Universidad de las Ciencias Informáticas

Facultad 1

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas.

Título:

"Arquitectura del módulo de pruebas de la suite de Ingeniería de Software"

Autor: Carlos Julio Marrero Gutiérrez

Tutoras:

Ing. Mónica María Albo Castro

Ing. Yusleydi Fernández Del Monte

Ing. Mairim Delgado Muñiz

La Habana

2013

Declaración de autoría.

Declaro ser el autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Ing. Yusleydi Fernández Del Monte

_____

Firma del Tutor

Ing. Mairim Delgado Muñiz

_____

Firma del Tutor

Carlos Julio Marrero Gutiérrez

_____

Firma del Autor

Agradecimientos

Ante todo agradezco a las personas más importantes, mis padres Elisa y Julián por guiarme por caminos de bien en la vida y ayudarme a crecer como la persona que hoy en día soy.

A mi hermano Luis Miguel que siempre está a mi lado y aunque es más pequeño que yo, siempre se preocupa por mí y me brinda su ayuda en todo momento.

A todas las personas de mi familia que confiaron siempre en mí y me dieron su apoyo incondicional.

A todos mis amigos que en diferentes momentos desempeñaron un papel importante en mi vida, sobre todo a Marlen, Zaidee y José Manuel, que desde mi llegada a la universidad siempre han estado a mi lado y son como hermanos.

A mis tutoras que estuvieron en todo momento siempre con su apoyo y ayuda.

A todos aquellos que contribuyeron a mi formación profesional.

A todas aquellas personas, que me han ayudado en todo el transcurso de mi vida, a las que siempre estaré agradecido y que ellas saben quiénes son, sin necesidad de nombrarlas.

A Dios por darme fuerzas para seguir adelante sin importar los obstáculos que se presenten y por permitir que mis seres queridos estén siempre conmigo.

Dedicatoria

A mis padres y hermano por sus esfuerzos incansables y por darme siempre lo mejor de ellos, sobre todo el amor, sentimiento que puede hacer cualquier cosa en la vida.

A todos los que en mi familia con su ayuda y preocupación estuvieron presentes cada momento.

A todos los amigos que me apoyaron, por enseñarme una amistad verdadera y confiar siempre en mí así como yo confío en ellos.

Resumen.

Con el proceso de desarrollo de un software se torna de vital importancia aplicar pruebas sistemáticas en sus diferentes etapas, debido a que la probabilidad de cometer errores es bastante alta. Los procesos de pruebas deben establecerse respetando los estándares y procedimientos concordados, así como exigencias del cliente. El proceso de pruebas de la distribución GNU/Linux Nova carece de facilidades para la ejecución e integración de los procedimientos de diseño, evaluación y comunicación de los resultados, afectando la fluidez y calidad del proceso. El objetivo es diseñar un módulo que permita automatizar la gestión del proceso de pruebas de la distribución GNU/Linux y se integre a la suite de ISW. Para lograr este objetivo se realiza un estudio de los procesos de pruebas a diferentes distribuciones de software libre, así como los indicadores de calidad definidos en los distintos estándares internacionales. Se incluyen las métricas para cada uno de estos indicadores y una nueva actividad en el proceso que es precisamente donde se realiza el cálculo de las mismas para dar una evaluación cuantitativa de los resultados. Con la conformación de una arquitectura adecuada, se logra una estructura con propiedades que brindan opciones al nivel de sistemas similares que realizan pruebas automatizadas. Se puede constatar la efectividad de la propuesta de la arquitectura a partir de los resultados obtenidos luego de aplicarle un método de evaluación.

Palabras claves: arquitectura, distribución GNU/Linux Nova, indicadores de calidad, métricas, proceso de pruebas.

Índice de contenido

Índice de figuras

Introducción

En la sociedad actual, las tecnologías de la información y las comunicaciones juegan un papel fundamental. Como consecuencia la industria del software aumenta vertiginosamente, debido a la creciente demanda de aplicaciones que facilitan la interacción de los usuarios con dichas tecnologías.

Es por esto que las empresas que se dedican al desarrollo de software están obligadas a elaborar mayor cantidad de las mismas, con un menor tiempo de producción. Las empresas se enfrascan entonces en una competencia por brindar al mercado los mejores productos así como estar entre los más novedosos y vendidos. Por tanto la necesidad de que al salir al mercado tengan una buena terminación y calidad, para su correcto funcionamiento en cualquier empresa, institución o lugar que se necesite.

La calidad se logra con la implementación de rigurosos procesos, entre ellos la realización de pruebas que pueden ser de tipo manuales o automáticas. En principio en la industria del software se probaban los programas de forma manual, esto se convertía en un proceso largo y tedioso. Además los productos eran probados por los desarrolladores de los mismos y estos podían obviar una serie de parámetros importantes por creer que su producto era perfecto y no poseería algún error, provocando problemas serios para la finalización con buena calidad del software. [1]

Existen organizaciones a nivel mundial que establecen estándares de calidad para la comercialización de los productos. En Cuba al igual que en otras partes del mundo se crean organizaciones que velan por la adaptación de esos estándares mundiales, así surge el Comité Técnico Nacional, adjunto a la Oficina Nacional de Normalización de Cuba (ONN). En la Feria Informática 2004, Cuba se declara como productora y comercializadora de software con la Industria Cubana del Software (INCUSOFT). La Universidad de las Ciencias Informáticas (UCI) desde sus inicios tuvo entre sus objetivos fomentar el incremento del desarrollo de software en Cuba, y a medida que se fue adentrando en el mercado fue necesario revisar y velar porque los productos tuvieran calidad, por lo que es creado Calisoft, organización que tiene como objetivo garantizar la máxima calidad del software desarrollado en la universidad. [2]

Entre los distintos tipos de software se destaca el Sistema Operativo (SO) este aparte de ser el software que se ejecuta en modo kernel, realiza dos funciones básicas: proporcionar a los programadores de aplicaciones un conjunto abstracto de recursos simples, en vez de los complejos conjuntos de hardware, y administrar estos recursos de hardware [3]. El bloqueo económico que mantiene EEUU contra Cuba impide comprar o actualizar diferentes productos como las versiones del SO Windows, que es un software privativo. Utilizar software sin ninguna licencia, beneficio legal, soporte o ayuda, trae como consecuencia que sea imposible exportar software cubano desarrollado sobre alguna plataforma privativa. En el año 2005 se define una política de migración hacia el software libre encaminada a lograr la independencia tecnológica, comenzando con la introducción de los sistemas operativos libres. La UCI desempeña un

papel fundamental en este proceso, a partir de la creación del proyecto Nova de la actual Facultad 1 en el año 2005, que tiene como objetivo crear una distribución que pueda sustituir los sistemas operativos privativos que se usan actualmente en las empresas cubanas. El mismo desarrolla tres líneas fundamentales: la versión para escritorio, la versión ligera para computadoras de bajas prestaciones y la versión para servidores.

La calidad como concepto en la distribución GNU/Linux Nova inicia después de que la versión Baire fuera liberada en el año 2009, donde los resultados fueron satisfactorios pero no completamente, pues el producto pudo tener mejor calidad, provocado porque fue probada empíricamente por los propios desarrolladores. Luego de esto es creado el grupo Nova Qalit, que tiene la tarea de velar por la calidad de los productos elaborados en Nova, iniciándose así las investigaciones para reunir los elementos necesarios para realizar las pruebas efectivamente. [1]

En el año 2011 el trabajo de tesis titulado “Proceso de pruebas de la distribución GNU/Linux Nova”, marcó los pasos iniciales para el ordenamiento del proceso de pruebas, el cual aplica las mejores prácticas propuestas por el modelo de calidad CMMI. Luego en el año 2012 se definieron los requisitos de calidad que debe cumplir una distribución GNU/Linux, teniendo en cuenta estándares de calidad internacionales como la NC ISO/IEC 9126.

El actual proceso de pruebas de la distribución cubana de software libre, ha pasado por varias actualizaciones en busca de lograr medir la mayor cantidad de características de calidad requeridas por este tipo de software. Por tanto los líderes de desarrollo de Nova comprobaron la utilidad del mismo, sin embargo consideran que no brinda facilidades para una correcta integración entre sus actividades y publicación de los resultados. Por esta situación se plantea como **Problema Científico**:

El proceso de pruebas utilizado actualmente por la distribución GNU/Linux Nova carece de facilidades para la ejecución e integración de los procedimientos de diseño, evaluación y comunicación de los resultados, lo cual que afecta la fluidez y calidad del proceso.

El **Objeto de Estudio** del presente trabajo de diploma son los procesos de pruebas de software, y se define así el siguiente **Campo de Acción**: automatización de la gestión de los procesos de pruebas a distribuciones GNU/Linux.

Objetivo General: Diseñar un módulo que permita automatizar la gestión del proceso de pruebas de la distribución GNU/Linux y se integre a la suite de ISW.

Para lograr este propósito se plantean los siguientes **Objetivos Específicos**:

- Caracterizar las herramientas más utilizadas en la industria de desarrollo de software internacional que automatizan, parcial o totalmente, los procesos de pruebas de software.
- Investigar las herramientas que realizan procesos de pruebas y agruparlas según el procedimiento al cual se ajustan.

- Definir las herramientas y tecnologías a utilizar o reutilizar durante la construcción del diseño de la herramienta de automatización del proceso de pruebas de las distribuciones GNU/Linux.
- Definir los requerimientos de la herramienta de automatización de la gestión proceso de pruebas de las distribuciones GNU/Linux.
- Diseñar el módulo de automatización de la gestión del proceso de pruebas de las distribuciones GNU/Linux, en función de los requerimientos determinados.
- Realizar la evaluación de la arquitectura del módulo de automatización de la gestión del proceso de pruebas de las distribuciones GNU/Linux.

Las **Tareas a cumplimentar** son:

- Determinación de los procedimientos que requiere ejecutar procesos de pruebas de una distribución GNU/Linux.
- Realización de un análisis comparativo de las herramientas estudiadas en función de definir cuales se pueden reutilizar.
- Determinación del lenguaje de programación y las tecnologías más apropiadas para el diseño de la herramienta en función de los elementos a reutilizar.
- Especificación de los requerimientos del módulo de automatización de la gestión del proceso de pruebas de las distribuciones GNU/Linux.
- Diseño de la arquitectura del módulo de automatización del proceso de pruebas de las distribuciones GNU/Linux.
- Evaluación de la arquitectura del módulo de automatización del proceso de pruebas de las distribuciones GNU/Linux.

Con el presente trabajo de diploma se defiende la idea que con el diseño de un módulo que se integre a la suite de ISW que permita automatizar de forma general la gestión del proceso de pruebas en Nova se puede asegurar la calidad de los productos que se obtengan en el proyecto así como lograr una integración de los procesos de prueba con los procesos de mantenimiento y liberación.

Métodos Teóricos que se utilizan para la realización de esta investigación:

- Analítico – Sintético: En el desarrollo de la investigación se analiza la documentación de diferentes herramientas con el tema de automatización, documentos, normas de calidad y otros artículos que estén relacionados con las pruebas de software, para extraer los elementos más significativos relacionados directamente con el tema y confeccionar la fundamentación de la investigación. Además se realizaron resúmenes, comparaciones y valoraciones de las características más relevantes relacionadas con las aplicaciones que automatizan estas pruebas y como están diseñadas.

- Inductivo – Deductivo: Con la realización de un análisis de las herramientas que se utilizan para las pruebas automatizadas se pudieron obtener los requerimientos necesarios para el diseño.

Métodos Empíricos que se utilizan para la realización de esta investigación:

- Experimental: En este método el investigador interviene sobre el objeto de estudio, en este caso el diseño de las herramientas automáticas de prueba de software, modificando a este directa o indirectamente para crear las condiciones necesarias que permitan revelar sus características fundamentales y sus relaciones esenciales. Este método es el más complejo y de gran eficacia.

El presente trabajo de diploma está estructurado en tres capítulos:

Capítulo 1: Fundamentación teórica de las pruebas de Software, en este capítulo serán analizados aquellos aspectos que son de suma importancia para la realización de los procesos de pruebas de software, además se hará un estudio del arte de los mismos, encaminados a la automatización de los procesos. Se podrá encontrar también las diferentes tecnologías y herramientas seleccionadas para dar cumplimiento al desarrollo del sistema, así como la metodología de desarrollo que se seguirá, unido al lenguaje y a la herramienta de modelado.

Capítulo 2: Características del sistema, en este capítulo se plantea cual será la arquitectura seleccionada para el desarrollo del sistema. Se especifica cómo será la representación arquitectónica y se presentan las diferentes vistas arquitectónicas que componen la descripción de la arquitectura planteada por la metodología de desarrollo utilizada. Además, podrá encontrar los estilos y patrones de diseño propuestos para elevar la calidad del software.

Capítulo 3: Evaluación de la arquitectura, en este capítulo se hará una evaluación de la arquitectura propuesta haciendo uso de los atributos de calidad y el método de evaluación seleccionado, describiéndose posteriormente los resultados obtenidos.

Capítulo 1: Fundamentación teórica de las pruebas de Software

Las pruebas de software forman parte del proceso de control de la calidad con investigaciones empíricas y técnicas en las cuales el punto fundamental es brindar información objetiva e independiente relacionada con la calidad del producto. Son principalmente un grupo de actividades dentro del desarrollo de software que pueden ser implementadas en el proceso en el momento que sea necesario. El objetivo principal de las pruebas es presentar a las personas responsables la información sobre la calidad que tiene el producto. Las pruebas son parte esencial del proceso de desarrollo del software con la función de detectar los errores lo antes posible.

La investigación está centrada en las pruebas de software, sus clasificaciones en cuanto a niveles, métodos y lo que evalúa según sus tipos. Los procesos de pruebas y la relación que tienen con especificaciones de calidad contempladas en estándares internacionales. Además existen muchas herramientas que realizan los procesos de pruebas de forma automática y entre estas una gran variedad tienen código abierto, se hace una relación de las fundamentales para aprovechar sus funcionalidades y potencialidades. Se realiza una descripción del proceso de pruebas a Nova, además de hacer un análisis del entorno de desarrollo a utilizar.

1.1 Pruebas de Software.

Según Pressman las pruebas de software son un conjunto de actividades que se planean con anticipación y se realizan de manera sistémica. Por tanto, se debe definir una plantilla para las pruebas de software, un conjunto de pasos en que se puedan incluir técnicas y métodos específicos del diseño de casos de prueba. [4]

Las pruebas se pueden clasificar desde distintos puntos de vista:

En cuanto a los Niveles de Pruebas estos pueden especificar el alcance que tendrá la prueba de software y se pueden clasificar en [5]:

- Pruebas unitarias, que detectan los errores en los datos, lógica y algoritmos.
- Pruebas de componentes o integración, detectan errores de interfaces y relaciones entre componentes.
- Pruebas de funcionalidad que detectan errores en la implementación de requerimientos.
- Pruebas de sistema que detectan fallas en el cubrimiento de los requerimientos.
- Pruebas de aceptación, detectan fallas en la implementación del sistema.
- Pruebas de instalación que detectan fallas en la instalación del sistema.

En cuanto a los Métodos de Pruebas que son la manera de realizar diferentes pruebas a un determinado sistema informático y así comprobar que se están cumpliendo los requerimientos y las funcionalidades, estos se pueden clasificar en [6]:

- Pruebas de caja blanca, con acceso al código fuente (datos y lógica). Se trabaja con entradas, salidas y el conocimiento interno.
- Pruebas de caja negra, son funcionales sin acceso al código fuente de las aplicaciones, se llevan a cabo sobre la interfaz del software y se trabaja con entradas y salidas.
- *Top Down* que requieren *Stubs*[1] para suplantar los módulos inferiores aún no implementados, estos se quitan a medida que se van desarrollando los diferentes módulos.
- *Botton Up* que escriben pruebas que brindan el contexto de ejecución a los módulos, se prueban los módulos y se desarrolla e integran funcionalidades del módulo superior.
- *Test* incrementales con un testeo continuo distribuyendo las pruebas de integración en la interacción diaria del código compartido.

En cuanto a lo que evalúa están los tipos de pruebas del sistema, estas tienen el objetivo de verificar el ingreso, procesamiento, así como la recuperación apropiada de los datos e implementación de las reglas de negocios. El enfoque de las pruebas del sistema debe de estar en los requisitos tomados de los casos de uso, funciones de negocios y reglas. Se determina con las pruebas de sistema un aseguramiento para que la aplicación alcance sus objetivos de negocio. Están basadas en técnicas de caja negra. La prueba de sistema es compleja porque intenta validar un número de características al mismo tiempo. Con la prueba de sistema se incluye:

- Prueba de Funcionalidad enfocada en los requisitos funcionales, reglas del negocio y basadas directamente en los casos de uso.
- Prueba de Usabilidad que detecta problemas relacionados con la conveniencia y la práctica con el sistema desde el punto de vista del usuario.
- Prueba de Documentación y Procedimientos consiste en evaluar la exactitud y claridad de la documentación del usuario.
- Prueba de Seguridad y Control garantiza la seguridad deseada.
- Prueba de Volumen somete al sistema a grandes volúmenes de datos para determinar los límites que causa que el sistema falle y hasta que volumen puede manejar.
- Prueba de Esfuerzo (*Stress*) propone encontrar errores debidos a recursos bajos o en la

1 Trozo de código usado como sustituto de alguna otra funcionalidad. Un stub puede simular el comportamiento de código existente o ser el sustituto temporal para un código aún no desarrollado.

completitud de los mismos.

- Prueba de Recuperación aseguran que una aplicación o sistema se recupere de una variedad de anomalías de hardware, software o red.
- Prueba de Múltiples sitios tiene como propósito evaluar el correcto funcionamiento del sistema o subsistema en múltiples instalaciones. [7]

En los sistemas web está recomendado realizar las siguientes pruebas de sistema: Usabilidad, Prueba de Esfuerzo, Funcionalidad y Humo. Este último no especificado en los puntos anteriores toma éste nombre porque su objetivo es probar el sistema constantemente buscando que saque “humo” o falle. Permite detectar problemas que por lo regular no son detectados en las pruebas normales, en algunos proyectos este tipo de prueba va junto con las pruebas funcionales y van de extremo a extremo de la aplicación sin ser exhaustivas.

Los objetivos de las pruebas del sistema son los siguientes:

- Reducir los riesgos y la baja calidad.
- Detectar los errores en *realeases* tempranos y de manera fácil.
- Garantizar poco esfuerzo en la integración final del sistema.
- Probar el sistema constantemente.
- Asegurar los resultados de las pruebas unitarias.

1.2 Procesos de pruebas de Software.

El proceso de pruebas de software es el proceso de ejecutar un programa con la intención de encontrar fallos. Es una tarea técnica que precisa del dominio del lenguaje de programación en el que el artefacto a probar fue generado, también del conocimiento necesario para comprender la arquitectura del sistema implementado y de las implicaciones de tipo lógico que su diseño pueda suponer. En las grandes empresas es una práctica obligatoria, sin embargo en la pequeña y mediana empresa tiene dificultad la implementación de métodos de pruebas, incluso conociendo que el coste de corregir un error en un sistema de software va en aumento a medida que avanza el desarrollo del mismo. [8]

Para la realización de un proceso de pruebas es necesario apoyarse en estándares de calidad definidos que permiten establecer una guía común para obtener productos con la calidad requerida. Como producto el software debe cumplir con una serie de requisitos para poder ser comercializado a nivel internacional y cumplir con una serie de indicadores de calidad. Estos han sido definidos con estándares construidos por organizaciones de reconocimiento a nivel internacional, dedicadas al estudio y elaboración de las normas. A continuación se muestran algunos estándares fundamentales que pueden resultar de importancia para el presente trabajo.

1.2.1 Capacity Maturity Model Integration (CMMI).

Según el *Software Engineering Institute* (SEI)**,** CMMI nos brinda una mejora de procesos que proporciona a las organizaciones los elementos esenciales de procesos efectivos que ayudan a elevar su desempeño, incluyendo la identificación de las fortalezas y debilidades de la organización en cuanto a los procesos y a la realización de cambios para su transformación. Existen tres variantes de CMMI: CMMI-DEV (desarrollo), CMMI-SVC (servicios) y CMMI-ACQ (adquisiciones) para interés de la investigación es el primero el que provee guías para medir, monitorizar y administrar los procesos de prueba. [9]

Las pruebas de software son un tema un poco más amplio que también se le denomina Verificación y Validación, estas abarcan muchas de las actividades de aseguramiento de la calidad del software. La Verificación es el conjunto de actividades que aseguran que el software implemente correctamente una función específica. La Validación es un conjunto diferente de actividades que aseguran que el software construido corresponde con los requisitos del cliente.

Verificación (VER): Tiene como objetivo demostrar que el producto cumple los requisitos que se especificaron. La verificación se realiza sobre productos de trabajo y sobre el producto final, los de trabajo son resultado de diversas actividades y no solamente de la codificación. Se verifican los documentos generados con respecto a los requisitos acordados en un inicio y durante todo el desarrollo del proyecto. Probar el código no está enmarcado solamente en verificar, las pruebas del software son una parte del concepto de verificación. Los productos del proyecto, y el código, deben analizarse para entonces decidir si se incorporan entre los que van a ser verificados, entre ellos están los documentos técnicos, los planes, los paquetes o productos comerciales, herramientas a utilizar y elementos de formación.

Validación (VAL): Tiene como objetivo demostrar que un producto o componente de producto se ajusta a su uso previsto cuando se sitúa en un entorno determinado. En la validación intervienen los usuarios finales u otros involucrados en el proyecto, que tienen autoridad para revisar los productos obtenidos y rechazarlos si no satisfacen las necesidades del usuario. Tradicionalmente se hacen cuando el producto está terminado y sólo sobre el software. Deberían seleccionarse trabajos intermedios, para ser validados, que puedan dar una visión de si el producto final cumplirá las expectativas del usuario y facilitar de esta forma ir validando con anterioridad, paulatina e incrementalmente durante todo el transcurso del proyecto. [10]

1.2.2 NC ISO/IEC 9126 [11]

Este estándar establece una serie de características para determinar el grado de calidad de un producto de software. Entre estas se definen:

Funcionabilidad: De acuerdo a la norma esta característica describe que hace el software para satisfacer las necesidades, sin importarle el cuándo ni el cómo.

- Idoneidad: Es la capacidad del software para mantener un conjunto apropiado de funciones para las tareas y objetivos especificados por los usuarios.
- Precisión: Capacidad del software para proporcionar efectos o resultados correctos o convenidos con el grado de exactitud necesario.
- Seguridad: Capacidad del producto de software para proteger la información y los datos, para que personas o sistemas desautorizados no puedan leer o modificar los mismos, y las personas o sistemas autorizados tenga el acceso a ellos.
- Interoperabilidad: Capacidad del producto de software para interactuar recíprocamente con uno o más sistemas especificados.
- Conformidad con la funcionalidad: Capacidad del software de apegarse a la normas, leyes, regulaciones y similares que se le apliquen en términos relativos a la funcionalidad.

Confiabilidad: La capacidad del producto de software para mantener un nivel de ejecución especificado cuando se usa bajo las condiciones especificadas.

- Madurez: Capacidad del producto de software de evitar un fallo total como resultado de haberse producido un fallo del software.
- Tolerancia ante fallos: Capacidad del software de mantener un nivel de ejecución o desempeño especificado en caso de fallos del software.
- Recuperabilidad: Capacidad del producto de software de restablecer un nivel de ejecuciones especificado y recuperar los datos directamente afectados en caso de fallo total.
- Conformidad con la confiabilidad: Capacidad del producto de software para adherirse a las normas que se le apliquen, convenciones, regulaciones, leyes y las prescripciones similares relativas a la confiabilidad.

Usabilidad: La capacidad del producto de software de ser comprendido, aprendido, utilizado y atractivo para el usuario, cuando se utilice bajo las condiciones especificadas.

- Comprensibilidad: capacidad del producto de software para permitirle al usuario entender si el software es idóneo, y cómo puede usarse para las tareas y condiciones de uso particulares.
- Cognoscibilidad: capacidad del producto de software para permitirle al usuario aprender su aplicación.
- Operabilidad: capacidad del producto de software para permitirle al usuario operarlo y controlarlo.
- Atracción: capacidad del producto de software de ser atractivo o amigable para el usuario.

- Conformidad con la usabilidad: capacidad del producto de software para adherirse a las normas, convenciones, guías de estilo o regulaciones relativas a la usabilidad.

Eficiencia: La capacidad del producto de software para proporcionar una ejecución o desempeño apropiado, aprovechando eficientemente los recursos y brindando una experiencia agradable al usuario bajo condiciones determinadas.

- Rendimiento: capacidad del producto de software para proporcionar apropiados tiempos de respuesta y procesamiento, así como tasas de producción de resultados al realizar su función bajo condiciones establecidas.
- Utilización de recursos: capacidad del producto de software para utilizar la cantidad y el tipo apropiado de recursos cuando realiza su función bajo las condiciones establecidas.
- Conformidad de la eficiencia: capacidad del producto de software de adherirse a las normas o convenciones que se relacionan con la eficiencia.

Mantenibilidad: La capacidad del producto de software de ser modificado. Las modificaciones pueden incluir las correcciones, mejoras o adaptaciones del software a cambios en el ambiente, así como en los requisitos y las especificaciones funcionales.

- Diagnosticabilidad: capacidad del producto de software de ser objeto de un diagnóstico para detectar deficiencias o causas de los fallos totales en el software, o para identificar las partes que van a ser modificadas.
- Flexibilidad: capacidad del producto de software para permitir la aplicación de una modificación especificada.
- Estabilidad: capacidad del producto de software para minimizar los efectos inesperados de las modificaciones realizadas.
- Contrastabilidad: capacidad del producto del software para permitir la validación de modificaciones realizadas a su código.
- Conformidad de la mantenibilidad: capacidad del producto de software para adherirse a las normas o convenciones que se relacionan con la mantenibilidad.

Portabilidad: capacidad de producto de software de ser transferido de un ambiente a otro.

- Adaptabilidad: capacidad del producto de software de ser adaptado a los ambientes especificados sin aplicar acciones o medios de otra manera que aquellos suministrados con el propósito de que el software cumpla sus fines.
- Instalabilidad: capacidad del producto de software de ser instalado en un ambiente especificado.
- Coexistencia: capacidad del producto de software de coexistir con otro software independiente en

un ambiente común y compartir los recursos comunes.

- Remplazabilidad: capacidad del producto de software de ser usado en lugar de otro producto de software especificado para los mismos fines y en el mismo ambiente.
- Conformidad con la portabilidad: capacidad del producto de software de adherirse a las normas o convenciones relativas a la portabilidad.

Dentro de esta familia de normas existen tres regulaciones:

ISO/IEC 9126-2 [12]

Esta norma define las métricas externas para la determinación de la calidad de un producto de software. Las métricas externas son aquellas que miden el comportamiento del sistema operativo donde el software a evaluar se ejecuta.

ISO/IEC 9126-3 [13]

Esta norma hace referencia a las métricas internas de la calidad del producto de software, las cuales se concentran en el producto de software en sí mismo. Está diseñada para proporcionarles herramientas a los usuarios de la métrica para determinar, a partir de un software no ejecutable o que esté en fase de desarrollo aun, la calidad del producto final.

ISO/IEC 9126-4 [14]

Este estándar proporciona métricas de calidad en el uso para la medición de los atributos definidos en la ISO/IEC 9126-1. Desarrolladores, evaluadores y administradores de la calidad podrían seleccionar métricas de esta norma para evaluar productos, definir requerimientos, medir aspectos de la calidad y otros propósitos, sin la intención de hacerlas cumplir estrictamente.

1.2.3 ISO/IEC 14598 [15]

Brinda un marco de trabajo para evaluar la calidad de todos los tipos de software, indicando los requisitos que serán medidos y analizados en este proceso. Implementar estándares que garanticen una correcta evaluación al software y mitigar los errores que se puedan presentar cuando se esté ejecutando. Este grupo de normas que son seis están diseñadas para complementar al grupo de las ISO/IEC 9126, de una forma abarcadora completamente para el proceso de la calidad de software.

- La primera parte es una introducción a las demás normas, explica la relación existente entre esta y el modelo de calidad propuesto por la ISO/IEC 9126 conjuntamente.
- La segunda parte se encarga de planificar y gestionar todo el proceso de evaluación. [16]
- La tercera parte la selección de los atributos que representan los requisitos de calidad. Esta se

encuentra fundamentalmente enfocada en definir y reportar los requerimientos que podrían ser útiles para predecir la calidad del producto final, aun cuando su desarrollo esté en fases intermedias. [17]

- La cuarta parte la norma describe el proceso de calidad y se muestran ejemplos de métodos de evaluación. [18]
- La quinta parte se encuentran descritas las herramientas para lograr una adaptación del proceso general de evaluación a un entorno específico. [19]
- La sexta parte de esta norma se hace una descripción de la documentación que puede llevarse en todo el proceso de calidad de software. [20]

1.2.4 ISO/IEC 25000 [21]

Estas normas surgen de la necesidad de actualizar a las ya decadentes 9126 y 14598, pues pertenecen a la primera generación de estándares de calidad de un producto de software. La norma 25000 al pertenecer a la segunda generación de estándares de calidad de productos de software tiene una serie de nuevas características:

- Este estándar está centrado en el lado del producto.
- El ciclo de vida de la calidad del producto de software se basa en tres fases principales, producto en uso, producto en operación y producto bajo desarrollo.
- Como es una revisión de la 9126 hereda de ella las mismas características de calidad de software.
- Se incorporan modificaciones en cuanto a la estructura respecto a la clasificación de las características y subcaracterísticas que ofrece la 9126.

Este grupo de normas se basan en la revisión y unificación de las normas ISO/IEC 9126 y la ISO/IEC 14598. Esta subdividida en:

- ISO/IEC 2500n: División de dirección de calidad.
- ISO/IEC 2501n: División del modelo de calidad.
- ISO/IEC 2502n: División de medida de calidad.
- ISO/IEC 2503n: División de requisitos de calidad.
- ISO/IEC 2504n: División de evaluación de calidad.

1.3 Herramientas de automatización del proceso de pruebas.

Una parte muy importante para los procesos de pruebas de software en la actualidad es su automatización para mejorar el trabajo de los desarrolladores y probadores así como una mejor terminación del producto, es por eso que la automatización de las pruebas debe ser considerada un proyecto con objetivos definidos

[22]. Existen numerosas herramientas que automatizan las pruebas de software para una mejora considerable a dichos procesos y entre ellas las de código abierto.

1.3.1 Test Works.

Para pruebas automatizadas en la mayoría de las plataformas basadas en UNIX[2] existe TestWorks, diseñado para trabajar de una forma independiente o también como conjunto de herramientas integradas que proporcionan un ambiente más eficiente. TestWorks para UNIX se compone de tres líneas de productos: [23]

TestWorks/Regresión incluye las herramientas CAPBAK, sistema de herramienta de captura/reproducción que permite al usuario registrar los movimientos del ratón, las actividades del teclado, llamadas *widgets* e información de verificación. CAPBAK/X sincronización automática de cambios de menor importancia en la aplicación y operaciones sensibles al tiempo. SMARTS se encarga del mantenimiento del software y del sistema de pruebas de regresión, organizando las pruebas en una estructura jerárquica para la ejecución con la ayuda de EXDIFF, que es un sistema de diferenciación extendida que compara la imagen del mapa de bits o archivos ASCII.

Todas estas herramientas operan en forma independiente o dentro de la suite de herramientas de regresión. TestWorks/Regresión proporciona una solución más completa para la automatización y, gestión de la ejecución y comprobación de pruebas en el texto y aplicaciones basadas en interfaz gráfica de usuario. Es ideal en aplicaciones host y cliente-servidor que utilizan la generación de carga automatizada desde una única estación de trabajo de varios usuarios. [24]

TestWorks/Cobertura garantiza la exhaustividad de los casos de prueba incluye a tres analizadores TCAT, S-TCAT y TCAT-PATH, además de una herramienta de observación de los datos de prueba T-SCOPE, que muestra de forma dinámica las ramas lógicas en el proceso de ejecución de una prueba. TestWorks/Cobertura es una suite integrada de los analizadores de cobertura, que trabaja en forma individual o como parte de la suite de herramientas de prueba. Con esta herramienta se verifica si la prueba es incompleta y con las mediciones proporcionadas si los casos de prueba, los niveles de prueba de unidad y la integración del sistema están correctos. Este producto permite que las pruebas disponibles se puedan centrar en el código no probado, en vez de generar pruebas redundantes o deficientes, ayudando a determinar cuando la prueba se ha realizado lo suficiente para la entrega del producto. [25]

TestWorks/Asesor utiliza las herramientas Métrica y TDGEN que establecen y miden con indicadores los parámetros de calidad y hacen un análisis del código fuente de anomalías, generando variedad en los

2 Es un sistema operativo portable, multitarea y multiusuario.

datos de prueba. TestWorks/Asesor realiza un análisis más profundo del código fuente, para detectar algún código complejo o potencialmente propenso a errores. Mejora la productividad, la gestión de recursos, la calidad y la previsibilidad. [25]

1.3.2 Herramientas de Open Source Testing.

Existen numerosas herramientas de código abierto, que asisten al equipo de *testing*. La utilización de estas herramientas facilita la ejecución de las pruebas pero no garantiza el éxito en su uso. En muchos casos, el tiempo de aprendizaje de las mismas o la correcta selección implica un esfuerzo adicional que no en todos los proyectos puede ser tenido en cuenta. Muchas herramientas están focalizadas en un tipo de prueba específica y otras son más generales, entre estas podemos encontrar:

Bugzilla Testopia: es una extensión de la gestión de Bugzilla. Se ha diseñado para ser una herramienta genérica para los casos de prueba de seguimiento, lo que permite a las organizaciones integrar las pruebas de informes de errores con sus resultados de la ejecución de casos de prueba. Está diseñada ante todo bajo la base de las pruebas de software**,** para así lograr un buen manejo de estas. [26]

Escenario de la prueba incremental: apoya los equipos de pruebas en el manejo de su complejidad y de forma adaptativa selecciona y prioriza las pruebas de acuerdo con los resultados obtenidos de las pruebas anteriores. Además brinda una guía a través de una sesión con escenarios de pruebas, que fueron generados sobre la marcha. [27]

MTS: Multi-Tester: define un simple dominio de lenguaje específico para el mantenimiento de las pruebas, que admite dos propiedades deseables: tiene el mando, la entrada y los resultados esperados todos en un solo archivo así como múltiples pruebas con pocos cambios se pueden generar de una misma fuente con resultados distintos. [28]

Requisitos y repositorio de gestión de pruebas (**RTMR):** se hace una gestión de los requisitos de software, se describen los escenarios y casos de prueba asociados y son ejecutados a través de campañas especificadas. Permite realizar un seguimiento de anomalías encontradas durante las pruebas realizadas, esto a través de un gestor de fallo interno y a través de un manejador de fallo externo. La gestión de versiones se hace por proyecto, por el requisito, según la hipótesis y casos de prueba, realizando así un seguimiento de los cambios en el software. [29]

RTH: es una herramienta web que contiene un enfoque más estructurado para las diferentes pruebas de software y aumenta la visibilidad del proceso con la creación de un repositorio común para todos los activos de prueba, que incluye los requisitos, casos de prueba, planes de prueba y resultados de las pruebas. [30]

Salomé-TMF: define si las pruebas serán manuales o automáticas, organiza los casos de prueba en la estructura jerárquica de árbol y parametriza el manual y las pruebas automáticas para llevarlas a cabo en varios ambientes diferentes. Es una herramienta de pruebas sencilla y potente que ofrece beneficios inmediatos en el tiempo gracias a la calidad y la creación de un marco único de referencia de la prueba para el proyecto y con la automatización de la realización de las pruebas. [26]

TestAutomation: gestiona los proyectos, requisitos, defectos, casos de prueba, el historial de ejecución de casos de prueba, y facilita las pruebas automatizadas. Crea el número de puestos de trabajo que se utilizaran para la prueba de cualquier tipo de funcionalidad de la interfaz de usuario, gestiona los casos de prueba y lleva un historial de ejecución así como brinda un informe sobre el estado que presenta el proyecto probado. [31]

Testitool: cada caso de prueba se puede asignar a un requerimiento funcional. Hacer una gestión de instancias individualmente de cada plan de pruebas, permitiendo seleccionar los casos de prueba que tendrá que ejecutar para cada caso concreto del plan de pruebas. Una generación de informes permitirá que cualquiera pueda ver varios informes del plan de pruebas, incluyendo en esto un informe de error, un informe sobre la marcha y el informe de tasa de fracaso así como también con Testitool se puede importar y exportar casos de prueba desde Excel. [32]

1.3.3 Phoronix Test Suite.

Contiene una plataforma de evaluación comparativa que proporciona un marco extensible para que las nuevas pruebas puedan ser fácilmente añadidas. El software es de fácil utilización y está diseñado para cumplir correctamente con dos puntos tanto cualitativos como cuantitativos. El Phoronix Test Suite se compone de un núcleo de procesamiento ligero con cada punto de referencia que consiste en un perfil basado en XML y secuencias de comandos relacionados con los recursos. Se basa en la prueba extensa, está compuesto por herramientas internas desarrolladas por Phoronix.com, mantiene una compatibilidad con el modo por lotes automatizada y soporta la descarga e instalación de pruebas automatizadas. [33]

1.3.4 Mago.

Es una herramienta desarrollada por el equipo de Gnome, contiene un corredor de pruebas que permite ejecutarlas constantemente, siendo capaz de presentar informes con los resultados de las mismas. Es una aplicación basada principalmente en *Linux Desktop Testing Project* (LDTP), un *framework* para pruebas de escritorio, programado en C. Para trabajar con Mago se requiere tener conocimientos del lenguaje Python para diseñar las pruebas pues se deben crear de forma manual en este lenguaje. Las pruebas están organizadas por varias carpetas de archivos XML, donde cada cual contiene uno o más casos de pruebas.

[34]
Se escogieron estas herramientas luego de la investigación, pues las mismas presentan las características y estructuras que pueden servir como base fundamental para incluir o reutilizar. Las herramientas que realizan pruebas automáticas, brindan un grupo de requisitos considerables, además de funcionalidades que pueden formar parte del módulo a desarrollar. Pudiendo lograr así que presente una mejor estructura, esté lo más completo posible en cuanto a funcionalidades, y que brinde facilidades a los probadores durante la realización de los procesos de pruebas.

1.4 Proceso de pruebas de Nova.

La distribución GNU/Linux Nova mantiene un equipo encargado del aseguramiento de la calidad, Nova Qalit, que ha definido un proceso de pruebas para sus productos, basado en un estudio sobre cómo se llevan a cabo las pruebas en otras distribuciones para aprovechar las buenas prácticas entre las que se encuentra comenzar las pruebas antes de liberar una imagen.

Primeramente se realiza la planificación del proceso de pruebas cuya actividad fundamental está centrada en una gestión del plan de pruebas, desde su creación al inicio del desarrollo de software, hasta que comienzan a desarrollarse las actividades específicas de las pruebas. Luego de esto los analistas del producto diseñan los casos de prueba a partir de los requisitos definidos, velando que estén correctamente elaborados. Durante la ejecución de las pruebas se incluyen pruebas automáticas, las que por el momento se realizan con la herramienta Phoronix Test Suite, que permite comparar los resultados obtenidos con otros sistemas operativos.

Las pruebas diseñadas se ejecutan al finalizar la construcción de una primera versión de todos los componentes, luego se pasa a una versión Beta o de pruebas después de varias iteraciones de pruebas y corrección. A esta versión se le realizan otro tipo de pruebas como son las pruebas de actualización, pruebas de rendimiento y el uso diario del sistema.

Por último antes de cerrar el ciclo de pruebas a la versión Beta se realizan lo que se conoce como pruebas de aceptación, las cuales tienen como objetivo comparar el producto con las necesidades actuales del cliente. En todos estos ciclos de pruebas al encontrar elementos que sean incorrectos o errores, se reportan como no conformidades para su corrección posterior. Luego de un período de tiempo y de corregidos los errores del Beta se libera la versión oficial la cual es distribuida a los usuarios.

Todo este proceso de pruebas es guiado por estándares de calidad que definen características de calidad y una guía de métricas establecida para evaluarlas. En esta última versión del proceso de pruebas se incluyen requisitos y métricas de calidad del software, los primeros determinan que características debe tener el producto para ser de calidad, pero sin contar con los indicadores asociados sería imposible

determinar cuantitativamente el cumplimiento del requisito de calidad.

Los indicadores constituyen medidas indirectas de las características del software y son la herramienta principal para lograr la medición y la regla que define cuando el producto está cumpliendo con la calidad respecto a una determinada característica. Las métricas propuestas para este proceso son básicas y se pueden ir incrementando a medida que se refinen los requisitos de calidad y la experiencia del equipo de pruebas aumente. Es aquí donde surge la actividad final, enmarcada en la evaluación de los resultados, que consiste en el cálculo de las métricas y la valoración de la calidad del producto al finalizar cada ciclo. [35]

1.5 Infraestructura tecnológica.

Las herramientas, tecnologías y metodologías de desarrollo de software son componentes fundamentales para la realización de cualquier producto, por lo que es de gran importancia el uso correcto de dichos componentes con el objetivo de obtener resultados con calidad. Se realiza un estudio sobre los diferentes conceptos, las principales herramientas de modelado para hacer la aplicación, los posibles lenguajes de modelado a utilizar y algunas de las metodologías de desarrollo para aplicaciones informáticas.

1.5.1 Metodología de desarrollo de software.

Es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información, este es un *framework* que consiste en: una filosofía de desarrollo de programas de computación con el enfoque del proceso de desarrollo de software y herramientas, modelos y métodos para asistir al proceso de desarrollo de software.

OpenUP

Contiene una filosofía pragmática y ágil de desarrollo, que está enfocada en la naturaleza colaborativa del desarrollo de software. Es un proceso con herramientas neutrales y de baja ceremonia que puede extenderse para alcanzar una amplia variedad de tipos de proyectos. Mantiene las características fundamentales de RUP como son:

- Desarrollo incremental.
- Utilización de casos de uso y escenarios.
- Manejo de riesgos.
- Diseño basado en la arquitectura.

El esfuerzo personal en un proyecto OpenUP se realiza en un ciclo de vida iterativo que estructura cómo entregar porciones de desarrollo estables, organizada en microincrementos. Estas representan unidades

cortas de trabajo que producen un ritmo estable y medible de progreso del proyecto basado en iteraciones generalmente expresado en semanas.
Estructura el ciclo de vida del proyecto en cuatro fases las cuales puede tener tantas iteraciones como sean necesarias: Inicio, Elaboración, Construcción y Transición.
OpenUP está dirigida por cuatro principios fundamentales:

- Balancear las prioridades involucradas para maximizar el valor para el *stakeholder*[3].
- Colaborar para alinear los intereses y compartir entendimiento.
- Enfocarse en la arquitectura tempranamente para minimizar riesgos y organizar el desarrollo.
- Evolucionar para obtener constantemente retroalimentación y mejorar. [36]

Se seleccionó OpenUp porque es un proceso unificado, ágil y liviano, con un enfoque iterativo e incremental dentro de un ciclo de vida estructurado y contiene un conjunto mínimo de prácticas que ayuda al equipo a ser más efectivo desarrollando software. Su proceso puede ser personalizado y extendido para distintas necesidades, que aparecen a lo largo del ciclo de vida del desarrollo de software, al ser incremental e iterativo es capaz de producir versiones y una de sus mayores ventajas es que puede ser utilizado en proyectos pequeños. Entonces los equipos tienen una interacción cara a cara diariamente, tomando sus propias decisiones sobre en qué necesitan trabajar, cuáles son las prioridades y cómo alcanzar las necesidades del *stakeholder* de la mejor manera, enfocándose fundamentalmente en reducir el riesgo de manera temprana en el ciclo de desarrollo del software. [37]

1.5.2 Lenguaje de modelado.

Es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar un diseño de software orientado a objetos. Se usan en combinación con una metodología de desarrollo de software para avanzar de una especificación inicial a un plan de implementación y para comunicar dicho plan a todo un equipo de desarrolladores. El uso de un lenguaje de modelado es más sencillo que la programación, pues existen menos medios para verificar efectivamente el funcionamiento adecuado del modelo.

UML

El Lenguaje Unificado de Modelado contiene un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan. Mientras que ha habido muchas notaciones y métodos usados para el diseño

3 Persona o entidad que está interesada en la realización de un proyecto o tarea, auspiciando el mismo ya sea mediante su poder de decisión o de financiamiento, o a través de su propio esfuerzo.

orientado a objetos, ahora los modeladores sólo tienen que aprender una única notación. UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware, y organizaciones del mundo real.

UML ofrece nueve diagramas en los cuales modelar sistemas:

- Diagramas de Casos de Uso para modelar los procesos *business*.
- Diagramas de Secuencia para modelar el paso de mensajes entre objetos.
- Diagramas de Colaboración para modelar interacciones entre objetos.
- Diagramas de Estado para modelar el comportamiento de los objetos en el sistema.
- Diagramas de Actividad para modelar el comportamiento de los Casos de Uso, objetos u operaciones.
- Diagramas de Clases para modelar la estructura estática de las clases en el sistema.
- Diagramas de Objetos para modelar la estructura estática de los objetos en el sistema.
- Diagramas de Componentes para modelar componentes.
- Diagramas de Implementación para modelar la distribución del sistema.

UML es una consolidación de muchas de las notaciones y conceptos orientados a objetos más usadas.

Se seleccionó UML porque mantiene una notación estándar y, semánticas esenciales para el modelado de un sistema orientado a objetos. Previamente, un diseño orientado a objetos podría haber sido modelado con cualquiera de la docena de metodologías populares, causando a los revisores tener que aprender las semáticas y notaciones de la metodología empleada antes que intentar entender el diseño en sí. Ahora con UML diseñadores de diferentes modelos, de sistemas diferentes, pueden entender cada uno los diseños de los otros. Los mecanismos de extensibilidad incorporados permiten a UML ser una especie de especificación abierta, estos son los llamados estereotipos que respresenta una distinción de uso. Puede ser aplicado a cualquier elemento de modelado, incluyendo clases, paquetes, relaciones de herencia, entre otros**.** [38]

1.5.3 Herramientas de modelado.

Son herramientas creadas para el desarrollo de la ingeniería de software, con el fin de desarrollar programas basándose en un conjunto de ayudas, utilizando técnicas de diseño y metodologías bien definidas, soportadas por las mismas. Además de realizar la planificación, pasando por el análisis y el diseño, para la generación del código fuente de los programas.

Visual Paradigm

Es una herramienta para desarrollo de aplicaciones utilizando modelado UML, ideal para Ingenieros de

Software, Analistas de Sistemas y Arquitectos de Software que están interesados en construcción de sistemas a gran escala y necesitan confiabilidad y estabilidad en el desarrollo orientado a objetos.
Visual Paradigm también ofrece:

- Navegación intuitiva entre la escritura del código y su visualización.
- Potente generador de informes en formato PDF/HTML.
- Documentación automática Ad-hoc.
- Ambiente visualmente superior de modelado.
- Sofisticado diagramador automáticamente de *layout*[4].
- Sincronización de código fuente en tiempo real o en demanda.

Se seleccionó Visual Paradigm porque proporciona un conjunto de ayudas para el desarrollo de programas informáticos. Ha sido concebida para soportar el ciclo de vida completo del desarrollo del software a través de la representación de todo tipo de diagramas. Fue diseñada para usuarios interesados en la construcción de sistemas de software de forma fiable a través de la utilización de un enfoque Orientado a Objetos. A pesar de ser una herramienta privada disponible en varias ediciones, posee una alternativa libre y gratuita, la versión Visual Paradigm UML 6.4 Community Edition. [39]

1.5.4 Diseño de interfaz gráfica.

Qt Designer es una biblioteca multiplataforma ampliamente usada para desarrollar aplicaciones con interfaz gráfica de usuario, así como también para el desarrollo de programas sin interfaz gráfica, como herramientas para la línea de comandos y consolas para servidores. Esta herramienta Qt Designer brinda una guía en el diseño y la construcción de interfaces gráficas de usuario (GUI) de los componentes Qt. Puede crear y personalizar sus *widget*[5] y pruebas de ellos con diferentes estilos y resoluciones. Los *widgets* y formas creadas con Qt Designer integran a la perfección con el código obtenido, con señales de Qt y el mecanismo de ranuras que le permite asignar fácilmente el comportamiento de los elementos gráficos. Todas las propiedades establecidas se puede cambiar de forma dinámica dentro del código. Además, características como la promoción *widget* y *plugins* personalizados permiten utilizar los propios componentes con Qt Designer. Qt proporciona una serie de clases para manejar automáticamente el

4 Suele utilizarse para nombrar al esquema de distribución de los elementos dentro un diseño.

5 Pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de widgets o Widget Engine. Entre sus objetivos están dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.

diseño, estas resuelven el reto de trazar los *widgets* automáticamente, proporcionando una interfaz de usuario que se comporta de manera predecible. [40]

Con el análisis de las pruebas de software, analizando como funcionan estas dentro de un proceso, los principales estándares de calidad que definen la utilización de métricas para determinar la calidad de un producto de software, se hizo una valoración del estado en el cual se encuentra el actual proceso de pruebas de Nova. El mismo está en concordancia con las necesidades de desarrollo, con una buena estrategia, procedimiento y organización, en su última versión se realizaron algunos cambios en su mejora continua con la inclusión en el diseño de los casos de prueba de una definición de los requisitos de calidad según los requerimientos y se insertó la selección de pruebas automáticas. En el proceso de valoración de las pruebas se incluyó la realización del cálculo de las métricas y la valoración de la calidad del producto al finalizar cada ciclo de pruebas. Además se realiza una comparación de los resultados con versiones anteriores y productos similares. El proceso de pruebas puede ser mejorado considerablemente con la facilitación de la integración entre sus actividades y publicación de los resultados, realizando una automatización de la gestión de estos aspectos vitales.

Las herramientas de pruebas automáticas que fueron investigadas brindaron una serie de requisitos y funcionalidades que pueden ser incluidos en la herramienta a desarrollar, para una mejor completitud de esta y que brinde la mayor cantidad de facilidades posibles a los probadores. Para el desarrollo se utilizará la metodología OpenUp que contiene un proceso unificado, ágil y liviano, así como una de sus mayores ventajas es que puede ser utilizado en proyectos pequeños. Como lenguaje de modelado el UML que contiene un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos con gran facilidad. La herramienta de modelado será el Visual Paradigm que proporciona un conjunto de ayudas y ha sido concebido para soportar el ciclo de vida completo del desarrollo del software con todo tipo de diagramas. Por último en el diseño de la interfaz gráfica se utilizará el Qt Designer, biblioteca multiplataforma ampliamente usada que brinda variadas facilidades para el diseño.

Capítulo 2. Características del sistema

Se realiza un análisis detallado de la solución propuesta del sistema para satisfacer las necesidades del cliente, describiendo cómo debe funcionar, sus potencialidades y características esenciales. Realizando luego una descripción de los requerimientos funcionales y no funcionales que debe tener el sistema. Con el análisis de diferentes patrones de arquitectura y diseño se construye una arquitectura adecuada y con las mejores características para su entendimiento y futura implementación. La descripción de la arquitectura se compone de un grupo de diagramas ordenados por las diferentes vistas que forman parte del sistema.

2.1 Propuesta del sistema.

El propósito fundamental de un proceso de pruebas es encontrar la mayor cantidad de errores y no conformidades en las primeras versiones funcionales de un software, lo que debe permitir corregirlos antes de liberarlo, incidiendo directamente en la calidad y el prestigio del producto. El sistema está estructurado como un módulo que automatiza la gestión del proceso de pruebas de un software, principalmente la evaluación de los resultados y la generación de los informes relacionados con este proceso.

En las distribuciones GNU/Linux no hay mucha divulgación sobre los procesos de pruebas pero Nova es un producto que se está promoviendo en el proceso de migración que está ocurriendo en Cuba. De ahí que un proceso de pruebas cómodo y que no requiera de altos conocimientos ingenieriles permite incrementar la socialización del mismo y aceptación del producto al incrementar las facilidades para la comunicación de los errores al equipo de desarrollo. Este sistema propuesto persigue los siguientes objetivos:

- Agilizar la generación de informes a partir de las plantillas predefinidas, sin que el usuario requiera estudiarse la estructura y elementos de cada una.
- Proveer facilidades para reportar las no conformidades o errores de forma cómoda para el usuario sin conocimiento técnico y entendible al equipo de desarrollo con una gestión de los resultados de las pruebas.
- Automatizar la evaluación de los resultados de las pruebas a partir del cálculo automático de las métricas definidas.

Estas son las principales características que presenta la gestión automatizada del proceso de pruebas de la cual se persigue realizar su arquitectura. Además de una gestión completa del plan de pruebas, así como de las iteraciones y ciclos que sea necesario realizar. Permite hacer modificaciones a los casos de

pruebas que sean introducidos y brindar una ayuda al usuario para un mejor entendimiento en el transcurso del trabajo con el sistema.

2.2 Especificación de los requisitos de la aplicación.

La especificación de los requisitos es uno de los flujos de trabajo más importantes, en él se establece qué tiene que hacer exactamente el sistema que se va a desarrollar. Los requisitos son las características que debe cumplir la aplicación y que el usuario final debe entender y aceptar.

En el proceso de selección de los requisitos se tomaron en cuenta primeramente los objetivos que se persiguen lograr en el futuro, y como estos pueden ser alcanzados transformados en funcionalidades. Todo esto se logra con una investigación previa de los sistemas que actualmente existen, las características que presentan en general cada uno de ellos, buscar aspectos contemplados en los objetivos propuestos para realizar una reutilización si fuera posible, así como ideas que puedan ser de importancia para incorporarlas. (Ver Anexo) (Para ampliar sobre requisitos ver en el expediente de proyecto los documentos 0113-Especificación de Requisitos de Software y 0129_Descripción de requisitos)

2.2.1 Requisitos funcionales.

A continuación aparece la lista de los requisitos funcionales y una descripción de los mismos, además de una especificación de los que son arquitectónicamente significativos:

RF 1 Gestionar plan de pruebas: Permite acceder al plan de pruebas del módulo de documentación y hacer una gestión del mismo. Este requisito es significativo para el sistema pues consiste en el primer paso a realizar para el inicio de las pruebas.

RF 1.1 Eliminar plan de pruebas: Permite eliminar un plan de pruebas dado un identificador.

RF 1.2 Adicionar plan de pruebas: Permite crear un plan de pruebas.

RF 1.3 Modificar plan de pruebas: Permite modificar las características de un plan de pruebas.

RF 1.4 Visualizar plan de pruebas: Permite visualizar un plan de pruebas.

RF 2 Gestionar ciclos de pruebas: Gestionar los ciclos de pruebas especificados en el plan de pruebas. Este requisito es significativo para el sistema pues consiste en la actividad que agrupa las iteraciones que sean necesarias para las pruebas.

RF 2.1 Mostrar ciclos de pruebas del plan de pruebas: Mostrar ciclos de pruebas del plan de pruebas por separado.

RF 2.2 Adicionar ciclo de prueba: Adicionar ciclo de pruebas y que este se adicione a su vez en el plan de pruebas.

RF 2.3 Modificar ciclo de pruebas: Dado un listado de ciclos de pruebas, seleccionar uno, hacer modificaciones en él y actualizar las mismas en el plan de pruebas.

RF 2.4 Eliminar ciclo de pruebas: Dado un listado de ciclos de pruebas, seleccionar uno eliminarlo y actualizar los cambios en el plan de pruebas.

RF 3 Gestionar iteración: Permite adicionar, eliminar, modificar o mostrar una iteración de un ciclo de pruebas. Este requisito es significativo para el sistema pues consiste en las actividades que se realizan directamente al caso de pruebas, está comprendido dentro de los ciclos de pruebas y se pueden realizar tantas iteraciones como sea necesario para las pruebas.

RF 3.1 Adicionar iteración. Permite adicionar una nueva iteración.

RF 3.2 Modificar iteración: Modificar las características de una iteración.

RF 3.3 Mostrar iteración: Mostrar las características de una iteración.

RF 3.4 Eliminar iteración: Eliminar una o varias iteraciones del ciclo de prueba al que pertenezca.

RF 4 Gestionar casos de pruebas: Permite adicionar, eliminar, modificar o mostrar casos de pruebas. Este requisito es significativo para el sistema pues consiste en gestionar todo un caso de pruebas, que es el artefacto por el cual se rigen para hacer las pruebas.

RF 4.1 Adicionar caso de prueba. Permite adicionar un nuevo caso de prueba.

RF 4.2 Modificar caso de pruebas: Modificar las características de un caso de pruebas.

RF 4.3 Mostrar detalles de un caso de pruebas: Mostrar las características de un caso de pruebas seleccionado.

RF 4.4 Eliminar caso de prueba: Dado un listado de casos de pruebas eliminar uno o varios casos de pruebas y actualizar este cambio en las iteraciones a las que ese caso de prueba este asociado.

RF 4.5 Listar casos de pruebas: Mostrar una lista con todos los casos de pruebas registrados organizados por tipo de pruebas, tipo de producto, por ciclo o por iteración.

RF 4.6 Comentar caso de pruebas: Permitir agregar comentarios a un caso de pruebas seleccionado.

RF 5 Gestionar resultado de las pruebas: Gestionar los resultados de las pruebas, que pueden ser satisfactorios o la detección de una No conformidad. Este requisito es significativo para el sistema pues consiste en obtener resultados de la realización de las pruebas y clasificarlos en dependencia de su tipo.

RF 5.1 Registrar el resultado de ejecutar un caso de caso de pruebas: Registrar los resultados de un caso de prueba luego de ser ejecutado.

RF 5.2 Mostrar el resultado de ejecutar un caso de pruebas: Mostrar las características del resultado de ejecutar un caso de pruebas.

RF 5.3 Listar resultados de ejecutar un mismo caso un mismo caso de pruebas: Listar los resultados de ejecutar un mismo caso de prueba organizados por probador, o fechas de registrado.

RF 5.4 Listar resultados de una iteración: Listar los resultados de ejecutar una iteración de prueba organizados por probador, o fechas de registrado.

RF 5.5 Listar resultados de un ciclo de pruebas: Listar los resultados de ejecutar un ciclo de pruebas organizados por iteraciones o fechas de registrado.

RF 6 Gestionar métricas: Permite adicionar, eliminar y modificar las métricas.

RF 7 Evaluar métricas de un ciclo: Realizar la evaluación de las pruebas según las métricas de calidad definidas para ese producto.

RF 8 Dar respuesta a una no conformidad: Muestra los detalles de una no conformidad seleccionada y da las opciones de respuesta de la no conformidad. Este requisito es significativo para el sistema pues consiste en darle un tratamiento a cada una en dependencia del resultado obtenido.

RF 9 Mostrar ayuda de la Suite de IS: Mostrar ayuda que permita al usuario ver como funcionan los procesos que automatiza la suite de ingeniería de software.

2.2.2 Requisitos no funcionales.

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener, también puede ser alguna restricción para cumplir una determinada funcionalidad. A continuación se presentan los requisitos no funcionales y especificaciones sobre cada uno de ellos:

Usabilidad

RnF 1. Los usuarios finales del módulo de pruebas de la Suite de IS son probadores, administradores de la calidad y desarrolladores de software, los cuales pueden tener niveles de conocimientos sobre esta área de medio, bajo o avanzado; por lo que el sistema debe proveer una guía que ayude a utilizar sus funcionalidades de forma intuitiva según los paradigmas de la ISW.

RnF 2. El módulo de pruebas de la Suite de IS debe ser fácil de utilizar.

La aplicación debe brindar a los usuarios las funcionalidades de forma intuitiva, con nombres sugerentes y señales que indiquen el flujo correcto de las acciones para permitirle ejercer sus actividades de una manera rápida y cómoda.

RnF 3. El módulo de pruebas de la Suite de IS es una aplicación de escritorio.

Debido a que se requiere de un software que sea rápido, que permita la gestionar el proceso de pruebas, el ingreso de grandes cantidades de datos, seguridad, y la continuidad del trabajo aun cuando no existe disponibilidad de la red.

RnF 4. La aplicación debe ser multiplataforma.

La aplicación es multiplataforma por lo que debe funcionar en sistemas operativos Windows, Linux y Macintosh.

RnF 5. Los usuarios finales deben ser capacitados.

Los usuarios finales deben recibir capacitación para que puedan utilizar correctamente todas las funcionalidades que provee el módulo de pruebas de la Suite de IS.

Confiabilidad

RnF 6. El sistema debe ser capaz de reanudar las operaciones si el funcionamiento del servidor de datos se ve afectado por la ausencia del fluido eléctrico.

Si el funcionamiento del sistema se ve interrumpido por la ausencia del fluido eléctrico en el área donde está alojado el servidor de datos, una vez que vuelva este, se reanudan las operaciones y los ordenadores clientes deben enviar la información con que se estaba trabajando de forma local al servidor de datos.

RnF 7. El sistema debe ser capaz de reanudar las operaciones si el funcionamiento de las PC clientes se ve afectado por la ausencia del fluido eléctrico.

Si el funcionamiento del sistema se ve interrumpido por la ausencia del fluido eléctrico en el área donde están alojadas las PC clientes, una vez que vuelva este, se debe actualizar nuevamente el sistema con la última información guardada en el servidor de datos.

RnF 8. El sistema debe ser capaz de reanudar las operaciones si el funcionamiento de las PC clientes se ve afectado por la ausencia de conexión a través de la red.

Si el funcionamiento del sistema se ve interrumpido por la ausencia de conectividad al servidor desde las PC clientes, una vez que se establezca la conexión, se debe actualizar en el servidor la información que el usuario había guardado hasta el momento.

RnF 9. El sistema debe ser capaz de reanudar las operaciones si el funcionamiento del servidor de datos se ve afectado por la ausencia de conexión a través de la red.

Si el funcionamiento del sistema se ve interrumpido por la ausencia de conectividad en el servidor, una vez que se establezca la conexión, se debe enviar una notificación al usuario y permitir actualizar en el servidor la información que el usuario había guardado hasta el momento.

RnF 10. El sistema debe permitir la impresión de los reportes generados.

Si se está trabajando con una impresora y esta se desconecta cuando se están haciendo operaciones con ella, el sistema debe mostrar un mensaje al usuario indicándole el incidente.

RnF 11. El tiempo de inactividad del sistema debe ser configurado por el usuario final.

El tiempo que el sistema debe estar inactivo debe ser configurable por el usuario final, y una vez que este se cumpla el sistema debe ser capaz de regresar a su estado inicial.

RnF 12. Las excepciones del sistema deben ser notificadas al usuario final.

Cuando ocurre una excepción el sistema debe mostrar un mensaje notificándola y mantenerse en el

estado que estaba antes de esta ocurrir si la excepción es leve, si por el contrario es una excepción crítica después de ser notificada el sistema debe cerrar.

RnF 13. La información debe estar disponible en todo momento para los usuarios finales.

La información guardada debe estar disponible siempre para los usuarios finales permitiendo que estos puedan acceder a ella siempre que deseen potenciando así la toma de decisiones.

Eficiencia

RnF 14. El sistema debe funcionar en hardware con bajas prestaciones.

El hardware con requerimientos mínimos sobre el que debe funcionar el módulo de pruebas de la Suite de IS se caracteriza por:

• En las PC clientes Micro Celeron 266, RAM 256MB.

• El servidor de datos especificado por el gestor PostgreSQL.

RnF 15. El sistema debe permitir que trabajen concurrentemente hasta 100 usuarios.

El sistema debe garantizar una concurrencia de 100 usuarios, es decir el trabajo sobre él de equipos grandes de probadores y que necesiten utilizar las mismas funcionalidades a la vez.

RnF 16. Para dar cumplimiento a los requisitos del sistema las transacciones deben tener una duración de 15 a 60 segundos.

Las transacciones que ocurren a la hora de ejecutar las operaciones deben tener una duración no mayor de 15 segundos y cómo máximo 60 segundos garantizando la respuesta rápida del sistema a las necesidades de los usuarios.

RnF 17. El tiempo de respuesta de la aplicación debe ser entre 5 y 60 segundos.

La aplicación ante las peticiones del usuario debe dar respuesta entre 5 y 60 segundos, permitiendo al usuario hacer sus operaciones de manera satisfactoria.

RnF 18. En las PC clientes el sistema debe consumir entre el 5% y el 40% de memoria.

En las PC clientes el sistema debe consumir entre el 5 y el 40% de memoria teniendo en cuenta que estas deben tener un tamaño de 512MB.

RnF 19. En el servidor el sistema debe consumir entre el 10% y el 50% del espacio en disco.

De acuerdo al volumen de información, las transacciones de las operaciones y la complejidad de las mismas en el servidor el sistema debe consumir entre el 10% y el 50% del espacio en disco duro, teniendo este un espacio mínimo de 10GB.

Soporte

RnF 20. El equipo de desarrollo debe brindar soporte por un periodo mínimo de un año.

El equipo de desarrollo debe brindar soporte por un periodo mínimo de un año y en los casos que lo requieran el tiempo de soporte debe ser mayor llegando este a un periodo máximo de dos años.

RnF 21. Si se encuentran no conformidades en el módulo de pruebas de la suite de IS estas deben ser corregidas en un periodo máximo de un mes.

Restricciones de diseño

RnF 22. El lenguaje de desarrollo debe ser compatible con PHP5 y versiones superiores.

RnF 23. Las funcionalidades deben habilitarse según estén generados los artefactos de las fases iniciales. Ejemplo: No se activa la funcionalidad de diseño de pruebas si no hay un plan de pruebas.

RnF 24. El sistema es un modulo que debe ser capaz de funcionar por si solo y a su vez integrarse como un plugin de la Suite.

Requisitos para la documentación de usuarios en línea y ayuda del sistema.

RnF 25. La ayuda del sistema debe brindarse en español y en ingles.

RnF 26. El sistema debe generar mensajes de ayuda para guiar el proceso a usuarios con bajos conocimientos.

2.3 Arquitectura Propuesta.

En un sentido amplio, la arquitectura del software es la definición de más alto nivel de la estructura de un software, que tiene la responsabilidad de definir los módulos principales, las responsabilidades que deben tener cada uno de estos y la interacción que existe entre dichos módulos. Además del control y flujo de datos, secuenciación de la información, protocolos de interacción y comunicación, así como la ubicación en el hardware. La arquitectura del software aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño. [41]

Estilos y patrones.

Con la imposición de algunos estilos arquitectónicos, las posibilidades de satisfacción de ciertos atributos de calidad del sistema pueden mejorar. Cada estilo contiene atributos de calidad, y la decisión de implementar alguno de los existentes depende de los requerimientos de calidad del sistema. Similar a esto se plantea el uso de los patrones de diseño y los patrones arquitectónicos para mejorar la calidad del sistema. El éxito de los patrones es la forma en que se alcanza de manera satisfactoria los objetivos de la ingeniería de software. Los patrones soportan el desarrollo, mantenimiento y evolución de sistemas complejos y de gran escala.

Los estilos y los patrones establecen un vocabulario común y brindan un soporte a los ingenieros para

conseguir una solución que haya sido aplicada con éxito anteriormente ante ciertas situaciones. Su aplicación en el diseño de la arquitectura del sistema juega un papel determinante para la satisfacción de los requerimientos de calidad. [42]

2.3.1 Estilos Arquitectónicos.

Para dar una definición de estilo existen diferentes clasificaciones dadas por varios autores. Definen estilo arquitectónico como una familia de sistemas de software en términos de un patrón de organización estructural, que contiene un vocabulario de componentes y tipos de conectores y un conjunto de restricciones de cómo pueden ser combinadas. Otras bibliografías expresan que son componentes y relaciones entre estos y con restricciones en su aplicación. Además de considerarse como un tipo particular de estructura fundamental para un sistema de software, en conjunto con un método asociado que especifica cómo construirlo. [43]

Estilos centrados en datos.

Estos estilos hacen especial énfasis en la integrabilidad de los datos. Puede ser apropiada para sistemas de acceso y actualización de datos en estructuras de almacenamiento. Los estilos asociados de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra, que en este último caso, contiene un grupo de características que se ajustan a la estructura que se desea para nuestro sistema.

Arquitecturas de Pizarra o Repositorio.

En esta arquitectura hay dos componentes principales, una estructura de datos que representa el estado actual y una colección de componentes independientes. En base a esto se han definidos dos subcategorías del estilo. La primera expresa que si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional. Como segunda, si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control.

Un sistema de pizarra se implementa para resolver problemas en los cuales las entidades individuales se manifiestan incapaces de aproximarse a una solución, o para los que no existe una solución analítica, o para los que sí existen pero es inviable por la dimensión del espacio de búsqueda. Todo modelo de este tipo consiste en tres partes, las fuentes de conocimiento necesarias para resolver el problema, la pizarra que representa el estado actual de la resolución del problema y la estrategia que regula el orden en que operan las fuentes. El estilo de pizarra tiene pleno sentido si tanto las fuentes de conocimiento con la pizarra se entienden como clases que son favorables para instanciarse en diversas variedades de objetos.

Este caso de arquitectura presenta una característica que es propia en el comportamiento que se quiere

para el sistema, que es la comunicación centralizada hacia una base de datos, esta se comunicará con una capa de control que manejara las diferentes clases que componen el sistema y que a su vez establecerá comunicación con la vista mediante la capa de control hacia la base de datos y no directamente. [43]

Estilos de llamada y retorno.

Estos estilos enfatizan en la modificabilidad y la escalabilidad. Son más generalizados en sistemas a gran escala. Los miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas en capas. Siendo la arquitectura en capas la que más se ajusta a las necesidades del sistema a implementar, al contar el mismo con un grupo de capas que establecerán comunicación entre estas, de diferentes maneras posibles.

Arquitectura en capas.

Las arquitecturas en capas constituyen uno de los estilos que aparecen con mayor frecuencia. Los conectores se definen mediante los protocolos que determinan las formas de la interacción. Las restricciones topológicas del estilo pueden incluir una limitante, que exige a cada capa operar sólo con capas adyacentes, y a los elementos de una capa entenderse sólo con otros elementos de la misma. Se supone que si esta exigencia no se cumple, disminuye la flexibilidad del conjunto y se complica su mantenimiento, porque el estilo pierde la posibilidad de reemplazar totalmente una capa sin afectar a las restantes. Casos representativos de este estilo son muchos de los protocolos de comunicación en capas, en ellos cada una proporciona un sustrato para la comunicación a algún nivel de abstracción.

Entre las ventajas del estilo en capas primeramente se tiene que soporta un diseño basado en niveles de abstracción crecientes, lo que a su vez permite a los desarrolladores la partición de un problema complejo. El estilo admite las optimizaciones y refinamientos que sean necesarios. Proporciona una amplia reutilización. 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 brinda la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. [43]

2.3.2 Patrones Arquitectónicos.

Los patrones arquitectónicos son plantillas que describen los principios estructurales globales que construyen las distintas arquitecturas de software viables. Plantean una organización estructural fundamental para un sistema de software, expresando un conjunto de subsistemas predefinidos, especificando responsabilidades y organizando las relaciones entre ellos. Los patrones no se proponen

descubrir ni expresar nuevos principios de la ingeniería de software. Intentan codificar el conocimiento, las expresiones y los principios ya existentes. Proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. Cada patrón ayuda a lograr una propiedad específica del sistema global como es la adaptabilidad de la interfaz de usuario. Los patrones no introducen ideas novedosas, son una mera codificación de lo más usado hasta la actualidad.

Se pueden agrupar según sus propiedades en varias categorías y dentro de estas los diferentes tipos de patrones. En este caso se toman los llamados Del fango a la estructura y los Sistemas interactivos, como los indicados por las características que presentan, para adaptarse a la estructura que tomará el sistema.

Del fango a la estructura.

Estos proporcionan una ayuda para evitar un mar de componentes u objetos. En particular brindan apoyo a una descomposición controlada de una tarea del sistema global en subtareas cooperantes. En esta categoría el indicado que se incluye es el patrón Blackboard.

Blackboard.

Es útil para problemas en los cuales no se conoce ninguna estrategia de solución determinística factible en la transformación de datos en estructuras de alto nivel. En este patrón varios subsistemas especializados ensamblan sus conocimientos para construir una solución posiblemente aproximada o parcial. La arquitectura está sobre la idea de una colección de programas especializados e independientes que trabajan sobre una estructura de datos común, donde el estado de progreso actual es evaluado por un componente de control central.

El sistema se divide en un componente que tiene como nombre pizarra, una colección de fuentes de conocimiento, y un componente de control. La pizarra es el almacenamiento central de datos que guarda también los elementos del espacio de solución y el control de datos. Proporciona una interfaz que habilita a todas las fuentes de conocimiento para leer y escribir en ella. Las fuentes de conocimiento son subsistemas separados e independientes que resuelven aspectos específicos del problema global y modelan el dominio del mismo. El componente de control monitorea los cambios en la pizarra y decide qué acción tomará luego, esto utilizando las evaluaciones de la fuente de conocimiento.

El patrón Blackboard aporta las siguientes ventajas:

Reusabilidad: Las fuentes de conocimiento son independientes para ciertas tareas. Los requisitos previos para rehusar son centrados en que la fuente de conocimiento y el sistema subyacente entiendan el mismo protocolo y datos.

Cambiabilidad y mantenibilidad: Las fuentes de conocimiento son individuales, el algoritmo de control y la

estructura de datos centrales están estrictamente separados.
Tolerancia a fallos y robustez: En esta arquitectura todos los resultados son sólo hipótesis.
Experimentación: El patrón experimenta con la mayor cantidad posible de algoritmos diferentes, y también permite intentar con diferentes heurísticas de control. Esto ocurre en dominios en que no existe alguna posible solución y una búsqueda completa del espacio de la solución no es factible. [44]

Sistemas Interactivos.

En esta categoría entra el patrón de Presentation–Abstraction–Control (PAC). Este apoya la estructuración de sistemas de software que ofrecen la interacción usuario-computadora. El centro de los sistemas interactivos está basado en los requerimientos funcionales del sistema. El objetivo es robustecer la utilidad de una aplicación mediante un alto grado de interacción del usuario, generalmente, con la ayuda de interfaces de usuario gráficas.

Presentación-Abstracción-Control.

El patrón arquitectónico define una estructura para los sistemas de software interactivos en forma de jerarquía de agentes cooperantes. En esta arquitectura de cooperación de agentes, cada uno se especializa en una tarea específica, y todos juntos proporcionan la funcionalidad del sistema. Cada agente depende de todos los agentes de niveles más altos y así hasta llegar al agente de nivel superior.
Presenta tres componentes: presentación, abstracción, y control. Esta subdivisión realiza una separación de los aspectos de interacción usuario-computadora del agente central funcional y su comunicación con otros agentes. Cada componente del patrón PAC tiene una tarea específica. La presentación proporciona la conducta visible del agente, la abstracción mantiene el modelo de datos que fundamenta al agente y proporciona la funcionalidad para operar sobre esos datos y el componente control conecta los componentes presentación y abstracción proporcionando la funcionalidad de comunicación con otros agentes.
Se compone de agentes de nivel superior, de nivel intermedio y de nivel inferior. El nivel superior proporciona el centro funcional del sistema, además incluye las partes de la interfaz del usuario que no pueden ser asignadas a subtareas particulares. En el nivel medio se representan cualquier combinación o relación entre los agentes de niveles más bajos, pudiendo mantener varias vistas de los mismos datos. Los agentes de nivel inferior representan su propio contenido semántico de los conceptos en los cuales los usuarios del sistema pueden actuar.
Este patrón presenta varias ventajas:
Cambiabilidad y extensibilidad: Los cambios dentro de los componentes presentación o abstracción de un agente no afectan a otros agentes en el sistema. Esto permite modificar individualmente el modelo de

datos que está debajo de un agente o cambiar su interfaz de usuario. Nuevos agentes se integran fácilmente en una arquitectura PAC existente sin mayores cambios para el resto. Todos ellos se comunican entre sí a través de una interfaz predefinida.
Multitarea: Los agentes pueden distribuirse fácilmente en diferentes hilos, procesos, o máquinas.
Separar intereses: Cada agente mantiene su propio estado y datos en forma coordinada e independiente de otro agente. [44]

Luego del análisis de los diferentes patrones arquitectónicos escogidos para su utilización en la arquitectura del sistema, se hace necesaria la integración del modelo en 3 capas de Presentación Abstracción Control y el Blackboard, para lograr la arquitectura que se necesita. Con el primero obtenemos una estructura lineal entre las partes que forman la arquitectura, fluyendo una comunicación desde la clase encargada de los datos, pasando por la controladora y llegando a las interfaces. Por parte del segundo se integrará junto con el anterior, pues este funciona como una pizarra que contiene los datos e interactúa con todas las interfaces por medio de las controladoras de las clases y regirá como el modelo o base de datos de la aplicación.

2.4 Patrones de Diseño.

Los patrones de diseño son soluciones bien documentadas que los desarrolladores emplean apoyados en la experiencia de haberlas utilizado con éxito, para dar solución a nuevos problemas. Un patrón de diseño está estructurado en clases que se presentan en forma repetida en distintos diseños orientados a objetos, las cuáles son empleadas para resolver un problema de manera flexible y adaptable. Los patrones de diseño brindan una forma eficaz de compartir la experiencia dentro de la comunidad de programación, así como se han transformado en una técnica difundida para la reutilización de conocimiento de diseño de software. [44]

2.4.1 Patrones GRASP: para asignar responsabilidades.

La calidad del diseño de la interacción de los objetos y la asignación de responsabilidades presentan gran variación. Una implementación hábil se funda en los principios cardinales que rigen un buen diseño orientado a objetos. En los patrones GRASP se codifican algunos de ellos, que se aplican en la interacción y cuando se asignan las responsabilidades. A continuación una relación de los seleccionados para ser aplicados al sistema.

Experto.

Asigna una responsabilidad al experto en información, es decir la clase que cuenta con la información necesaria para cumplir la responsabilidad. Cuando se definen interacciones entre los objetos durante el

diseño orientado a objetos, se toman las decisiones sobre la asignación de responsabilidades a las clases. Haciendo esto de forma adecuada el sistema es más fácil de entender, ampliar y mantener. Además se pueden reutilizar diferentes componentes en futuras aplicaciones. Es un patrón que se usa más que cualquier otro al asignar responsabilidades, ofrece una analogía con el mundo real porque se asignan responsabilidades a personas que disponen de la información para llevar a cabo una tarea.

Se escogió este patrón pues brinda beneficios importantes al conservar el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta otro patrón GRASP, el bajo acoplamiento que favorece a tener un sistema más robusto y de fácil mantenimiento. El comportamiento se distribuye entre las clases que cuentan con la información requerida, así se crean clases sencillas y mas cohesivas que son más fácil de mantener y comprender. Se ve aquí la vinculación con otro patrón la alta cohesión.

Creador.

Asigna a la clase B la responsabilidad de crear una instancia de la clase A en algunos de los casos siguientes:

- B agrega los objetos de A.
- B contiene los objetos de B.
- B registra las instancias de los objetos de A.
- B utiliza específicamente los objetos de A.
- B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado, por tanto B es un experto en la creación de A.

La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. Cuando está bien asignado puede soportar un bajo acoplamiento, el encapsulamiento, la reutilización y presenta una mayor claridad.

Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos, el propósito fundamental es encontrar un creador que se debe de conectar con el objeto producido en cualquier evento.

Se escogió este patrón porque se obtienen beneficios brindando un soporte a un bajo acoplamiento, lo que supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase creada tiende a ser visible a la clase creador, debido a las asociaciones actuales que llevan a elegirla como él parámetro adecuado.

Bajo acoplamiento.

Consiste en asignar una responsabilidad para mantener un bajo acoplamiento. Este es una medida de como una clase está conectada con otras clases, que las conoce y que recurre a ellas. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño. Estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento. Soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad.

Se escogió este patrón porque ante todo a de incluirse como uno de los principales del diseño que influyen en la decisión de asignar responsabilidades. Para apoyar una mejor reutilización de los componentes, se tiene en cuenta las metas de rehúso antes de intentar disminuir al mínimo el acoplamiento. Entre los beneficios que trae contamos con que las clases no se afectan por cambios de otros componentes, son más fáciles de entender por separado y de reutilizar.

Alta cohesión.

Se asignan responsabilidades de modo que la cohesión siga siendo alta. Esta es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo grande. Este patrón es la meta principal que ha de buscarse en todo momento, es evaluativo y el desarrollador lo aplica al valorar sus decisiones de diseño.

Se escogió este patrón pues trae beneficios al mejorar la claridad y facilidad con que se entiende el diseño. Se puede simplificar el mantenimiento y las mejoras en funcionalidad. Muy seguido se genera un bajo acoplamiento y tiene la ventaja de una gran funcionalidad al soportar mayor capacidad de reutilización, porque una clase que sea muy cohesiva puede dedicarse a un propósito bien específico. [44]

2.4.2 Patrones GoF.

Los patrones GoF se descubren como una forma indispensable de enfrentarse a la programación a raíz del libro *"Design Patterns Elements of Reusable Software"* de Erich Gamma, Richard Helm, Ralph Jonson y John Vlissides. Luego de la publicación empezó a difundirse con más fuerza la idea de patrones de diseño. Recopilaron y documentaron 23 patrones de diseño aplicados usualmente por expertos diseñadores de software orientado a objetos, estos se clasifican en 3 grandes categorías basadas en su propósito como son los creacionales, estructurales y de comportamiento. De cada una de estas clasificaciones se escogen las que se ajustan a la arquitectura**.**

Creacionales.

Tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es de abstraer el

proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.

Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.

Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

Estructurales.

Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades.

Adapter (Adaptador): Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.

Flyweight (Peso ligero): Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.

Comportamiento.

Los patrones de comportamiento ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

Chain of Responsibility (Cadena de responsabilidad): Permite establecer la línea que deben llevarlos mensajes para que los objetos realicen la tarea indicada.

Mediator (Mediador): Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.

Observer (Observador): Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él. [45]

Se escogen este grupo de patrones GoF porque brindan un grupo de características necesarias para el diseño. Estos proponen soluciones a problemas concretos, no son teorías genéricas. En ocasiones tienen más utilidad con algunos lenguajes de programación y en otras son aplicables a cualquier lenguaje. Se utilizan en situaciones frecuentes. Debido a que se basan en la experiencia acumulada al resolver problemas reiterativos. Ayudan a construir software basado en la reutilización, a construir clases reutilizables. Al aplicar un patrón, el código resultante no tiene por qué delatar el patrón o patrones que lo inspiró.

2.6 Vistas arquitectónicas del módulo.

Es muy complejo capturar la arquitectura software en un sólo modelo o diagrama. Para manejar esta

complejidad se representan diferentes aspectos y características de la arquitectura en múltiples vistas. Una vista es “una presentación de un modelo, la cual es una descripción completa de un sistema desde una particular perspectiva” (Kruchten, 1995). El modelo más aceptado a la hora de establecer las vistas necesarias para describir una arquitectura software es el modelo 4+1. Cada vista es descrita usando su particular y más adecuada notación, los arquitectos pueden escoger cierto estilo arquitectónico o patrón arquitectónico, permitiendo la coexistencia de múltiples estilos en un sistema. Este método ayuda en múltiples puntos de vista pues se pueden abordar por separado las preocupaciones de las distintas partes interesadas de la arquitectura. [46]

2.6.1 Vista de Casos de Uso.

Esta vista va a ser representada por los casos de uso, que ayudan a unir las cuatro vistas, así desde un caso de uso podemos ver cómo se van mezclando las vistas, con esto tenemos una trazabilidad de componentes, clases, equipo y paquetes.

Luego del análisis de todas las actividades que va a realizar el sistema, en conjunto con los interesados del mismo y contando con las funcionalidades que son más necesarias para el buen desarrollo de la arquitectura y futuro correcto funcionamiento, se llega a la conclusión del siguiente Diagrama de Casos de Uso del Sistema:

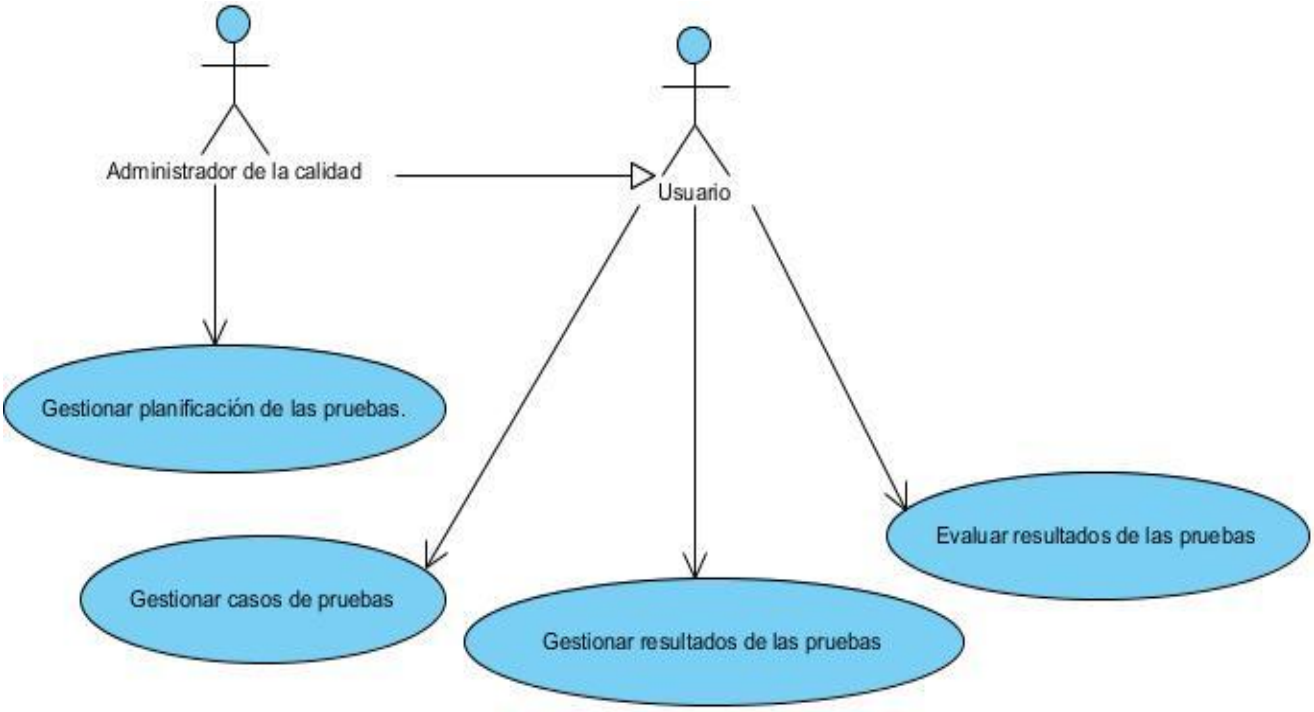


Figura 1. Diagrama de Casos Uso del Sistema.

El anterior diagrama muestra el comportamiento de los actores y casos de uso del sistema. En los actores se puede evidenciar como al Usuario que inicializa los casos de uso Gestionar casos de pruebas, Gestionar resultados de las pruebas y Evaluar resultados de las pruebas, se le aplica el patrón múltiples actores. Pues este realiza las funciones de Probador y Desarrollador, los cuales tienen actividades en común en el sistema, evitando así que más de un actor con el mismo rol inicie el mismo caso de uso. Por su parte el rol Administrador de la calidad, que también realiza esas actividades anteriormente mencionadas, inicia el caso de uso Gestionar planificación de las pruebas, que es una actividad que solo él tiene privilegios para hacer.

Descripción de los actores involucrados en el sistema:

Administrador de la calidad: Es el encargado de planificar las pruebas y diseñar las estrategias de pruebas, así como de supervisar y evaluar todo el proceso.

Probador: Es el responsable del diseño de los casos de pruebas y, del registro y asignación de las no conformidades detectadas después de ejecutar un caso de prueba.

Desarrollador: Tienen la posibilidad de observar los casos de pruebas, los resultados de ejecutarlos y emitir comentarios sobre ellos, así como darles respuesta.

Descripción de los casos de uso involucrados en el sistema:

Gestionar planificación de las pruebas: es el encargado de realizar toda una gestión del plan de pruebas que se elabora al inicio de comenzar la realización de las mismas. Gestiona los ciclos de pruebas así como las iteraciones que sean necesarias ejecutar durante todo el proceso.

Gestionar casos de pruebas: realiza toda una gestión con los casos de pruebas que son introducidos por parte del diseñador de los casos de pruebas, además de permitir listar los casos de pruebas y poder hacer comentarios sobre ellos.

Gestionar resultados de las pruebas: encargado de registrar los resultados de las pruebas, permitiendo luego de esto poder listar y mostrar los mismos, al tener la ocurrencia de una no conformidad se tiene la posibilidad de poder dar respuesta a la misma según la clasificación que le corresponda.

Evaluar resultados de las pruebas: Realiza una evaluación de las pruebas según las métricas de calidad definidas para cada producto, permitiendo también una gestión de las métricas para su correcto manejo y utilización.

2.6.2 Vista Lógica.

La vista lógica trata el análisis y la especificación de los requisitos funcionales, es decir, lo que el sistema debería proporcionar en términos de servicios a sus usuarios con las funciones y servicios que se han definido. El sistema se descompone en un conjunto de abstracciones clave tomadas mayormente del dominio del problema, en forma de objetos o clases. Esta descomposición no es sólo por el bien del análisis funcional, sino también sirve para identificar los mecanismos comunes y elementos de diseño a través de las diferentes partes del sistema.

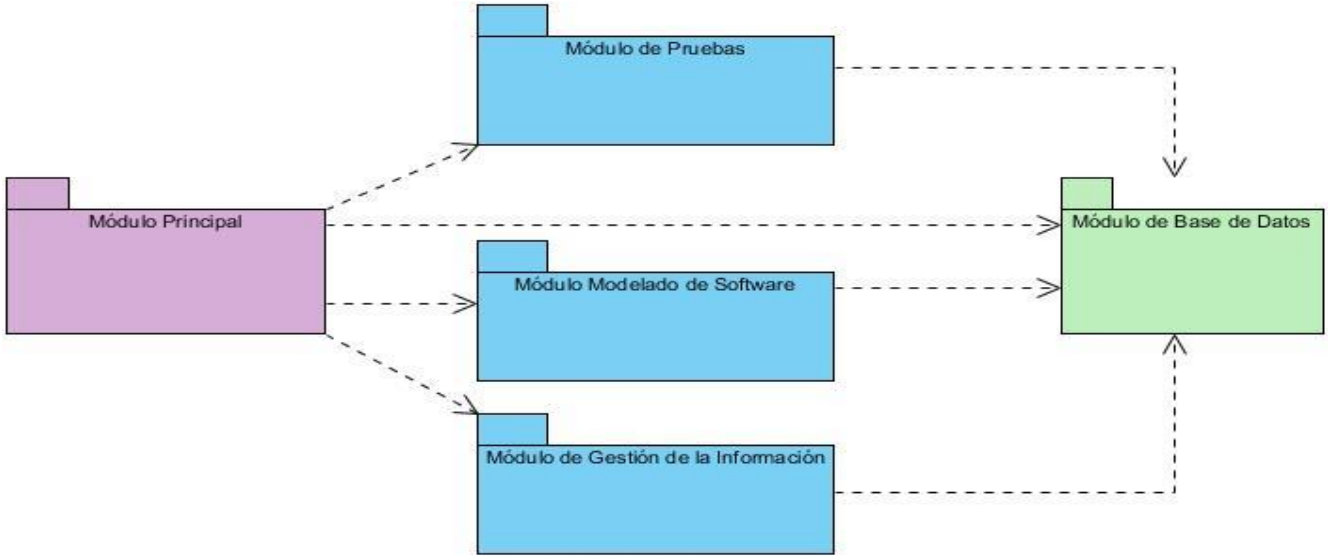


Figura 2. Diagrama de Paquetes de la suite de ingeniería de software.

Para definir la arquitectura del módulo se parte de la arquitectura que conforma la suite de ISW en general, detallando luego el paquete del módulo de pruebas.

Módulo Principal: Integra el funcionamiento de todos los módulos a la vista de los usuarios, garantizando que estos tengan los privilegios adecuados según su rol, y permitiendo la gestión general de un proyecto determinado.

Módulo de Pruebas: El propósito fundamental de un proceso de pruebas es encontrar la mayor cantidad de errores y no conformidades en las primeras versiones funcionales de un software, lo que permitiría corregirlos antes de liberarlo, incidiendo directamente en la calidad y el prestigio del producto. El sistema de pruebas de software pretende automatizar el proceso de pruebas de un software, principalmente la evaluación de los resultados y la generación de los informes relacionados con este proceso. En primera

instancia se especializará en el proceso que llevan a cabo las distribuciones GNU/Linux y posteriormente se ampliará hacia un proceso más genérico para todo tipo de software.

Módulo de Modelado de Software: Permite modelar software teniendo en cuenta todas las perspectivas desde las que se puede analizar este tipo de producto, para esto brinda opciones para utilizar la notación UML, así como para detallar los negocios de las organizaciones a través de BPMN, se pueden representar además modelos de las vistas lógicas y físicas de las bases de datos y la trazabilidad de los requisitos.

Módulo de Gestión de la Información: Conjunto de funcionalidades que permiten guardar el historial de un proceso de desarrollo de software en productos de trabajo predefinidos, así como las planificaciones y los resultados que se van obteniendo de ejecutarlas, elementos que sirven de evidencias para la toma de decisiones en el proyecto y en otros similares.

Módulo Bases de Datos: Es el núcleo del sistema pues tiene la tarea fundamental de almacenar e integrar toda la información que va a ser procesada en el restos de los módulos.

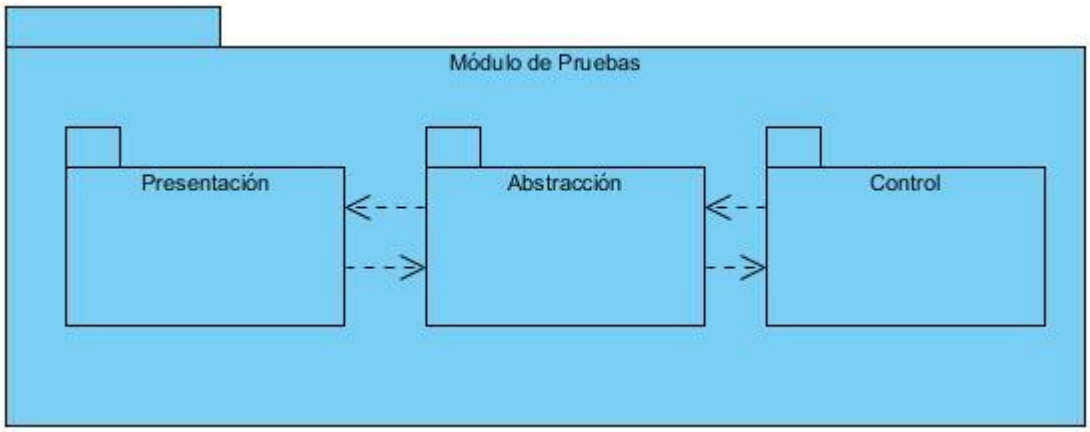


Figura 3. Diagrama General de Paquetes del Módulo de Pruebas.

El anterior diagrama muestra una vista general del módulo de pruebas, con la estructura de los diferentes paquetes que lo componen, aplicando el patrón seleccionado anteriormente Presentación Abstracción Control.

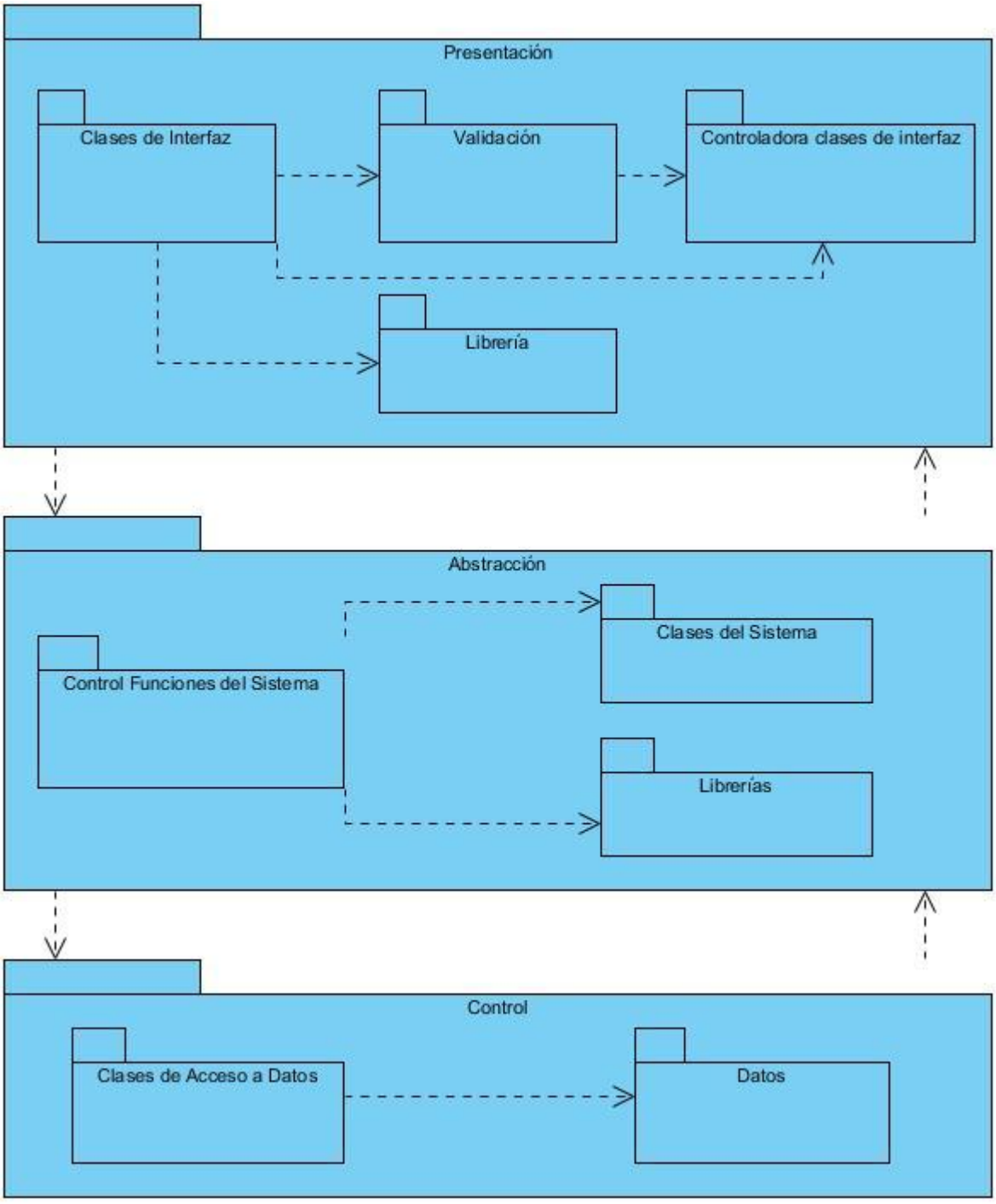


Figura 4. Diagrama de Paquetes del Módulo de Pruebas.

El Módulo de Pruebas basado en el patrón de arquitectura PAC está estructurado por los siguientes paquetes:

Presentación: Contiene el paquete de Interfaz Principal que se conecta con el resto de las interfaces de la aplicación, que cada una de ellas gestionan la información según su tipo.

Abstracción: Contiene los paquetes que almacenan la información en dependencia de la clase interfaz de gestionar a la que estén conectados.

Control: Contiene los paquetes de control de la información en dependencia de la clase abstracción a la que estén conectados.

El siguiente diagrama de clases brinda las distintas interacciones entre las clases que forman parte de la aplicación. Mostrando los atributos que las componen y métodos asociados a cada una de ellas.

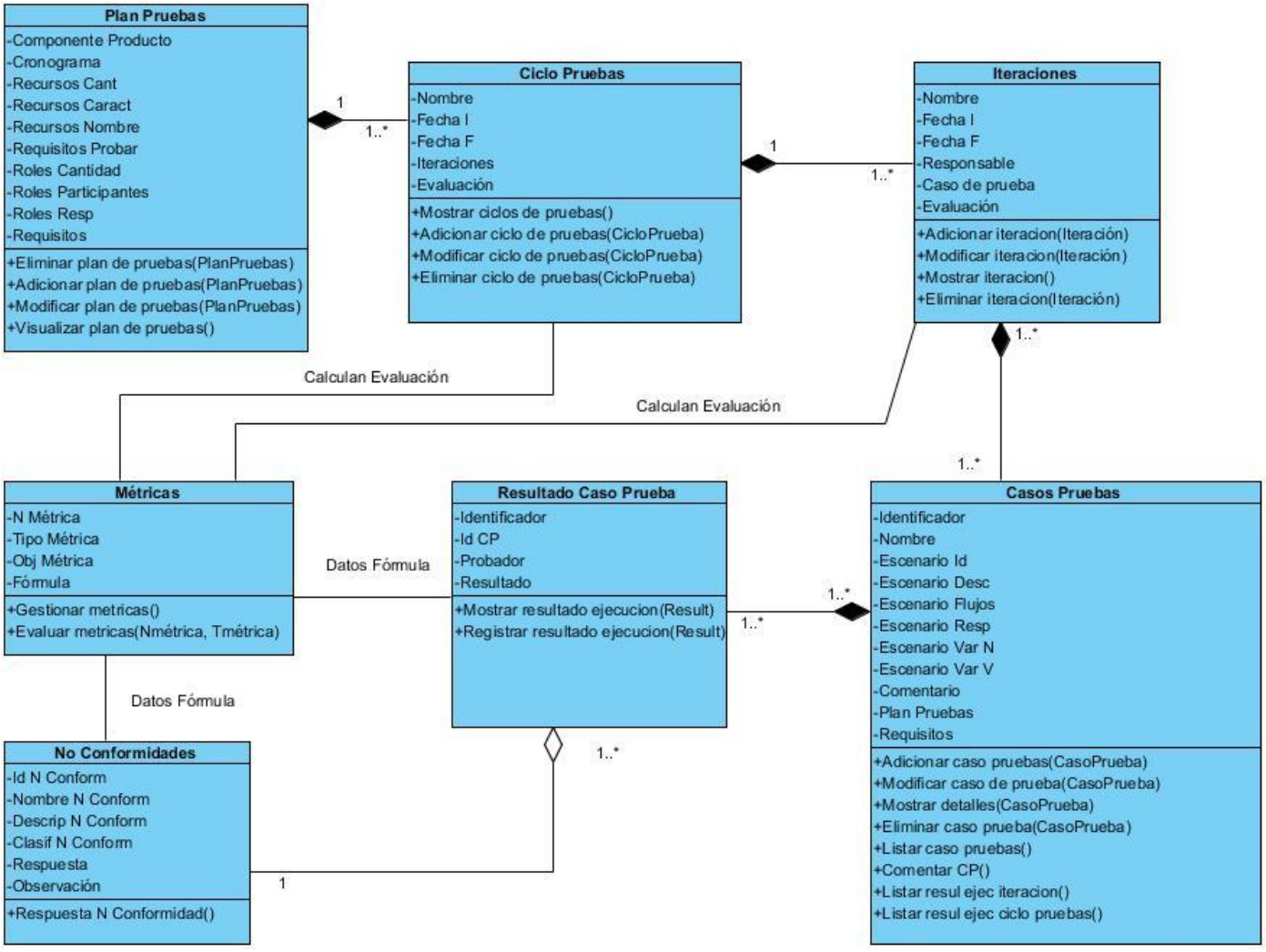


Figura 5. Diagrama de Clases.

2.6.3 Vista de Implementación.

La vista de implementación está compuesta por una organización de los módulos en términos de paquetes y capas, así como puede incluirse también la trazabilidad de la vista lógica. Se representa por un diagrama de componentes o especificaciones de paquetes, que son básicamente un subconjunto del modelo de despliegue.

El diagrama que a continuación se muestra está estructurado por un grupo de componentes y a cada uno le corresponde una funcionalidad del sistema respecto a las interfaces que las realizan. Además de los siguientes componentes:

PyQt4.py: utiliza para crear las diferentes interfaces y hacer su generación a código Python.
Validaciones.py: utiliza para validar los datos que introduzcan los usuarios.
ControladoraVisual.py: controla el flujo de datos dentro de la capa de presentación y la de la lógica del negocio.
Controladora.py: controla el flujo de datos correspondiente a la capa Abstracción y su relación con la capa de Presentación y la de Control.
ControladoraModelo.py: controla el flujo de datos entre el componente Controladora y el PythonRelatorio.
PythonRelatorio: librería de Python-Relatorio.
ControladoraDatos.py: controla el flujo de datos de las capas de Presentación y Control.
Datos.sql: corresponde a los datos guardados en la base de datos del módulo.
PlanPruebas.ui: pertenece a la clase Plan de Pruebas, encargada de la gestión completa de los planes de pruebas que se crean.

CiclosPruebas.ui: pertenece a la clase Ciclos de Pruebas, encargada de la gestión completa de los ciclos de pruebas.

Iteraciones.ui: pertenece a la clase Iteraciones, encargada de la gestión completa de las iteraciones que se realizan.

CasosPruebas.ui: pertenece a la clase Casos de Pruebas, encargada de la introducción y reporte de casos de pruebas.

ResultCasoPrueba.ui: pertenece a la clase Resultado de Caso de Pruebas, encargada de la gestión de los resultados de los mismos.

NConformidades.ui: pertenece a la clase No Conformidades, encargada de realizar el tratamiento a las no conformidades encontradas, según el tipo que represente para el sistema.

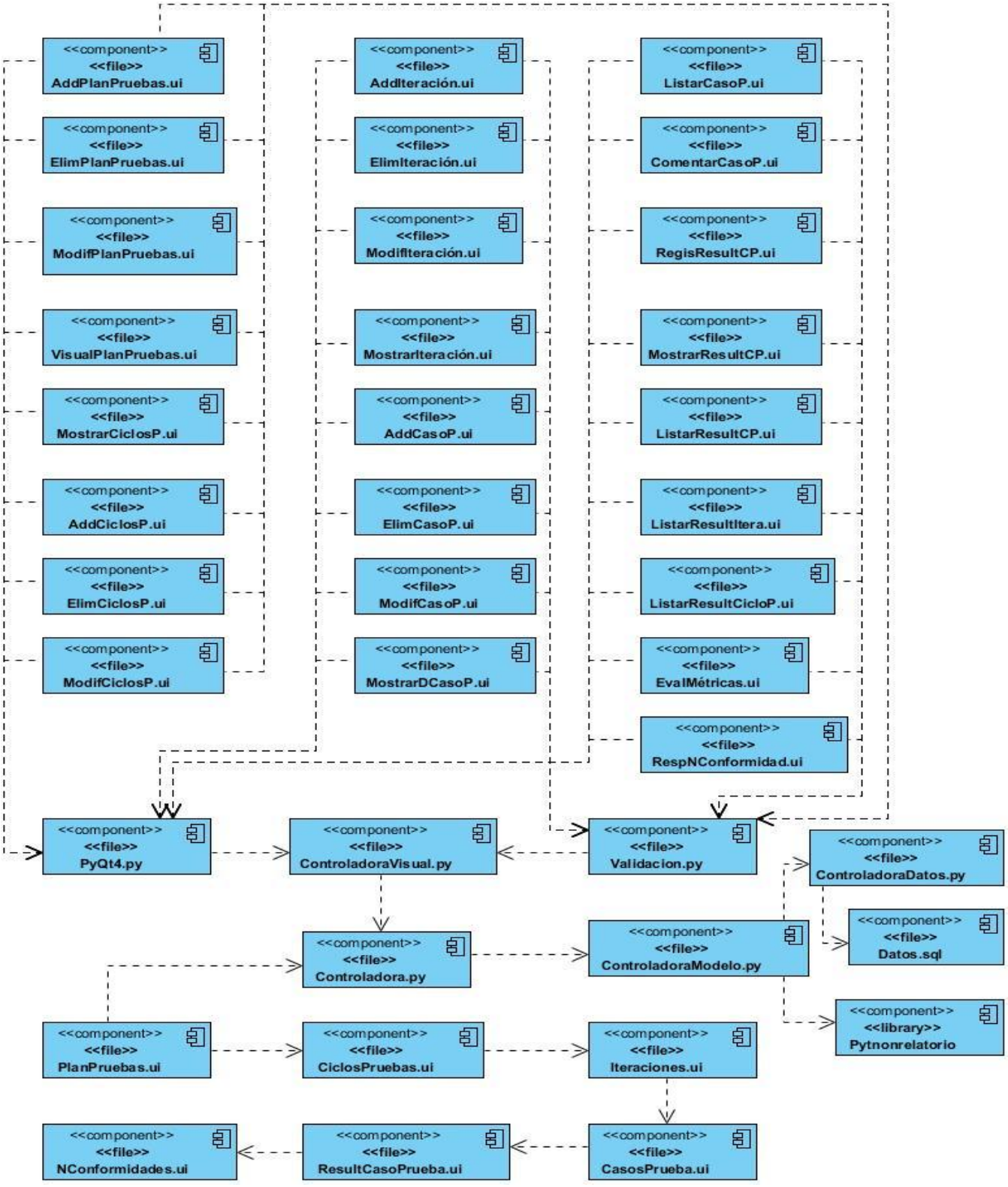


Figura 6. Diagrama de Componentes.

2.6.4 Vista de Despliegue.

La vista de despliegue o desarrollo se enfoca en la organización de los módulos software en el entorno real de trabajo. Se va a mostrar principalmente como está dividido nuestro sistema de software en componentes, y muestra las dependencias entre estos. El software es empaquetado en pequeños pedazos como librerías de programa, subsistemas, componentes, etc. También va a mostrar la organización y las dependencias entre el conjunto de componentes, y como se comunican entre ellos.

Figura 7. Diagrama de Despliegue.

Servidor de Base de Datos: Es el núcleo de almacenamiento del sistema. Tiene la tarea fundamental de almacenar e integrar toda la información que va a ser procesada en el resto de los módulos. El servidor de base de datos está especificado por el gestor PostgreSQL.

PC Cliente: Son las estaciones de trabajo que el usuario utiliza para acceder a la aplicación y realizar lo que necesite según las diferentes tareas.

Impresora: Es el dispositivo que realiza el proceso de impresión de la documentación generada por la aplicación en diferentes planillas definidas. Imprimirán las que sea necesario tener en formato duro, luego de que se realicen determinado grupo de pruebas.

Se utiliza el protocolo de comunicación entre el Servidor de BD y la PC Cliente TCP/IP, como el indicado para el intercambio de los datos e información entre estos y con la impresora será a través de USB.

El capítulo detalla cómo debe ser el funcionamiento del sistema, sus futuras potencialidades y principales características. La descripción de cada uno de los requerimientos funcionales aportó a la conformación del diseño de los diagramas que estructuran como se visualizará la aplicación, con cada una de sus funciones específicas y también las propiedades que presenta, especificadas en la descripción de los requisitos no funcionales. El análisis de diferentes patrones de arquitectura y diseño arrojó la construcción de una arquitectura adecuada y con características que muestran facilidades y técnicas de mejora continua. Esto

fue logrado con la integración de dos tipos de patrones arquitectónicos, el modelo en 3 capas, de Presentación Abstracción Control y el Blackboard. Del primero se escogió su estructura de funcionamiento en capas y de manera lineal, y del segundo caso su forma de comunicación en modelo de datos centralizado. La descripción de toda la arquitectura se estructuró en un grupo de diferentes vistas, dentro de las cuales se ordenaron los diagramas que estructuran el sistema, evitando así la complejidad de capturar la arquitectura en un sólo diagrama o modelo. Por lo que por este método se brinda una ayuda al abordar por separado las preocupaciones de las partes interesadas en la arquitectura.

Capítulo 3. Evaluación de la arquitectura.

Para obtener una arquitectura robusta, depende entre otras cosas, de la evaluación que se le aplique. El primer paso para la evaluación de una arquitectura es conocer qué es lo que se quiere evaluar. De esta forma, es posible establecer la base para la evaluación. Resulta interesante el estudio de la evaluación de una arquitectura si las decisiones que se toman sobre la misma determinan los atributos de calidad del sistema, es entonces posible evaluar las decisiones de tipo arquitectónico con respecto al impacto sobre estos atributos. Al termino de la investigación sobre los métodos para la evaluación de la arquitectura, se seleccionó el más adecuado y brindando una descripción del funcionamiento y características del mismo. Al aplicar el método, inicialmente se tiene un compendio de los atributos de calidad escogidos, se hace una priorización así como un ordenamiento de los escenarios, y estos a su vez son organizados en un árbol de utilidad. Finalmente con la identificación de los riesgos se toman las decisiones correctas para la corrección o mitigación de estos.

3.2 Métodos de evaluación de la arquitectura.

Un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema, para la búsqueda de conflictos que pueda presentar la arquitectura y de esta manera encontrar más rápidamente sus soluciones. Luego de un estudio de los métodos de evaluación de arquitecturas de software que existen se escogió el Método de Análisis de Desventajas de la Arquitectura (*Architecture Trade-off Analysis Method*, ATAM), como el indicado por las características que presenta y que se ajustan al trabajo que se realiza. El mismo sigue un enfoque orientado hacia la mitigación de riesgos con la ubicación de los atributos de calidad que son de interés y que pueden afectar las decisiones arquitectónicas que sean tomadas.

Método de Análisis de Desventajas de Arquitectura (ATAM).

El método ATAM está inspirado en tres áreas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM[6]. El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas. El propósito

6 El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es el primero que fue ampliamente promulgado y documentado. El método fue originalmente creado para el análisis de la modificabilidad de una arquitectura.

fundamental es: evaluar las consecuencias de las decisiones arquitectónicas en base en los atributos de calidad. Es un método que está enmarcado en puntos de análisis como son, descubrir los riesgos analizando las alternativas que podrían crear problemas en el futuro en algunos atributos de calidad. Descubrir no riesgos para las decisiones que promueven cualidades que ayudan a cumplir los objetivos del negocio. Descubrir puntos de sensibilidad para alternativas donde un ligero cambio hace una diferencia significativa en algunos atributos de calidad. Y de último descubrir las desventajas y decisiones que afectan a más de un atributo de calidad.

Al realizar un análisis de ATAM se conseguirán beneficios como:

- Identificar riesgos en una fase temprana del ciclo de vida.
- Aclarar cuáles son los atributos de calidad más importantes.
- Mejorar la documentación de la arquitectura.
- Documentar las decisiones de la arquitectura base.
- Aumentar la comunicación entre los interesados.

El resultado que se obtiene es una mejora considerable a las arquitecturas, teniendo como finalidad evaluar las consecuencias de las decisiones arquitectónicas a través de los atributos de calidad. ATAM facilita la interacción entre las múltiples partes interesadas, lo que lleva a la identificación de riesgos, las sensibilidades, y desventajas. Para lograr el funcionamiento de ATAM es necesario que exista una arquitectura con un alcance definido y manejable. Entre las precondiciones para su aplicación se encuentran: Los miembros del equipo de revisión analizarán los artefactos arquitectónicos. El arquitecto debe preparar una arquitectura de presentación. Los clientes deben preparar una presentación de los objetivos del negocio. El equipo de evaluación revisará los artefactos de la arquitectura, presentaciones y leerá el material antes para familiarizarse con el dominio.

ATAM presenta una estructura de 4 fases y 9 pasos:

Presentación:

1- Presentar el ATAM. El método describe a las partes interesadas. Los pasos en resumen, las técnicas que serán utilizadas para la obtención y análisis así como las salidas de la evaluación.

2- Presentar las pautas del negocio. El líder del proyecto describe los objetivos que motivan el esfuerzo de desarrollo. Las funciones más importantes del sistema, todas las restricciones técnicas, la mayoría de los *stakeholders* y las guías de la arquitectura.

3- Presentar la arquitectura. El arquitecto presenta un panorama general de la arquitectura. Pueden ser las restricciones técnicas, otros sistemas con los que se debe interactuar y el estilo arquitectónico utilizado para hacer frente a los requisitos de los atributos de calidad. De esta forma el equipo de evaluación comienza el análisis y la captura de todos los riesgos.

Investigación y Análisis:

4- Identificar las propuestas arquitectónicas. El ATAM centraliza el análisis de una arquitectura en el entendimiento de sus propuestas arquitectónicas, en este paso son capturadas por el equipo de evaluación pero no son analizadas.

5- Generar el árbol de utilidad de los atributos de calidad. Este paso es crucial, pues guía el resto del análisis. Sin esta guía, los evaluadores pueden perder su tiempo analizando aspectos de la arquitectura que no son de interés para los *stakeholders*. Identificar, priorizar y refinar los atributos de calidad más importantes, cuyo objetivo es la construcción de un árbol de utilidad.

6- Analizar las propuestas arquitectónicas. El equipo de evaluación identifica los enfoques arquitectónicos desde el punto de vista de los atributos de calidad para identificar los riesgos, no riesgos, puntos sensibles y desventajas.

Pruebas:

7- Lluvia de ideas y priorización de escenarios. Los interesados generan los escenarios apoyándose para facilitar el trabajo, en la realización de una lluvia de ideas, obteniendo un conjunto más amplio de los escenarios. A este conjunto de escenarios se le da prioridad a través de un proceso de votación de la totalidad de los participantes del grupo de interesados. Se agregan los nuevos escenarios al árbol de utilidad.

8- Analizar las propuestas arquitectónicas. Identificar los enfoques de arquitectura impactados por los escenarios generados en el paso anterior. Este paso continúa el análisis iniciado en el paso 6, usando los escenarios nuevos, y luego se documentan.

Presentación de informes:

9- Presentar los resultados. Sobre la base de la información recogida en ATAM como son el documento de propuestas arquitectónicas, el conjunto de escenarios priorizados, el árbol de utilidad, los riesgos descubiertos y no riesgos documentados.

El equipo evaluador presenta las conclusiones a las partes interesadas que se reunieron y escribe un informe detallando esta información junto con las propuestas para las estrategias de mitigación. [47]

3.3 Aplicación del método escogido a la arquitectura.

Con los métodos de evaluación arquitectónicos se considera la concepción de un sistema de software a un alto nivel de abstracción, con base en una visión arquitectónica. Algunos de estos métodos tienen como objetivo la generación de una arquitectura que responda a un conjunto de requisitos iniciales. Otros están dirigidos a evaluar configuraciones arquitectónicas ya existentes para escoger la mejor de acuerdo a un conjunto de requisitos. Todos se ubican en las fases tempranas del proceso de diseño de software. En la presente investigación fue seleccionado el método ATAM, este se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados.

Vinculado a esto se tendrán los atributos de calidad definidos por la ISO/IEC 9126, para una mejora considerable de la aplicación de la evaluación de la arquitectura.

3.3.1 Atributos a medir según ISO/IEC 9126.

Los autores plantean que la especificación de los atributos de calidad haciendo uso de un modelo basado en estándares internacionales ofrece una vista amplia y global de los atributos de calidad, tanto a usuarios como arquitectos del sistema, para efectos de la evaluación. El método contempla las actividades descritas a continuación:

1. Analizar los requerimientos funcionales y no funcionales principales del sistema, para establecer las metas de calidad.
2. Utilizar el modelo de calidad ISO/IEC 9126 adaptado para arquitecturas de software. Algunas métricas pueden definirse con mayor nivel de detalle.
3. Presentar las arquitecturas candidatas iniciales.
4. Construir la tabla comparativa para las arquitecturas candidatas.
5. Establecer prioridades para las subcaracterísticas de calidad tomando en cuenta los requerimientos de calidad del sistema.
6. Analizar los resultados obtenidos y resumidos en la tabla, de acuerdo con las prioridades establecidas.

Los atributos de calidad pertenecen al Método de comparación de arquitecturas basada en el modelo ISO/IEC 9126 adaptado para arquitecturas de software. Estos atributos fueron analizados en el Capítulo 1, Epígrafe 1.2.2. [41]

Funcionabilidad

Idoneidad.

La aplicación puede tener problemas con la capacidad para mantener un grupo de tareas que han sido solicitadas por los usuarios, además de objetivos especificados.

Precisión.

La aplicación tiene las capacidades que son requeridas para que se puedan obtener resultados correctos con buena exactitud y a la vez que sean los convenidos.

Seguridad.

Buena capacidad de la aplicación para la protección de la información y de datos.

Las distintas funcionalidades se van mostrando en dependencia del usuario que esté autenticado y los permisos que el mismo tenga.

Interoperabilidad.

La aplicación tiene una mediana interacción con otros sistemas que se especifiquen.

Conformidad con la funcionalidad.

Se encuentra regido por leyes, normas y regulaciones en cuanto sus funcionalidades.

Confiabilidad

Madurez.

La aplicación está estructurada con una arquitectura que evita que al tener un fallo en el sistema el mismo no colapse totalmente.

Tolerancia ante fallos.

La aplicación debe de estar estructurada con una arquitectura adecuada para el mantenimiento de un correcto nivel de ejecución ante cualquier fallo de software.

Recuperabilidad.

La aplicación debe presentar las condiciones necesarias para recuperar sus capacidades luego de un fallo, así como de los datos que contenga la misma.

Conformidad con la confiabilidad.

Aplicación apegada a las reglas de confiabilidad.

Usabilidad

Comprensibilidad.

La aplicación debe presentar características que sean útiles para que el usuario pueda realizar sus tareas de una manera más entendible.

Cognoscibilidad.

Aplicación con características que facilitan al usuario aprender rápido a trabajar con la misma.

Operabilidad.

Aplicación con características que permiten operar y controlar el sistema.

Atracción.

La aplicación terminará siendo atractiva y amigable para el usuario.

Conformidad con la usabilidad.

Aplicación apegada a las normas o convenciones de usabilidad.

Eficiencia

Rendimiento.

La aplicación propiciará buenos tiempos de respuesta y procesamiento de la información.

Utilización de recursos.

Aplicación que utilizará cantidad y tipos apropiados de recursos para la realización de tareas.

Conformidad de la eficiencia.

Aplicación apegada a las normas o convenciones de eficiencia.

Mantenibilidad

Diagnosticabilidad.

Aplicación sin muchas capacidades para encontrar dentro del mismo, deficiencias o causas de fallos.

Flexibilidad.

La aplicación permite que se le realicen diferentes modificaciones que sean necesarias.

Estabilidad.

La aplicación puede minimizar efectos ante cambios que se realicen.

Conformidad de la mantenibilidad.

Aplicación apegada a las normas o convenciones de mantenibilidad.

Portabilidad

Adaptabilidad.

La aplicación presenta capacidades para la adaptación a otros ambientes, tales como Windows y GNU/Linux.

Instalabilidad.

Aplicación con características adecuadas que permiten ser instalada en el ambiente que se necesite.

Coexistencia.

Tiene las capacidades como para coexistir con otro software y compartir recursos comunes.

Remplazabilidad.

Puede reemplazar a otro software que requiera los mismos recursos y ambiente que el.

Conformidad con la portabilidad.

Aplicación apegada a las normas o convenciones de portabilidad.

3.3.2 Priorización y ordenamiento de escenarios.

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste. Consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas, configuración, etc. El contexto describe qué sucede en el sistema al momento del estímulo. La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado. Los escenarios proveen un vehículo que permite concretar y entender atributos de calidad. [41]

Se debe ordenar los escenarios ya que de esta manera los arquitectos cuentan con más orientación para tomar decisiones arquitectónicas, y los *stakeholders* pueden estar más conscientes de lo que esperan del sistema, y obtener una idea sobre en qué medida va a ser satisfecho. Una vez que se ha decidido incluir en el árbol de utilidad los escenarios más importantes, se procede a asignar un orden a los escenarios de calidad de un sistema utilizando los criterios siguientes:

1. Evaluar el riesgo de no considerar el escenario. Se deben responder las preguntas: ¿qué pasa si este escenario no se cumple?, ¿cuántas personas se ven afectadas?, ¿es posible compensar el no responder a este escenario?
2. Evaluar el esfuerzo necesario para lograr cumplir con el escenario. En este caso se determina que cambios o integraciones de nuevos componentes se deben realizar para satisfacer el escenario. Los escenarios más difíciles son aquellos en los que se debe consumir más tiempo de análisis y el resto debe ser guardado como parte del registro. [48]

Escenario 1: La aplicación propiciará buenos tiempos de respuesta y procesamiento de la información.
Atributos: Eficiencia - Rendimiento.
Estímulo: Guardar cambios realizados al ciclo de pruebas.
Respuesta: Muestra una ventana de confirmación para guardar los cambios realizados.

Escenario 2: La aplicación permite que se le realicen diferentes modificaciones que sean necesarias.
Atributos: Mantenibilidad - Flexibilidad.
Estímulo: Realizar cambios en la estructura del plan de pruebas por parte del programador.
Respuesta: Los cambios son realizados y aceptados por la aplicación.

Escenario 3: La aplicación puede minimizar efectos ante cambios que se realicen.
Atributos: Mantenibilidad - Estabilidad.
Estímulo: Realizar cambios en la estructura de las iteraciones por parte del programador.
Respuesta: Muestra la interfaz para realizar cambios a las iteraciones.

Escenario 4: La aplicación puede tener problemas con la capacidad para mantener un grupo de tareas que han sido solicitadas por los usuarios, además de objetivos especificados.
Atributos: Funcionabilidad - Idoneidad.
Estímulo: Varias tareas son solicitadas por el usuario que interactúa en la aplicación.
Respuesta: Procesamiento de las tareas en el orden en que son solicitadas por el usuario.

Escenario 5: Buena capacidad de la aplicación para la protección de la información y de datos.
Atributos: Funcionabilidad - Seguridad.
Estímulo: Guardar cambios realizados.
Respuesta: Los cambios son guardados en la base de datos.

Escenario 6: Las distintas funcionalidades se van mostrando en dependencia del usuario que esté autenticado y los permisos que el mismo tenga.
Atributos: Funcionabilidad - Seguridad.
Estímulo: Realizar cambios en el plan de pruebas.
Respuesta: Los cambios son guardados en la base de datos.

Escenario 7: Autenticación de los usuarios para entrar en la aplicación.
Atributos: Funcionabilidad - Seguridad.
Estímulo: Accede a la aplicación el Administrador de calidad.
Respuesta: Muestra la aplicación con las funcionalidades que le corresponden al Administrador de la Calidad.

Escenario 8: La aplicación debe de estar estructurada con una arquitectura adecuada para el mantenimiento de un correcto nivel de ejecución ante cualquier fallo de software.
Atributos: Confiabilidad – Tolerancia ante fallos.
Estímulo: Crear un nuevo plan de pruebas.
Respuesta: Muestra la interfaz para la creación de un nuevo plan de pruebas.

Escenario 9: La aplicación debe presentar las condiciones necesarias para recuperar sus capacidades luego de un fallo, así como de datos.
Atributos: Confiabilidad – Recuperabilidad.

Estímulo: Visualizar plan de pruebas.

Respuesta: Muestra la interfaz para la visualización de un plan de pruebas.

Escenario 10: La aplicación debe presentar características que sean útiles para que el usuario pueda realizar sus tareas de una manera más entendible.

Atributos: Usabilidad – Comprensibilidad.

Estímulo: Adicionar iteración.

Respuesta: Muestra la interfaz para adicionar una iteración.

Escenario 11: Aplicación con características que permiten operar y controlar el sistema.

Atributos: Usabilidad – Operabilidad.

Estímulo: Realizar cambios al ciclo de pruebas.

Respuesta: Muestra la interfaz para realizar cambios al ciclo de pruebas.

Escenario 12: La aplicación presenta capacidades para la adaptación a otros ambientes, tales como Windows y GNU/Linux.

Atributos: Portabilidad – Adaptabilidad.

Estímulo: Se trabaja con la aplicación en el sistema operativo Windows.

Respuesta: La aplicación trabaja sobre el sistema operativo Windows.

3.3.3 Árbol de utilidad.

Un árbol de utilidad es un vehículo para el manejo de los requisitos de atributos de calidad específicos. Selecciona los objetivos de calidad para ser los nodos del alto nivel (por lo general, el rendimiento, modificabilidad, seguridad y disponibilidad). La salida de un árbol de utilidad es la caracterización y priorización de los requisitos de atributos de calidad.

Se construye el árbol de utilidad con las salidas de la priorización y ordenamiento de requisitos. La prioridad del escenario define en este método como un par (X,Y) en el cual X define el esfuerzo de satisfacer el escenario, y la Y indica los riesgos que se corren al excluirlos del árbol de utilidad. Los posibles valores para X e Y son A (Alto), B (Bajo) y M (Medio). El árbol de utilidad generado se toma como un plan para el resto de la evaluación que realiza el método. Indica además al equipo evaluador dónde ocupar su tiempo y dónde probar aproximaciones y riesgos arquitectónicos. [48]

Característica	Subcaracterística	Prioridad	Escenario
Eficiencia	Rendimiento	(M,A)	La aplicación propiciará buenos tiempos de

			respuesta y procesamiento de la información.
Mantenibilidad	Flexibilidad	(A,A)	La aplicación permite que se le realicen diferentes modificaciones que sean necesarias.
	Estabilidad	(M,A)	La aplicación puede minimizar efectos ante cambios que se realicen.
Funcionabilidad	Idoneidad	(A,A)	La aplicación puede tener problemas con la capacidad para mantener un grupo de tareas que han sido solicitadas por los usuarios, además de objetivos especificados.
	Seguridad	(M,M)	Buena capacidad de la aplicación para la protección de la información y de datos.
	Seguridad	(B,M)	Las distintas funcionalidades se van mostrando en dependencia del usuario que este autenticado y los permisos que el mismo tenga.
	Seguridad	(B,M)	Autenticación de los usuarios para entrar en la aplicación.
Confiabilidad	Tolerancia ante fallos	(M,A)	La aplicación debe de estar estructurada con una arquitectura adecuada para el mantenimiento de un correcto nivel de ejecución ante cualquier fallo de software.
	Recuperabilidad	(M,B)	La aplicación debe presentar las condiciones necesarias para recuperar sus capacidades luego de un fallo, así como de datos.
Usabilidad	Comprensibilidad	(M,B)	La aplicación debe presentar características que sean útiles para que el usuario pueda realizar sus tareas de una manera más entendible.
	Operabilidad	(B,M)	Aplicación con características que permiten operar y controlar el sistema.

Portabilidad	Adaptabilidad	(M,A)	La aplicación presenta capacidades para la adaptación a otros ambientes, tales como Windows y GNU/Linux.

Tabla 1. Árbol de Utilidad.

3.3.4 Riesgos identificados y toma de decisiones en función de estos.

Luego de la aplicación del método ATAM a la arquitectura de la aplicación, se identificaron como resultado los riesgos que se especifican en la siguiente tabla. Tomándolos como base, se realiza un análisis de la arquitectura propuesta buscando las características que puedan mitigar o dar solución a cada uno de ellos.

Riesgo Identificado	Decisión
Malos tiempos de respuesta y procesamiento de la información.	Con un Bajo Acoplamiento se soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad. Se genera un bajo acoplamiento con la ventaja de una gran funcionalidad al soportar mayor capacidad de reutilización. Una clase que tenga la propiedad de ser muy cohesiva puede dedicarse a un propósito bien específico.
Problemas con la capacidad para mantener un grupo de tareas que han sido solicitadas por los usuarios, además de objetivos especificados.	El patrón Presentación Abstracción Control y la ventaja de ser multitarea, proporciona que los agentes pueden distribuirse fácilmente en diferentes hilos, procesos, o máquinas.
Estructura incorrecta de la arquitectura para el mantenimiento de un alto nivel de ejecución ante cualquier fallo de software.	El patrón arquitectónico de pizarra funciona como almacenamiento central de datos, que guarda también los elementos del espacio de solución y el control de datos.

Los efectos ante cambios que se realizan, no son minimizados rápidamente.	El estilo en capas que se aplica admite tantas optimizaciones y refinamientos como sean necesarios. En el patrón Presentación Abstracción Control, los cambios dentro de los componentes presentación o abstracción de un agente no afectan a otros agentes en el sistema. Permitiendo que se puedan modificar individualmente el modelo de datos que está debajo de un agente o cambiar su interfaz de usuario. El patrón de diseño bajo acoplamiento favorece a tener un sistema más robusto y de fácil mantenimiento.
Aplicación con una buena capacidad para la protección de la información y de los datos.	Al contar con dos roles en el sistema, Administrador de la calidad y Usuarios, se crean dos tipos de accesos separados, cada uno con permisos diferentes, teniendo acceso según sus categorías. Cumpliendo así estrictamente con normas de seguridad.

Tabla 2. Riesgos y decisiones.

En el presente capítulo se mostró como los atributos de calidad son elementos claves a tener en cuenta para el desarrollo de una arquitectura sólida, por lo cual conocer sus prioridades es un elemento fundamental que permite definir los conflictos que generan las decisiones tomadas por los arquitectos para alcanzarlos. La aplicación de la norma ISO/IEC 9126 garantizó elevar la calidad de las pruebas realizadas. Al evaluar la arquitectura propuesta mediante el método ATAM, se pudo observar como la arquitectura satisface las necesidades del sistema a desarrollar, al detectarse poca cantidad de riesgos. Dichos riesgos no son tan graves para el sistema pues se cuentan con decisiones arquitectónicas para ayudar a mitigar los mismos.

Conclusiones

La investigación desarrollada en la realización de este trabajo de diploma y el análisis de la documentación estudiada, permitió el cumplimiento de los objetivos trazados. La utilización de un proceso de pruebas bien definido aumenta la calidad y el prestigio de Nova. Teniendo en cuenta las características de la distribución GNU/Linux y tomando como guía fundamentalmente las características de calidad del estándar ISO/IEC 9126, se definieron los requisitos de calidad con los que deben cumplir los productos resultantes del proyecto. Con la elaboración del proceso de pruebas automatizado se obtuvo una mayor organización y planificación a la hora de realizar las pruebas. A partir de la definición de métricas externas que evalúan el comportamiento de los indicadores de calidad, se aplicaron los mismos en el proceso de pruebas automatizadas de la distribución cubana de software libre GNU/Linux Nova, logrando que se pueda evaluar cuantitativamente la calidad a través de la comprobación del cumplimiento de estos indicadores de calidad. Se obtuvieron los requerimientos arquitectónicos, los patrones de arquitectura así como las tecnologías y herramientas para dar soporte al desarrollo del sistema, quedando definida la línea base de la arquitectura necesaria para el sistema. Haciendo uso del método de evaluación ATAM, se obtuvieron resultados satisfactorios en la evaluación de la arquitectura del módulo.

Recomendaciones.

Luego de analizados los resultados obtenidos en el presente trabajo, se realizan las recomendaciones siguientes:

- Continuar refinando los requisitos de calidad de la distribución GNU/Linux Nova.
- Continuar profundizando el estudio de las herramientas que realizan pruebas automatizadas, en aras de aumentar la exigencia de los requisitos de calidad, referentes a la seguridad, para la inclusión de nuevas funcionalidades en posteriores versiones.
- Realizar nuevas investigaciones sobre los procesos de pruebas de las distribuciones GNU/Linux para continuar enriqueciendo aún más el proceso propuesto.
- Realizar la implementación del módulo de pruebas de la suite de ISW a partir de la arquitectura definida.

Referencia Bibliográfica.

1. Delgado Morejón, Yusdel Pablo. Indicadores de calidad en el subproceso de pruebas automáticas de la distribución GNU-Linux Nova. Universidad de las Ciencias Informáticas. Cuba. 2012.
2. Santos Hernández, Vismar. La Industria del software. Estudio a nivel Global y América Latina. Consultado en noviembre 2012. Disponible en [http://www.eumed.net/cursecon/ecolat/la/09/vsh.htm]
3. Tanenbaum, Andrew S. Sistemas Operativos Modernos, 2009.
4. Pressman, Roger S. Ingeniería de Software. Un enfoque práctico. Sexta Edición. 2005.
5. materias.fi.uba.ar/7548/Pruebas-Intro, it-Mentor. “capacitación y guía para el desarrollo de software”, 2010. Consultado en diciembre 2012. Disponible en [http://www.materias.fi.uba.ar/7548/Pruebas-Intro.pdf]
6. indalog.ual.es/mtorres/LP/Prueba.pdf, Técnicas de prueba, 2010. Consultado en enero 2013. Disponible en [http://www.indalog.ual.es/mtorres/LP/Prueba.pdf]
7. Abad Londoño, Jorge Hernán. Ingeniería de Software: Tipos de Pruebas de Software, 2005. Consultado en enero 2013. Disponible en [http://www.ing-sw.blogspot.com/2005/04/tipos-de-pruebas-de-software.html]
8. Ruiz Tenorio, Roberto. Las Pruebas de Software y su Importancia en las Organizaciones. Universidad Veracruzana. Veracruz. 2010. Consultado en enero 2013. Disponible en [http://www.cdigital.uv.mx/bitstream/123456789/28540/1/Ruiz%20Tenorio.pdfl]
9. Cervantes Humberto. Arquitectura de Software y CMMI, mayo-julio 2012. Consultado en enero 2013. Disponible en [http://www.sg.com.mx/revista/arquitectura-software-y-cmmi]
10. Calidad y Software, Testing (I) - Las pruebas según el CMMi, abril 2012. Consultado en enero 2013. Disponible en [http://www.calidadysoftware.blogspot.com/2012/04/testing-i-las-pruebas-segun-el-cmmi.html]
11. NC ISO/IEC 9126-1:2005 Consultado en: enero 2013. Disponible en [http://calisoft.uci.cu/tmp/documentos/normas/iso/NC-ISO-IEC%209126-1.pdf]
12. ISO/IEC 9126-2:2003. Consultado en: enero 2013. Disponible en [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22750]
13. ISO/IEC 9126-3:2003. Consultado en: enero 2013. Disponible en [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22891]
14. ISO/IEC 9126-4:2004. Consultado en: enero 2013 Disponible en

[http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39752]

15. ISO/IEC 14598-1:1999. Consultado en: enero 2013. Disponible en [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=24902]
16. ISO/IEC 14598-2:2000. Consultado en: enero 2013. Disponible en [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=24903]
17. ISO/IEC 14598-3:2000. Consultado en: enero 2013. Disponible en [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=24904]
18. ISO/IEC 14598-4:1999. Consultado en: enero 2013. Disponible en [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=24905]
19. ISO/IEC 14598-5:1998. Consultado en: enero 2013. Disponible en [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=24906]
20. ISO/IEC 14598-6:2001. Consultado en: enero 2013. Disponible en [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=24907]
21. ISO/IEC 25000:2005 Consultado en: enero 2013. Disponible en [http://www.iso.org/iso/catalogue_detail.htm?csnumber=35683]
22. TestWorks para UNIX, 2004. Consultado en septiembre 2012. Disponible en [http://www.soft.com/Products/index.html]
23. Esmite Ignacio, Farías Mauricio, Farías Nicolás, Pérez Beatriz. Automatización y Gestión de las Pruebas Funcionales usando Herramientas Open Source, 2010. Consultado en diciembre 2012. Disponible en [http:// www.ces.com.uy/documentos/AutomCACIC07.pdf]
24. Software TestWorks Test Tool Suite, TestWorks for UNIX Version 3.1,2004. Consultado en julio 2012. Disponible en [http://www.soft.com/Products/index.html]
25. OpenSourceTesting, Bugzilla Testopia, junio 2012. Consultado en septiembre 2012. Disponible en [http://www.mozilla.org/projects/testopia/]
26. OpenSourceTesting, Escenario de la prueba incremental, junio 2011. Consultado en septiembre 2012. Disponible en [http://sourceforge.net/projects/istt/]
27. OpenSourceTesting, MTS: Multi-Tester, 2011. Consultado en septiembre 2012. Disponible en [http://www.cubewano.org/]
28. Jorge Emmanuel, OpenSourceTesting, RTMR Requisitos y repositorio de gestión de pruebas, noviembre 2011. Consultado en septiembre 2012. Disponible en [http://rtmr.net/]
29. OpenSourceTesting, RTH-Turbo, junio 2011. Consultado en septiembre 2012. Disponible en [http://code.google.com/p/rth-turbo/]

30. OpenSourceTesting, Salomé-TMF, diciembre 2007. Consultado en septiembre 2012. Disponible en [https://wiki.ow2.org/salome-tmf/]
31. OpenSourceTesting, TestAutomation, diciembre 2010. Consultado en septiembre 2012. Disponible en [http://sourceforge.net/projects/testautomation/]
32. OpenSourceTesting, Testitool, marzo 2010. Consultado en septiembre 2012. Disponible en [http://www.majordojo.com//testitool/]
33. Phoronix Test Suite, Sitio Oficial, 2012. Consultado en octubre 2012. Disponible en [http://phoronix-test-suite.com/]
34. Monteagudo Suárez, Lorena. Proceso de pruebas de la distribución GNU/Linux Nova. Universidad de las Ciencias Informáticas. Cuba. 2011.
35. Ing. Albo Castro Mónica Ma, Ing. López Rangel Rosa Maria, Ing. Morejon Delgado Yusdel P. Indicadores y Metricas de Calidad en el Proceso de Pruebas de la distribucion GNU/Linux Nova. Universidad de las Ciencias Informáticas. Cuba. 2012.
36. Lic. Gimson, Loraine. Metodologías ágiles y desarrollo basado en conocimiento. Junio de 2012. Consultado en febrero 2013. Disponible en [http://sedici.unlp.edu.ar/bitstream/handle/10915/24942/Documento_completo__.pdf?sequence=1]
37. Visual Paradigm para UML. Consultado en febrero 2013. Disponible en [http://www.software.com.ar/visual-paradigm-para-uml.html]
38. Popkin Software and Systems. Modelado de Sistemas com UML. Consultado en febrero 2013. Disponible en [http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/doc-modelado-sistemas-uml.pdf]
39. Visual Paradigm para UML. Consultado en febrero 2013. Disponible en [http://www.visual-paradigm.com/]
40. Qt Designer Manual. Consultado en febrero 2013. Disponible en [http://doc.qt.digia.com/stable/designer-manual.html]
41. Camacho Erika, Cardeso Favio, Nuñez Gabriel. Arquitecturas de Software. Guía de estudio. Abril 2004. Consultado en abril 2013. Disponible en [http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf]
42. Reynoso Carlos, Kiccillof Nicolás. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. Versión 1.0. Universidad de Buenos Aires. Argentina. Marzo 2004. Consultado en abril 2013. Disponible en [http://carlosreynoso.com.ar/archivos/arquitectura/Estilos.PDF]
43. Sandra Almeira Adriana, Perez Cavenago Vanina. Arquitectura de Software: Estilos y Patrones.

Universidad Nacional De La Patagonia San Juan Bosco. Argentina. Marzo 2007. Consultado en abril 2013. Disponible en [http://www.roa.unp.edu.ar:8080/graduate/bitstream/123456789/203/1/Tesina%20Arquitectura%20de%20Soft.pdf]

44. Larman Craig, UML y Patrones Introducción al análisis y diseño orientado a objetos. Primera Edición. 1999.
45. Geek the Planet, Patrones GoF. Mayo 2011. Consultado en abril 2013. Disponible en [http://geektheplanet.net/5462/patrones-gof.xhtml]
46. Kruchten P, Architectural Blueprints The “4+1” View Model of Software Architecture. IEEE Software. Noviembre 1995. Consultado en abril 2013. Disponible en [http://www.computer.org/csdl/mags/so/1995/06/s6042-abs.html]
47. Díaz Verdecia Osvaldo, Quevedo Campins Virgen Damaris. Una guía práctica de Arquitectura de Software. Universidad de las Ciencias Informáticas. Cuba. Mayo 2009.
48. González Aleksander, Mijares Marizé, Mendoza Luis E., Grimán, Pérez Anna María. Método de Evaluación de Arquitecturas de Software Basadas en Componentes (MECAB). Universidad Simón Bolívar. Venezuela. Consultado en abril 2013. Disponible en [http://www.lisi.usb.ve/publicaciones/03%20evaluacion/evaluacion_09.pdf]

Bibliografía.

Calidad y Software, Testing (I) - Las pruebas según el CMMi, abril 2012. Consultado en enero 2013. Disponible en [http://www.calidadysoftware.blogspot.com/2012/04/testing-i-las-pruebas-segun-el-cmmi.html]

Camacho Erika, Cardeso Favio, Nuñez Gabriel. Arquitecturas de Software. Guía de estudio. Abril 2004. Consultado en abril 2013. Disponible en [http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf]

Casanovas Joseph, Usabilidad y Arquitectura del Software, Julio de 2004. Consultado en abril 2013. Disponible en [http://www.alzado.org/articulo.php?id_art=355]

Delgado Morejón, Yusdel Pablo. Indicadores de calidad en el subproceso de pruebas automáticas de la distribución GNU-Linux Nova. Universidad de las Ciencias Informáticas. Cuba. 2012.

Díaz Verdecia Osvaldo, Quevedo Campins Virgen Damaris. Una guía práctica de Arquitectura de Software. Universidad de las Ciencias Informáticas. Cuba. 2009.

Fernández del Monte, Yusleydi. Metodología Nova OpenUp para el desarrollo de distribuciones GNU/Linux. Versión 2. 2012.

Hernández León Rolando Alfredo, Coello González Sayda. El Proceso de Investigación Científica. Ciudad de La Habana. Editorial Universitaria. 2011.

Ing. Albo Castro Mónica Ma, Ing. López Rangel Rosa Maria, Ing. Morejon Delgado Yusdel P. Indicadores y Metricas de Calidad en el Proceso de Pruebas de la distribucion GNU/Linux Nova. Universidad de las Ciencias Informáticas. Cuba. 2012.

Kruchten P, Architectural Blueprints The “4+1” View Model of Software Architecture. IEEE Software. Noviembre 1995. Consultado en abril 2013. Disponible en [http://www.computer.org/csdl/mags/so/1995/06/s6042-abs.html]

Larman Craig, UML y Patrones Introducción al análisis y diseño orientado a objetos. Primera Edición. 1999.

Osorio Castro Eneysi , Torres Camilo Asdrúbal, Propuesta de arquitectura para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Universidad de las Ciencias Informáticas. Cuba. 2010.

Pressman, Roger S. Ingeniería de Software. Un enfoque práctico. Sexta Edición. 2005.

Ruiz Tenorio, Roberto. Las Pruebas de Software y su Importancia en las Organizaciones. Universidad Veracruzana. Veracruz. 2010. Consultado en enero 2013. Disponible en [http://www.cdigital.uv.mx/bitstream/123456789/28540/1/Ruiz%20Tenorio.pdfl]

Software TestWorks Test Tool Suite, TestWorks for UNIX Version 3.1,2004. Consultado en julio 2012. Disponible en [http://www.soft.com/Products/index.html]

OpenSourceTesting, junio 2012. Consultado en septiembre 2012. Disponible en [http://www.mozilla.org/projects//]

Sandra Almeira Adriana, Perez Cavenago Vanina. Arquitectura de Software: Estilos y Patrones. Universidad Nacional De La Patagonia San Juan Bosco. Argentina. Marzo 2007. Consultado en abril 2013. Disponible en [http://www.roa.unp.edu.ar:8080/graduate/bitstream/123456789/203/1/Tesina%20Arquitectura%20de%20Sft.pdf]

Glosario de términos.

Nota:

Glosario son términos con significado.

Las siglas son ACRÓNIMOS

Artefactos: Un artefacto es un producto tangible resultante del proceso de desarrollo de software.

Árbol de Utilidad: esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno.

Artefacto: Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.

Calidad de software: Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con características implícitas que se espera de todo software desarrollado profesionalmente.

Casos de uso: Fragmentos de funcionalidad del sistema**.**

CMMI: Capability Maturity Model Integration (CMMI). Modelo para la mejora de procesos que proporciona a las organizaciones los elementos esenciales para procesos eficaces. Las mejores prácticas CMMI se publican en los documentos llamados modelos. En la actualidad hay dos áreas de interés cubiertas por los modelos de CMMI: Desarrollo y Adquisición. Independientemente de la constelación\modelo que opta una organización, las prácticas CMMI deben adaptarse a cada organización en función de sus objetivos de negocio.

Frameworks: Es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

HTML: HyperText Markup Language (Lenguaje de Marcas de Hipertexto). Lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

ISO, IEEE: Son los organismos encargados de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación. Su función principal es la de buscar la estandarización de normas de productos y seguridad para las empresas u organizaciones a nivel internacional.

Licencia GNU GPL: Esta licencia GNU General Public License (GPL) se aplica en la mayoría de los programas realizado por la Fundación del software libre (Free Software Foundation FSF) y en

cualquier otro programa en los que los autores quieran aplicarla. También, muchos otros programas están cubiertos por la GNU Lesser General Public License (LGPL) e igualmente puede usarla para cubrir sus programas. Está diseñada para garantizar su libertad de compartir y modificar el software libre para garantizar la libertad de sus usuarios.

Servidores: En informática, un servidor es una computadora que, formando parte de una red, provee servicios a otras computadoras denominadas clientes.

Solución: es la propuesta de la solución definitiva al problema, es analizada con respecto a su: posible impacto, viabilidad y conveniencia.

SEI: Software Engineering Institute (SEI). Instituto federal estadounidense de investigación y desarrollo, fundado por el Congreso de los Estados Unidos en 1984 para desarrollar modelos de evaluación y mejora en el desarrollo de software, que dieran respuesta a los problemas que generaba al ejército estadounidense la programación e integración de los sub-sistemas de software en la construcción de complejos sistemas militares. Financiado por el Departamento de Defensa de los Estados Unidos y administrado por la Universidad Carnegie Mellon.

Stakeholders: Cualquier persona o entidad que es afectada por las actividades de una organización o empresa; por ejemplo, los trabajadores de esa organización, sus accionistas, las asociaciones de vecinos, sindicatos, organizaciones civiles y gubernamentales, etc.

Open Source: Cualidad de algunos software de incluir el código fuente en la distribución del programa. En general se usa para referirse al software libre.

Plugin: Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

Proveedor: es la empresa u organismo que proporciona los servicios solicitados por el cliente.

Releases: Nueva versión de una aplicación informática.

Rol: Define el comportamiento y responsabilidad de un individuo o grupo de individuos en un sistema.

RF: Requisito Funcional.

RNF: Requisito No Funcional.

UML: Unified Modeling Language. Lenguaje Unificado de Modelado.

Usuarios: las personas que utilizan el servicio.

XML: Extensible Markup Language (Lenguaje de Marcas). Metalenguaje extensible de etiquetas, no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

Anexos.

Especificación de Requisito Adicionar Plan de Pruebas

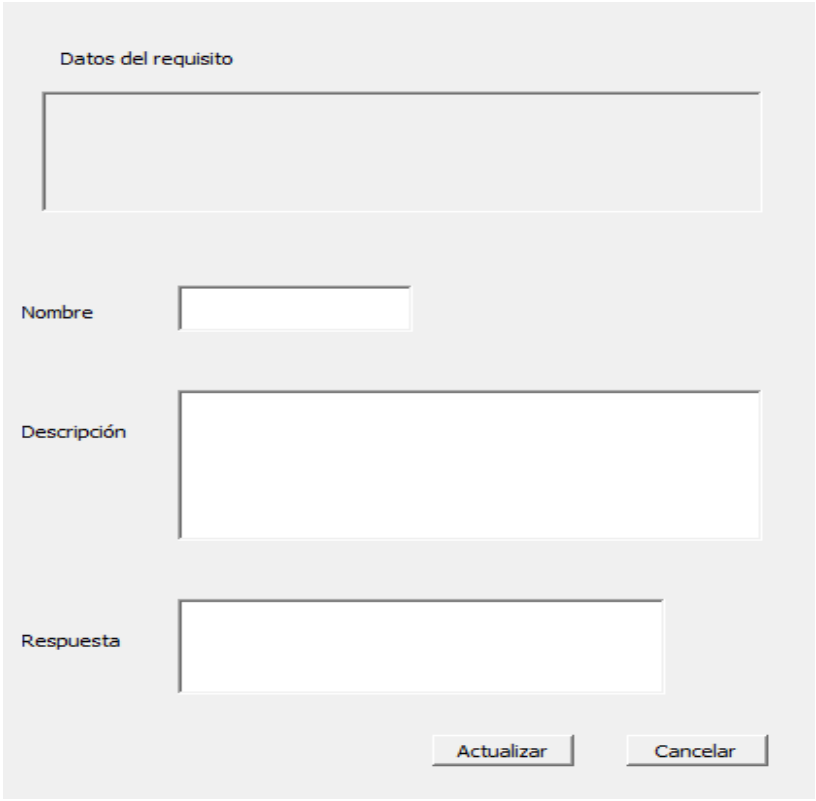
Descripción textual del requisito

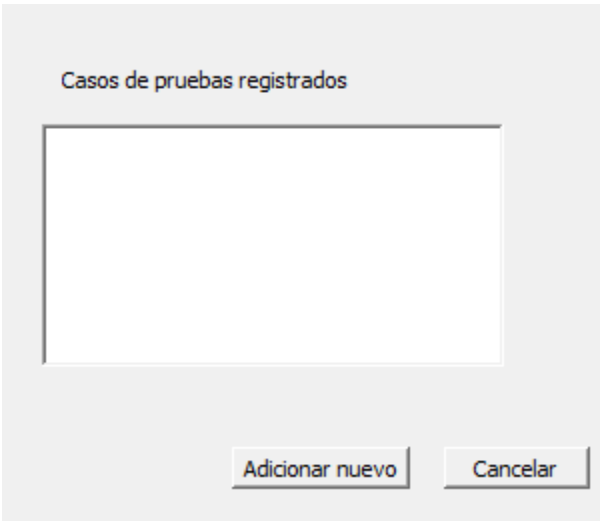
Precondiciones	Existe un proyecto para un producto o componente.
Flujo de eventos	
Flujo básico Adicionar Plan	
1.	Se accede a la sección del plan de pruebas.
2.	Mostrará una lista de todos los planes de pruebas existentes con la opción para crear uno nuevo.
3.	Mostrará una nueva ventana similar a un asistente de instalación, donde se listan los productos y/o componentes registrados en el sistema.
4.	Se selecciona el producto o componente para el que se planificará el proceso de pruebas y se pasa al siguiente paso.
5.	Mostrará en la ventana una nueva vista, que lista los roles que incluye la metodología asociada a ese producto y/o componente.
6.	Se seleccionan los roles que intervendrán en el proceso de pruebas.
7.	Muestra en la misma vista al lado de los roles seleccionados la opción para introducir la cantidad y la lista de responsabilidades asociadas al rol en una lista.
8.	Se introducen la cantidad de personas, por cada rol, que participaran en el proceso de pruebas.
9.	Se selecciona para cada rol las responsabilidades específicas que desempeñaran en el proceso de pruebas y se pasa al siguiente paso.
10.	Mostrará en la ventana una nueva vista, para introducir los datos de los recursos a utilizar en el proceso de pruebas: nombre, características y cantidad, y debajo se muestra la opción que permite adicionar un nuevo recurso.
11.	Se introduce el nombre del primer recurso, ejemplo: PC Clientes, Servidor de Datos, Servidor Web, etc.
12.	Se introducen las características del primer recurso, ejemplo: 256 RAM, HDD 160Gb, HP Deskjet 2550 J510, etc.
13.	Se introduce la cantidad de equipos de ese recurso que debe haber disponible para el proceso de pruebas.
14.	Si hay más recursos se selecciona la opción de adicionar recurso y se introducen los datos al igual que para el primer recurso y cuando termine con todos los recursos pasa al siguiente paso.
15.	Mostrará en la ventana una nueva vista, con una lista de todos los requisitos del producto o

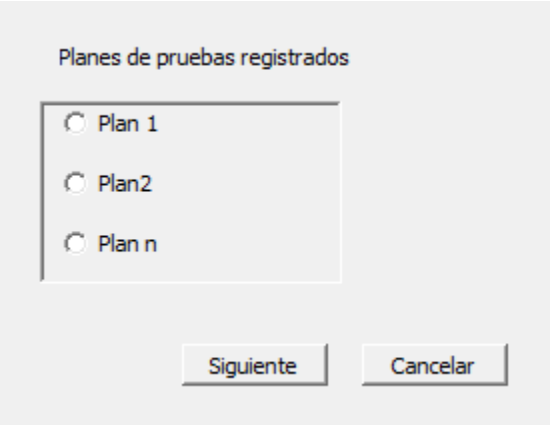
	componente seleccionado inicialmente.	
16.	Se selecciona los requisitos que se estarán analizando durante el proceso de pruebas y se termina la primera versión del plan de pruebas.	
Pos-condiciones		
1.	Plan de pruebas creado.	
Flujos alternativos		
Flujo alternativo 13.a No hay más recursos		
1.	Si no hay más recursos se pasa directamente al paso 15.	
Pos-condiciones		
1.	Plan de pruebas creado.	
Validaciones		
1.	Al menos un rol seleccionado, con al menos 1 persona y una responsabilidad.	
2.	Al menos un recurso PC Cliente, con sus respectivos datos.	
3.	Al menos un requisito seleccionado.	
Conceptos	Componente y/o producto	Visible en Interfaz: Garantizar que se visualicen los componentes y/o productos registrados en el sistema. Componente visual: lista de radiobuttoms.
	Roles	Visible en Interfaz: Garantizar que se visualicen los roles de la metodología asociada al producto y/o componente seleccionado. Componente visual: lista de chekbox.
	Responsabilidades de los Roles	Visible en Interfaz: Garantizar que se visualicen las responsabilidades asociadas al rol seleccionado. Componente visual: lista de chekbox.
	Requisitos	Visible en Interfaz: Garantizar que se visualicen los requisitos asociados al producto y/o componente seleccionado. Componente visual: lista de chekbox.
	Roles Cantidad	Cantidad de personas que desempeñaran los roles que participarán en la prueba. Entera: int Debe de entrar alguna cantidad para cada rol, valor mínimo 1.

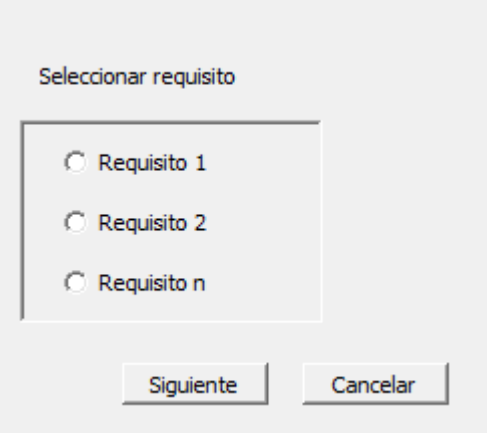
	Recurso Nombre	El nombre del recurso a emplear. Ejemplo: PC Cliente, Servidor de Datos, Impresora, etc. Texto: string Debe existir al menos un recurso que represente las PC donde se ejecutaran las pruebas al software.
	Recursos características	Las características del recurso como el modelo, en el caso de las PC y los Servidores la cantidad de memoria RAM, disco duro, etc. Texto: string Debe estar asociado al nombre del recurso, se sugiere que se muestre al lado.
	Recursos Cantidad	Cantidad del recursos descrito que se utilizará en el proceso. Entero: int Debe estar asociado al nombre del recurso, se sugiere que se muestre al lado.
	Requisitos Probar	Los requerimientos que serán probados durante el proceso de pruebas planificados. Se mostrará un listado con los requisitos del sistema, al lado de cada uno se tendrá la opción con un checkbox de escoger los que se estarán analizando durante las pruebas, en cada etapa. Debe seleccionarse al menos un requisito en la primera versión del documento.
	Cronograma	Actividades del proceso de pruebas, incluidos los ciclos e iteraciones.
Requisitos especiales	*Son los requisitos no funcionales específicos para el requisito. Por ejemplo, estándares de intercambio de información.*	
Asuntos pendientes	*Posibles mejoras al requisito.*	

Prototipo elemental de interfaz gráfica de usuario









Especificación de Requisito Modificar caso de prueba

Descripción textual del requisito

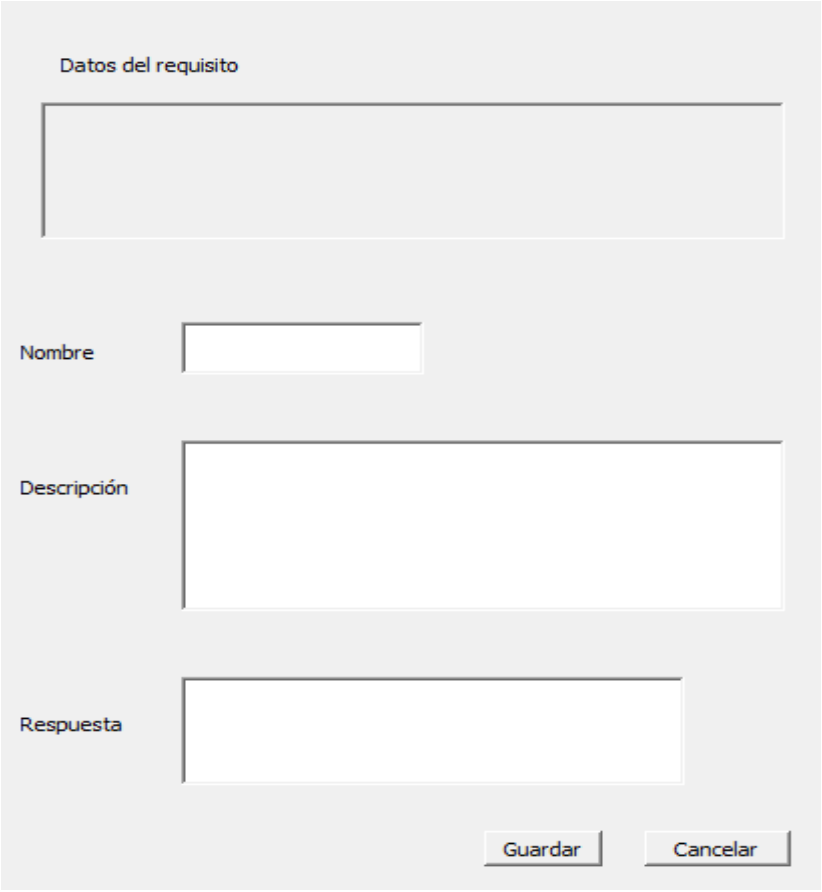
Precondiciones	Existe al menos un plan de pruebas y un caso de prueba previamente creado.
Flujo de eventos	
Flujo básico Modificar CP	
1.	Se accede a la sección de diseño de las pruebas.
2.	Mostrará una lista de todos los casos de pruebas diseñados hasta el momento cada uno con la opción de modificar.
3.	Seleccionar el caso de prueba al que se desea realizar alguna modificación.
4.	Mostrará en una nueva pestaña los datos del caso de prueba en formato editable.
5.	Modificar los elementos necesarios en el caso de prueba y confirmar.
Pos-condiciones	
1.	Caso de pruebas actualizado.

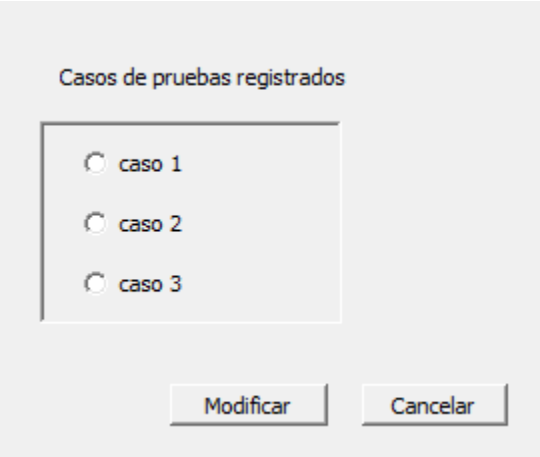
Flujos alternativos	
Flujo alternativo 5.a No se tienen modificaciones.	
1.	Cancelar la operación y cerrar la pestaña.
Flujo alternativo 5.b Agregar nuevo escenario.	
1.	Seleccionar la opción de agregar nuevo escenario.
2.	Mostrará los elementos del formulario necesarios para entrar datos del escenario nuevo.
3.	Se introduce el identificador del escenario que comienza con el identificador del caso de prueba.
4.	Se introduce una descripción de en qué consiste el caso de prueba.
5.	Se introducen los pasos a seguir para ejecutar la prueba, es decir el flujo del escenario.
6.	Se introduce la respuesta que se espera del software una vez ejecutado el flujo y se valora si es necesario otro para volver al paso 1 de este flujo.
Flujo alternativo 5.c Alguna variable requiere más valores.	
1.	Seleccionar la opción de agregar más valores.
2.	Se introduce el nuevo valor de la variable en cuestión y se valora si es necesario otro para volver al paso 1 de este flujo.
Flujo alternativo 5.d Algún escenario requiere más variables.	
1.	Seleccionar la opción de agregar una nueva variable.
2.	Mostrará los elementos del formulario necesarios para entrar los datos de la nueva variable y la opción de introducir más de un valor en caso que lo requiera.
3.	Se introduce el nombre de la variable el cual debe ser representativo.
4.	Se introduce el valor que tomará la variable y se valora si es necesario otra para volver al paso 1 de este flujo.
Flujo alternativo 5.e Está asociado con alguna iteración.	
1.	Mostrará además una lista de las iteraciones del sistema, organizadas por ciclo.
2.	Revisar las iteraciones a las que estará asociado, es decir, en las que se debe ejecutar.
3.	Actualizar las iteraciones seleccionadas o desmarcadas.
Pos-condiciones	
1.	Caso de pruebas actualizado.
Validaciones	
1.	El identificador del escenario debe comenzar con el identificador del caso de prueba. Ejemplo: CP 1, Esc 1.1.

Conceptos	Iteraciones	Visible en Interfaz:

		Visualizar todas las iteraciones organizadas por ciclo. Componente visual: Lista de checbox.
	Casos pruebas	Listado de todos los casos de pruebas. Listado de casos de pruebas para seleccionar uno: Listado de radiobuttoms. Solo se puede seleccionar un caso de prueba a la vez
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

Prototipo elemental de interfaz gráfica de usuario





Para analizar el resto de los requisitos consultar en el expediente de proyecto los documentos 0113-Especificación de Requisitos de Software y 0129_Descripción de requisitos.