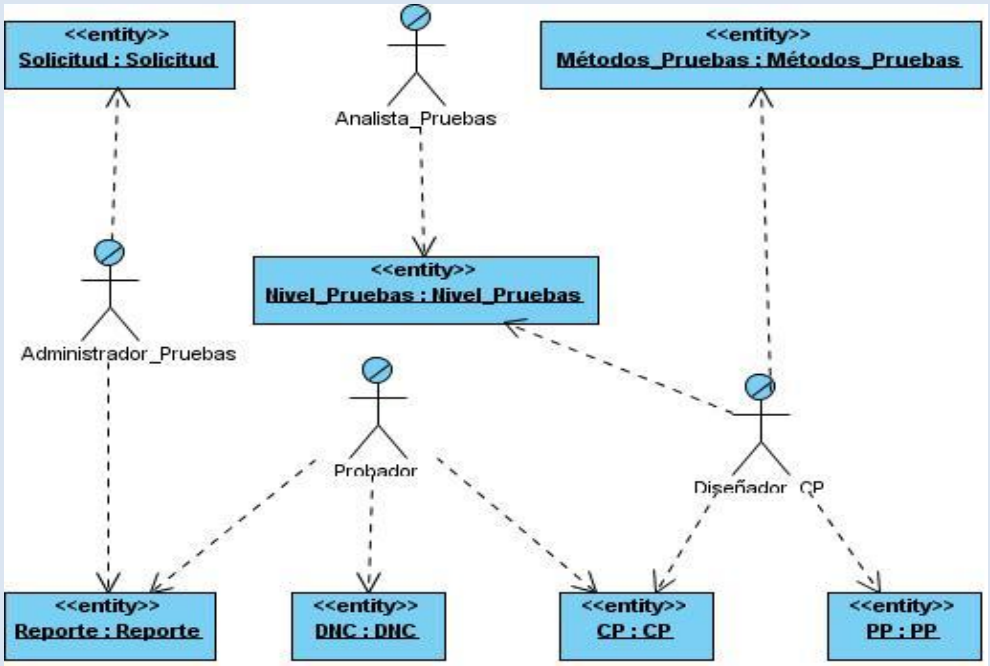
Ilustración 4: Diagrama de Actividades CU Gestionar Pruebas



2.3.6- Diagrama de Objetos del Negocio.

Es un modelo de objetos que describe cómo colaboran los trabajadores y las entidades del negocio dentro del flujo de trabajo del proceso de negocio.

Ilustración 5: Diagrama de Objetos del Negocio.



2.3.7- Reglas del Negocio.

Se identificaron las siguientes reglas que debe cumplir la aplicación a desarrollar, a fin de respetar y garantizar las restricciones que existen en el negocio:

- ✓ Solo pueden realizar solicitudes de Proceso de Prueba los Líderes de Proyectos, además los mismos deben pertenecer al PHA.

- ✓ El Administrador de Pruebas es el encargado de manipular toda la información correspondiente a las solicitudes realizadas por los Líderes de Proyecto. Es el que determina si las mismas cumplen

con todas las especificaciones y si se cuenta en ese momento con los recursos necesarios para comenzar el proceso de pruebas.

- ✓ El Administrador de Pruebas es el encargado de informarle al Líder de Proyecto solicitante la aprobación o no de la solicitud realizada.

- ✓ Si la solicitud del proceso de pruebas es aprobada, el Administrador de Pruebas es el encargado de conformar el Equipo de Pruebas que va a llevar a cabo el proceso en el proyecto solicitante. Si aun no están definidos los roles, debe asignarlos a cada uno de los integrantes en dependencia de la experiencia y desempeño en trabajos anteriores.

- ✓ El Analista de Pruebas es el encargado de identificar y definir correctamente los Niveles y Tipos de pruebas que requiere el proyecto solicitante, además es el encargado de monitorear el progreso del proceso y los resultados en cada ciclo.

- ✓ El Diseñador de Casos de Pruebas es el que define los métodos de prueba a realizarse, asegura la realización de una implementación exitosa de las mismas. Identifica las técnicas apropiadas, herramientas e instrucciones para implementar las pruebas necesarias y encauzar los recursos correspondientes para las pruebas.

- ✓ El Equipo de Pruebas debe cumplir al pie de la letra con la estrategia establecida, de manera que se logre una uniformidad en todo el proceso.

- ✓ Los miembros del equipo solo pueden desempeñar un único rol durante el proceso.

- ✓ Todos los artefactos resultantes estarán relacionados entre sí con otros artefactos que le preceden, para poder realizar uno de estos es necesario haber completado el inmediato superior.

- ✓ El artefacto “Plan de Pruebas” será realizado por el Diseñador de Casos de Pruebas.

- ✓ El artefacto “Casos de Prueba” será realizado por el Diseñador de Casos de Pruebas.

- ✓ El Probador lleva a cabo las pruebas, ejecuta los Casos de Pruebas definidos por el Diseñador de Casos de Pruebas, además el encargado de generar el artefacto “Documento de No Conformidades” y al final de cada jornada de trabajo deberá completar otro artefacto, el reporte diario.

- ✓ El probador debe registrar el reporte diario el mismo día que realiza las pruebas.

- ✓ Será posible conocer el estado del proceso de pruebas de cada uno de los proyectos y el estado en general.

- ✓ Las solicitudes de reportes pueden ser realizadas por cualquier usuario interesado.

2.4- Requerimientos.

Este es un flujo de trabajo que se realiza en la Fase de Inicio, según lo establecido por RUP. Tiene como objetivos fundamentales identificar, enunciar y clasificar los requerimientos, además obtener las propiedades o cualidades que el producto debe cumplir, obtener un listado de las capacidades o funciones del sistema y por último determinar actores y casos de uso.

2.4.1- Especificación de los Requerimientos de Software.

El análisis de requisitos es una de las tareas más importantes en el ciclo de vida del desarrollo de software, puesto que en ella se determinan los “planos” de la nueva aplicación.

El análisis de requisitos se puede definir como el proceso del estudio de las necesidades de los usuarios para llegar a una definición de los requisitos del sistema, hardware o software, así como el proceso de estudio y refinamiento de dichos requisitos, definición proporcionada por el IEEE. (G. Piattini, 1996)

Asimismo, se define requisito como una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado. (G. Piattini, 1996)

Esta definición se extiende y se aplica a las condiciones que debe cumplir o poseer un sistema o uno de sus componentes para satisfacer un contrato, una norma o una especificación.

Los requisitos pueden ser clasificados de acuerdo con el modelo FURPS+ (B.Grady, 1992) que es un Modelo de clasificación de requerimientos cuyo nombre es el acrónimo en inglés de las diferentes categorías (Functionality, Usability, Reliability, Performance, Supportability) fue desarrollado por Robert B. Grady de y en la actualidad es uno de los más utilizados.

- **Funcional** (Functional): Características, capacidades y algunos aspectos de seguridad
- **Facilidad de uso** (Usability): Factores Humanos (interacción), ayuda, documentación.
- **Fiabilidad** (Reliability): Frecuencia de fallos, capacidad de recuperación de un fallo y grado de previsión.
- **Rendimiento** (Performance): Tiempos de respuesta, productividad, precisión, disponibilidad, uso de los recursos.
- **Soporte** (Supportability): Adaptabilidad, facilidad de mantenimiento, internacionalización, facilidad de configuración.

El "+" en FURPS+ indica requisitos adicionales, tales como:

- **Implementación**: Limitación de recursos, lenguajes y herramientas, hardware.
- **Interfaz**: Restricciones impuestas para la interacción con sistemas externos.
- **Operaciones**: Gestión del sistema, pautas administrativas, puesta en marcha
- **Empaquetamiento:** Forma de distribución.
- **Legales**: Licencia, derechos de autor, etc.

Se utilizó FURPS+ como una lista de chequeo para comprobar que se cubren la mayoría de los requisitos, con el objetivo de reducir al máximo los riesgos de no tener en cuenta características importantes del sistema.

2.4.2- Requisitos Funcionales.

Los Requisitos Funcionales **(RF)** definen el comportamiento interno del software y otras funcionalidades específicas que muestran cómo los casos de uso serán llevados a la práctica, además describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Para el sistema propuesto se registraron los siguientes RF:

RF1- Gestionar Proyectos.

RF1.1- Gestionar Proyecto

RF1.1.2 Adicionar Proyecto.

RF1.1.2.- Buscar Proyecto.

RF1.1.3- Modificar información del Proyecto.

RF1.1.4- Eliminar Proyecto.

RF1.2- Gestionar Módulo.

RF1.2.1- Adicionar Módulo.

RF1.2.2- Buscar Módulo.

RF1.2.3- Modificar información del Módulo.

RF1.2.4- Eliminar Módulo.

RF2- Gestionar Equipo de Pruebas.

RF2.1- Crear Equipo.

RF2.2- Adicionar Integrante.

RF2.3- Eliminar Integrante.

RF2.2- Buscar Equipo.

RF3- Gestionar Proceso de Pruebas.

RF3.1- Gestionar PP.

RF3.1.1- Adicionar Plan de Pruebas.

RF3.1.2- Buscar Plan de Pruebas.

RF3.1.3- Modificar información del Plan de Pruebas.

RF3.1.4- Eliminar Plan de Pruebas.

RF3.2- Gestionar CP.

RF3.2.1- Adicionar Caso de Pruebas.

RF3.2.2- Buscar Caso de Pruebas.

RF3.2.3- Modificar información del Caso de Pruebas.

RF3.2.4- Eliminar Caso de Pruebas.

RF3.3- Gestionar DNC.

RF3.3.1- Adicionar No Conformidades.

RF3.3.2- Buscar No Conformidades.

RF3.3.3- Modificar No Conformidades.

RF3.3.4- Eliminar No Conformidades.

RF3.4- Registrar Reporte.

RF3.4.1- Adicionar Reporte.

RF3.4.2- Modificar Reporte.

RF3.4.3- Buscar Reporte.

RF3.5- Solicitar Reportes.

RF3.6- Gestionar Solicitud de Pruebas

RF3.6.1- Solicitar Pruebas.

RF3.6.2- Modificar Solicitud.

RF3.6.3- Buscar Solicitud.

RF3.6.4- Eliminar Solicitud

RF4- Administrar Sistema.

RF4.1- Gestionar Usuarios.

RF4.1.1- Adicionar Usuarios.

RF4.1.2- Modificar Usuarios.

RF4.1.3- Buscar Usuarios.

RF4.1.3- Asignar Rol.

RF4.1.4- Cambiar Contraseña.

RF4.1.5- Eliminar Usuarios.

RF4.2- Gestionar Líder.

RF4.2.1- Adicionar Líder de Proyecto.

RF4.2.2- Buscar Líder de Proyecto.

RF4.2.3- Modificar información del Líder de Proyecto.

RF4.2.4- Eliminar Líder de Proyecto.

RF5- Asignar Tareas.

RF5.1- Adicionar Tarea.

RF5.2- Ver Tareas.

RF5.2.1- Ver Tareas Activas.

RF5.2.2- Ver Mis Tareas.

RF5.2.2.1- Resolver Tarea.

RF5.2.2.2- Reasignar Tarea.

RF5.2.3- Ver Todas las Tareas.

En este prototipo de CCALPHA no se le implementarán completamente los **RF5-** Asignar Tarea y el **RF1.2-** Gestionar Módulo, los mismos serán implementados en el próximo ciclo de desarrollo de RUP.

2.4.3- Requisitos No Funcionales.

Los Requerimientos No Funcionales **(RNF)** son propiedades o cualidades que el producto debe tener. Tienen que ver además con características que de una u otra forma puedan limitar el sistema. Estas propiedades son las características que hacen al producto atractivo, usable, rápido, confiable, seguro, robusto, portable, etc.

Normalmente estos RNF están vinculados a RF. Una vez conocido lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser. El levantamiento de RNF para el sistema propuesto aparece a continuación:

Tabla 6: Requisitos No Funcionales

Requisitos No Funcionales	
No	**RNF1**
Categoría	**Interfaz**
Descripción	El prototipo de aplicación tendrá un diseño sencillo con el propósito de la comprensión y el acceso a los materiales complementarios y de consulta sobre el proceso de pruebas, además de otras informaciones de interés para los usuarios. La interfaz no contendrá muchas imágenes para agilizar el tiempo de respuesta. La aplicación será seria, formal y de navegación intuitiva.
No	**RNF2**
Categoría	**Seguridad**
Descripción	El sistema controla los diferentes niveles de acceso a usuarios, identifica los usuarios antes de que pueda realizar cualquier acción sobre el sistema, estos tendrán acceso a las opciones que le son referidas según su rol. Garantiza que la información sea consultada solo por los usuarios que tienen permisos. La validación de los campos deberá ser inmediata en caso de ser posible y al momento de introducción de datos por parte de los usuarios del Sistema. Se hacen validaciones de la información tanto en el cliente como en el servidor, no obstante los usuarios acceden de manera rápida y operativa al sistema sin que los requerimientos de seguridad se conviertan en un retardo o impedimento para ellos.
No	**RNF3**
Categoría	**Confidencialidad**
Descripción	Se vela por la integridad de los datos en el sistema a través de la consistencia de la información. Toda la información estará protegida del acceso no autorizado, solo los administradores tendrán acceso pleno a ella y los demás usuarios podrán transformarla

	en dependencia del nivel de acceso que tenga el rol que desempeña.
No	**RNF4**
Categoría	**Rendimiento**
Descripción	La conexión entre el cliente-servidor será constante, brindando una alta disponibilidad. La respuesta a las solicitudes de los usuarios debe realizarse en un margen de tiempo pequeño, evitando la acumulación de tareas pendientes.
No	**RNF5**
Categoría	**Políticos-Culturales**
Descripción	Todos los mensajes y textos que la aplicación muestre deberán aparecer en idioma español.
No	**RNF6**
Categoría	**Soporte**
Descripción	El producto software deberá recibir mantenimiento y actualización cada cierto tiempo y cuando ocurra cualquier fallo imprevisto.
No	**RNF7**
Categoría	**Usabilidad**
Descripción	La aplicación permitirá la conexión concurrente de varios usuarios desde cualquier computadora que comparta la misma red que los servidores. Los mismos podrán ser administrados y mantenidos desde cualquier computadora en red. Se validará que toda la información introducida en la aplicación sea correcta y esté completa. El sistema deber permitir ser monitoreado en cualquier momento.
No	**RNF8**
Categoría	**Software**
Descripción	Se hará uso del Servidor HTTP Apache, Servidor de Base de Datos MySQL, se utilizará como herramienta CASE el Visual Paradigm for UML Community Edition V3.4, el IDE de desarrollo será el Aptana Studio Community Edition y la herramienta para el desarrollo

	web será el Quanta Plus. Para el funcionamiento del cliente y servidor serán necesarios el Sistema Operativo Linux o Windows.
No	**RNF9**
Categoría	**Hardware**
Descripción	Se necesitan como requerimientos mínimos, una computadora con procesador Pentium II o superior, una memoria RAM de 128 o superior, disco duro preferiblemente de 5 GB o superior, la conexión de red admite cualquier velocidad.
No	**RNF10**
Categoría	**Restricciones en el diseño y la implementación.**
Descripción	Aplicación Web desarrollada con la tecnología para creación de páginas Web dinámicas PHP5 del lado del servidor y con HTML, CSS y Java Script de la parte del cliente.
No	**RNF11**
Categoría	**Documentación.**
Descripción	El sistema brinda a los usuarios documentación en línea de modo que si el usuario presenta alguna duda sobre las pruebas, pueda acudir a la misma en todo momento.
No	**RNF12**
Categoría	**Legales**
Descripción	El sistema se basa en estándares que se rigen por normas internacionales, no incumple con normas o leyes establecidas en nuestro país.

2.5- Definición de los Actores del Sistema.

El Actor del Sistema es una entidad externa, representando un rol de una o varias personas, un equipo o un sistema automatizado que interactúa con el sistema, por lo que no forma parte del sistema. Puede intercambiar información o ser un recipiente pasivo de información. (Callejas Cuervo, et al.)

Tabla 7: Actores del Sistema

Actores del Sistema	Descripción
Administrador de Pruebas	Es el responsable de planificar y administrar los recursos. Acepta o no las solicitudes realizadas por los líderes de proyecto, crea el Equipo de Pruebas y le designa su respectivo Jefe de Equipo, asigna las tareas que va a desempeñar cada miembro del equipo así como las responsabilidades dentro del proceso.
Analista de Pruebas	Es el responsable de identificar y definir los niveles y tipos de pruebas requeridas, monitorear el progreso de la prueba y el resultado en cada ciclo.
Diseñador de Casos de Pruebas	Es el responsable de definir el método de prueba y asegurar su implementación exitosa. Identifica las técnicas de pruebas apropiadas. Crea el Plan de Pruebas y los Casos de Pruebas.
Probador	Es el responsable durante las actividades principales de las pruebas, el cual incluye la conducción de las pruebas necesarias y el registro del resultado de la prueba en la aplicación, así como el Documento de No Conformidades y el reporte del resumen diario del trabajo realizado.
Administrador del Sistema	Es el encargado de gestionar la información relacionada con los líderes de proyectos, los proyectos y los demás usuarios del sistema. Será el encargado de administrar la aplicación, tendrá acceso único a realizar las configuraciones del sistema y configurar los niveles de accesos de los usuarios.
Líder de Proyecto	Individuo que se encuentra al frente de un proyecto productivo y es el encargado de solicitar que se realice el proceso de pruebas en su proyecto. Recibe los reportes y resultados finales del proceso de pruebas.

Usuario	Persona que no va a tener ningún rol dentro del sistema, solamente tendrá navegación básica en el mismo y podrá consultar la información disponible en la página de inicio.

2.6- Definición de los CU del Sistema.

Solo se muestra la definición de los CU más importantes, las descripciones de los demás CU se encuentran a partir del Anexo 5 hasta el Anexo 10.

Tabla 8: Definición CUS: Gestionar Proyecto.

Nombre de Caso de Uso	**Gestionar Proyecto**
Actor	Administrador del Sistema
Descripción	El CU inicia cuando el Administrador del Sistema accede a la interfaz de administración de la aplicación web y selecciona la opción: Gestionar Proyecto, el sistema muestra las opciones: adicionar, eliminar y modificar proyecto.
Referencia	**RF1.2**

Tabla 9: Definición CUS: Gestionar Usuarios.

Nombre de Caso de Uso	**Gestionar Usuarios**
Actor	Administrador del Sistema
Descripción	El CU inicia cuando el Administrador del Sistema accede a la interfaz de administración de la aplicación web y selecciona la opción: Gestionar Usuarios, el sistema muestra las opciones: adicionar, eliminar y modificar usuarios del sistema.
Referencia	**RF5.1**

Tabla 10: Definición CUS: Registrar Reporte Diario.

Nombre de Caso de Uso	**Registrar Reporte Diario**
Actor	Probador
Descripción	El CU inicia cuando el Probador, luego de haber terminado la jornada laboral, escoge la opción: Registrar Reporte, el sistema muestra la interfaz correspondiente donde aparecen las opciones: adicionar y modificar reportes.
Referencia	**RF3.4**

Tabla 11: Definición CUS: Gestionar Solicitud de Pruebas.

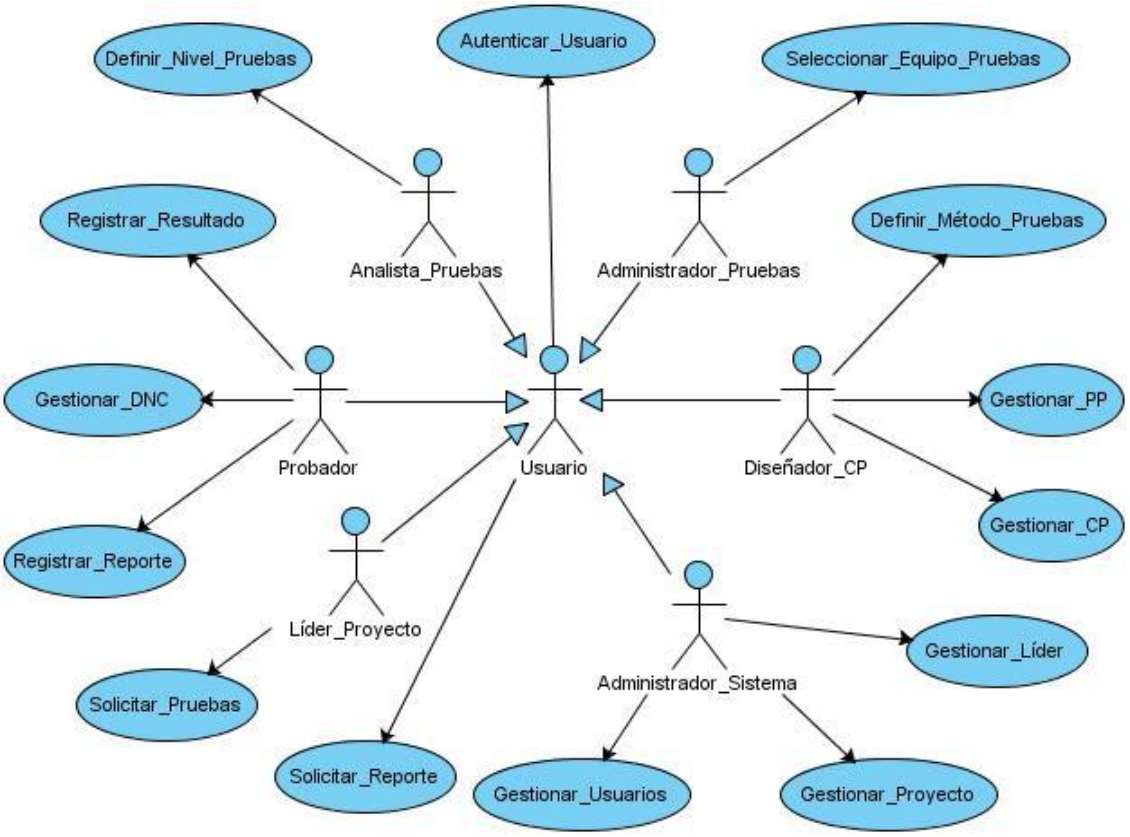
Nombre de Caso de Uso	**Gestionar Solicitud de Pruebas**
Actor	Líder de Proyecto
Descripción	El CU inicia cuando el Líder de proyecto accede a su interfaz principal y escoge la opción: Solicitar Pruebas, el sistema muestra la interfaz correspondiente y el Líder de Proyecto procede a completar los campos que corresponden a la solicitud. Además el Líder de Proyecto tiene las opciones de eliminar, actualizar y buscar las solicitudes que ya hayan sido introducidas.
Referencia	**RF3.6**

2.7- Diagrama de CU del Sistema.

El diagrama de CU del Sistema es la representación gráfica de parte o el total de los actores y casos de uso del sistema, incluyendo sus interacciones. Todo sistema tiene como mínimo un diagrama de casos de uso principales, que es una representación gráfica del entorno del sistema (actores) y su funcionalidad principal (casos de uso). Un diagrama de casos de uso muestra, por tanto, los distintos requisitos funcionales que se esperan de una aplicación o sistema y cómo se relaciona con su entorno.

A continuación se muestra el Diagrama de CU del Sistema:

Ilustración 6: Diagrama de CU del Sistema.



CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL SISTEMA

3.1- Introducción.

El presente capítulo describe los dos flujos de trabajos definidos por RUP que deben realizarse en la fase de Elaboración: Análisis y Diseño.

Se exponen los diagramas de clases de análisis como parte de la realización de los CU arquitectónicamente significativos, junto con los diagramas de clases del diseño y sus diagramas de secuencias. Se presenta una breve descripción de las clases identificadas durante el flujo de trabajo de diseño, y de las tablas y atributos que conforman la base de datos donde será almacenada la información que deba persistir en los procesos del negocio.

3.2- Modelo de Análisis.

Este flujo de trabajo definido por RUP tienes como objetivos transformar los RF en un diseño de clases, analizando las relaciones e interacción que existe entre ellos y teniendo en cuenta en el proceso una arquitectura robusta que permita adaptar el sistema al entorno de implementación que se está desarrollando.

Consiste en obtener una primera visión del sistema que se preocupa de ver “qué” hace, de modo que sólo se interesa por los RF. Por otro lado, el diseño es un refinamiento del análisis que tiene en cuenta los RNF, en definitiva “cómo” cumple el sistema sus objetivos.

Es un modelo conceptual, genérico respecto al diseño. Posee tres estereotipos conceptuales sobre las clases: Control, Entidad e Interfaz. Además es dinámico al no estar muy centrado en la secuencia. Otra de sus características es que puede no ser mantenido durante todo el ciclo de vida del software

3.2.1- Clases del Análisis.

Estos diagramas nos permiten identificar las clases que utiliza el sistema y sus relaciones, además es posible identificar los atributos y sus métodos, estas clases normalmente se convierten en las tablas de la Base de Datos, por lo que el diagrama servirá como base para próximas fases. Los diagramas de clases

son utilizados durante el proceso de análisis y diseño, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

Las clases del análisis se centran en los RF y son evidentes en el dominio del problema porque representan conceptos y relaciones del dominio. Tienen atributos y entre ellas se establecen relaciones de asociación, agregación/composición, generalización/ especialización y tipos asociativos.

A continuación se muestran lo Diagramas de las Clases del Análisis más importantes, los restantes diagramas se encuentran a partir del Anexo 11 hasta el Anexo 17.

Ilustración 7: Diagrama de Clases Análisis: Gestionar Proyecto.

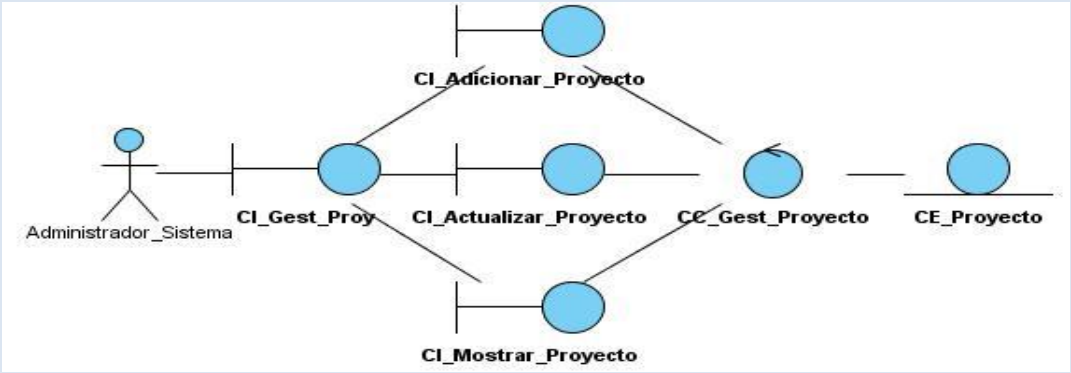


Ilustración 8: Diagrama de Clases Análisis: Gestionar Usuarios.

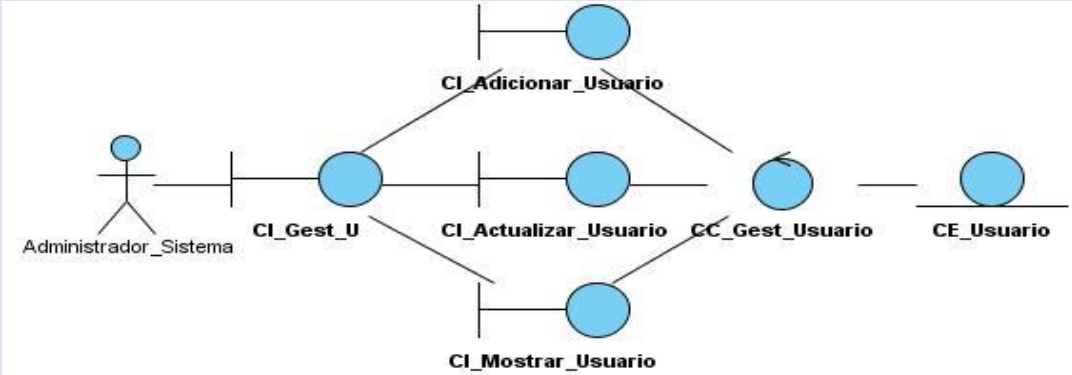


Ilustración 9: Diagrama de Clases Análisis: Registrar Reporte Diario.

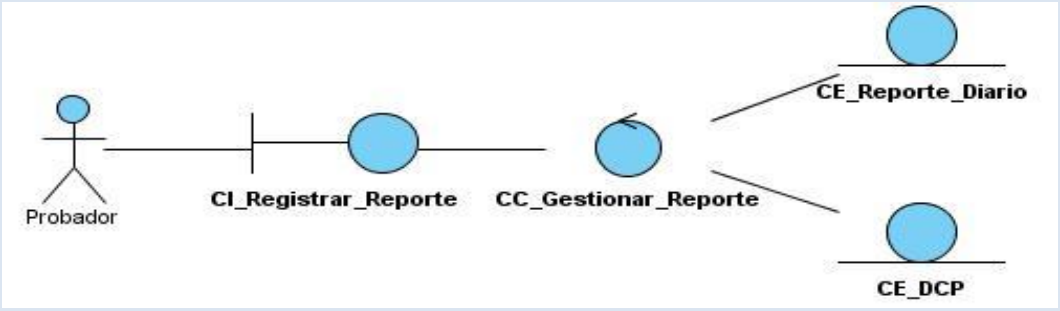
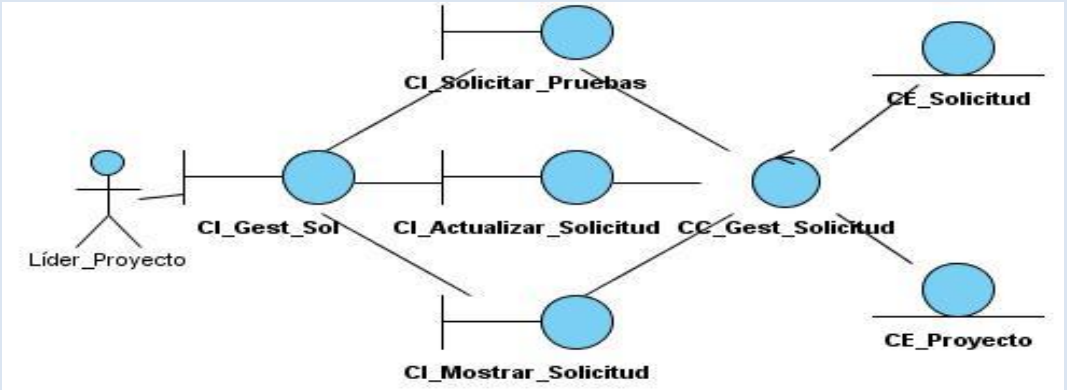


Ilustración 10: Diagrama de Clases Análisis: Gest Solicitud de Pruebas.



3.3- Modelo de Diseño.

La fase de diseño expande y detalla los modelos del análisis, tomando en cuenta todas las implicaciones y restricciones técnicas. El propósito del diseño es especificar una solución que trabaje y pueda ser convertida fácilmente en código fuente y construir una arquitectura simple y fácilmente extensible. Las clases definidas en el análisis fueron detalladas, y se añadieron nuevas clases para manejar áreas técnicas como base de datos, interfaz de usuario, comunicación, dispositivos, etc.

Este modelo es el punto de partida de la Implementación. Descompone las tareas de implementación en sub-tareas manejables que, en muchos casos, se pueden realizar de forma simultánea. Es una abstracción del sistema. La estructura del sistema no puede cambiar, aunque sí los detalles de la implementación

Como sus características fundamentales se tiene que es un modelo físico, no genérico y específico para una implementación. Puede tener cualquier número de estereotipos físicos sobre las clases en dependencia del lenguaje en el que se haya decidido implementar la aplicación. Es un modelo dinámico, centrado en la secuencia. Debe ser mantenido durante todo el ciclo de vida del software y finalmente da forma al sistema mientras que intenta preservar la estructura definida por el modelo de análisis lo más posible.

3.3.1- Diagramas de Interacción en el Diseño.

En el diseño orientado a objetos la interacción es el comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos dentro de un contexto para lograr un propósito. Un mensaje es la especificación de una comunicación entre objetos (unidad de comunicación entre objetos).

Los objetos interactúan para realizar colectivamente los servicios ofrecidos por las aplicaciones. Los diagramas de interacción muestran cómo se comunican los objetos en una interacción. El UML define dos tipos de diagramas de interacción: los Diagramas de Colaboración y los Diagramas de Secuencia. En este trabajo se realizaron los Diagramas de Secuencia de los CU arquitectónicamente significativos, y dada la existencia de CU que se diferencian solo en los parámetros de entrada, se realizó un ejemplo que resume los demás flujos de secuencias. *(Ver Anexos a partir la Ilustración 41- Anexo 5).*

3.3.2- Clases del Diseño.

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones. El modelo de CU y los diagramas de interacción aportan información para establecer las clases, objetos, atributos y operaciones.

A continuación se muestran los Diagramas de Clases del Diseño más importantes, los restantes diagramas se encuentran a partir del Anexo 18 hasta el Anexo 24.

Ilustración 11: Diagrama de Clase Diseño: Gestionar Proyecto.

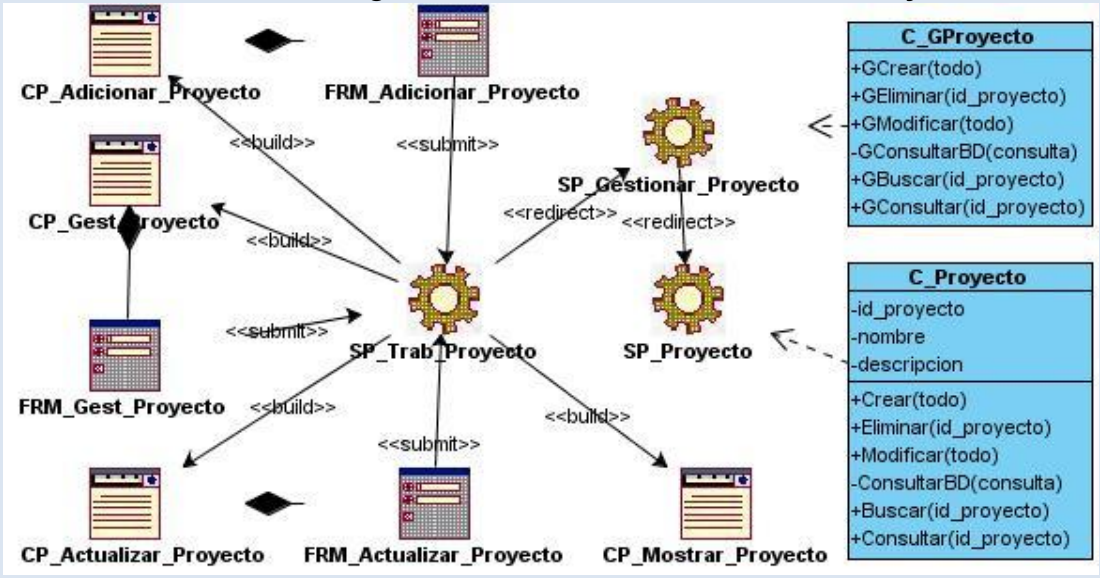


Ilustración 12: Diagrama de Clase Diseño: Gestionar Usuarios.

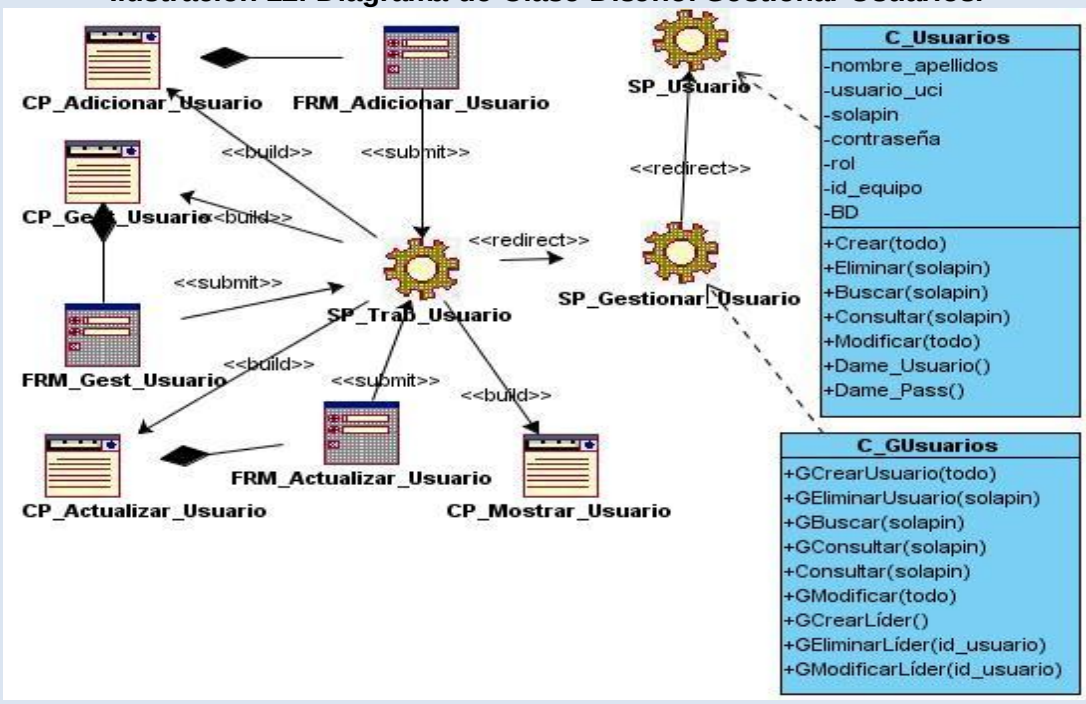


Ilustración 13: Diagrama de Clase Diseño: Registrar Reporte Diario.

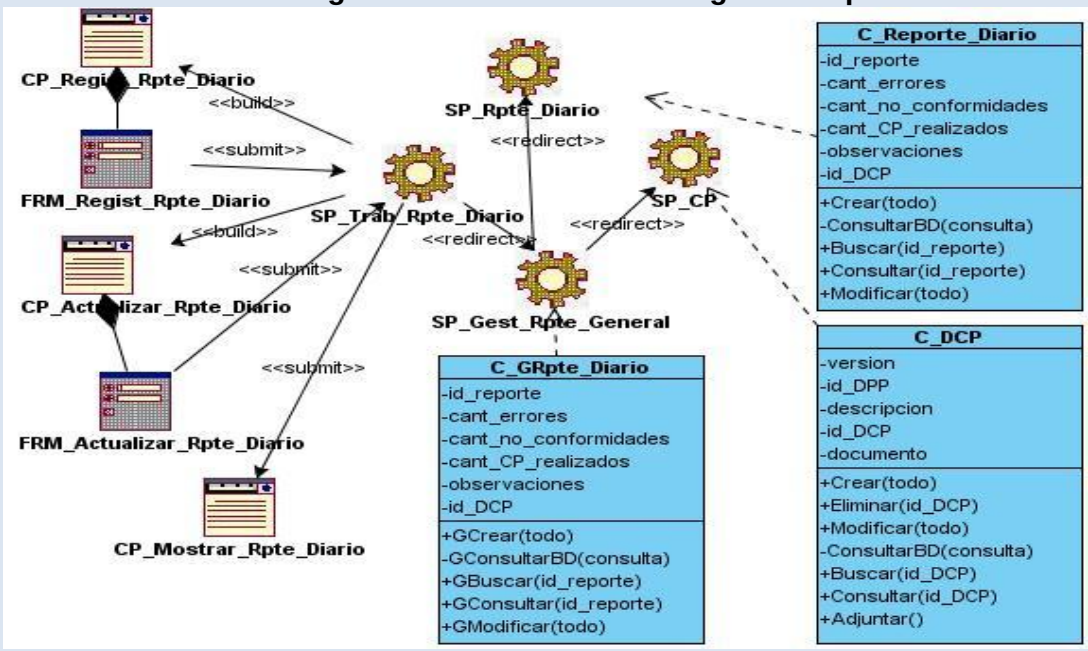
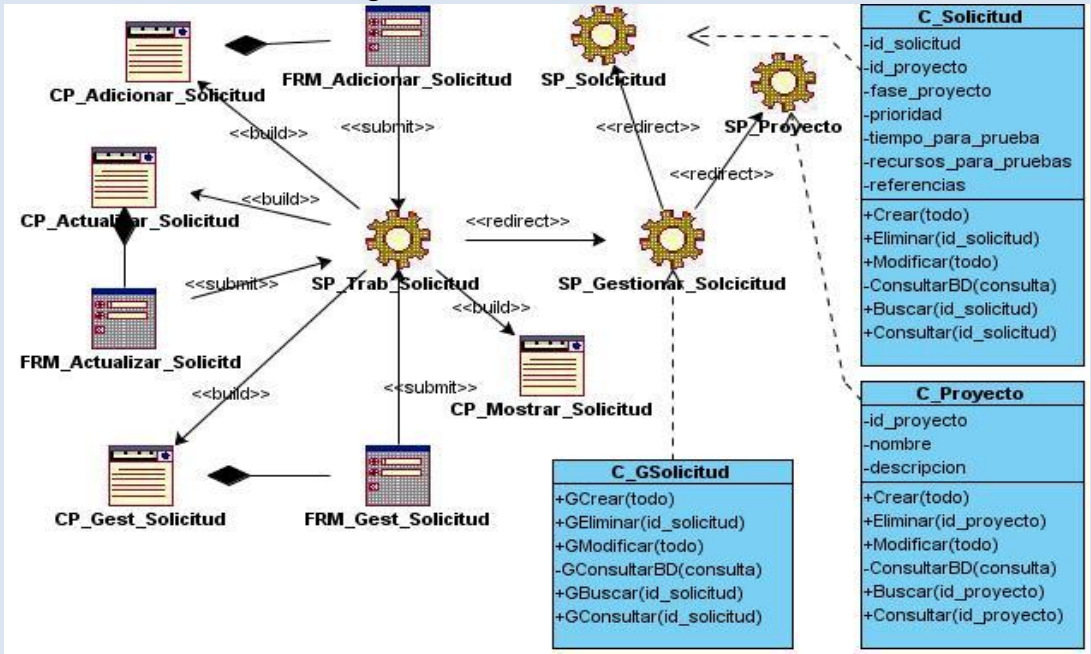


Ilustración 14: Diagrama de Clase Diseño: Gestionar Solicitud.



3.3.3- Descripción de las Clases del Diseño.

A continuación se describen las clases del diseño más importantes, las restantes descripciones se encuentran a partir del Anexo 31 hasta el Anexo 35.

Tabla 12: Descripción Clase de Diseño: C_Usuario.

Nombre:	**C_Usuario**
Tipo de Clase:	Entidad
Atributos:	**Tipo:**
nombre_apellidos	Varchar
usuario_uci	Varchar
solapin	Integer
contraseña	Varchar
rol	Varchar
id_equipo	Integer
Para Cada Responsabilidad	
Nombre:	**Crear(todo)**
Descripción:	Para adicionar un usuario al sistema dados todos sus atributos.
Nombre:	**Eliminar(solapin)**
Descripción:	Para eliminar un usuario del sistema dado su identificador.
Nombre:	**ConsultarBD(consulta)**
Descripción:	Para realizar las diferentes consultas a la Bases de Datos.
Nombre:	**Buscar(solapin)**
Descripción:	Para determinar existencia de un usuario dado su identificador.
Nombre:	**Consultar(solapin)**
Descripción:	Mostrar usuarios dado su identificador.

Nombre:	**Modificar(Todo)**
Descripción:	Para modificar y actualizar los datos de los usuarios registrados.
Nombre:	**Dame_Usuario()**
Descripción:	Para pedir el usuario.
Nombre:	**Dame_Pass()**
Descripción:	Para pedir el contraseña.

Tabla 13: Descripción Clase de Diseño: C_Proyecto.

Nombre:	**C_Proyecto**
Tipo de Clase:	Entidad
Atributos:	**Tipo:**
id_proyecto	Integer
nombre	Varchar
descripción	Varchar
Para Cada Responsabilidad	
Nombre:	**Crear(todo)**
Descripción:	Para adicionar un nuevo proyecto con todos sus atributos.
Nombre:	**Eliminar(id_proyecto)**
Descripción:	Para eliminar un proyecto existente dado su identificador único.
Nombre:	**Modificar(todo)**
Descripción:	Para modificar y actualizar un proyecto dados sus atributos.
Nombre:	**ConsultarBD(consulta)**
Descripción:	Para realizar diferentes consultas a la Base de Datos.
Nombre:	**Buscar(id_proyecto)**
Descripción:	Para determinar existencia de un proyecto dado su identificador.
Nombre:	**Consultar(id_ proyecto)**

Descripción:	Mostrar proyecto dado su identificador.

Tabla 14: Descripción Clase de Diseño: C_Reporte_Diario.

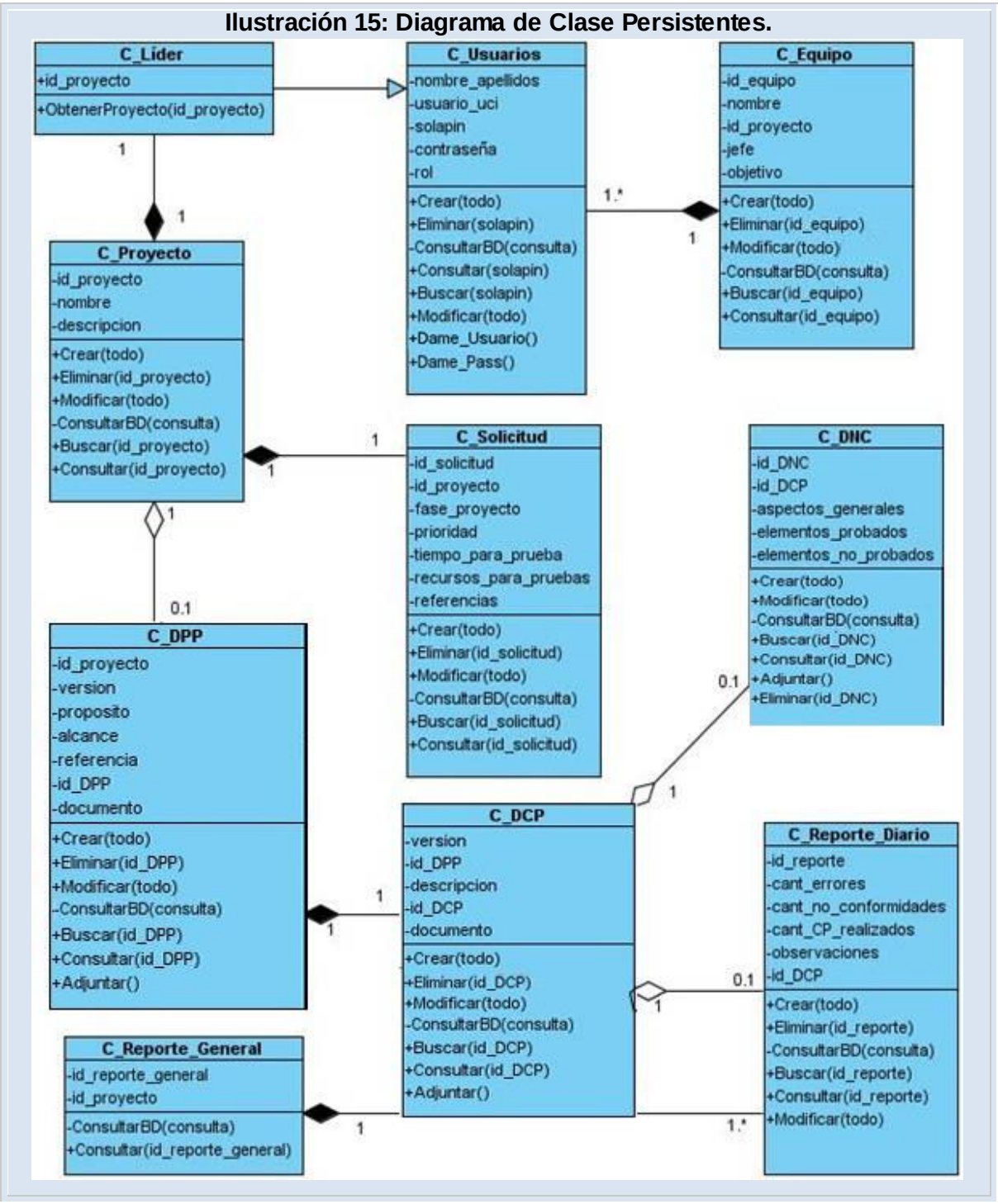
Nombre:	**C_Reporte_Diario**
Tipo de Clase:	Entidad
Atributos:	**Tipo:**
id_reporte	Integer
cant_errores	Integer
cant_no_conformidades	Integer
cant_CP_realizados	Integer
id_DCP	Integer
observaciones	Varchar
Para Cada Responsabilidad	
Nombre:	**Crear(todo)**
Descripción:	Para adicionar un nuevo Reporte Diario dados sus atributos.
Nombre:	**Eliminar(id_reporte)**
Descripción:	Para eliminar un Reporte Diario dado su identificador.
Nombre:	**Modificar(todo)**
Descripción:	Para modificar y actualizar un Reporte Diario dados sus atributos.
Nombre:	**ConsultarBD(consulta)**
Descripción:	Para realizar diferentes consultas a la Base de Datos.
Nombre:	**Buscar(id_reporte)**
Descripción:	Para determinar existencia de un Reporte Diario dado su identificador.
Nombre:	**Consultar(id_reporte)**
Descripción:	Para mostrar un reporte dado su identificador.

Tabla 15: Descripción Clase de Diseño: C_Solicitud.

Nombre:	**C_Solicitud**
Tipo de Clase:	Entidad
Atributos:	**Tipo:**
id_solicitud	Integer
id_proyecto	Integer
fase_proyecto	varchar
prioridad	Integer
tiempo_pruebas	Integer
recursos_pruebas	Varchar
referencias	Varchar
Para Cada Responsabilidad	
Nombre:	**Crear(todo)**
Descripción:	Para adicionar una nueva Solicitud dados sus atributos.
Nombre:	**Eliminar(id_solicitud)**
Descripción:	Para eliminar una Solicitud dado su identificador.
Nombre:	**Modificar(todo)**
Descripción:	Para modificar y actualizar una Solicitud dados sus atributos.
Nombre:	**ConsultarBD(consulta)**
Descripción:	Para realizar diferentes consultas a la Base de Datos.
Nombre:	**Buscar(id_solicitud)**
Descripción:	Para determinar existencia de una Solicitud dado su identificador.
Nombre:	**Consultar(id_solicitud)**
Descripción:	Para mostrar una solicitud dado su identificador.

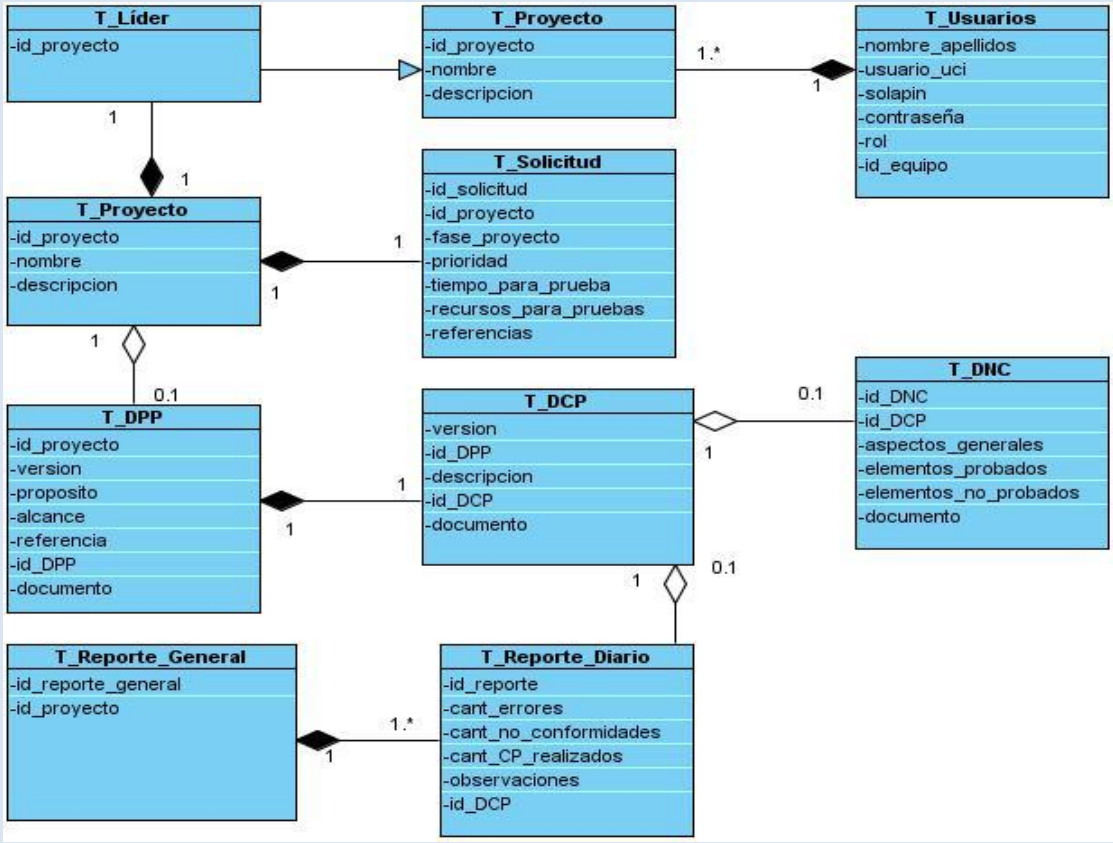
3.3.4- Diagrama de Clases Persistentes.

Ilustración 15: Diagrama de Clase Persistentes.



3.3.5- Modelo Físico de Datos.

Ilustración 16: Modelo Físico de Datos.



3.3.6- Descripción de las Tablas y Atributos.

A continuación se describen las tablas más importantes del Modelo Físico de Datos, las demás tablas aparecen a partir del Anexo 36 hasta el Anexo 40.

Tabla 16: Descripción de Tabla: T_Usuario.

Nombre:	**T_Usuario**	
Descripción:	En esta tabla se almacenan los usuarios y sus atributos.	
Atributos:	**Tipo:**	**Descripción:**
nombre_apellidos	Varchar (255)	Nombre y apellidos del usuario.

usuario_uci	Varchar (255)	Usuario para los servicios de la UCI.
solapin	Integer (11)	Numero del solapín entregado en la UCI.
contraseña	Varchar (255)	Contraseña para entrar al Sistema.
rol	Varchar (255)	Rol que ocupa el usuario dentro del Sistema.
id_equipo	Integer (11)	Identificador único del Equipo de pruebas al que pertenece.

Tabla 17: Descripción de Tabla: T_Proyecto.

Nombre:	**T_Proyecto**	
Descripción:	En esta tabla se almacenan todos los Proyectos que pertenecen al PHA.	
Atributos:	**Tipo:**	**Descripción:**
id_proyecto	Integer (11)	Identificador único del Proyecto.
nombre	Varchar (255)	Nombre del Proyecto.
descripción	Varchar (510)	Breve descripción del Proyecto.

Tabla 18: Descripción de Tabla: T_Reporte_Diario.

Nombre:	**T_Reporte_Diario**	
Descripción:	En esta tabla se almacenan los datos de los reportes diarios que realiza cada Probador luego de terminar la jornada de trabajo diario.	
Atributos:	**Tipo:**	**Descripción:**
id_reporte	Integer (11)	Identificador único del Reporte Diario
cant_errores	Integer (11)	Cantidad de errores encontrados durante la jornada de trabajo diaria.
cant_no_conform	Integer (11)	Cantidad de No Conformidades encontradas durante la jornada de trabajo diaria.
cant_CP_realiz	Integer (11)	Cantidad de Casos de Pruebas realizados

		durante la jornada de trabajo diaria.
id_DCP	Integer (11)	Identificador único del Diseño de Casos de Pruebas al que pertenece el Reporte Diario.
observaciones	Varchar (255)	Observaciones durante la jornada de trabajo diaria.

Tabla 19: Descripción de Tabla: T_Solicitud.

Nombre:	**T_Solicitud**	
Descripción:	En esta tabla se almacenan los datos descriptivos del proyecto que solicita las pruebas, así como los datos necesarios para las pruebas.	
Atributos:	**Tipo:**	**Descripción:**
id_solicitud	Integer (11)	Identificador único de la Solicitud.
id_proyecto	Integer (11)	Identificador único del proyecto que emitió la solicitud.
fase_proyecto	varchar (255)	Fase en la cual se realizarán las pruebas al proyecto.
prioridad	Integer (11)	Orden de prioridad definido por el Líder de Proyecto.
tiempo_pruebas	Integer (11)	Cantidad de días que se utilizarán para las pruebas.
recursos_pruebas	Varchar (255)	Recursos disponibles para la realización de las pruebas(Hardware,Software,RRHH)
referencias	Varchar(255)	Documentos a consultar para ejecutar las pruebas.

3.4- Tratamiento de Errores.

Uno de los principales requisitos exigidos por los clientes en todo tipo de aplicaciones es el correcto tratamiento de errores. En la medida que este aspecto sea tomado en cuenta mejorará la calidad del sistema y el usuario se sentirá satisfecho.

El sistema posee un grupo de validaciones constantes, de la información que entra al mismo. El objetivo de reducir las posibilidades de que se introduzca información errónea, por parte de los usuarios, a la Base de Datos que está vinculada al sistema. Cuando el usuario cometa algún error se le comunicará inmediatamente a través de mensajes de error los cuales informarán claramente al usuario lo que está sucediendo.

3.5- Seguridad.

Otro de los aspectos importantes a tener en cuenta cuando se esté desarrollando una aplicación seria, es el tema de la seguridad de los datos que se manejan. Es conveniente contar en el sistema con las medidas necesarias para asegurar la integridad de la información en todo momento. Esto permite lograr un alto nivel de confiablidad en el sistema. El sistema realiza el control de la seguridad principalmente mediante el control de acceso de los usuarios. Otro aspecto que contribuye a la seguridad del sitio es el hecho de que cada usuario registrado tiene definido un rol que es el que establece su nivel de acceso de y la información que podrá manejar en cada momento además de las acciones que podrá realizar.

3.6- Interfaz de Usuario.

El estilo visual de la interfaz debe ser bastante serio pero a la vez agradable, adecuándose a las actividades a realizar en la misma, además debe ser sencillo e intuitivo. Por otra parte, la organización de la información debe estar equilibrada mostrando el mismo orden de la misma en todas las páginas web. La cantidad de elementos que aparecerán en la pantalla será mínima, solo la necesaria en cada caso, y cada elemento debe conservar el mismo estilo en cuanto al tamaño, color, forma y alineación en la pantalla.

Las funcionalidades del sistema van apareciendo en la medida que se realiza cada actividad, brindando comodidad al usuario y evitando la necesidad de memorizar cada paso del proceso de pruebas.

Se utilizará “Arial” como fuente base para toda la tipografía de la aplicación ya que es una fuente con una excelente legibilidad y el texto estará marginado por ambos lados connotando formalidad. El tamaño de la fuente será de 13 puntos. Se hará uso del formato “Negrita” en el caso de que sea necesario resaltar alguna información importante.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBAS

4.1-Introducción:

El flujo de trabajo de Implementación, definido por RUP, describe cómo los elementos del modelo del diseño se implementan en términos de componentes, y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue. Este flujo está fuertemente determinado por el lenguaje de programación.

El siguiente capítulo tiene como objetivos principales definir la organización del sistema en términos de Subsistemas de Implementación organizados en capas. Además se Integran los componentes en un sistema ejecutable.

Los diagramas de despliegue y componentes conforman lo que se conoce como un modelo de implementación al describir los componentes a construir, y su organización y dependencia entre nodos físicos en los que funcionará a aplicación.

4.2- Modelo de Implementación.

Este modelo describe como los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, ficheros de código fuente, ejecutables etc. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y como dependen de los componentes unos de otros.

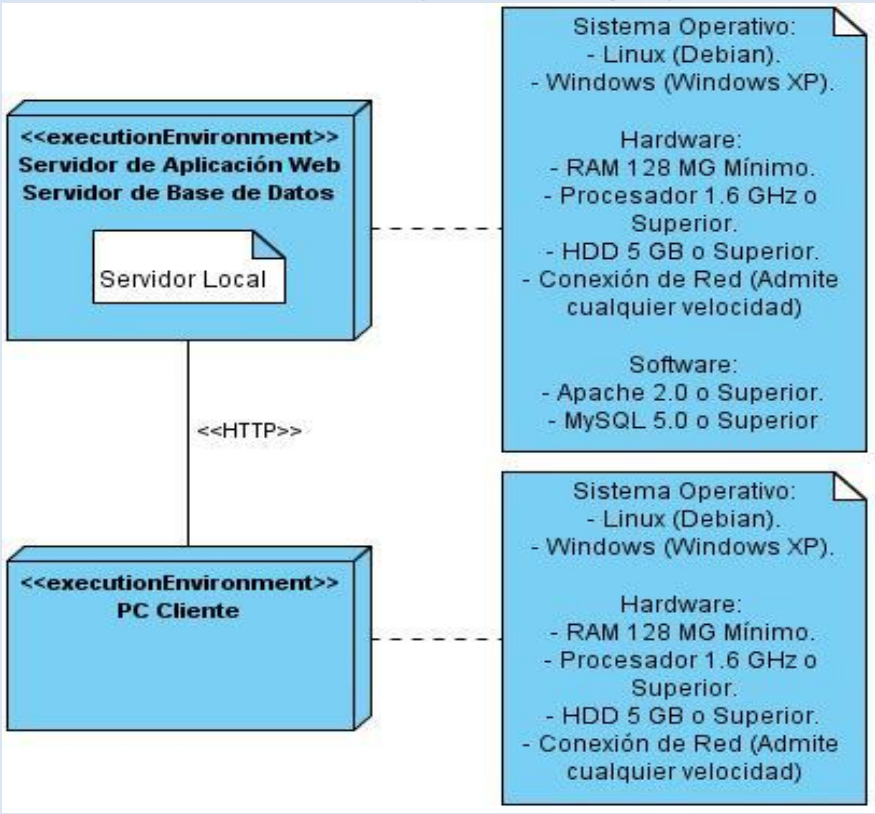
4.3- Diagramas de Implementación.

Los Diagramas de Implementación, a diferencia de los estáticos y de los dinámicos, no describen la funcionalidad del software, sino su estructura general con vistas a su construcción, ejecución e instalación. Son dos: El **Diagrama de Componentes:** que muestra cuales son las diferentes partes del software. El **Diagrama de Despliegue:** que describe la distribución física de los diferentes partes del software en tiempo de ejecución. (Campderrich Falgueras, et al., 2003)

4.3.1- Diagrama de Despliegue.

Un diagrama de Despliegue muestra cómo y dónde se desplegará el sistema. Las máquinas físicas y los procesadores se representan como nodos, y la construcción interna puede ser representada por nodos o artefactos embebidos. Como los artefactos se ubican en los nodos para modelar el despliegue del sistema, la ubicación es guiada por el uso de las especificaciones de despliegue. (G. Piattini, 1996)

Ilustración 17: Diagrama de Despliegue.



4.3.2- Diagrama de Componentes.

Los diagramas de componentes se distinguen de otros tipos de diagramas por su contenido. Estos contienen componentes, interfaces y relaciones entre ellos y como todos los diagramas pueden contener paquetes utilizados para agrupar elementos del modelo. Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes, sean éstos componentes de código fuente, binarios o ejecutables.

Los diagramas de componentes en muchos aspectos se pueden considerar como un diagrama de clases a gran escala. Cada componente en el mismo debe ser documentado con un diagrama de componentes más detallado, uno de clases, o uno de casos de uso. Estos se utilizan para modelar código fuente, versiones ejecutables, bases de datos físicas, entre otros.

Ilustración 18: Diagrama de Paquetes.

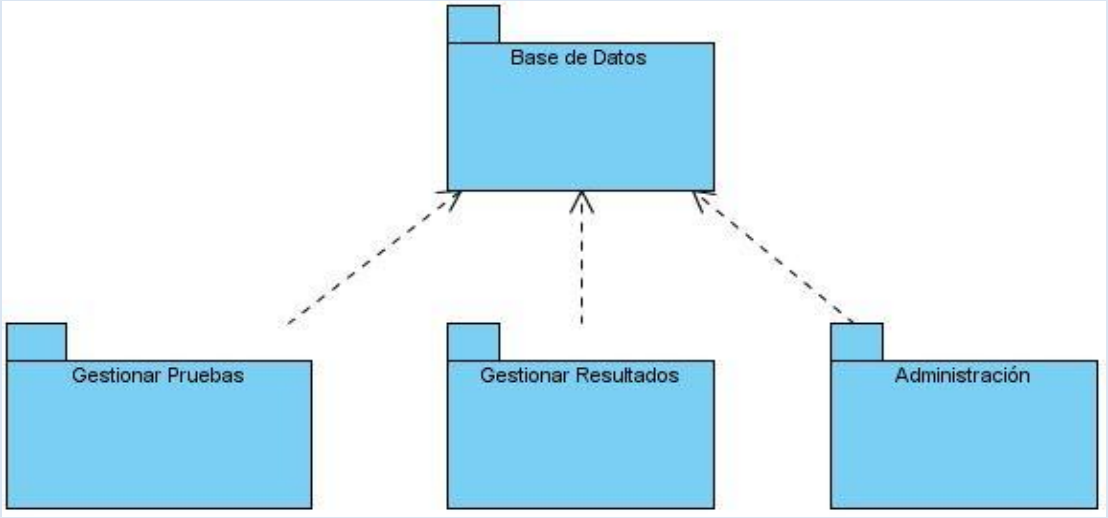


Ilustración 19: Diagrama de Componentes: Administración.

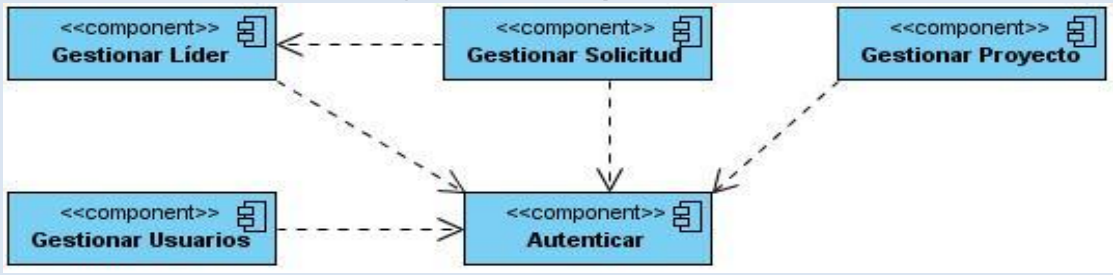


Ilustración 20: Diagrama de Componentes: Pruebas.

Ilustración 21: Diagrama de Componentes: Resultados.

4.4- Pruebas

Uno de los flujos de trabajo de apoyo definido por RUP es el de Pruebas, actividad en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos, los resultados son observados y registrados, y se realiza una evaluación de los diferentes aspectos del sistema o componente. Este flujo de trabajo se realiza en las fases de Elaboración, Construcción y Transición. Los artefactos claves que se elaboran en esta actividad son el Plan de Pruebas y la Estrategia de Pruebas.

Objetivos del Flujo de Trabajo de Prueba

- Encontrar y documentar los defectos que puedan afectar la calidad del software.
- Validar que el software trabaje como fue diseñado.
- Validar y probar los requisitos que debe cumplir el software en ese momento.
- Validar que los requisitos fueron implementados correctamente

4.5- Plan de Pruebas.

La construcción de un buen Plan de Pruebas es la piedra angular, y en consecuencia, el principal factor crítico de éxito para la puesta en práctica de un proceso de pruebas que permita entregar un software de mejor nivel. No obstante a que cada esfuerzo o proceso de pruebas puede ser diferente y específico, la mayor parte de los proyectos informáticos, sean de nuevos desarrollos o de mantenimiento de aplicaciones, tienen un marco común para la realización de las pruebas.

El Plan de pruebas describe la estrategia, recursos y planificación de las pruebas; proporciona el marco dentro del cual el equipo de prueba desarrolla las pruebas trabajando con los recursos y la planificación dada.

4.5.1- Introducción.

El desarrollo de aplicaciones informáticas implica una serie de actividades de desarrollo de software en las que es bastante común cometer errores. Esta es la principal razón por la que se necesita probar el software periódicamente con el objetivo de poder encontrar y solucionar estos errores aun cuando hay tiempo, y cuando su solución es viable al no implicar grandes atrasos en la terminación del software.

Cada proceso de pruebas a realizar debe ser planificada con antelación, para lograr de esta manera que los resultados alcanzados sean los esperados y que el proceso de pruebas tenga la calidad necesaria y cumpla con los objetivos por los que se realiza.

4.5.2- Propósito.

El propósito de este Plan de Pruebas es definir la estrategia de pruebas para la aplicación CCALPHA, además se pretende determinar con antelación los recursos necesarios para el mismo, definir los procedimientos, guías, plantillas que serán utilizadas y exponer los RF y CU que serán probados.

4.5.3- Alcance.

El alcance de este Plan de Pruebas comprende la planificación para la ejecución de las Pruebas de Sistema a la aplicación CCALPHA. Estas pruebas son usualmente conducidas para comprobar que todos los módulos trabajan juntos sin error. Examina si la aplicación cumple con los requerimientos de la organización, su utilidad, seguridad y desempeño.

4.5.4- Especificaciones de Software y Hardware.

La aplicación hace uso de la siguiente lista de Software y Hardware:

Software:

- Sistema Operativo Linux o Windows.

- Navegador de HTML.
- Gestor de Base de Datos MySQL.
- Servidor Web Apache.

Hardware:

- Computadora Servidor.
- Computadora Cliente.
- Red de Computadoras.

4.5.5- Descripción de los Requisitos.

En esta sección del Plan de Pruebas se listan todos los RF que serán probados. Cualquier requisito no incluido en esta lista queda fuera del alcance del Plan de Pruebas y del proceso de pruebas en general.

RF a probar:

- Adicionar Usuario.
- Adicionar Proyecto.
- Registrar Reporte.
- Solicitar Pruebas.

4.5.6- Estrategia de Pruebas.

Luego de realizar un minucioso análisis sobre el estado de terminación de la aplicación a probar y los objetivos de la realización del Plan de Pruebas, se determinó que en este caso la estrategia de pruebas quedaría definida de la siguiente forma:

Definición del Nivel de Pruebas: Pruebas de Sistema.
Definición del Tipo de Pruebas: Pruebas de Funcionalidad.
Definición del Método de Prueba: Prueba de Caja Negra.
Definición de las Técnicas de Pruebas: Particiones de Equivalencia, Análisis de Valores Límites y Conjetura de Errores.

Esta estrategia de pruebas intenta encontrar errores de las siguientes categorías:

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

4.5.7- Proceso de Pruebas.

El proceso de pruebas estará compuesto por tres actividades fundamentales:

1- **Preparar la ejecución de las pruebas:**
   - Evaluar documentación con la que se cuenta
   - Comprobar que están listos todos los recursos previstos.
   - Asignar las tareas al equipo de pruebas.
   - Preparar los documentos de trabajo (rediseñar los casos de prueba de ser necesario).

2- **Medición.**
   - Ejecutar las pruebas planificadas (Registrar los resultados).
   - Intercambiar información periódicamente.
   - Evaluar el progreso de las pruebas.
   - Reasignar tareas entre los miembros del equipo de pruebas según sea necesario.
   - Comunicar periódicamente a los desarrolladores las no conformidades detectadas.

3- **Evaluación.**
   - Valorar los resultados alcanzados con las pruebas en cada iteración.

Al concluir la ejecución de todos los casos de prueba se le enviará a los desarrolladores un documento “Resumen de no conformidades detectadas”; estos deben corregir las no conformidades detectadas y enviar al equipo de pruebas el documento “Respuesta a las no conformidades detectadas”. Cuando todas las no conformidades hayan sido cerradas, se vuelve a instalar la nueva versión de la aplicación y se comienza una nueva iteración de pruebas.

4.5.8- Casos de Pruebas.

Son un conjunto de condiciones o variables bajo las cuales se determina si los requisitos de una aplicación son parcial o completamente satisfactorios.

CP 1- Adicionar Usuario.

Descripción:

Este caso de pruebas le permite al Administrador del Sistema adicionar un nuevo Usuario.

Flujo Central:

- El Administrador del Sistema ejecuta la aplicación en el navegador web.
- El sistema carga la interfaz principal de la aplicación.
- El Administrador del Sistema inicia su sesión y se muestra su interfaz de trabajo.
- El Administrador del Sistema solicita adicionar un Usuario.
- El sistema muestra la interfaz que permite adicionar un Usuario.
- El Administrador del Sistema introduce todos los datos solicitados y acepta.
- El sistema muestra un mensaje indicando que la acción fue realizada satisfactoriamente y muestra la interfaz de trabajo del Administrador del Sistema.

Condiciones de Ejecución:

- El Administrador del Sistema debe estar autenticado en el sistema.

Iteraciones:

Ver las Iteraciones en Anexo 41.

CP 2- Adicionar Proyecto.

Descripción:

Este caso de pruebas le permite al Administrador del Sistema adicionar un nuevo Proyecto.

Flujo Central:

- El Administrador del Sistema ejecuta la aplicación en el navegador web.
- El sistema carga la interfaz principal de la aplicación.
- El Administrador del Sistema inicia su sesión y se muestra su interfaz de trabajo.
- El Administrador del Sistema solicita adicionar un Proyecto.
- El sistema muestra la interfaz que permite adicionar un Proyecto.
- El Administrador del Sistema introduce todos los datos solicitados y acepta.
- El sistema muestra un mensaje indicando que la acción fue realizada satisfactoriamente y muestra la interfaz de trabajo del Administrador del Sistema.

Condiciones de Ejecución:

- El Administrador del Sistema debe estar autenticado en el sistema.

Iteraciones:

Ver las Iteraciones en Anexo 42.

CP 3- Registrar Reporte.

Descripción:

Este caso de pruebas le permite al Probador registrar un nuevo Reporte.

Flujo Central:

- El Probador ejecuta la aplicación en el navegador web.
- El sistema carga la interfaz principal de la aplicación.
- El Probador inicia su sesión y se muestra su interfaz de trabajo.
- El Probador solicita adicionar un Reporte.
- El sistema muestra la interfaz que permite registrar un Reporte.
- El Probador introduce todos los datos solicitados y acepta.
- El sistema muestra un mensaje indicando que la acción fue realizada satisfactoriamente y muestra la interfaz de trabajo del Probador.

Condiciones de Ejecución:

- El Probador debe estar autenticado en el sistema.

Iteraciones:

Ver las Iteraciones en Anexo 43

CP 4- Solicitar Pruebas.

Descripción:

Este caso de pruebas le permite al Líder de Proyecto solicitar un proceso de pruebas.

Flujo Central:

- El Líder de Proyecto ejecuta la aplicación en el navegador web.
- El sistema carga la interfaz principal de la aplicación.
- El Líder de Proyecto inicia su sesión y se muestra su interfaz de trabajo.
- El Líder de Proyecto seleccionar la opción de Solicitar Pruebas.
- El sistema muestra la interfaz que permite Solicitar Pruebas.
- El Líder de Proyecto introduce todos los datos solicitados y acepta.
- El sistema muestra un mensaje indicando que la acción fue realizada satisfactoriamente y muestra la interfaz del Líder de Proyecto

Condiciones de Ejecución:

- El Líder de Proyecto debe estar autenticado en el sistema.

Iteraciones:

Ver las Iteraciones en Anexo 44.

CONCLUSIONES

Luego de culminada la investigación se desarrolló un prototipo de aplicación informática que mejora en gran medida el proceso de pruebas en el PHA, al permitir un mejor control y gestión del mismo. Como parte de este proceso se le dio cumplimiento a los objetivos propuestos y es posible plantear las siguientes conclusiones:

- Se analizaron detalladamente los aspectos teóricos conceptuales correspondientes al objeto de estudio y al campo de acción planteados. Para ello se consultaron diversas bibliografías y los resultados fueron expuestos a lo largo de este documento.

- El prototipo de aplicación se desarrolló siguiendo lo establecido por RUP, que fue la metodología seleccionada; se utilizaron representaciones UML para la modelación en todas las fases del proyecto.

- Se seleccionaron y utilizaron las herramientas necesarias para el desarrollo teniendo en cuenta en todo momento las numerosas ventajas de la utilización de software libre, además de que, tanto las herramientas para el desarrollo, como la aplicación final, fueran multiplataforma.

- Se logró una correcta identificaron los procesos principales del negocio, se definieron los requerimientos del sistema, tanto funcionales como no funcionales, estructurándose además, el modelo de casos de uso del sistema.

- Se diseñó la Base de Datos que cumple con todos los requisitos de seguridad y confiabilidad, permite gestionar eficientemente los datos obtenidos durante el proceso de pruebas en el PHA.

- Se realizó el análisis y diseño del sistema a través de sus diagramas de clases correspondientes, además de otros diagramas que simplificaron y permitieron la implementación del prototipo de aplicación web.

Por tanto se puede concluir que el objetivo general, y las tareas de la investigación propuestas para el presente trabajo investigativo han sido cumplidos satisfactoriamente incluyéndose una serie de recomendaciones que deben tenerse en cuenta para el desarrollo futuro de este trabajo.

RECOMENDACIONES

Una de las principales características del ciclo de vida de RUP, es concebir un desarrollo de software iterativo e incremental. Propone que cada fase se desarrolle en iteraciones. Las iteraciones hacen referencia a pasos en los flujos de trabajo, y los incrementos, al crecimiento del producto.

En el presente trabajo se realizó la primera iteración, la misma involucró actividades de la mayoría de los flujos de trabajo, y al final se obtuvo un prototipo, por lo que quedaron algunas cuestiones que, por diversos factores como el tiempo y la prioridad, no fueron analizadas a profundidad. A continuación se relacionan algunos aspectos que se consideran necesarios para darle continuidad a este trabajo:

- Seguir investigando acerca del objeto de estudio y del campo de acción planteados para tener un conocimiento más profundo de los mismos.

- Realizar los demás flujos de Trabajo de apoyo establecidos por RUP.

- Mejorar la interfaz gráfica del prototipo, así como las funcionalidades implementadas hasta el momento.

- Implementar una versión estable que cuente con todas las funcionalidades necesarias.

- Incorporar un módulo de asignación de tareas efectivo que no dependa ni se realice mediante las otras herramientas con las que cuanta el PHA para este objetivo.

- Incorporar módulos para gestionar auditorías, inspecciones y revisiones a los productos software.

- Dar mantenimiento periódicamente tanto al sistema como a la Base de Datos, con el objetivo de extender su vida útil.

- Fomentar el desarrollo de aplicaciones similares en el Polo de Realidad Virtual, en el resto de la Universidad y en los demás lugares que se produzcan o se prueben productos software.

- Lograr una buena capacitación sobre el uso de la herramienta y sobre el proceso de pruebas a los usuarios que van a estar relacionados directamente con el proceso de pruebas y a los líderes de los proyectos del PHA.

BIBLIOGRAFIA

1. **De Antonio Jiménez, Angélica.** *Gestión, Control y Garantía de la Calidad de Software.*

2. **S. Pressman, Roger.** *Ingeniería de Software un enfoque práctico.* 5ta Edición.

3. **M. Minguet Melián, Jesús y Hernández Ballesteros, Juan Francisco.** *La calidad del software y su medida.* 1ra. s.l. : EDITORIAL UNIVERSITARIA RAMON ARECES, 2003.

4. **H. Canós, José, Letelier, Patricio y Penadés, Mª Carmen.** *Metodologías Ágiles en el Desarrollo de Software.*

5. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Educacion S.A, 2000. 84-7829-036-2.

6. **Callejas Cuervo, Mauro, Baquero, Yovany y Moreno, Oscar.** *Herramientas libres para modelar software.*

7. **G. Piattini, Mario.** *Análisis y diseño detallado de aplicaciones informáticas de gestión.* Madrid : RA-MA Editorial, 1996.

8. **Campderrich Falgueras, Benet y Maspons, Ramon.** *Ingeniería del software.* s.l. : UOC, 2003.

9. **Berzal Galiano, Fernando.** *Diseño de arquitecturas software.*

TRABAJOS CITADOS

B.Grady, Robert. 1992. *Software Metrics for Project Management and Process Improvement.* New Jersey : Prentice Hall, 1992.

Berzal Galiano, Fernando. *Diseño de arquitecturas software.*

Callejas Cuervo, Mauro, Baquero, Yovany y Moreno, Oscar. *Herramientas libres para modelar software.*

Campderrich Falgueras, Benet y Maspons, Ramon. 2003. *Ingeniería del software.* s.l. : UOC, 2003.

De Antonio Jiménez, Angélica. *Gestión, Control y Garantía de la Calidad de Software.*

G. Piattini, Mario. 1996. *Análisis y diseño detallado de aplicaciones informáticas de gestión.* Madrid : RA-MA Editorial, 1996.

EA 7.0. *Guía de Usuario de Enterprise Architect 7.0.* EA 7.0.

H. Canós, José, Letelier, Patricio y Penadés, Mª Carmen. *Metodologías Ágiles en el Desarrollo de Software.*

ISO 9000. *ISO 9000:2000.*

ISO 9001. *ISO 9001:2000.*

ISO 9004. *ISO 9004:2000.*

ISO 9126. *ISO 9126:2000.*

ISO. 8402. *ISO 8402:2000.*

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Educacion S.A, 2000. 84-7829-036-2.

M. Minguet Melián, Jesús y Hernández Ballesteros, Juan Francisco. 2003. *La calidad del software y su medida.* 1ra. s.l. : EDITORIAL UNIVERSITARIA RAMON ARECES, 2003.

Raja Prado, Elena. 2007. *Actas de Talleres de Ingeniería del Software y Bases de Datos.* 2007. Vol. 1.

Real Academia Española. 2001. *Diccionario de la lengua española.* 22. 2001. 2.

S. Pressman, Roger. *Ingeniería de Software un enfoque práctico.* 5ta Edición.

IEEE 1990. *Standar 610 de la IEEE.* IEEE 1990.

ANEXOS

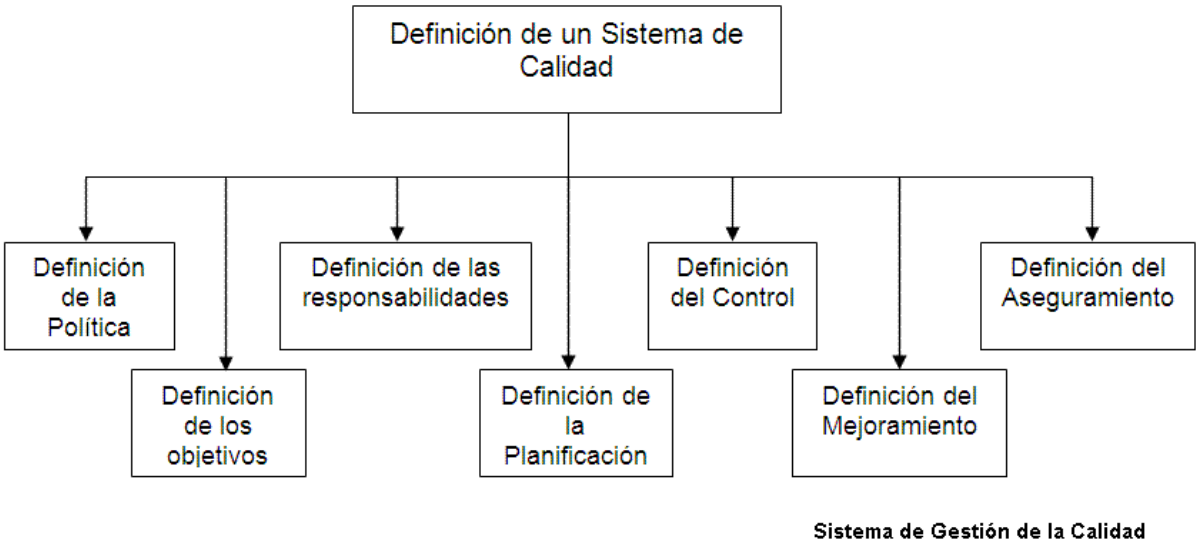


Ilustración 22: Anexo1- Sistema de Gestión de Calidad.

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

Ilustración 23: Anexo 2- Diferencia entre metodologías.

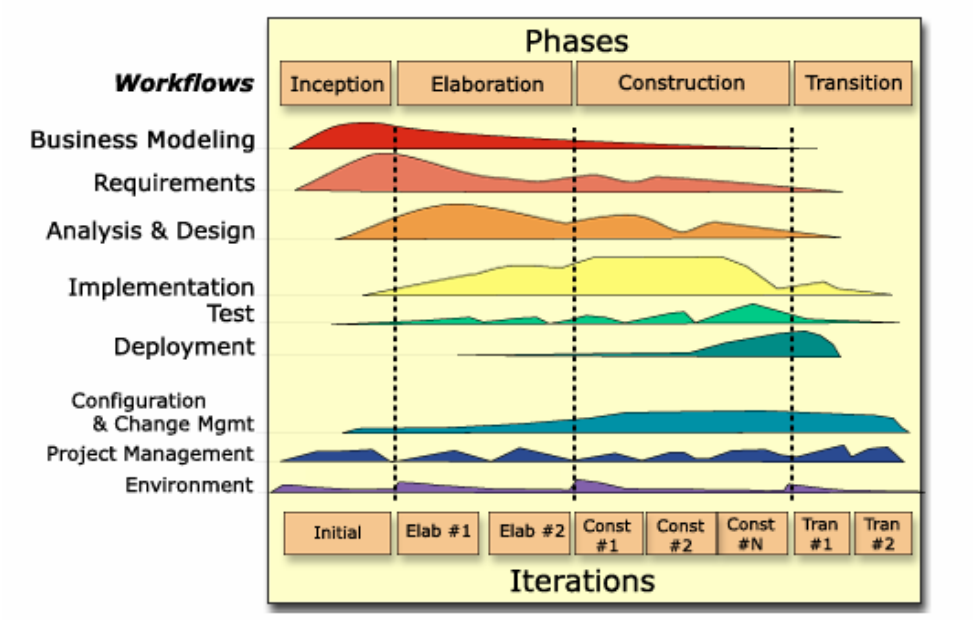


Ilustración 24: Anexo 3- Fases de RUP.

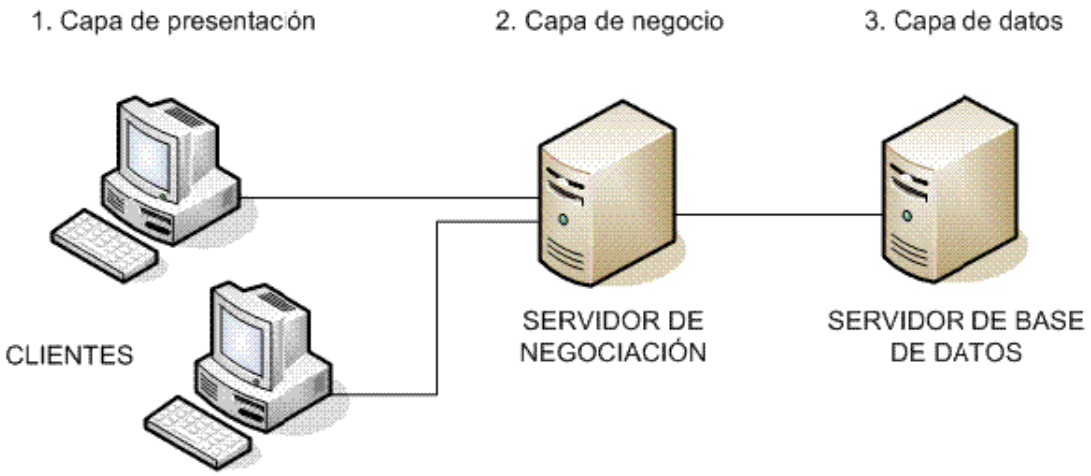


Ilustración 25: Anexo 4- Arquitectura Multicapas.

Tabla 20: Anexo 5- DefCUS: Gestionar Líder de Proyecto.

Nombre de Caso de Uso	**Gestionar Líder de Proyecto**
Actor	Administrador del Sistema
Descripción	El CU inicia cuando el Administrador del Sistema accede a la interfaz de administración de la aplicación web y selecciona la opción de Gestionar Líder, el sistema muestra las opciones: adicionar, eliminar y modificar líder.
Referencia	**RF1.1**

Tabla 21: Anexo 6- DefCUS: Seleccionar Equipo de Pruebas.

Nombre de Caso de Uso	**Seleccionar Equipo de Pruebas**
Actor	Administrador de Pruebas
Descripción	El CU inicia cuando el Administrador de Pruebas accede a la interfaz de la aplicación correspondiente a la creación del equipo de pruebas, el sistema muestra los datos correspondientes a todos los usuarios registrados, tales como el nombre, rol que desempeñan y la disponibilidad. El Administrador de pruebas selecciona el equipo de pruebas en dependencia de las necesidades del equipo a probar y de la disponibilidad de cada usuario del sistema.
Referencia	**RF2**

Tabla 22: Anexo 7- DefCUS: Gestionar Plan de Pruebas.

Nombre de Caso de Uso	Gestionar Plan de Pruebas
Actor	Diseñador de Casos de Pruebas
Descripción	El CU inicia cuando el Diseñador de Casos de Pruebas accede a la interfaz donde se gestiona el Plan de Pruebas. El sistema muestra las opciones: adicionar, modificar y eliminar plan de pruebas.
Referencia	RF3.1

Tabla 23: Anexo 8- DefCUS: Gestionar Casos de Pruebas.

Nombre de Caso de Uso	Gestionar Casos de Pruebas
Actor	Diseñador Casos de Pruebas
Descripción	El CU inicia cuando el Diseñador de Casos de Pruebas accede a la interfaz donde se gestionan los Casos de Pruebas. El sistema muestra las opciones: adicionar, modificar y eliminar casos de pruebas.
Referencia	RF3.2

Tabla 24: Anexo 9- DefCUS: Gestionar Documento de No Conformidades.

Nombre de Caso de Uso	Gestionar Documento de No Conformidades
Actor	Probador
Descripción	El CU inicia cuando el Probador accede a la interfaz donde se registran las no conformidades observadas en las pruebas, el sistema muestra los campos: adicionar, modificar y eliminar no conformidades.
Referencia	RF3.3

Tabla 25: Anexo 10- DefCUS: Solicitar Reportes General.

Nombre de Caso de Uso	Solicitar Reportes General

Actor	Líder de Proyecto
Descripción	El CU inicia cuando el Líder de proyecto escoge la opción: Solicitar Reporte, el sistema muestra todos los reportes realizados por los probadores hasta el momento.
Referencia	**RF3.5**

Ilustración 26: Anexo 11- DiagCA: Autenticar Usuario

Ilustración 27: Anexo 12- DiagCA: Gestionar Líder de Proyecto.

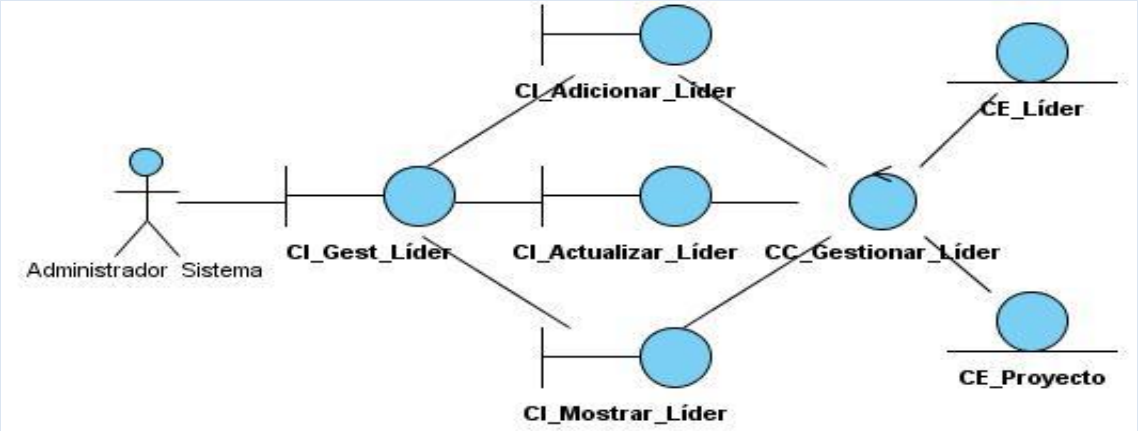


Ilustración 28: Anexo 13- DiagCA: Gestionar Equipo de Pruebas.

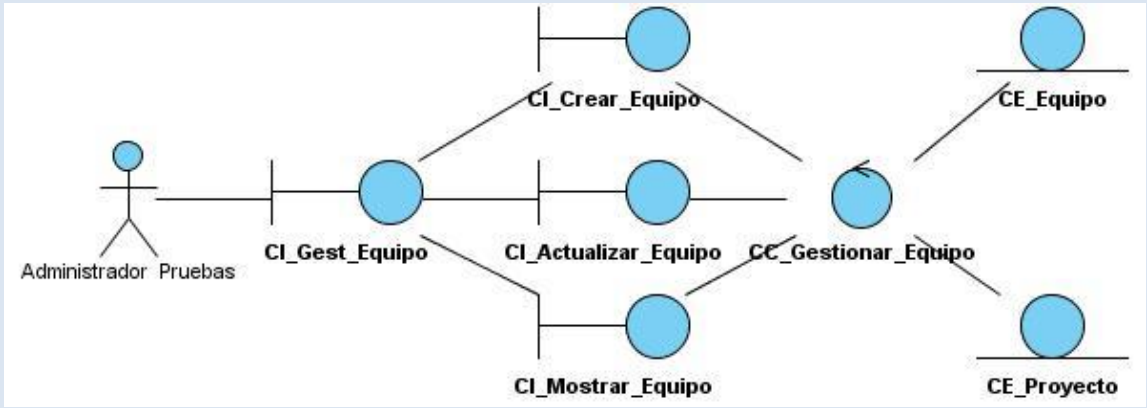


Ilustración 29: Anexo 14- DiagCA: Gestionar Plan de Pruebas.

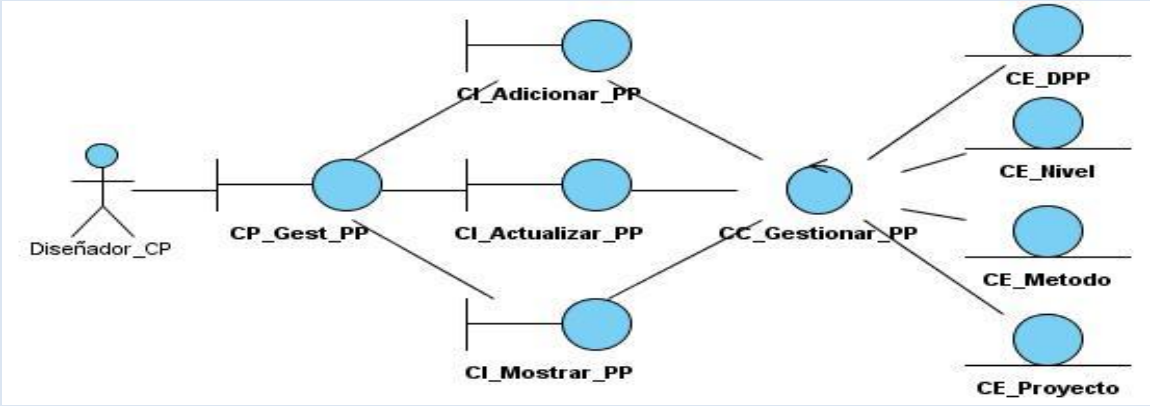


Ilustración 30: Anexo 15- DiagCA: Gestionar Casos de Pruebas.

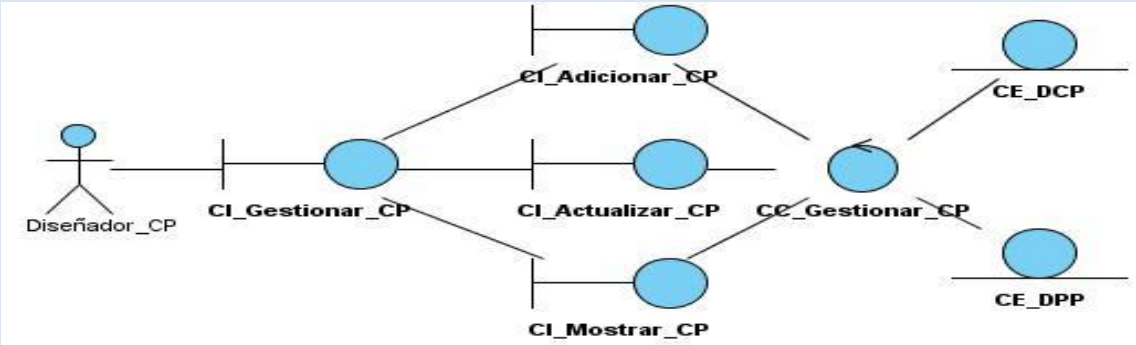


Ilustración 31: Anexo 16- DiagCA: Gestionar No Conformidades.

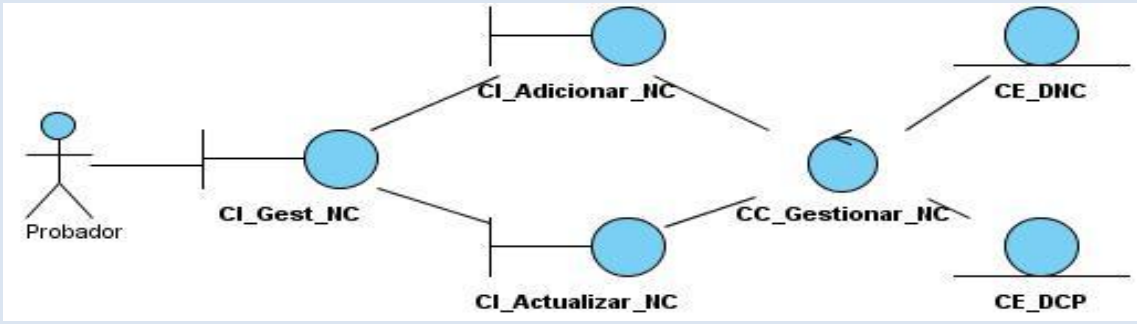


Ilustración 32: Anexo 17- DiagCA: Solicitar Reporte General.

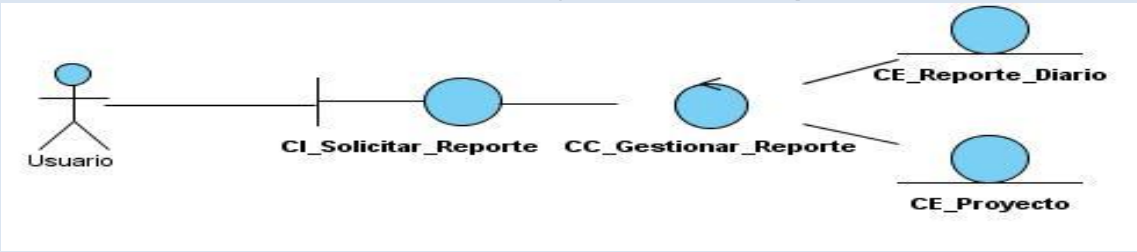


Ilustración 33: Anexo 18- DiagCD: Autenticar Usuario.

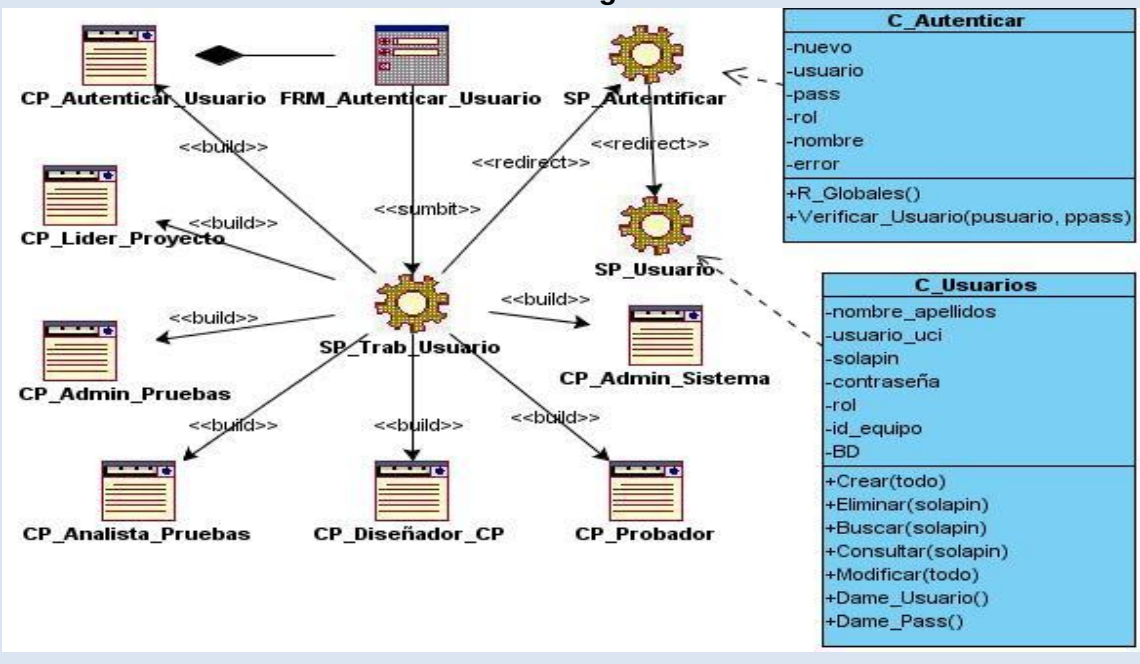


Ilustración 34: Anexo 19- DiagCD: Gestionar Líder.

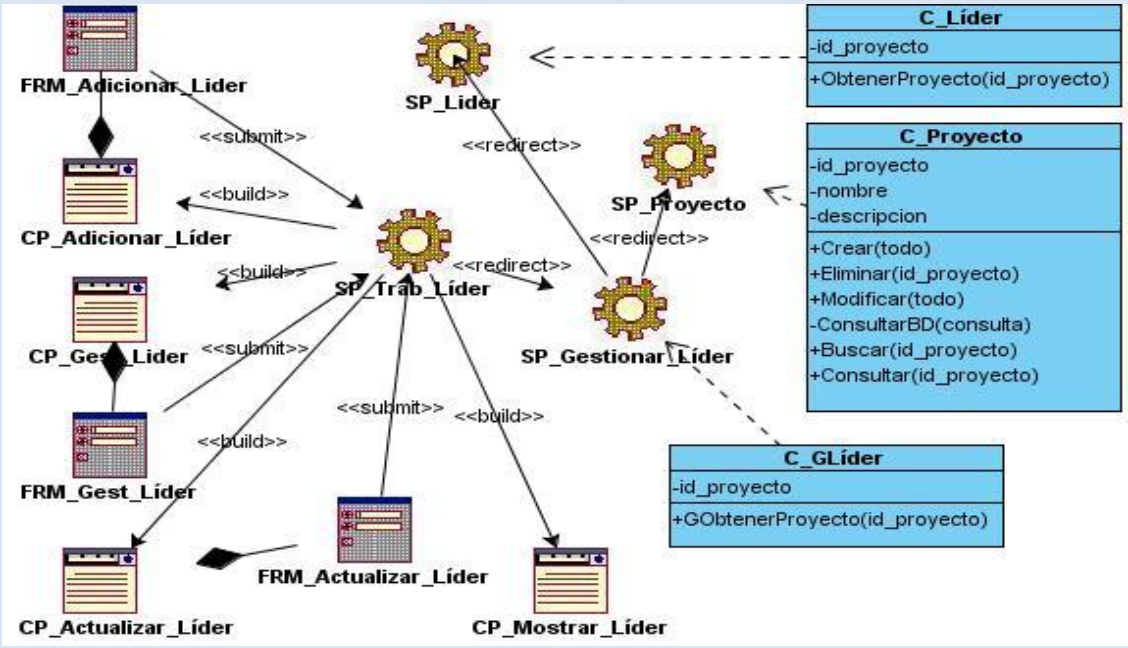


Ilustración 35: Anexo 20- DiagCD: Gestionar Equipo de Pruebas.

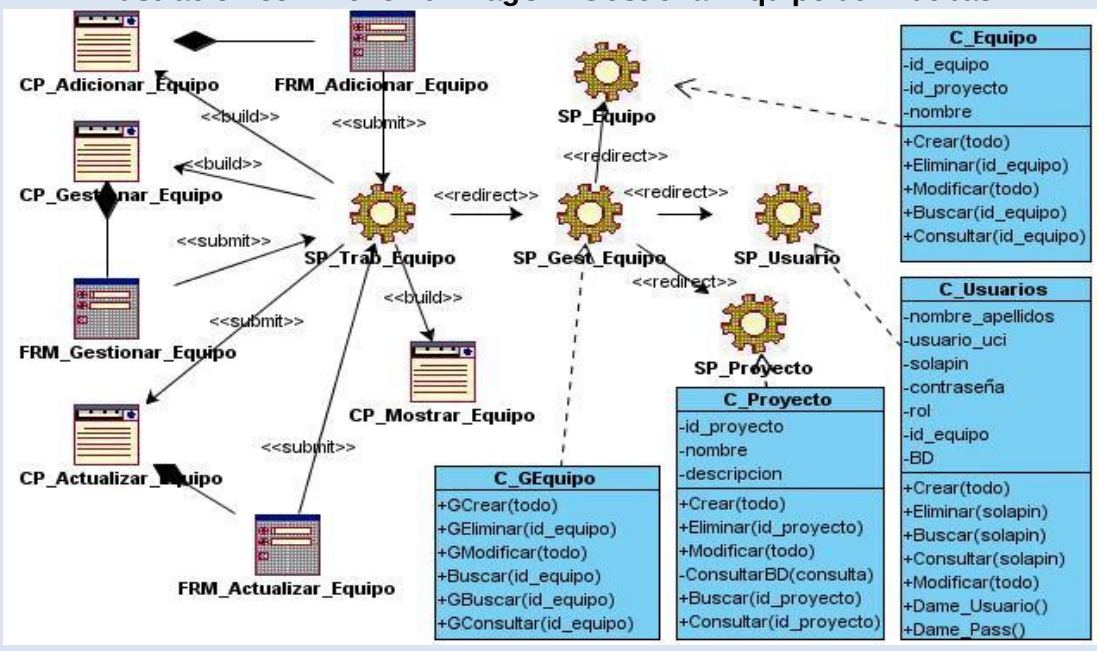


Ilustración 36: Anexo 21- DiagCD: Gestionar Plan de Pruebas.

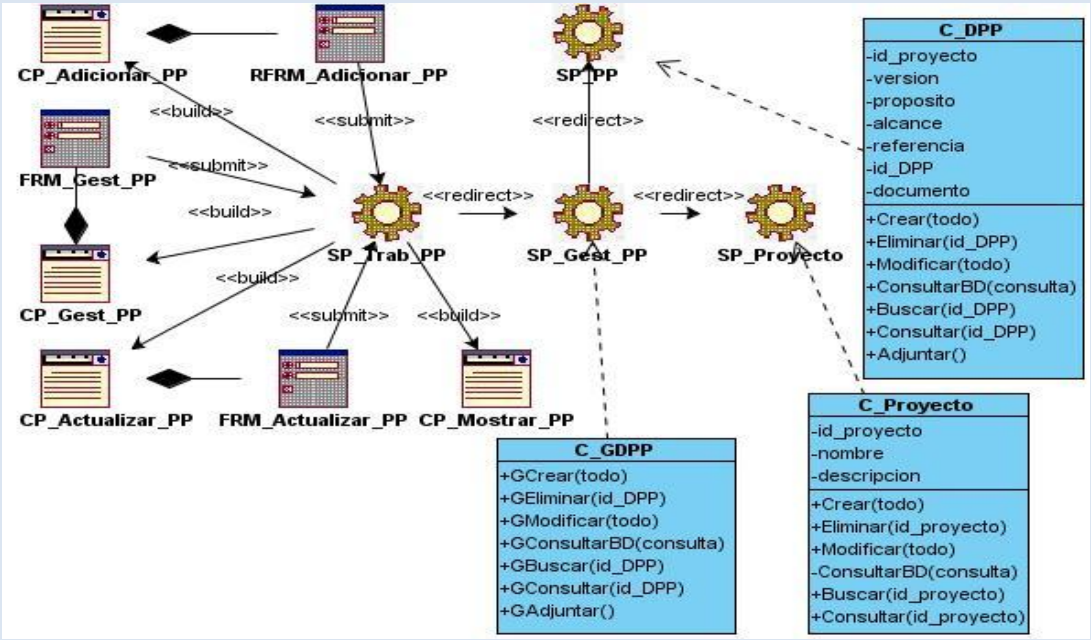


Ilustración 37: Anexo 22- DiagCD: Gestionar Casos de Pruebas.

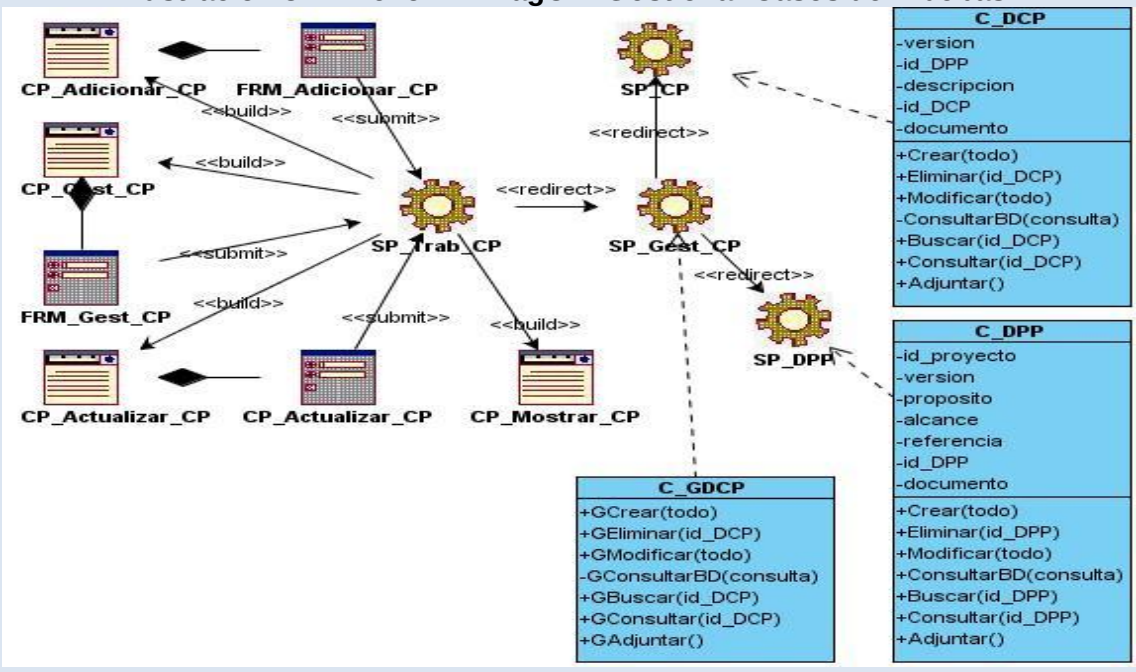


Ilustración 38: Anexo 23- DiagCD: Gestionar DNC.

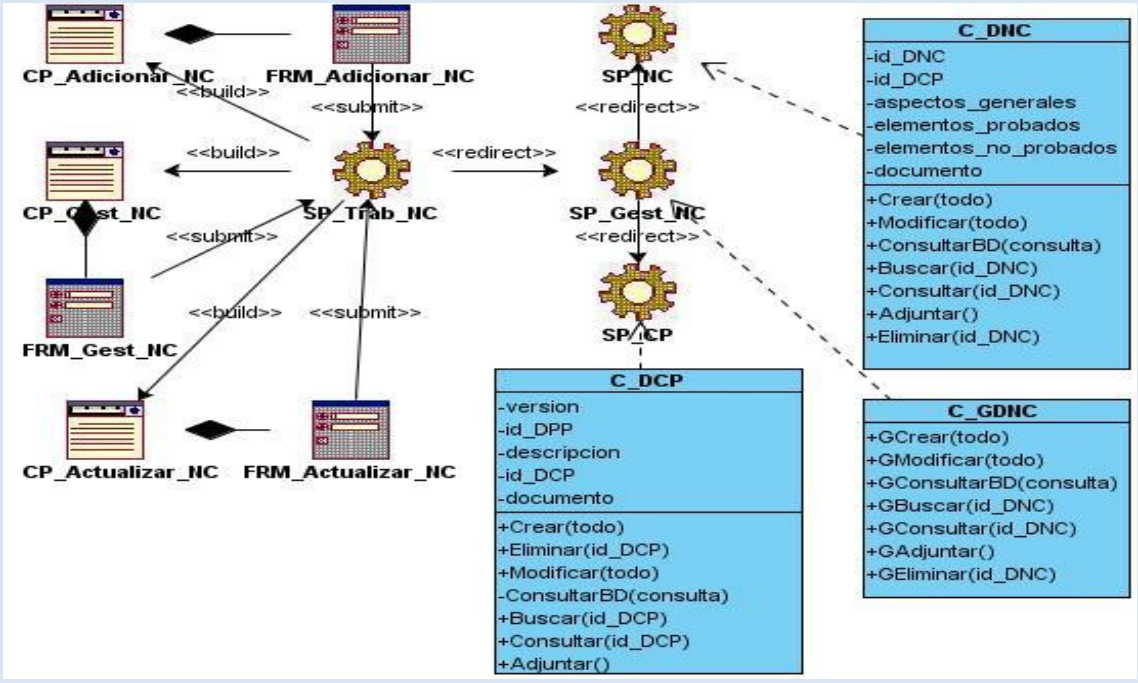


Ilustración 39: Anexo 24- DiagCD: Solicitar Reporte General.

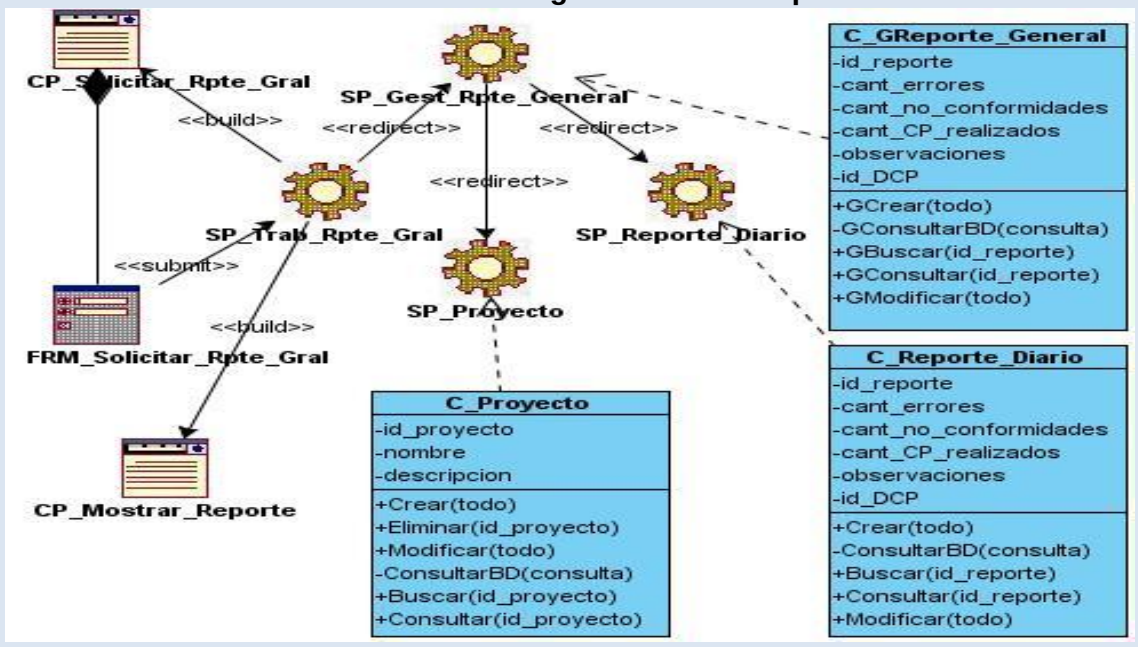


Ilustración 40: Anexo 25 Diag de Sec: Autenticar Usuario.

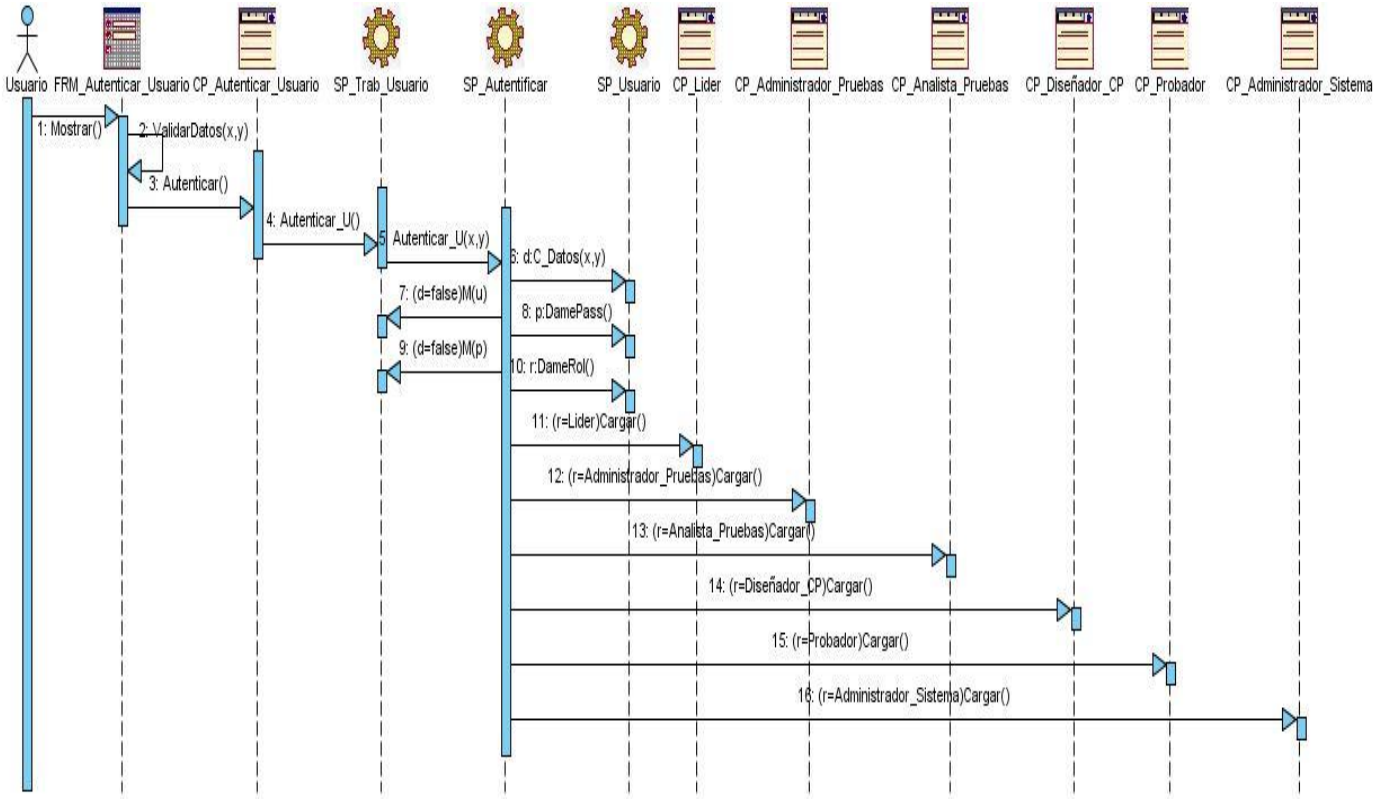


Ilustración 41: Anexo 26 Diag de Sec: Gestionar Reporte Diario.

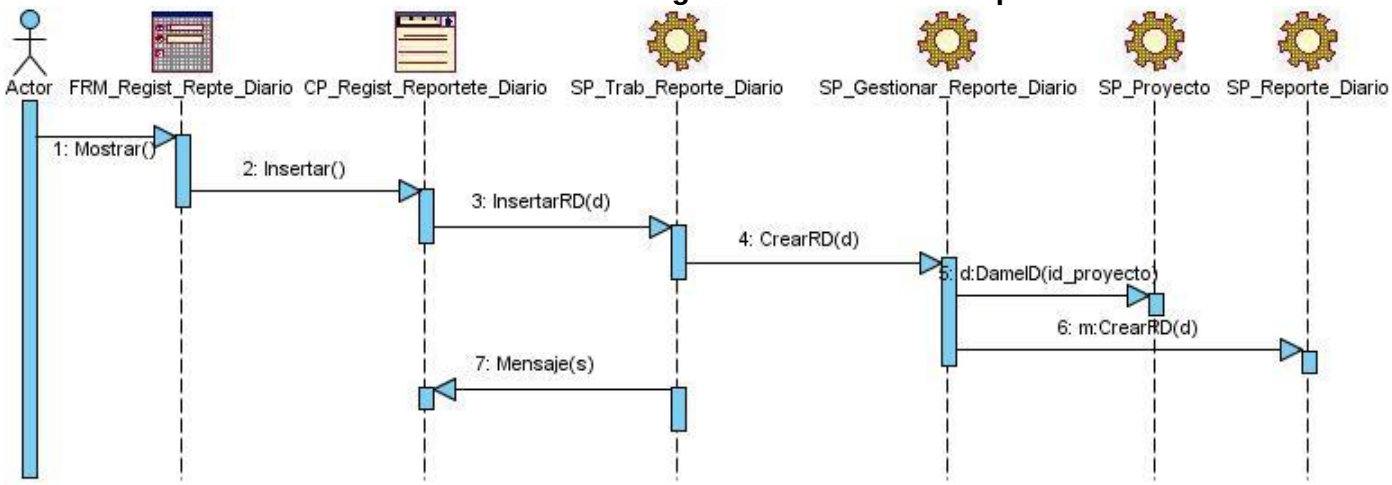


Ilustración 42: Anexo 27 Diag de Sec: Solicitar Reporte General.

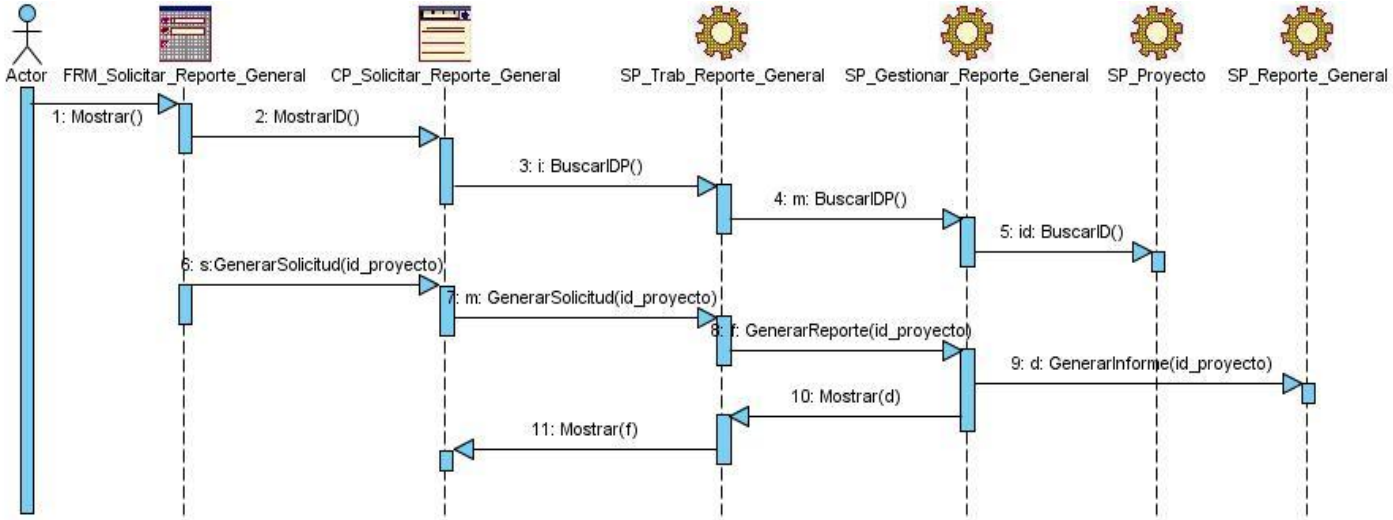


Ilustración 43: Anexo 28 Diag de Sec: Adicionar Usuario.

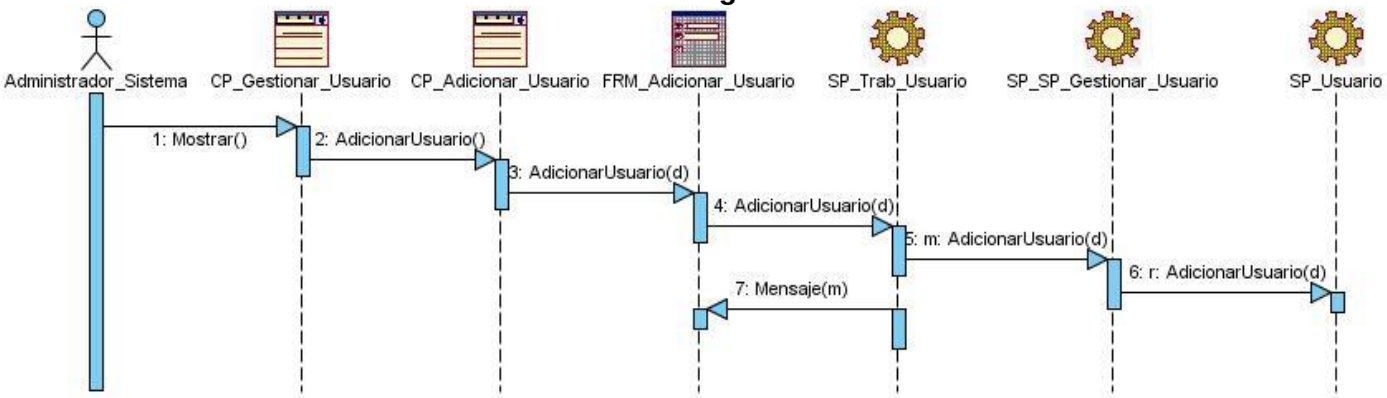


Ilustración 44: Anexo 29 Diag de Sec: Eliminar Usuario.

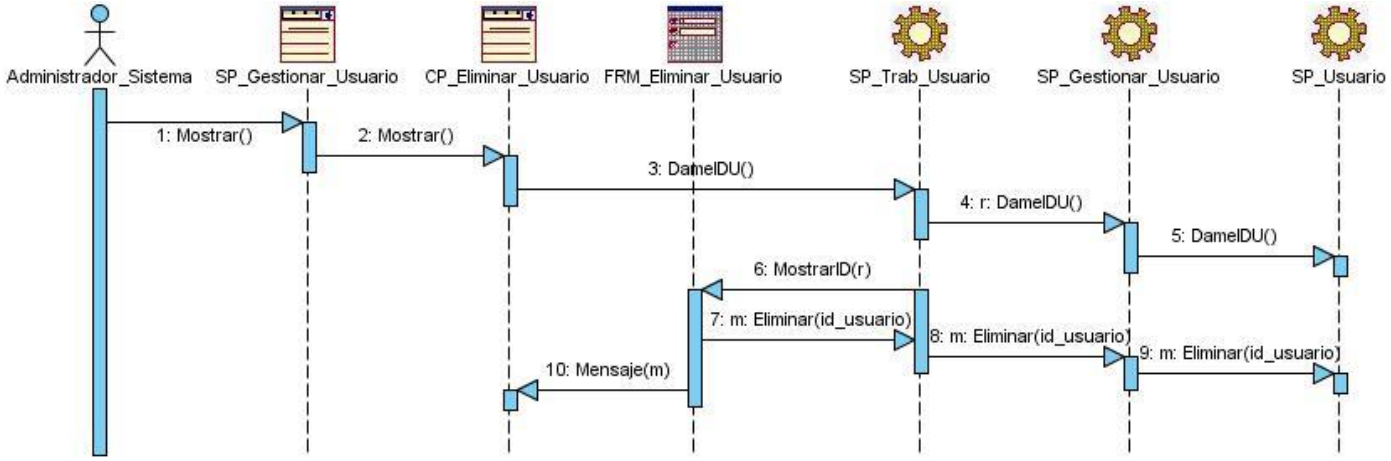


Ilustración 45: Anexo 30 Diag de Sec: Modificar Usuario.

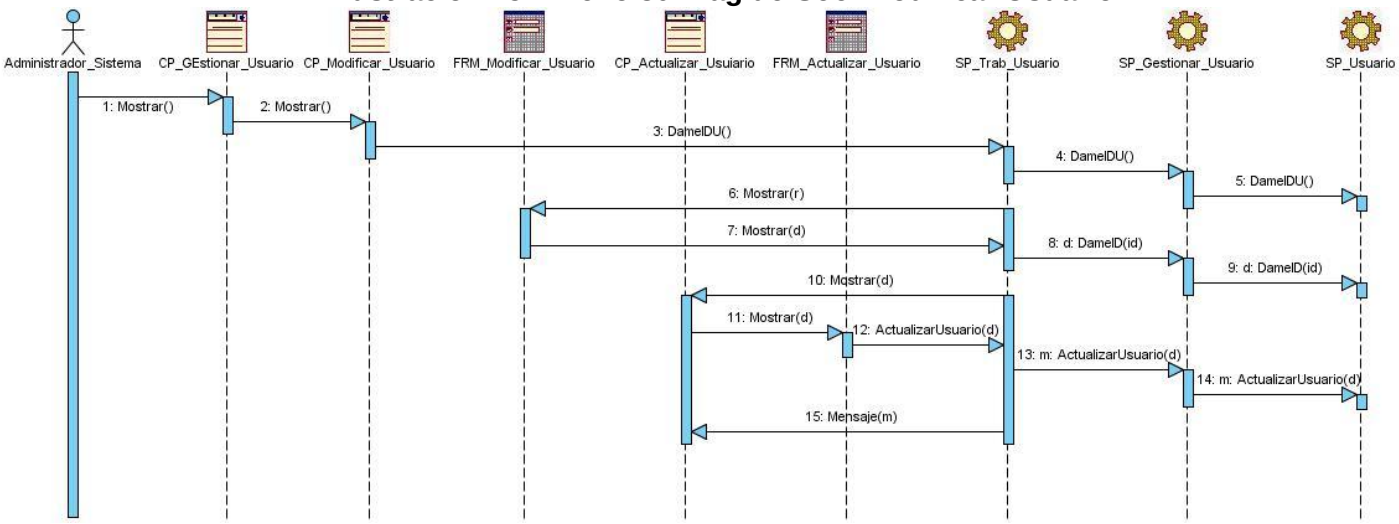


Tabla 26: Anexo 31 - Descripción Clase de Diseño: C_Lider

Nombre:	**C_Lider**
Tipo de Clase:	Entidad
Atributos:	**Tipo:**
id_proyecto	Integer

Para Cada Responsabilidad	
Nombre:	**ObtenerProyecto(id_proyecto)**
Descripción:	Para obtener el proyecto al que pertenece el líder.

Tabla 27: Anexo 32- Descripción Clase de Diseño: C_Equipo

Nombre:	**C_Equipo**
Tipo de Clase:	Entidad
Atributos:	**Tipo:**
id_equipo	Integer
nombre	Varchar
id_proyecto	Integer
jefe	Varchar
objetivo	Varchar
Para Cada Responsabilidad	
Nombre:	**Crear(todo)**
Descripción:	Adicionar un nuevo equipo de pruebas dados sus atributos.
Nombre:	**Eliminar(id_equipo)**
Descripción:	Para eliminar un equipo existente dado su identificador.
Nombre:	**Modificar(todo)**
Descripción:	Para modificar y actualizar los equipos dados sus atributos.
Nombre:	**ConsultarBD(consulta)**
Descripción:	Para realizar diferentes consultas a la Base de Datos.
Nombre:	**Buscar(id_equipo)**
Descripción:	Para determinar existencia de equipo.
Nombre:	**Consultar(id_equipo)**

Descripción:	Mostrar equipo dado su identificador.

Tabla 28: Anexo 33- Descripción Clase de Diseño: C_DPP

Nombre:	**C_DPP**
Tipo de Clase:	Entidad
Atributos:	**Tipo:**
id_DPP	Integer
id_proyecto	Integer
version	Varchar
proposito	Varchar
alcance	Varchar
referencias	Varchar
documento	Varchar
Para Cada Responsabilidad	
Nombre:	**Crear(todo)**
Descripción:	Adicionar un nuevo Documento de Plan de Pruebas dados sus atributos.
Nombre:	**Eliminar(id_DPP)**
Descripción:	Para eliminar un Documento de Plan de Pruebas dado su identificador.
Nombre:	**Modificar(todo)**
Descripción:	Para modificar y actualizar los datos del Documento de Plan de Pruebas dados sus atributos.
Nombre:	**ConsultarBD(consulta)**
Descripción:	Para realizar diferentes consultas a la Base de Datos.
Nombre:	**Buscar(id_DPP)**

Descripción:	Para determinar existencia de Documento de Plan de Pruebas dado su identificador.
Nombre:	**Consultar(id_DPP)**
Descripción:	Mostrar Documento de Plan de Pruebas dado su identificador.
Nombre:	**Adjuntar()**
Descripción:	Para adjuntar los archivos correspondientes al Documento de Plan de Pruebas.

Tabla 29: Anexo 34- Descripción Clase de Diseño: C_DCP

Nombre:	**C_DCP**
Tipo de Clase:	Entidad
Atributos:	**Tipo:**
id_DCP	Integer
id_DPP	Integer
version	Varchar
descripcion	Varchar
documento	Varchar
Para Cada Responsabilidad	
Nombre:	**Crear(todo)**
Descripción:	Adicionar un nuevo Diseño de Casos de Pruebas dados sus atributos.
Nombre:	**Eliminar(id_DCP)**
Descripción:	Para eliminar un Diseño de Casos de Prueba dado su identificador.
Nombre:	**Modificar(todo)**
Descripción:	Para modificar y actualizar los datos del Diseño de Casos de

	Pruebas dados sus atributos.
Nombre:	**ConsultarBD(consulta)**
Descripción:	Para realizar diferentes consultas a la Base de Datos.
Nombre:	**Buscar(id_DCP)**
Descripción:	Para determinar existencia de Diseño de Casos de Pruebas.
Nombre:	**Consultar(id_DCP)**
Descripción:	Para mostrar Diseños de Casos de Pruebas dado su identificador.
Nombre:	**Adjuntar()**
Descripción:	Para adjuntar los archivos correspondientes al Diseño de Casos de Pruebas.

Tabla 30: Anexo 35- Descripción Clase de Diseño: C_DNC

Nombre:	**C_DNC**
Tipo de Clase:	Entidad
Atributos:	**Tipo:**
id_DNC	Integer
id_DCP	Integer
aspectos_generales	Varchar
elementos_probados	Varchar
elementos_no_probados	Varchar
documento	Varchar
Para Cada Responsabilidad	
Nombre:	**Crear(todo)**
Descripción:	Para adicionar un nuevo Documento de No conformidades dados sus atributos.
Nombre:	**Eliminar(id_DNC)**

Descripción:	Para eliminar un Documento de No Conformidades dado su identificador.
Nombre:	**ConsultarBD(consulta)**
Descripción:	Para realizar diferentes consultas a la Base de Datos.
Nombre:	**Modificar(todo)**
Descripción:	Para modificar y actualizar los datos de los Documentos de No Conformidades dado su identificador.
Nombre:	**Buscar(id_DNC)**
Descripción:	Para determinar existencia del Documento de No Conformidades dado su identificador.
Nombre:	**Consultar(id_DNC)**
Descripción:	Para mostrar un Documento de No Conformidades dado su identificador.
Nombre:	**Adjuntar()**
Descripción:	Para adjuntar los archivos correspondientes al Documento de No Conformidades.

Tabla 31: Anexo 36- Descripción de Tabla: T_Lider.

Nombre:	**T_Lider**	
Descripción:	En esta tabla se almacenan todos los Líderes de los diferentes Proyectos existentes en el PHA.	
Atributos:	**Tipo:**	**Descripción:**
id_proyecto	Varchar (255)	Identificador único del Proyecto al que pertenece el Líder de Proyecto.

Tabla 32: Anexo37- Descripción de Tabla: T_Equipo.

Nombre:	**T_Equipo**	
Descripción:	En esta tabla se almacenan los Equipos que se crean para llevar a cabo el proceso de pruebas en un Proyecto.	
Atributos:	**Tipo:**	**Descripción:**
id_equipo	Integer (11)	Identificador único del Equipo de Pruebas.
nombre	Varchar (255)	Nombre del Equipo de Pruebas.
id_proyecto	Integer (11)	Identificador único del Proyecto en el que va a trabajar el Equipo de Pruebas.
jefe	Varchar (255)	Nombre del usuario Jefe del Equipo de Pruebas.
objetivo	Varchar (255)	Objetivos para los que se crea el Equipo.

Tabla 33: Anexo 38- Descripción de Tabla: T_DPP.

Nombre:	**T_DPP**	
Descripción:	En esta tabla se almacenan los datos más significativos de los Documentos de Planes de Pruebas y la dirección física del Documento de Plan de Pruebas.	
Atributos:	**Tipo:**	**Descripción:**
id_DPP	Integer (11)	Identificador único del Documento de Plan de Pruebas.
id_proyecto	Integer (11)	Identificador único del Proyecto al que pertenece el Plan de Pruebas.
version	Float	Versión actual del Plan de Pruebas.
proposito	Varchar (255)	Propósito que se persigue con el Plan de Pruebas.
alcance	Varchar (255)	Alcance que tendrá el Plan de Pruebas.

referencias	Varchar (255)	Referencias utilizadas para realizar el Plan de Pruebas.
documento	Varchar (255)	Nombre y dirección física del Plan de Pruebas.

Tabla 34: Anexo 39- Descripción de Tabla: T_DCP.

Nombre:	**T_DCP**	
Descripción:	En esta tabla se almacenan los datos más significativos de los Diseños de Casos de Pruebas y la dirección física del documento Diseño de Casos de Pruebas.	
Atributos:	**Tipo:**	**Descripción:**
id_DCP	Integer (11)	Identificador único del Diseño de Casos de Pruebas.
id_DPP	Integer (11)	Identificador único del Documento de Plan de Pruebas al que pertenece el Diseño de Casos de Pruebas.
version	Float	Versión actual del Diseño de Casos de Pruebas.
documento	Varchar (255)	Nombre y dirección física del Diseño de Casos de Pruebas Físico.

Tabla 35: Anexo 40- Descripción de Tabla: T_DNC.

Nombre:	**T_DNC**	
Descripción:	En esta tabla se almacenan los datos más significativos de los Documentos de No Conformidades y la dirección física del Documentos de No Conformidades.	
Atributos:	**Tipo:**	**Descripción:**

id_DNC	Integer (11)	Identificador único del Documento de No Conformidades.
id_DCP	Integer (11)	Identificador único del Diseño de Casos de Pruebas al que pertenece el Documento de No Conformidades.
aspect_generales	Varchar (255)	Breve descripción de los aspectos generales que se tratan en el Documento de No Conformidades.
elementos_prob	Varchar (255)	Lista de elementos que fueron probados.
element_no_prob	Varchar (255)	Lista de elementos que no fueron probados.
documento	Varchar (255)	Nombre y dirección física del Documento de No Conformidades.

Tabla 36: Anexo 41- Iteraciones: CP Adicionar Usuario.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Adicionar Nuevo Usuario con todos los datos correctos(nombre, usuario, contraseña, número de solapín y el rol que ocupa)		El Sistema adiciona el nuevo usuario y muestra un mensaje indicando que la acción se realizó satisfactoriamente.	El Sistema adicionó el nuevo usuario y mostró un mensaje indicando que la acción se realizó satisfactoriamente.	
	Adicionar Usuario anteriormente adicionado, con todos los datos correctos(nombre, usuario, contraseña, número de solapín y el rol que ocupa)	El Sistema no permite adicionar el usuario, muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió adicionar el usuario y mostró un mensaje indicando que el usuario ya existía.	

	Adicionar Usuario llenando solamente el campo: Nombre.	El Sistema no permite adicionar el usuario, muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió adicionar el usuario y mostró un mensaje indicando que faltaban datos por llenar.	
	Adicionar Usuario llenando solamente los campos: Nombre, usuario, contraseña y número de solapín.	El Sistema no permite adicionar el usuario, muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió adicionar el usuario y mostró un mensaje indicando que faltaban datos por llenar.	
	Adicionar Usuario llenando los campos con datos incorrectos para los que deben estar validados. (en el campo: nombre una cadena de números, en el campo usuario: una cadena de números, en el campo solapin: una cadena de letras, en el campo rol: una cadena de números)	El Sistema no permite adicionar el usuario, muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió adicionar el usuario y mostró un mensaje indicando los datos introducidos no eran correctos.	

Tabla 37: Anexo 42- Iteraciones: CP Adicionar Proyecto.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Adicionar Nuevo Proyecto con todos los datos correctos (identificador, nombre y descripción).		El Sistema adiciona el nuevo Proyecto y muestra un mensaje indicando que la acción se realizó satisfactoriamente.	El Sistema adicionó el nuevo Proyecto y mostró un mensaje indicando que la acción se realizó satisfactoriamente.	
	Adicionar Proyecto anteriormente adicionado, con todos los datos correctos (identificador, nombre y descripción).	El Sistema no permite adicionar el Proyecto y muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió adicionar el Proyecto y mostró un mensaje indicando que el mismo ya existía.	

	Adicionar Proyecto llenando solamente el campo: Identificador.	El Sistema no permite adicionar el Proyecto, muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió adicionar el Proyecto y mostró un mensaje indicando que faltaban datos por llenar.	
	Adicionar Proyecto llenando solamente los campos: Identificador y Nombre.	El Sistema no permite adicionar el Proyecto, muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió adicionar el Proyecto y mostró un mensaje indicando que faltaban datos por llenar.	
	Adicionar Proyecto llenando los campos con datos incorrectos para los que deben estar validados. (en el campo identificador: números negativos y en el campo: nombre una cadena de números)	El Sistema no permite adicionar el Proyecto, muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió adicionar el Proyecto y mostró un mensaje indicando los datos introducidos no eran correctos.	

Tabla 38: Anexo 43- Iteraciones: CP Registrar Reporte.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Registrar Nuevo Reporte con todos los datos correctos (identificador, cantidad de errores encontrados, cantidad de no conformidades, cantidad de CP realizados, observaciones y el DCP al que pertenece).		El Sistema registra el nuevo Reporte y muestra un mensaje indicando que la acción se realizó satisfactoriamente.	El Sistema registró el nuevo Reporte y mostró un mensaje indicando que la acción se realizó satisfactoriamente.	
	Registrar Reporte con sin llenar los campos necesarios.	El Sistema no permite registrar el Reporte y muestra un mensaje indicando que la	El Sistema no permitió registrar el Reporte y mostró un mensaje indicando que faltaban	

		acción no se pudo realizar y las causas.	campos por llenar.	
	Registrar Reporte llenando solamente los campos: Identificador, cantidad de errores, cantidad de CP y las observaciones.	El Sistema no permite registrar el Reporte, muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió registrar el Reporte y mostró un mensaje indicando que faltaban datos por llenar.	
	Registrar Reporte llenando los campos con datos incorrectos para los que deben estar validados, (identificador: cadena de números, cantidad de errores: cadena de letras, cantidad de no conformidades: cadena de letras, cantidad de CP: cadena de letras)	El Sistema no permite registrar el Reporte, muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió registrar el Reporte y mostró un mensaje indicando los datos introducidos no eran correctos.	

Tabla 39: Anexo 44- Iteraciones: CP Realizar Solicitud.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Realizar Solicitud de Pruebas llenando correctamente todos los datos solicitados (identificador, fase del proyecto, prioridad, tiempo para las pruebas, recursos disponibles y referencias)		El Sistema permite realizar la Solicitud y muestra un mensaje indicando que la acción se realizó satisfactoriamente.	El Sistema registró la nuevo Solicitud y mostró un mensaje indicando que la acción se realizó satisfactoriamente.	
	Realizar Solicitud llenando correctamente solo los campos: identificador, fase del proyecto, prioridad y tiempo para las pruebas.	El Sistema no permite realizar la Solicitud y muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió realizar la Solicitud y mostró un mensaje indicando que faltaban campos por llenar.	

	Realizar Solicitud llenando los campos con datos incorrectos para los que deben estar validados, (identificador: cadena de letras, fase del proyecto: cadena de números, prioridad: una letra, tiempo para las pruebas: números negativos)	El Sistema no permite realizar la Solicitud, muestra un mensaje indicando que la acción no se pudo realizar y las causas.	El Sistema no permitió realizar la Solicitud y mostró un mensaje indicando los datos introducidos no eran correctos.	

GLOSARIO

A:

Artefacto: Producto tangible del proyecto que es producido, modificado y usado por las actividades.

Autenticar: Efectuar un procedimiento que garantice la autenticidad y, por lo tanto, la legalidad de un documento, de un procedimiento o de un hecho, en este caso de una persona a un sistema informático.

Automatización: Realización de una combinación específica de acciones por una máquina, sin la ayuda de personas.

B:

Base de Datos: Es un conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su posterior uso.

Bucle: es una sentencia que se realiza repetidas veces a un trozo aislado de código, hasta que la condición asignada a dicho bucle deje de cumplirse.

C:

Caso de Uso: Secuencias de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de las secuencias.

Clase: Es la declaración o abstracción de un objeto cuando se programa según el paradigma de orientación a objetos.

Código: Es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa.

Código abierto (Open Source): Relativo al software para el cual el código fuente está disponible en forma gratuita.

Código fuente: Es un texto escrito generalmente por una persona que se utiliza como base para generar otro código con un compilador o intérprete para ser ejecutado por una computadora.

D:

Componente: Es la unidad de construcción elemental del diseño físico.

Depuración: Es el proceso de identificar y corregir errores de programación.

E:

Estrategia: Es un conjunto de acciones que se llevan a cabo para lograr un determinado fin.

F:

Framework: Es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado.

I:

Implementar: Poner en funcionamiento, aplicar métodos, medidas, etc., para llevar algo a cabo.

Interfaz: Parte de un programa que permite el flujo de información entre un usuario y la aplicación, o entre la aplicación y otros programas o periféricos. Esa parte de un programa está constituida por un conjunto de comandos y métodos que permiten estas intercomunicaciones.

M:

Metodología: Se refiere a los métodos de investigación que se siguen para alcanzar una gama de objetivos en una ciencia.

Modelo Vista Controlador (MVC): Es una arquitectura de software que separa el modelo de datos de una aplicación, la interfaz de usuario, y la lógica de control en tres distintos componentes de forma que las

modificaciones al componente de la vista pueden ser hechas con un mínimo impacto en el componente del modelo de datos.

Módulo: Es una parte de un programa de ordenador**.**

Multiplataforma: Poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas.

N:

Negocio: Cualquier ambiente o entorno en cual está enmarcado el problema.

P:

Paradigma: Es un modelo o patrón en cualquier disciplina científica u otro contexto epistemológico.

Patrones: Describe un problema que ocurre una y otra vez en nuestro entorno y describe también el núcleo de la solución al problema, de forma que puede re-utilizarse continuamente.

Programa: Es una secuencia de instrucciones que una computadora puede interpretar y ejecutar.

Prototipo: Es un sistema que sin tener todas las funcionalidades que debe disponer en su versión final, es lo suficientemente estable como para dar a conocer de forma simple los requisitos más esenciales.

R:

Requerimientos: Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.

S:

Sistema: Es un conjunto de elementos que interactúan entre sí con el fin de apoyar las actividades de una empresa o negocio. Es una aplicación informática.

Sistema Informático: Es aquel sistema que se encarga del manejo de información en la computadora, a través de la cual el usuario controla las operaciones que realiza el procesador.

Software: Conjunto de instrucciones escritas en un determinado lenguaje, que dirigen a un ordenador para la ejecución de una serie de operaciones, con el objetivo de resolver un problema que se ha definido previamente.

T:

Tecnologías de la información y la comunicación (TIC): Son un conjunto de servicios, redes, software y dispositivos que tienen como fin la mejora de la calidad de vida de las personas dentro de un entorno, y que se integran a un sistema de información interconectado y complementario.