Universidad de las Ciencias Informáticas

Facultad 6

Título: "PROPUESTA DE UN SISTEMA PARA REALIZAR PRUEBAS DE RENDIMIENTO A APLICACIONES CLIENTE-SERVIDOR"

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

Autores: Yusneydi Zayas Barrios
Tatiana Álvarez Acosta

Tutores: MSc. Longendri Aguilera Mendoza
Ing. Adisley Reyes Crespo

Consultante: Ing. Cesar Raúl García Jacas

Junio, 2010

"Nunca consideres el estudio como una obligación sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber."

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos 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 _____.

<nombre autor>

_____

Firma del Autor

<nombre tutor>

_____

Firma del Tutor

Datos de Contacto

Autores:

Tatiana Álvarez Acosta

Universidad de las Ciencias Informáticas
Email: taacosta@estudiantes.uci.cu

Yusneydi Zayas Barrios

Universidad de las Ciencias Informáticas
Email: yzbarrios@estudiantes.uci.cu.

Tutores:

Longendri Aguilera Mendoza

Licenciado en Ciencias de la Computación

Máster en Bioinformática.

Email: loge@uci.cu.

Adisley Reyes Crespo

Ingeniero en Ciencias Informáticas.

Email: areyesc@uci.cu.

Consultante:

César Raúl García Jacas

Ingeniero en Ciencias Informáticas.

Email: crjacas@uci.cu.

AGRADECIMIENTOS

Queremos agradecerles primeramente a nuestros tutores Adisley y Longendri, por su entrega, por el conocimiento que nos han transmitido y por saber guiarnos en este difícil camino para que todo saliera bien, gracias.

*A **mami y a papi** por ser luz y guía en mi camino, por enseñarme que aunque todo parezca perdido siempre hay una salida, sólo hay que proponérselo, por ser la razón por la que hoy yo pueda ser ingeniera y convertir su esfuerzo y sacrificio en realidad, por confiar en mi siempre y darme las fuerzas para seguir adelante, los quiero mucho.*

*A **mi tata** por ser "mi hermanito mayor" y ser mi ejemplo en todo momento, por estar ahí siempre que lo necesito, por enseñarme que las cosas se hacen bien sino no se hacen, por poder contar con él siempre, estoy y estaré orgullosa de ti, te quiero.*

*A **mi abuelita**, que aunque no esté hoy, la llevo siempre en mi corazón y si hoy soy lo que soy, en parte, es gracias a ella. Ojalá estuviera presente y pudiera abrazar y besar a su ingeniera, pero donde quiera que esté, sé que está orgullosa de mí.*

*A **Aylén**, mi cuñada preferida, por poder contar contigo en todo, por regalarme dentro de poco esa criaturita preciosa que será mi sobrinita y por ser más que cuñada, una amiga.*

*A mi tía **Graciela** por preocuparse por mí y saber darme consejos para seguir adelante, a mi primo **Rafe** por sus locuras y por siempre hacerme reír aunque pareciera enojada, gracias.*

*A mis tías **Luisa y Migdalia** por todo su amor, cariño y preocupación, gracias.*

*A **mis otros tíos y primos** por preocuparse por mí en estos años y brindarme su apoyo.*

*A mi pareja de tesis **"yuya"** que más que eso es mi amiga y compañera, gracias por confiar en mí, ¡lo logramos!*

*A mis primeras amigas de la UCI, **Ingrita, May, Lexyta, Mary,** por todos estos años compartidos, las fiestas, charlas, sus ratos tristes pero más grandes los felices, consejos y por haber estado cuando las necesité, por confiar en mi y por brindarme todo su apoyo y cariño, **y a mis demás amigos que no fueron los primeros pero llegaron para quedarse, Eri, Gianny, la Yanet, Yanersi, Imi y Yanari,** por ser sinceros y tan especiales, por llenar todos estos años mi corazón de alegría y valentía para seguir adelante, nada hubiese sido igual sin todos ustedes, los quiero mucho.*

A mi ***gente del 6109, Landy, Rigo, Michel y Carli*** *por tantos ratos inolvidables y por llenar mi corazón de recuerdos lindos y duraderos de cada uno.*

A mi ***grupo 6503*** *por los ratos divertidos, aunque fue un año solito son lo máximo.*

A ***César****, por saber guiarnos por un buen camino, gracias por confiar en nosotras y darnos tú apoyo incondicional en todo momento.*

A ***Bety*** *por ayudarnos incondicionalmente en todo momento, gracias.*

A ***Daniel, julio, el bola y Jesús****, gracias por hacerme saber que puedo contar con ustedes, y darme aires de optimismo y que con perseverancia todo se logra.*

A ***Raúl, Enrique, Lenio, Geno, Alberto, Rafa, Maikel, Yosvany, Eri y Eduardo*** *gracias por su preocupación, comprensión y ayuda en todo momento, sin dudas compartir con ustedes fue algo excepcional.*

A las ***chicas del 76103, Mary, Elisa, Ali, Haymel, Nela, Mariela y Yumi*** *incluyendo a "los chicos, Carlos (Pelu), Osmay y Yasiel, por tantos ratos divertidos e inolvidables, gracias.*

A todas las personas magníficas que tuve el privilegio de conocer y compartir en esta universidad, les agradezco.

En fin, le agradezco a Dios por hacer este sueño realidad y a todos los que una vez preguntaran ¿y la tesis cómo va?

Tatiana

*A **Dios** por permitir hacer mi sueño realidad.*

A las personas que me ayudaron a salir adelante en los momentos más difíciles de mi carrera.

*A **mi compañera de tesis** Tatiana por compartir conmigo el reto más importante de mi vida y por que más que una compañera, con su cariño y comprensión demostró ser mi amiga.*

*A **César, Daniel, yusdenis, Erislan, Betty, Albertico** por su ayuda incondicional.*

*A mis amigos que estuvieron a mi lado en los malos y los buenos momentos**, Elisa, Ana niuska, Taty, Yanara, leydisbel, Roxi, Evelyn, Reinaldo, Susy, Imi, Aliekna,** mi gran amigo **Roberto**, a mi amiga **Dayana** por no olvidarse nunca de mí a pesar de la distancia que nos separa, en fin a todos los amigos que la UCI me dio la oportunidad de tener.*

*A **Yuleydis** más conocida como yuya le agradezco todo el cariño y el amor de amiga que me dio hace 4 años y que hoy aunque no estemos cerca sigue dándome su apoyo ,cariño y principalmente su verdadera amistad, a ella que más que una amiga es mi hermana .*

*A mis amis de pinar **Arlenis y Yaima**, por estar compartiendo conmigo este momento, por darme su cariño, apoyo y dedicación.*

*A **mis padres** por darme la dicha de ser su hija y apoyarme y amarme siempre.*

***A mi novio Luis** por hacerme feliz.*

A todas las personas que de una forma u otra hicieron posible este resultado.

Yusneydi

DEDICATORIA

A mi mamá y a mi papá porque todo lo soy hoy, se lo debo infinitamente a ellos.
A mi hermano por ser un gran ejemplo para mí.

Tatiana

A una persona muy especial que hoy no esta aquí con nosotros, pero me acompañó a lo largo de mis cuatro cursos anteriores, dándome apoyo, amor ,en los momentos mas difíciles de mi carrera ahí estaba ella extendiéndome su mano y aconsejándome, dándome fuerzas para seguir adelante. Mi tesis va dedicada especialmente a ella mi abuelita del alma Tata.

A mi mamita por apoyarme en todo, por darme la vida, por convertirme en esta persona que soy hoy, por no perder la fe en mí, por darme su cariño incondicional.

A mi papá por sus buenos consejos por guiarme en la vida, por apoyarme en mis decisiones, por comprenderme en todo por su buen corazón, por ser como yo.

A mi hermanito por llenar mi vida de felicidad con su presencia, por ser el más pequeño y consentido de mis seres queridos, por ser mí angelito.

A una persona que llegó en mi vida hace 10 meses, motivo por el cual no deja de ser especial, ya que en este tiempo supo colmarme de amor, cariño, supo darme todo lo que una persona necesita para ser feliz, a mi novio Luis Javier.
A mi familia que siempre espero lo mejor de mi.

Yusneydi

RESUMEN

En la Universidad de las Ciencias informáticas[1](UCI), para las pruebas de rendimiento de un software, el Laboratorio Industrial para Pruebas de Software (LIPS) utiliza la herramienta JMeter. Esta herramienta funciona simulando hilos de ejecución o lo que es lo mismo, simulando los usuarios que se conectarán a una aplicación determinada en un ordenador.

El presente trabajo de diploma tiene el propósito de analizar la propuesta de una herramienta capaz de potenciar las pruebas de rendimiento en aplicaciones cliente-servidor. La herramienta que se propone tiene el fin de lograr la simulación de conexiones en hilos en un ambiente más real. La propuesta consiste en probar el nivel de carga de usuario que es capaz de soportar un servidor a partir de varias máquinas simulando hilos de ejecución sobre ella, para ello se utiliza la plataforma de tareas distribuidas(T-arenal), que se encuentra desplegada actualmente en la Universidad cuyo ambiente distribuido se aprovechará para potenciar las pruebas de rendimiento. Además, se realiza una propuesta de la arquitectura y posteriormente se evalúa la misma mediante el método ATAM[2].

PALABRAS CLAVE

Hilos de ejecución, prueba, rendimiento.

[1] Universidad cubana donde se estudia la carrera de Ingeniería en Ciencias Informáticas y donde se origina la problemática a la que se le da solución en el presente trabajo.

[2] ATAM (Architecture Tradeoff Analysis Method, del inglés), método para la evaluación de la arquitectura que se centra en la búsqueda de enfoques arquitectónicos que son los medios empleados por la arquitectura para alcanzar los atributos de calidad.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En nuestros días, se evidencia un desarrollo acelerado en las tecnologías y las comunicaciones. A raíz de esto, se hace cada vez más importante la producción de software, el cual alcanza diariamente mayor aceptación y demanda a nivel mundial. Sin embargo, la calidad de las soluciones constituye uno de los mayores problemas que se presentan actualmente.

Cuba no ha estado exenta de este desarrollo, debido a que la informática en los últimos años ha cobrado auge en el país, logrando insertarse en el mercado informático mundial. Esto implica que cada día las empresas destinadas al desarrollo de software, tengan como reto brindar una respuesta eficaz, eficiente, rápida y con calidad a los clientes que cada vez son más exigentes.

Entre las entidades productoras de software en Cuba se encuentran: La Empresa Cubana Nacional de Software (DeSoft), La Empresa Cubana Productora de Software para la Técnica Electrónica (Softel) y La Universidad de las Ciencias Informáticas (UCI). Esta última como empresa desarrolladora de software, se encarga de la formación de ingenieros bien preparados a partir de la vinculación estudio trabajo como modelo de formación, teniendo como misión producir software con eficiencia.

Para lograr una buena calidad en el software, la Universidad tiene la ventaja de contar con Calisoft (Centro para la excelencia en el desarrollo de Proyectos Tecnológicos) perteneciente al Ministerio de la Informática y las Comunicaciones, que se encarga de certificar la calidad de los productos de software que en dicha institución se realizan, además de facilitar la implementación de las mejores prácticas en el proceso de desarrollo y mantenimiento de un software. Para esto cuenta con un laboratorio industrial de pruebas de software (LIPS). El mismo, asegura que cada artefacto haga lo adecuado en todo momento una vez realizadas todas las pruebas al software, es decir, que el software esté al 100% para ser entregado al cliente.

Para asegurar que un producto está libre de errores, que responda a las necesidades del cliente y que funcione correctamente en el entorno para el que fue creado, se realizan varias pruebas. Entre estas pruebas se encuentran las pruebas de unidad, que se basan en verificar que una unidad funcione correctamente por sí misma sin tener en cuenta las otras partes del sistema; se procede con las de integración de los módulos, detectando así cualquier error de funcionalidad del sistema; seguidamente se

realizan las pruebas de sistema para verificar el diseño detalladamente y por último las pruebas de aceptación para validar los requerimientos especificados por el cliente.

En general todos los niveles de prueba son importantes a la hora de verificar que el software se ha desarrollado correctamente, pero para verificar que se ha cumplido con los requisitos de rendimiento especificados, el nivel de prueba a usar es el de pruebas a nivel de sistema. Las pruebas de sistema están dirigidas a verificar el programa final, después que todos los componentes de software y hardware han sido integrados. Estas incluyen pruebas de funcionalidad, de usabilidad, de comunicación, de entorno, de instalación, de recuperación, pruebas de seguridad, pruebas de volumen y rendimiento.

Uno de los pasos para garantizar que el producto sea óptimo, es el desarrollo de un correcto procedimiento de pruebas de rendimiento al software, para mejorar la calidad del mismo. Estas pruebas de rendimiento se realizan con el fin de medir mediante una carga de trabajo, qué partes de un sistema provocan que el mismo rinda deficientemente y de esta forma obtener el punto de ruptura del sistema.

Actualmente en la Universidad se realizan diversas aplicaciones informáticas para empresas tanto nacionales como internacionales. Gran parte de estas aplicaciones se basan en arquitectura cliente-servidor, donde el cliente es una máquina que solicita una determinada petición al servidor y este le proporciona respuesta a dicha petición.

La realización de pruebas de rendimiento a este tipo de sistema es fundamental, ya que mediante ellas se puede verificar qué volumen de carga es capaz de soportar un servidor ante peticiones clientes, ya sea de forma esperada o muy por encima de lo esperado. En el mundo hay grandes servidores que necesitan de las pruebas de rendimiento debido a las miles de conexiones que se ejecutan en determinados instantes de tiempo, ejemplo de ellos son: Google, Yahoo, AltaVista, entre otros.

En la universidad, el laboratorio Industrial de pruebas de software (LIPS), utiliza la herramienta JMeter para realizar las pruebas de rendimiento y devuelve resultados estadísticos que muestran el desempeño del sistema, los tiempos de respuesta, la carga de usuario, el uso de recursos y en general el comportamiento del software bajo determinadas condiciones de trabajo a las que se someta. Su funcionamiento se basa en grabar escenarios para simular conexiones en hilos sobre la aplicación a probar desde un ordenador. Estos hilos de ejecución se realizan gradualmente de 10 en 10, de 20 en 20, y

así paulatinamente o ejecuciones a la vez que pudiera ser un grupo de 100 hilos para probar la carga que es capaz de soportar el sistema, ya sea de forma esperada o muy por encima de lo esperado.

Esta herramienta es potente en cuanto a pruebas se refiere, porque no sólo realiza pruebas de rendimiento sino también pruebas funcionales y de regresión. A pesar de esta ventaja, trae consigo que al realizar las pruebas de rendimiento se torne muchas veces lenta e incluso llegue a bloquearse con apenas 150 usuarios a simular, debido a que todo este proceso se realiza en la misma máquina.

Disponer de una herramienta que permita un mayor nivel de concurrencia en el servidor, contribuiría a potenciar las pruebas de rendimiento para que los ingenieros del LIPS lleguen más rápido al punto de ruptura de las aplicaciones que requieren de un gran número de conexiones.
A partir de aquí podemos definir como **problema científico** ¿Cómo potenciar la realización de las pruebas de rendimiento que se realizan en la Universidad por el LIPS?

Como **objeto de estudio** se define el proceso de Pruebas de calidad de software

El **campo de acción** se enmarca en el proceso de pruebas de rendimiento a aplicaciones cliente-servidor.

Para dar solución al problema se define como **objetivo general:** Desarrollar una propuesta de una herramienta que potencie las pruebas de rendimiento a aplicaciones cliente-servidor generando múltiples hilos de ejecución en varias estaciones de trabajo.

Objetivos Específicos:

- ✓ Realizar el análisis de una herramienta que realice pruebas de rendimiento a aplicaciones cliente-servidor.
- ✓ Definir la propuesta de arquitectura de la herramienta.

Para dar cumplimiento a este objetivo se proponen las siguientes **tareas de la investigación:**

- ✓ Revisión del estado del arte acerca de sistemas existentes en el mundo para realizar pruebas de rendimiento.
- ✓ Estudio de herramientas que permitan aprovechar los recursos de cómputo que se encuentran disponibles en la universidad.

- ✓ Elaboración del modelo de dominio del sistema.
- ✓ Definición de los requisitos funcionales y no funcionales de la herramienta.
- ✓ Desarrollo del Diagrama de Casos de Uso del sistema.
- ✓ Descripción de los Casos de Uso del sistema.
- ✓ Descripción de la vista de caso de uso.
- ✓ Descripción de una propuesta de la vista lógica y vista de despliegue.
- ✓ Descripción de los patrones de diseño propuestos.
- ✓ Selección de los patrones arquitectónicos.
- ✓ Evaluación de la arquitectura.

Capítulo 1. Fundamentación Teórica, en este capítulo se realiza un estudio sobre la calidad del software y sus conceptos más significativos, los tipos de pruebas que se le realizan a los sistemas informáticos y las herramientas que se utilizan para medir el rendimiento en un software. También se realiza un estudio sobre la arquitectura de un software así como el estudio de métodos para la evaluación de la misma. Además, se explican las tecnologías y herramientas utilizadas para dar solución al problema planteado.

Capítulo 2. Características del sistema, en este capítulo se realiza una descripción detallada de la solución propuesta. Se desarrolla un modelo de dominio donde se analiza cada uno de los conceptos y entidades presentes en el negocio. Se definen los requisitos funcionales y no funcionales de la herramienta, se presenta el diagrama de caso de uso del sistema, así como las descripciones textuales de los casos de uso para comprender mejor el funcionamiento del sistema.

Capítulo 3: Propuesta de la Arquitectura, en este capítulo se define elementos como el patrón arquitectónico, el tipo de arquitectura a utilizar en la herramienta y la modelación de la vista de casos de uso. También se describe la propuesta de la vista lógica y la vista de despliegue. Además, se realiza la evaluación de la arquitectura haciendo uso del método de análisis de acuerdo de la arquitectura (ATAM).

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se realiza un estudio de la calidad del software y sus conceptos más significativos, detallando los tipos de pruebas por niveles que existen, haciendo énfasis en las pruebas de sistema y dentro de sus funcionalidades, rendimiento. También se realiza un estudio sobre arquitectura de un software y sus elementos principales como los estilos y patrones arquitectónicos, así como el estudio de métodos para la evaluación de la misma, como son el Método de Análisis de Arquitecturas de Software (SAAM), Método de Análisis de Acuerdos de Arquitectura (ATAM) y Método de Revisiones de Diseños Activas (ARID). Se realiza un estudio sobre las herramientas que se utilizan para medir el rendimiento en sistemas informáticos utilizadas en el mundo y más detalladamente en la Universidad de Las Ciencias Informáticas. Además, se explica la metodología de desarrollo, tecnologías y herramientas utilizadas para dar solución al problema planteado.

1.1 Calidad de software

En la actualidad uno de los problemas en la esfera de la informática, es la calidad del software. La obtención de un software con calidad implica la utilización de metodologías o procedimientos a lo largo de todo el desarrollo del mismo en vista de lograr un producto con eficiencia.

La palabra calidad de software tiene múltiples definiciones, entre ellas se encuentran: La calidad de software es la “Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente” (1).

El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) definió la calidad de software como: “grado con el que un sistema, componente o proceso cumple con los requerimientos especificados y las necesidades o expectativas del cliente o usuario” (2). Se puede definir que la calidad es sinónimo de la comprobación constante del software en cada fase de su desarrollo, apoyándose en las pruebas que se le hacen a lo largo de su ciclo de vida con el fin de lograr un mayor grado de satisfacción del cliente.

1.2 Pruebas de calidad del Software

Las pruebas de calidad del software son un eslabón fundamental en el proceso de desarrollo del software. Permiten evaluar el producto para conocer en qué condiciones se encuentra el software y de esta forma detectar errores que pudieran generar comportamientos erróneos durante la ejecución del mismo.

Según el IEEE, prueba se define como: “Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente” (2).

Los principales objetivos que se deben tener en cuenta a la hora de realizar pruebas de calidad a un software son:

- ✓ Encontrar defectos en un software.
- ✓ Diseñar pruebas a lo largo de todo el ciclo de desarrollo del software para así corregir la mayor cantidad de errores posibles y ganar en tiempo y esfuerzo.
- ✓ Ofrecer un producto altamente seguro y confiable.
- ✓ Verificar que el software satisface las necesidades del cliente.

El proceso de pruebas en un software tiene éxito si descubre defectos en el mismo, de lo contrario fracasa. A pesar de que las pruebas no pueden asegurar que un software está correcto, ya que sólo pueden demostrar que existen defectos en el software, son parte fundamental dentro del ciclo de desarrollo antes de ser entregado el producto final.

En el proceso de pruebas al software existen niveles en los que se ejecutan diferentes tipos de prueba con objetivos específicos. Teniendo en cuenta esto, las pruebas se agrupan por niveles de acuerdo a las diferentes etapas del proceso de desarrollo.

1.2.1 Prueba de Unidad

Las pruebas de unidad se concentran en la lógica del procesamiento interno. Son utilizadas para probar el correcto funcionamiento de un módulo de código, de ahí que su objetivo es aislar cada parte del programa y mostrar que las partes individuales son correctas por sí solas.

1.2.2 Prueba de Integración

Estas pruebas son realizadas una vez que terminen las pruebas de unidad y preceden a la integración del sistema. Algunas veces llamadas integración y testeo, se basan principalmente en combinar y testear componentes individuales como un grupo, para corregir errores que pueden surgir a partir de la integración de esos componentes. No son verdaderamente pruebas de sistema, porque los componentes no están implementados en el ambiente operativo (3).

1.2.3 Pruebas de Sistema

Las pruebas de sistema coinciden con las fases finales de las pruebas de integración. Tienen como propósito entrenar profundamente el sistema para verificar que se han integrado adecuadamente todos los elementos del sistema ya sean de hardware o software. Estas se realizan cuando el software está funcionando como un todo.

"La prueba de sistema, generalmente está constituida por una serie de pruebas diferentes cuyo propósito fundamental es ejercitar el sistema**.** Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas". (4)

Entre los tipos de pruebas más importantes en este nivel se encuentran:

- ✓ Funcionales: Las pruebas funcionales están desarrolladas bajo la perspectiva del usuario, confirmando que el sistema hace lo que los usuarios esperan que haga, es decir, que se cumplan satisfactoriamente los requisitos funcionales.
- ✓ De Usabilidad: Se intenta determinar si el sistema será fácil de utilizar para sus usuarios.
- ✓ De comunicación: Se prueban las interfaces de comunicación entre diferentes componentes del sistema.
- ✓ De rendimiento: Prueba el rendimiento del software en tiempo de ejecución. Determina si los tiempos de respuesta, tanto en condiciones normales como en condiciones especiales, se encuentran dentro de los límites predefinidos. Entre los tipos de prueba de rendimiento se encuentran:
  - ➢ Pruebas de carga: Una prueba de carga se ejecuta para comprender el comportamiento de una aplicación ante una carga determinada. Esta carga puede ser el número de usuarios esperado

ejecutando o un número de transacciones durante un tiempo determinado. El resultado de esta prueba nos dará el tiempo de respuesta de todas las transacciones críticas. Si la base de datos, servidor de aplicación también se monitorizan, entonces esta prueba puede mostrar potenciales problemas de botella en la aplicación.

- Pruebas de estrés: Estas pruebas son utilizadas normalmente para someter a la aplicación al límite de su funcionamiento, mediante la ejecución de un número de usuarios muy superior al esperado. Esta prueba tiene como finalidad el determinar la robustez de una aplicación cuando la carga es extrema y ayuda a administradores a determinar si la aplicación se comportará correctamente en dichas situaciones. Otro posible objetivo de este tipo de pruebas es determinar el límite real de la aplicación en cuanto a número de usuarios concurrentes, número de transacciones por segundo, entre otros.
- Pruebas de Resistencia (SOAK): Esta prueba se realiza con el fin de determinar si la aplicación puede mantener la carga esperada de manera continua y durante un largo tiempo.
- Pruebas de Picos: Este tipo de pruebas se realiza insertando la carga en el sistema en forma de "picos", que se irán lanzando en distintos momentos de la prueba. De esta forma se permitirá comprender el comportamiento de la aplicación ante cambios bruscos de carga.

- ✓ De volumen: Dedicadas a probar el software trabajando con grandes cantidades de datos, similares a las esperadas en producción.
- ✓ De seguridad: Se basa en verificar los mecanismos de protección.
- ✓ De entorno: Orientadas a probar las interacciones entre el sistema y otros sistemas en su entorno.
- ✓ De instalación: Se centra en asegurar que el sistema de software desarrollado se puede instalar en diferentes configuraciones de hardware y software y bajo condiciones excepciones, por ejemplo con espacio de disco insuficiente o continuas interrupciones.
- ✓ De recuperación: Se basa en forzar un fallo del software y verificar que la recuperación se lleva a cabo apropiadamente.

1.2.4 Prueba de Validación

Estas pruebas se realizan con el objetivo de comprobar que el software, una vez ensamblado, funciona de acuerdo a las expectativas del cliente. Se comprueba que el software satisface los requisitos funcionales,

de portabilidad, compatibilidad, recuperación de errores, facilidad de mantenimiento, entre otros, además de verificar que la documentación sea correcta.

Un elemento importante del proceso de validación es la revisión de la configuración. La intención de la revisión es asegurarse de que todos los elementos de la configuración del software se han desarrollado correctamente y están suficientemente detallados para soportar la fase de mantenimiento durante el ciclo de vida del software.

1.2.5 Prueba de Regresión

Se denominan pruebas de regresión a cualquier tipo de pruebas de software que intentan descubrir las causas de nuevos errores, ya sean carencias en la funcionalidad o discordancias funcionales con respecto al comportamiento esperado del software. Por lo tanto, en la mayoría de las situaciones del desarrollo de software se considera una buena práctica, que cuando se corrija un error, se grabe una prueba que exponga el mismo, y se vuelva a probar reiteradamente después de los cambios efectuados en el programa.

1.2.6 Prueba de Aceptación

Las pruebas de aceptación son básicamente pruebas funcionales sobre el sistema completo, ya que se ejecutan cuando el sistema funciona como un todo. Su objetivo es comprobar que el software está listo y que puede ser usado por usuarios finales para ejecutar todas las funcionalidades para las cuales fue creado.

En este trabajo se hace un enfoque en las pruebas de sistema, dentro de ellas, en rendimiento. Para la automatización de estas pruebas, se utilizan herramientas, las cuales se abordan a continuación.

1.3 Herramientas que se utilizan para pruebas de rendimiento

Existen herramientas que permiten medir el rendimiento de un sistema. A continuación se describen algunas de estas herramientas, destacando el hecho que todas son propietarias excepto JMeter que es libre y no sólo realiza pruebas de rendimiento sino también de regresión y funcionales.

1.3.1 QALoad

QALoad es una herramienta de pruebas para aquellas aplicaciones que están sometidas a gran carga de usuarios. QALoad ayuda a alcanzar cargas que simulan el comportamiento real del negocio, así como a validar que el sistema cumple aceptablemente con los niveles de servicio.

Es una herramienta de automatización de pruebas de carga para Web, Java, .NET, aplicaciones de planificación de recursos empresariales (ERP) y aplicaciones de gestión de relaciones con los clientes y ambientes distribuidos (CRM). Simula miles de usuarios desempeñando transacciones de negocio clave de una aplicación. Con esta herramienta, los equipos de prueba pueden rápidamente detectar problemas, optimizar el desempeño de los sistemas y ayudar a asegurar un despliegue de aplicaciones exitoso.

1.3.2 LoadRunner

Es una herramienta para realizar pruebas de carga de Mercury Interactive (empresa que originalmente desarrolló el producto), que permite mostrar el comportamiento y el rendimiento del sistema. Además, permite poner a prueba toda la infraestructura corporativa para identificar y aislar los posibles problemas mediante la simulación de la actividad de miles de usuarios, con lo que los equipos de desarrollo de aplicaciones y sitios Web pueden mejorar el rendimiento de las aplicaciones.

Los mismos scripts creados durante las pruebas, pueden volverse a utilizar para monitorizar la aplicación una vez terminada su implantación.

1.3.3 JMeter

JMeter es una herramienta Java libre y de código abierto dentro del proyecto Jakarta[3]. Permite realizar pruebas de rendimiento, regresión y pruebas funcionales sobre Aplicaciones Web, de manera que se simulan un número de hilos que se conectarán a una aplicación y ejecutarán de manera independiente, cada uno de ellos. Dichos hilos pueden ser considerados como distintos usuarios que se conectan a la aplicación.

Además, muestra los resultados de las pruebas en una amplia variedad de informes y gráficas y facilita a una rápida detección de los cuellos de botella existentes debido al tiempo de respuesta excesivo.

[3] El proyecto Jakarta es el que se encarga de crear y mantener todas las soluciones open source (open source o código abierto permite que varios programadores puedan leer, modificar y redistribuir el código fuente de un programa mejorándolo y corrigiendo sus errores) creadas para la plataforma Java.

Además, se destaca por su versatilidad, estabilidad, por su uso gratuito y porque además de realizar pruebas de rendimiento de carga a aplicaciones Web, también las realiza de estrés. Actualmente en la universidad se utiliza esta herramienta por el LIPS para realizar pruebas de rendimiento a aplicaciones con arquitectura cliente/servidor y medir el punto óptimo, que indica la ruptura del sistema mediante la degradación que va sufriendo en toda su estructura (servidores físicos, bases de datos, servidores de aplicaciones) a través de simulaciones de usuarios.

1.4 Sistemas Distribuidos

Un sistema distribuido se define, como una colección de computadoras separadas físicamente y conectadas entre sí por una red de comunicaciones distribuida. Cada máquina posee sus componentes de hardware y software que el usuario percibe como un único sistema (no necesita saber qué cosas están en qué máquinas).

Las pruebas de carga y estrés sobre aplicaciones distribuidas son de gran utilidad. Cuando se plantea realizar pruebas de carga y estrés, se necesita una aplicación que sea capaz de generar de manera simple, repetible, distribuida, suficiente y controlada, peticiones a la aplicación que se desea someter a estrés. Un sistema distribuido, propicia que las pruebas de carga y estrés se hagan en menor tiempo y aumenten la concurrencia en un servidor, a partir de simulaciones de hilos de ejecución.

1.4.1 Plataforma de Cálculo Distribuido T-arenal.

La Plataforma T-arenal, es un sistema distribuido que utiliza varios recursos de cómputo como estaciones de trabajo, que pueden ser heterogéneos en cuanto a hardware y software, para reducir considerablemente el tiempo en obtener la solución de grandes problemas que puedan ser descompuestos en tareas más pequeñas e independientes. No introduce un costo adicional, como la compra de nuevos equipos de cómputo, ya que utiliza las computadoras existentes en la Universidad que funcionan como estaciones de trabajo. Es multiplataforma, es decir, completamente independiente del sistema operativo y arquitectura de hardware que tengan las computadoras. Además, está basado en el modelo Cliente – Servidor (5).

Está dividido en tres componentes esenciales: servidor central, servidor de peticiones y cliente.

Servidor Central: Su principal función, es la de chequear y planificar la asignación de las diferentes tareas a los servidores de peticiones. Además de controlar información sobre los usuarios que pueden acceder e interactuar con el sistema, así como, almacenar los problemas a partir de los cuales se crearán las diferentes ejecuciones a servir. Realiza también el seguimiento de todos los eventos que ocurren durante su funcionamiento (5).

Servidor de Peticiones: Es el responsable de solicitar una tarea al servidor central, así como atender y controlar la realización de la misma. La tarea a ser atendida, será dividida en pequeñas subtareas denominadas unidades de trabajo. El principal aspecto del servidor de peticiones es repartir las diferentes unidades de trabajos entre los clientes, siempre y cuando estos estén autorizados a interactuar con el mismo. El servidor de peticiones procesa el resultado de cada una de las subtareas generadas, para finalmente construir el resultado de la tarea original (5).

Cliente: El cliente es el que realiza una solicitud de una unidad de trabajo al servidor de peticiones correspondiente. Realiza el procesamiento de la unidad de trabajo y posteriormente retorna el resultado obtenido, y así sucesivamente. Múltiples clientes pueden realizar peticiones de trabajo al servidor de peticiones (5).

Además, cumple con los aspectos esenciales de un sistema de cómputo distribuido: transparencia, eficiencia, flexibilidad, escalabilidad y fiabilidad (5).

- ✓ **Transparencia:** Oculta la naturaleza distribuida permitiendo que los usuarios interactúen con una aplicación de Desktop y trabajen como si se tratara solamente de una supercomputadora.
- ✓ **Eficiencia:** La solución a los problemas se obtiene mucho más rápido haciendo uso del sistema distribuido, que la respuesta que daría una simple computadora.
- ✓ **Flexibilidad:** Es lo suficientemente flexible, para que cualquier cambio a realizar no requiera la parada de todo el sistema y la recopilación de todo el código.
- ✓ **Escalabilidad:** Se comporta estable, tanto en redes pequeñas que grandes. También garantiza un adecuado mantenimiento y actualización, empleando el mínimo de personal.
- ✓ **Fiabilidad:** La protección que brinda al usuario es extrema, asegura al 100% que el problema del usuario será resuelto, independientemente de los problemas ajenos que existan, ya sean fallos de red, electricidad, apagado de máquinas.

- ✓ **Multiplataforma:** Al ser un sistema desarrollado completamente en Java, está listo para ser instalado sobre cualquier tipo de arquitectura de hardware y sistemas operativos.

Este sistema distribuido se encuentra desplegado actualmente en la UCI.

1.5 Arquitectura de software

El término arquitectura de software (AS) tiene sus orígenes en los años 60 del siglo XX, pero no es hasta la década de los 90 cuando alcanza gran popularidad, convirtiéndose en uno de los progresos más importantes dentro de la producción de software. La arquitectura de un software puede concebirse como aquella estructura del programa que cohesiona las funcionalidades más críticas y relevantes (necesarias para el sistema) y que sirve de soporte al resto de funcionalidades finales (necesarias para el usuario).

La AS permite representar la estructura de un sistema a un nivel mayor que el dado por la programación e incluso por el diseño. Constituye un modelo de como está estructurado dicho sistema y tiene como objetivo primario aportar elementos para ayudar a la toma de decisiones entre las personas involucradas en el desarrollo, como analistas, diseñadores, programadores y otros. También describe diversos aspectos del software de una forma comprensible utilizando modelos o vistas, los cuales pueden expresarse mediante varios lenguajes como el Lenguaje Unificado de Modelado (UML, de sus siglas en inglés), el mismo, ha sido adoptado de forma extendida para la mayoría de los modelos. Varios son los autores que han dado sus criterios y opiniones acerca de qué es la AS y a continuación se muestran algunos criterios, tomando los más completos u oficiales en el mundo, de las empresas productoras de software.

La definición oficial que sigue la IEEE Std1471-2000 plantea: “La arquitectura del software es la organización fundamental de un sistema formada por sus componentes[4], las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución” **¡Error! No se encuentra el origen de la referencia.**. Una definición reconocida es la de Clements: “La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema” (6).

[4] Bloques de construcción que conforman las partes de un sistema de software. A nivel de lenguajes de programación, pueden ser representados como módulos, clases, objetos o un conjunto de funciones relacionadas.

Es una vista estructural de alto nivel; define un conjunto de patrones y estilos o combinación de estilos para una solución; centra su atención en los requerimientos no funcionales, ya que los requerimientos funcionales se satisfacen mediante el modelado y diseño de la aplicación.

1.5.1. Arquitectura orientada a plug-ins

La arquitectura de plug-ins como su nombre lo indica está basada en plug-ins. Un plug-ins o complemento, es una aplicación adicional que se relaciona con otra para aportarle una funcionalidad nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de una Interfaz de programación de aplicaciones (API) que representa una interfaz de comunicación entre componentes de software (7). Además, la aplicación principal funciona independientemente de la aplicación adicional, lo que permite a los usuarios finales añadir y actualizar los complementos de forma dinámica sin necesidad de hacer cambios en la aplicación principal. A partir de esta arquitectura, se pueden adicionar otros plug-ins para incluir nuevas funcionalidades. Este tipo de arquitectura fue la seleccionada para la herramienta que se propone.

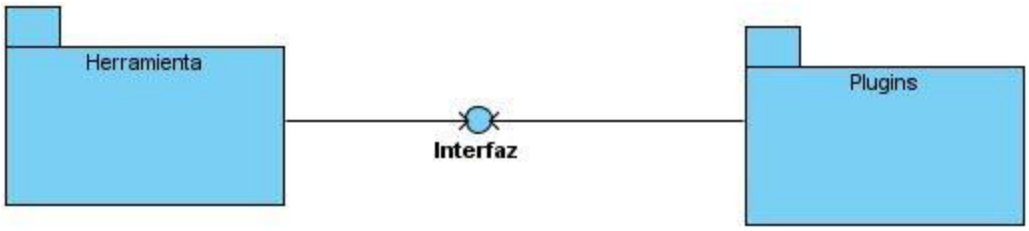

Figura 1. Propuesta de la arquitectura orientada a plug-ins en la herramienta.

1.6 Estilos y patrones arquitectónicos.

En algunas bibliografías referentes al estudio de los patrones arquitectónicos se les llama de la misma forma a ambos, pero la mayoría de los autores los ven de manera separada. Ambos se refieren a formas de estructurar los sistemas y cómo relacionar los componentes de estos, la diferencia radica en el nivel de abstracción en que se aplican. Los estilos están en un plano de abstracción más alto, definiendo cómo configurar la arquitectura, mientras que los patrones están más cercanos al diseño, incluso podría decirse que más cercano a algo físico, pues estos pueden representarse mediante código en un lenguaje de programación determinado.

1.6.1 Estilo Arquitectónico.

Cuando se define la arquitectura de software, hay que decidir que estilos arquitectónicos serán utilizados en la misma. Los estilos de arquitectura guían a la organización del sistema de software. Estos incluyen reglas y líneas a seguir para la organización de un sistema. Están vinculados a la forma más general en que está organizado un sistema de software, a las formas generales de la organización, mientras que los patrones están asociados a formas más concretas, que tienen que ver con la especialización. Los estilos arquitectónicos constituyen una generalización y abstracción de los patrones.

1.6.2 Patrón arquitectónico.

El patrón es una descripción del problema y la esencia de su solución, de forma que la solución pueda reutilizarse en diferentes situaciones, no constituye una especificación detallada, es una solución adecuada a un problema común. De ahí, que en el año 1977, Christopher Alexander expresa: “Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a ese problema, de tal manera que puedes usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces”.

Los patrones son formas de describir las mejores prácticas, buenos diseños, y encapsulan la experiencia de forma tal que es posible para otros el reutilizar dicha experiencia. Constituyen mecanismos cuyo objetivo es la solución de problemas que ocurren repetidamente dentro de un contexto muy bien definido.

1.7 Patrón arquitectónico cliente/servidor

Este patrón se divide en dos partes, la primera es la parte del servidor y la segunda la de un conjunto de clientes. El cliente y el servidor generalmente están localizados en diferentes sistemas, sin embargo pueden encontrarse en el mismo sistema. El cliente, es la entidad que hace la petición por un servicio y el servidor, es la entidad que provee el servicio correspondiente a la petición. El servicio debe procurar el resultado, el cual es retornado al cliente. Normalmente, el servidor es una máquina bastante potente que actúa de depósito de datos y funciona como un sistema gestor de base de datos (SGBD). Por otro lado, los clientes suelen ser estaciones de trabajo que solicitan varios servicios al servidor. Ambas partes deben estar conectadas entre sí mediante una red. A continuación, una representación gráfica de este tipo de patrón:

Figura 2. Representación del patrón arquitectónico cliente-servidor

1.8 Evaluación de la arquitectura

Para evaluar una arquitectura hay que tener en cuenta si las decisiones que se han tomado con respecto a la misma, están basadas en el logro de los atributos de calidad del sistema, de ser así, entonces la evaluación se centra en revisar qué tipo de impacto tiene la arquitectura elegida sobre los atributos de calidad que deben estar presentes en el sistema. No sirve de nada un sistema que no cumple con los atributos de calidad y con los requisitos no funcionales que se especificaron. Por lo que diseñar una correcta arquitectura va a determinar el éxito o fracaso de un sistema de software, en la medida que ésta cumpla o no con sus objetivos, debido a esto "…para reducir tales riesgos, y como buena práctica de ingeniería, es recomendable realizar evaluaciones a la arquitectura" (8).

De la evaluación de una arquitectura de software no se obtienen respuestas del tipo sí, no, bueno o malo, sino que explica cuáles son los puntos de riesgo que se corren al asumir la arquitectura propuesta. Uno de los puntos para el éxito de una arquitectura, está en que la misma, presente el menor número de riesgos posibles.

¿Qué es una Evaluación?

• Es un estudio de factibilidad que pretende detectar posibles riesgos, así como buscar recomendaciones para contenerlos.

• La diferencia entre evaluar y verificar, es que la evaluación se realiza antes de la implementación de la solución. La verificación es con el producto ya construido (9).

El objetivo de evaluar una arquitectura, es saber si puede habilitar los requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los stakeholders (10). A continuación, se hace un breve estudio de algunos métodos que existen en la actualidad para llevar a cabo esta tarea.

1.9 Métodos de evaluación de la arquitectura de software.

Existen varios métodos para evaluar la arquitectura, entre los más importantes se encuentran el Método de Análisis de Arquitecturas de Software (SAAM), Método de Análisis de Acuerdos de Arquitectura (ATAM) y el Método de Revisiones de Diseños Activas (ARID).

Método de Análisis de Arquitecturas de Software (SAAM): El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM), es el primer método de evaluación basado en escenarios que surgió, centra su atención en la modificabilidad, y puede ser usado para evaluar una arquitectura o evaluar y comparar varias. Kazman plantea que las salidas de la evaluación del método SAAM son las siguientes (10).

- ✓ Una proyección sobre la arquitectura de los escenarios, que representan los cambios posibles ante los que puede estar expuesto el sistema.
- ✓ Entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación.

Método de Análisis de Acuerdos de Arquitectura (ATAM): Está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. Es un método de evaluación basado en escenarios[5], que permite evaluar qué tan bien un atributo de calidad es soportado por la arquitectura y permite reconocer las decisiones arquitectónicas que afectan a más de un atributo de calidad (puntos de compromiso).

El método se concentra en la identificación de los estilos arquitectónicos, ya que estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad. El método confía fuertemente en la identificación de los *puntos de sensibilidad* y los riesgos de la arquitectura, ya que los puntos de sensibilidad ayudan a reforzar la arquitectura y a prevenirla de nuevos riesgos. Además, el método no sólo encuentra *riesgos* sino también *no riesgos* que demuestran que las decisiones arquitectónicas son apropiadas, es decir, que la arquitectura satisface los requerimientos de los atributos de calidad. Puede utilizarse en una fase temprana en el ciclo de vida del desarrollo de software. Al finalizar el desarrollo del método se obtienen los siguientes resultados:

- ✓ El documento de propuestas arquitectónicas.
- ✓ El conjunto de escenarios priorizados.
- ✓ El árbol de utilidad.
- ✓ Los riesgos descubiertos.
- ✓ Los no riesgos
- ✓ Los puntos de sensibilidad.
- ✓ Los puntos de trade-off.

Método de Revisiones de Diseños Activas (ARID)

El método ARID fue creado para evaluar diseños parciales de la arquitectura. En ARID coinciden los métodos ATAM; visto anteriormente; y Active Design Review (ADR). ADR es utilizado para la evaluación de la calidad y la completitud en la documentación de diseños detallados de unidades del software como los componentes o módulos, a esto se le une la idea del uso de escenarios. El método ARID evalúa mejor la factibilidad de la arquitectura.

[5] Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con el propio sistema. consta de tres partes: el estímulo(es la parte del escenario que describe la iteración del involucrado con el sistema), el contexto (describe que sucede en el momento del estímulo) y la respuesta (describe cómo debería responder el sistema ante el estímulo).

Tomando en cuenta la importancia de la evaluación de la AS y profundidad de los métodos mencionados en este epígrafe, se certificarán las decisiones arquitectónicas de la descripción propuesta mediante el método de evaluación ATAM porque demostró ser el más acorde a las necesidades del trabajo, el más documentado y con mejores explicaciones en la bibliografía consultada y además, porque se puede utilizar para evaluar una arquitectura en etapas tempranas del desarrollo de software.

1.10 Atributos de Calidad

Es importante determinar cuáles atributos de calidad deben alcanzarse en la descripción arquitectónica ya que posteriormente serán usados para verificar el cumplimiento de ellos en la propuesta arquitectónica. Entre los modelos de calidad más importantes propuestos se encuentran: McCall (1977), FURPS (1987), Dromey (1996) e ISO/IEC 9126, este último adaptado para arquitecturas de software propuesto por Losavio en el año 2003, será el que se propone en la herramienta.

El estándar ISO/IEC 9126 ha sido desarrollado en un intento de identificar los atributos claves de calidad para un producto de software (11). Este estándar es una simplificación del Modelo de McCall e identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software. El estándar, provee una descomposición de las características en subcaracterística, que se muestran en la siguiente tabla.

Tabla 1. Atributos de calidad - Modelo ISO/IEC 9126

Características	Subcaracterística
Funcionalidad	Adecuación, Exactitud, Interoperabilidad y Seguridad.
Confiabilidad	Madurez, Tolerancia a fallos y Recuperabilidad.
Usabilidad	Entendibilidad, Capacidad de aprendizaje y Operabilidad.
Eficiencia	Comportamiento en tiempo y Comportamiento de recursos.
Mantenibilidad	Analizabilidad, Modificabilidad, Estabilidad y Capacidad de pruebas.
Portabilidad	Adaptabilidad, Instalabilidad y Reemplazabilidad

1.11 Metodología, tecnologías y herramientas

Para la realización de cualquier proyecto de software, se debe definir qué metodología y herramienta de desarrollo se ajusta al proyecto. No existe una metodología de software universal, esta debe adecuarse a las características específicas de cada proyecto (recursos y equipo de desarrollo) que exigen que el proceso sea configurable. La propuesta de la metodología, tecnología y herramienta que se muestra a continuación, surgen a partir de la relación que existirá entre la herramienta y la plataforma de tareas distribuidas (t-arenal) para su funcionamiento, además, porque t-arenal es una aplicación programada en java utilizando eclipse como entorno de desarrollo. Todas son tecnologías y herramientas libres, de ahí que sean las propuestas para el desarrollo de la herramienta.

1.11.1 Metodología

Existen varias metodologías de desarrollo de software, cada una con sus propias características, aunque objetivamente persigan lo mismo. Ejemplos de éstas son: El Proceso Unificado de Rational (RUP), Programación Extrema (XP), OpenUP.

Se decide utilizar OpenUP como metodología de desarrollo siguiendo los estándares propuestos por el proyecto Plataforma de Tareas Distribuidas. Además, preserva las características esenciales de RUP que incluye el desarrollo interactivo, casos de uso y escenarios de conducción de desarrollo, la gestión de riesgo y el enfoque centrado en la arquitectura. También define un proceso de desarrollo de software mínimo y completo. Mínimo porque solamente lo fundamental es incluido dentro del proceso y completo porque define un conjunto de componentes que guían y definen dicho proceso de desarrollo hasta la obtención del producto. Es extensible, de manera que se pueden añadir artefactos, actividades u otro componente a la metodología, según lo requiera el sistema que se desarrolla.

Además de lo antes expuesto, OpenUP es un proceso dirigido a gestión y desarrollo de proyectos de software basados en un desarrollo iterativo, ágil e incremental apropiado para proyectos pequeños y de bajos recursos. Además, es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo.

1.11.2 Herramienta para la modelación visual del sistema

Las herramientas CASE (Computer-Aided Software Engineering) son generalmente aplicadas a cualquier sistema o colección de herramientas que ayudan a automatizar el proceso de diseño y desarrollo del

software. Impiden a los programadores tratar tan directamente con el hardware, permitiéndoles trabajar con un alto nivel de abstracción en la definición del sistema.

Para la modelación visual del sistema se emplea Visual Paradigm para UML en su versión 6.4. Es una herramienta Case profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Es una herramienta fácil de usar e instalar. Permite modelar todos los tipos de diagramas de clases, código inverso, generar código fuente desde diagramas y documentación en HTML/PDF.

1.11.3 Lenguaje de Modelado

Para dar solución al problema planteado se utiliza como lenguaje de modelado UML (Lenguaje Unificado de Modelado) como se plantea en el epígrafe anterior. UML, es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software (12). Además, es una notación unificada con la que se permite lograr un entendimiento entre los usuarios y los desarrolladores.

1.11.4 Entorno de Desarrollo Integrado

El entorno integrado de desarrollo (IDE) Eclipse es un entorno de desarrollo de Java sobre el que se pueden montar herramientas de desarrollo para cualquier lenguaje, mediante la implementación de módulos (plug-ins, en inglés) adecuados. La arquitectura de módulos permite que el entorno de desarrollo soporte otros lenguajes de programación además de java, así como introducir otras aplicaciones que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML como el Visual Paradigm for UML, editores visuales de interfaces, ayuda en línea para librerías, herramientas para el control de versiones y otras. Presenta un interfaz amigable, además de ser un software libre y multiplataforma.

1.11.5 Tecnología utilizada para la programación

Java, es toda una tecnología orientada al desarrollo de software con la cual se puede realizar cualquier tipo de programa. Una de las principales características que favoreció el crecimiento de Java, es ser multiplataforma, esto significa que el mismo programa escrito para Linux puede ser ejecutado en Windows sin ningún problema. Además, Java se caracteriza por ser simple, ya que elimina al 50% los errores más comunes de programación en los lenguajes C y C++ al eliminar muchas de las características de éstos,

entre las que destacan: necesidad de liberar memoria, definición de tipos, aritmética de punteros, entre otros. Además de las características mencionadas con anterioridad, constituye un lenguaje “simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico”.

1.12 Conclusiones

En este capítulo se realizó un estudio sobre la calidad de un software y los niveles de prueba por los que transita un sistema informático a lo largo de su desarrollo. Además de las herramientas que existen en el mundo y en la UCI, para automatizar la realización de pruebas de rendimiento a un software. Conjuntamente, se seleccionó la arquitectura orientada a plug-ins que presentará la herramienta y como método de evaluación de la arquitectura propuesta se seleccionó el Método ATAM. También se seleccionó OpenUP como metodología de desarrollo, Visual Paradigm for UML como herramienta CASE para la modelación visual del sistema con UML de lenguaje de modelado, Eclipse como IDE y la tecnología Java como lenguaje de programación. Se utiliza la plataforma de tareas distribuidas por ser un sistema distribuido que está desplegado actualmente en la UCI y que utiliza cientos de estaciones de trabajos para su funcionamiento.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Introducción

En el presente capítulo se expone una descripción detallada de la solución propuesta. Aborda lo referente al funcionamiento del negocio. Se desarrolla un modelo de dominio donde se analiza cada uno de los conceptos y entidades presentes en el negocio. Se plantea cada uno de los requisitos funcionales y no funcionales que debe tener la herramienta, se presenta el diagrama de CU del sistema, así como la descripción teórica de los CU.

2.1 Propuesta del Sistema

Actualmente en la UCI, en el LIPS, se realizan pruebas de rendimiento al software enfocadas en carga y estrés. Para la realización de estas pruebas, se utiliza la herramienta JMeter. La aplicación de la herramienta JMeter en el LIPS, se hace desde un único ordenador generando conexiones en hilos. La creación de hilos de ejecución consume recursos de la computadora, como la memoria RAM, de ahí que físicamente exista un límite máximo de hilos y conexiones posibles a crear en un determinado intervalo de tiempo.

Es por esto, que se define la propuesta de una herramienta que se enfoque en medir el rendimiento de un sistema en cuanto a carga y estrés inicialmente, que logre aumentar el nivel de concurrencia en el servidor para llegar más rápido al punto de ruptura de la aplicación a probar, aprovechando los recursos de cómputo de la Universidad. En esta propuesta se utilizarían varias estaciones de trabajo generando hilos de ejecución sobre un servidor. En lugar de ser 100 hilos de ejecución en un mismo ordenador cada cierto tiempo, contar con más estaciones de trabajo propiciaría que se pueda distribuir esta carga de peticiones entre ellas. De esta forma se lograría aumentar el nivel de concurrencia en el servidor de la aplicación que se desee probar, la prueba podría realizarse en menos tiempo ya que la carga sería distribuida y se llegaría más rápido al punto de ruptura de la aplicación a probar.

¿De qué forma sería posible esto?

Se utilizaría la plataforma de tareas distribuidas t-arenal, aprovechando su ambiente distribuido explicado anteriormente.

Ejemplo de la solución propuesta

Se cuenta con el Servidor Central de t-arenal, un servidor de peticiones, y 3 clientes pidiendo tareas, además del servidor de la aplicación a probar. Este servidor central de t-arenal almacena la tarea que pudiera ser, simular 150 hilos de ejecución sobre la aplicación a probar. Esta tarea es enviada al servidor de peticiones, quién se encarga de dividirla en pequeños subproblemas, (cada cliente realizará 50 hilos de ejecución) y las distribuye entre sus clientes. Cada cliente realiza el procesamiento de su tarea y envía la respuesta al servidor de peticiones y éste se encarga de coleccionar el resultado de cada uno de los subproblemas, para construir el resultado final de la tarea original, y se la envía al servidor central de t-arenal.

En total si cada PC es capaz de soportar por sus características físicas 50 hilos de ejecución, si hay 3 PC, entonces sería 3 * 50 hilos de ejecución que serían las 150 ejecuciones que no se realizarían desde un mismo ordenador, por lo tanto, se ganará en tiempo de ejecución, aumentará el nivel de concurrencia en el servidor de la aplicación a probar y las máquinas clientes no sufrirán degradación en su sistema.

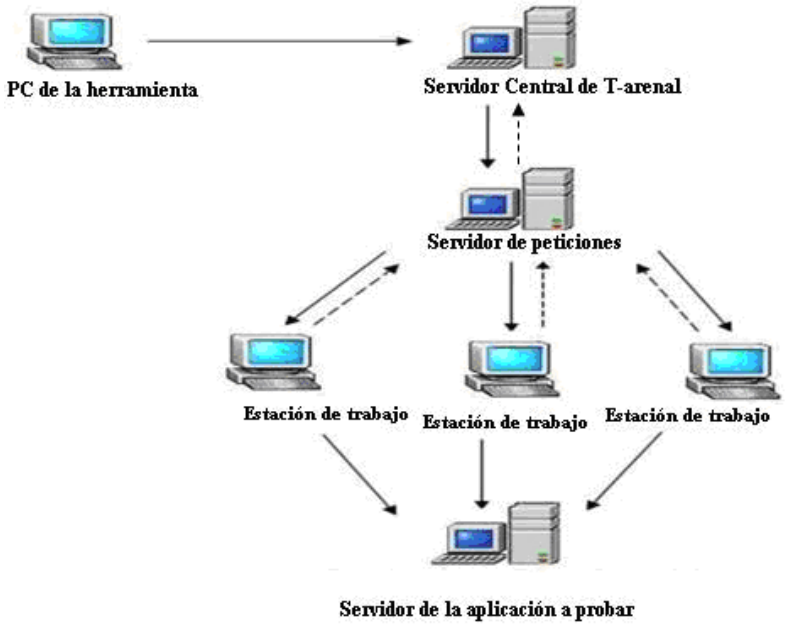

Figura 3. Solución Propuesta.

2.2. Modelo de Dominio

El modelo de dominio representa una secuencia lógica de conceptos, es decir no representa conceptos propios de un sistema de software, sino de la propia realidad física del mismo. Puede utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema.

2.2.1 ¿Por qué modelo de dominio?

Se decide realizar modelo de dominio debido a la poca estructuración de los procesos de negocio. Para poder entender en el contexto que se sitúa el sistema, se describe el funcionamiento de la herramienta mediante una serie de conceptos y sus relaciones, agrupándolos en un modelo de dominio.

2.3 Modelo de dominio

A continuación, se muestra el modelo de dominio con los principales conceptos que se identifican en el análisis de la herramienta propuesta.

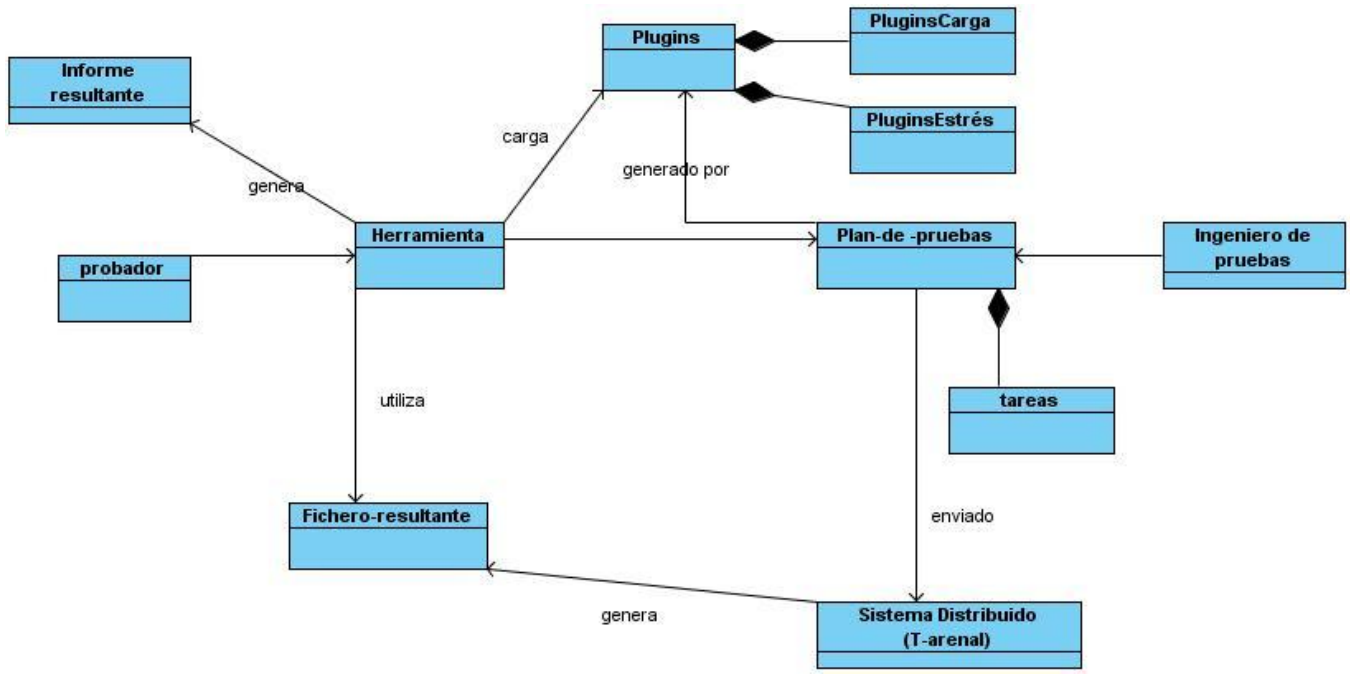

Figura 4. Modelo de Dominio

Para una mejor comprensión, a continuación se describen cada una de las clases que intervienen en el diagrama mostrado:

- ✓ **Herramienta**

Es una representación en el modelo de dominio de la herramienta que se propone. La cual va a contener todas las funcionalidades necesarias para realizar las pruebas de carga y estrés.

- ✓ **Plan de pruebas**

Representa el conjunto de funcionalidades básicas para ejecutar la prueba, de manera que éstas sean realizadas satisfactoriamente. Va a estar constituida por tareas.

- ✓ **Tareas**

Las tareas van a estar dentro del plan de prueba, que son las que van a ser enviadas al sistema distribuido, que es quién se encarga de ejecutar la tarea. Cada tarea va a contener una cantidad de usuarios a simular, una opción para escribir la URL de la aplicación que se quiere probar, entre otras opciones.

- ✓ **Sistema Distribuido**

Este sistema distribuido se conoce con el nombre de Plataforma de Tareas Distribuidas (t-arenal) explicada anteriormente, este sistema va a ser el encargado de recibir las tareas, almacenarlas y enviarlas a los **servidores de peticiones,** los cuales se encargarán de dividir la tarea en unidades de trabajo y distribuirla a los **clientes**. El servidor de peticiones, solamente podrá atender una tarea al mismo tiempo, pero además, podrá colaborar con otro servidor de peticiones en la realización de una tarea. Existirán uno o varios servidores de peticiones, según las necesidades del usuario final.

- ✓ **Informe de Resultado**

El informe de resultado lo genera la herramienta, a partir de un fichero resultado que se toma del sistema distribuido, una vez realizada la tarea.

- ✓ **Plug-ins:** Se refiere a los plug-ins que van a ser cargados sobre la herramienta, que inicialmente serán de **carga** y **estrés**.

2.4 Especificación de los Requisitos del Software.

El flujo de trabajo de requerimientos es uno de los más importantes, porque en él se establece qué es lo que tiene que hacer exactamente el sistema que se construya, de modo que los usuarios finales tienen

que comprender y aceptar los requisitos que se especifiquen. Se dividen en dos grupos: los requisitos funcionales y los requisitos no funcionales.

2.4.1 Requisitos funcionales

Una vez descrito el modelo de dominio, para poder identificar qué debe hacer el sistema y entender su funcionamiento, es fundamental conocer los requisitos funcionales que el sistema debe cumplir. Los requerimientos funcionales deben comprenderlo tanto los desarrolladores como los usuarios.

RF1. Autenticar usuario: El sistema debe permitir al usuario autenticarse.

RF2. Crear usuario: El sistema debe permitirle al administrador crear un usuario nuevo.

RF3. Eliminar usuario: El sistema debe permitirle al administrador eliminar un usuario existente.

RF4. Modificar usuario: El sistema debe permitirle al administrador modificar un usuario existente.

RF5. Crear un plan de prueba: El sistema debe permitirle al ingeniero de prueba crear un plan de prueba.

RF6. Definir tarea: El sistema debe permitirle al ingeniero de prueba definir la tarea en la herramienta.

RF7. Eliminar tarea: El sistema debe permitirle al ingeniero de prueba eliminar una tarea.

RF8. Modificar tarea: El sistema debe permitirle al ingeniero de prueba modificar una tarea ya existente.

RF9. Cargar fichero: El sistema debe ser capaz de cargar un fichero con el resultado de la prueba y generar un informe de resultado de la prueba para un mejor entendimiento de los involucrados.

RF10. Adicionar un plug-ins: El sistema debe ser capaz de adicionar plug-ins.

RF11. Eliminar plug-ins: El sistema debe ser capaz de eliminar plug-ins.

2.4.2 Requisitos no funcionales

Una vez analizado los requisitos funcionales, se hace necesario analizar los requisitos no funcionales, que no son más que las propiedades o cualidades que el producto debe tener. Entre los requisitos no funcionales del sistema se encuentran:

Apariencia o interfaz externa

- ✓ El sistema debe contar con una interfaz amigable, donde el usuario pueda orientarse fácilmente.
- ✓ Cada rol tendrá una interfaz diferente con las funciones que le corresponden.

Usabilidad

- ✓ La herramienta tendrá un ambiente sencillo y será fácil de manejar para los usuarios.
- ✓ La herramienta estará dirigida a usuarios con un nivel medio o superior de informática.

Fiabilidad

- ✓ La interacción con el sistema, estará sometida a un proceso de autenticación del usuario.
- ✓ Deben implementarse mecanismos para garantizar la respuesta ante posibles fallos ya sean fallos de red, electricidad, apagado de máquinas, entre otros que puedan surgir, además de posibles fallos a la hora de cargar los plug-ins sobre el sistema.

Seguridad

- ✓ Confidencialidad: Se requiere de usuario y contraseña para poder acceder a la información de la herramienta.
- ✓ Disponibilidad: En caso de tener el usuario y la contraseña, se le garantiza poder acceder a la herramienta en todo momento dependiendo del tipo de usuario que se autentique.
- ✓ Integridad: En dependencia del usuario que se autentique serán los permisos que tendrá dentro de la aplicación y las acciones a las que tendrá acceso de forma que todo cambio o modificación en el sistema será responsabilidad de un usuario según su rol.

Eficiencia

- ✓ La carga de plug-ins sobre el sistema debe realizarse satisfactoriamente.
- ✓ La solución a los problemas se obtiene más rápidamente haciendo uso del sistema distribuido, que la respuesta que daría una simple computadora.

Portabilidad

- ✓ El sistema es multiplataforma, razón por la cual podrá ser utilizado en Windows o Linux.

Software

- ✓ Debe estar instalada la máquina virtual de Java SDK 1.5.x o superior.
- ✓ Tiene que poder funcionar la plataforma de tareas distribuidas.

2.5 Modelo de Casos de Usos.

2.5.1 Actores del sistema

Los actores representan a terceros fuera del sistema que interactúan con él. En el sistema que se describe se identificaron los siguientes actores:

Tabla 2. Descripción de los actores del sistema a automatizar.

Actores	Descripción
Ingeniero de pruebas	Representa al usuario que va a construir el plan de pruebas.
Probador	Representa al usuario que va a ejecutar la prueba.
Usuario	Es el usuario encargado de la autenticación en el sistema.
Administrador	Es el usuario encargado de las actividades administrativas.

2.5.2 Casos de uso del sistema

Los casos de uso del sistema que aparecen a continuación, tienen como objetivo satisfacer los requisitos funcionales descritos con anterioridad. Para ello, se utilizó el patrón de casos de uso CRUD, que gestiona la información de crear, eliminar, modificar y buscar. El patrón CRUD puede ser parcial o completo dependiendo de la totalidad de la información gestionada. Para definir los casos de uso y confeccionar el diagrama de casos de uso se utilizó el CRUD parcial en el caso de uso Administrar tarea, donde se agrupan las funcionalidades eliminar y modificar tarea y en el caso de uso Gestionar usuario, donde se agrupan las funcionalidades de crear, eliminar, y modificar usuario.

1. **CU Autenticar usuario.**
2. **CU Gestionar usuario.**
3. **CU Crear plan de prueba.**
4. **CU Definir tarea.**
5. **CU Administrar tarea.**
6. **CU Cargar fichero.**
7. **CU Adicionar un plug-ins.**
8. **CU Eliminar un plug-ins.**

2.5.3 Diagrama de Casos del sistema a automatizar.

En el siguiente diagrama se representa la relación entre los actores y los casos de uso del sistema.

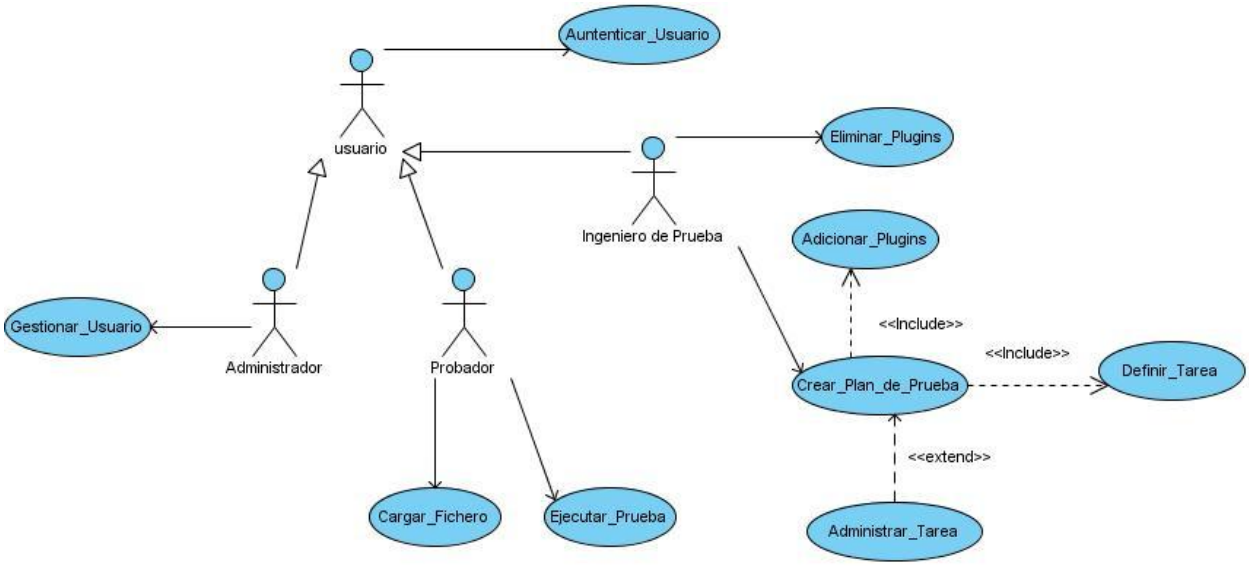

Figura 5. Diagrama de Caso de Uso del sistema

2.5.4 Descripción de los casos de uso del sistema más significativos.

A continuación se describen los casos de uso más importantes del sistema.

Tabla 3. Descripción del Caso de uso Autenticar usuario

Caso de Uso:	Autenticar usuario
Actores:	Usuario (administrador, probador, ingeniero de prueba).
Resumen:	El caso de Uso inicia cuando un usuario decide interactuar con el sistema. El sistema realiza las verificaciones de los datos enviados y finalmente lo acepta o lo rechaza. Termina el caso de uso
Precondiciones:	El usuario debe tener acceso a realizar cualquier cambio a la hora de autenticar un usuario.
Referencias	RF1
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1. El actor se autentica en el sistema.	1.1. El sistema verifica que el usuario exista.
	1.2. El sistema verifica que la contraseña del usuario es correcta.
	1.4 El sistema muestra la sesión del usuario.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	1.1. Si el usuario no existe, el sistema emite un mensaje de error “usuario incorrecto”.
	1.2. Si la contraseña es incorrecta, el sistema emite un mensaje de error “contraseña incorrecta”.
Poscondiciones	Queda autenticado el usuario en el sistema.

Tabla 4. Descripción del caso de uso Adicionar plug-ins.

Caso de Uso:	Adicionar plug-ins
Actores:	Ingeniero de prueba
Resumen:	El caso de uso inicia cuando el usuario decide adicionar un plug-ins sobre la herramienta.
Precondiciones:	El actor debe tener todos los permisos pertinentes para realizar esta acción.
Referencias	RF11
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El actor selecciona la opción "adicionar plug-ins" en la herramienta.	1.1. El sistema muestra una ventana para que el actor seleccione la ruta donde se encuentra el plug-ins.
2. El actor selecciona el plug-in y selecciona el botón aceptar para que sea cargado en la herramienta.	2.1. El sistema carga el plug-in sobre la herramienta mostrando un mensaje “el plug-ins ha sido cargado correctamente en la herramienta”.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	2.1. En caso de existir un error el sistema muestra el mensaje “vuelva a cargar el plug-ins sobre la herramienta porque ocurrió un error”.
Poscondiciones	Se adiciona un plug-ins.

Tabla 5. Descripción del caso de uso Definir Tarea.

Caso de Uso:	Definir tarea
Actores:	Ingeniero de Prueba
Resumen:	El caso de uso una vez cargado el plug-ins en la herramienta. El actor llena los campos que muestra el sistema para definir la tarea. Termina el caso de uso.
Precondiciones:	Debe haberse autenticado correctamente y tener los permisos

	pertinentes para realizar dicha acción.
Referencias	RF6
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1 El ingeniero de prueba inicia la acción de definir tarea una vez cargado el plug-ins sobre el sistema.	1.1. El sistema muestra un panel con los campos a llenar por parte del ingeniero de prueba tales como: - nombre de la prueba. - fecha de realización de la prueba. - tiempo de arranque de la prueba. - tiempo de finalización de la prueba. -rango de IP de las PC a utilizar. - número de peticiones a ejecutar. - URL de la aplicación a probar. -Tipo de plug-in.
2. El ingeniero de prueba llena los datos del panel y selecciona el botón guardar_ enviarTarea que se encargará de enviar la tarea al sistema distribuido y guardarla en la herramienta con nombre, fecha y hora en que se realizó.	2.1. El sistema envía y guarda los datos. Termina el caso de uso.

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	2.1. En caso de error el sistema muestra un mensaje “Debe volver a llenar los datos”.

Poscondiciones	Queda definida la tarea a realizar.

Tabla 6. Descripción del caso de uso Cargar Fichero.

Caso de Uso:	Cargar fichero
Actores:	Probador
Resumen:	El caso de uso se inicia una vez que el probador carga el fichero sobre la herramienta para devolver un informe resultante de la prueba realizada. Termina el Caso de Uso.
Precondiciones:	El actor debe tener los permisos pertinentes para realizar esta acción.
Referencias	RF9
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El actor selecciona la opción Cargar Fichero del menú de la herramienta.	1.1. El sistema muestra una ventana con las tareas enviadas al sistema distribuido.
2. El actor selecciona la tarea de la que quiere ver el resultado y selecciona el botón. Conectar_SD.	2.1. El sistema se conecta al sistema distribuido y descarga el fichero de la tarea seleccionada.
3. El actor selecciona el fichero y lo descomprime.	3.1. El sistema guarda el fichero con los mismos datos de la tarea a la que pertenece.
4. El actor elige el fichero y selecciona el botón GenerarInformeResultado.	4.1. El sistema muestra un informe con los resultados de la prueba los cuales estarán dados por: - Tiempo que demora en realizar los hilos de ejecución. - Cantidad de hilos que se realizaron

	sin fallos sobre la aplicación a probar. - Tiempo de finalización de la prueba.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	2.1. En caso contrario el sistema muestra un mensaje “Error en la conexión al sistema distribuido, vuelva a realizar la operación”.
Poscondiciones	Se carga el fichero sobre la herramienta y se muestra un informe de resultado de la prueba realizada.

2.6 Conclusiones

En este capítulo se han descrito una serie de conceptos y entidades para la modelación del dominio, se han visto las relaciones o vínculos que existen entre cada uno de estos, mediante el diagrama del modelo de dominio. Se definieron los actores que interactúan con el sistema. Se tuvieron en cuenta también, las funcionalidades que debe tener el software y las cualidades del mismo planteadas como requisitos funcionales y requisitos no funcionales respectivamente. Se realizó una descripción de cada uno de los casos de uso contemplados en el diagrama de casos de uso del sistema.

CAPITULO 3: PROPUESTA DE LA ARQUITECTURA

Introducción

Este capítulo describe la representación arquitectónica del sistema, haciendo énfasis en el patrón de arquitectura utilizado y el tipo de arquitectura de la herramienta, así como una propuesta de los patrones de diseño que se evidencian inicialmente en la arquitectura propuesta.

3.1 Representación arquitectónica

La misión principal de la arquitectura del sistema es mostrar una panorámica general de los elementos que lo integran, para ello se utilizan las vistas. El modelo de representación arquitectónico está basado en el modelo de las 4 + 1 vistas. Se describirá la Vista de Casos de Uso y una propuesta de la Vista Lógica y Vista de Despliegue, teniendo en cuenta el alcance del trabajo de diploma.

3.1.1 Patrón arquitectónico

En la figura 6, se muestra el patrón utilizado cliente-servidor entre la herramienta y el sistema distribuido. La *PC_herramienta* y los *Clientes_t-arenal* actúan como cliente. El *Servidor-t-arenal* siempre se comporta como servidor, en el momento de enviarle la respuesta de la tarea a la PC_herramienta y para darle una tarea al *Servidor de Peticiones*. El Servidor de peticiones se comporta como cliente y como servidor, como cliente, cuando solicita una tarea al *Servidor_t-arenal* y como servidor cuando distribuye la tarea a los *clientes_t-arenal* que éstos solicitan.

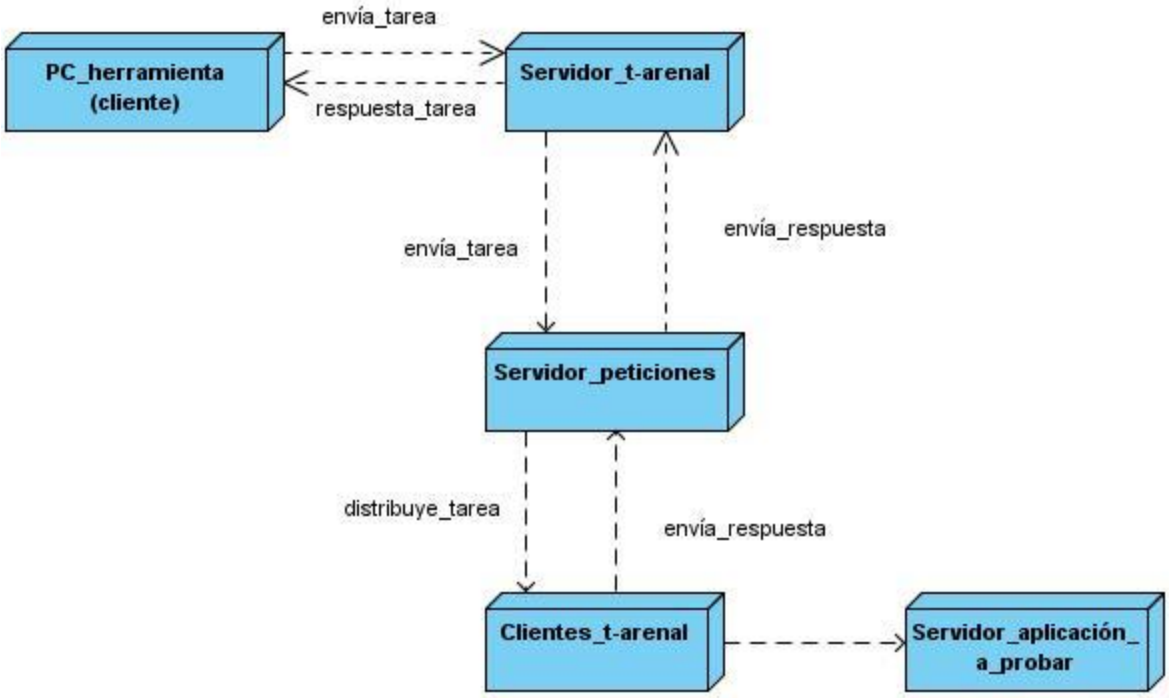

Figura 6. Patrón arquitectónico cliente – servidor aplicado al sistema

3.1.2 Propuesta de patrones de diseño.

A continuación, se proponen los patrones de diseño que se tienen en cuenta para la arquitectura propuesta.

Experto: Consiste en asignar la responsabilidad al experto en información, la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Este patrón se podría utilizar en la herramienta en la clase principal, donde se van a implementar todos los métodos importantes para el funcionamiento de la herramienta, esta clase llevará el nombre de *CPrincipal*.

Creador: Guía la asignación de responsabilidades relacionadas con la creación de objetos.

Se asigna la responsabilidad de que una clase B cree un objeto de la clase A. La clase B crea las instancias de A si:

- B agrega los objetos de A
- B contiene los objetos de A
- B registra las instancias de los objetos de A.

- B tiene los datos de inicialización que serán enviados a la clase A cuando este objeto sea creado.

Este patrón se manifiesta en la clase principal de la herramienta *Cprincipal* ya que ella va a instanciar a las clases que necesite para la implementación de los métodos principales.

Bajo acoplamiento: El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, las conoce y recurre a ellas. Delimita pocas relaciones entre clases, de forma tal, que en caso de producirse una modificación en alguna de ellas se tenga la mínima repercusión posible en el resto de las clases.

Este patrón se evidencia ya que no existe una estrecha relación entre las clases de la herramienta con las clases que responden a los plug-ins, un cambio en las clases de la herramienta no afectará a las clases que demuestran las funciones de los plug-ins.

Alta cohesión: La cohesión es una medida de cuán 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 enorme. Una clase con alta cohesión tiene la responsabilidad de realizar una única labor identificable en el sistema.

Este patrón está enmarcado en la herramienta debido a que cada clase tendrá sus responsabilidades en la herramienta. No existirá gran dependencia entre clases.

Fachada: El patrón fachada simplifica el acceso a un conjunto de clases proporcionando una única clase que todos utilizan para comunicarse con dicho conjunto de clases. Este patrón de diseño se puede implementar como una interfaz o como una clase abstracta, sin especificar los detalles de implementación. Este patrón se evidencia en la herramienta mediante la clase interfaz, la cual sirve de fachada entre la herramienta y los plug-ins.

3.2 Arquitectura orientada a plug-ins.

La herramienta se desarrollará bajo la utilización de una arquitectura orientada a plug-ins. La utilización de esta, permite añadir nuevos plug-ins a la herramienta de modo que puedan crear o personalizar sus funcionalidades propias. La herramienta estará enfocada en la realización de pruebas de rendimiento mediante la carga de plug-ins, inicialmente estará orientada a carga y estrés, pero en un futuro si se quisiera realizar otros tipos de pruebas de rendimiento se adicionarían plug-ins con sus funcionalidades propias.

3.3 Vistas arquitectónicas

Las vistas de la arquitectura constituyen una síntesis de los modelos que están en la línea base de la arquitectura. Una vista, es la representación de un conjunto coherente de elementos arquitectónicos y sus relaciones, en este sentido la vista describe parte de la arquitectura del sistema. A continuación, se representa la vista de caso de uso de la arquitectura propuesta.

3.3.1 Vista de Caso de Uso

La vista de casos de uso contiene los casos de usos más significativos para la arquitectura, ya que describen las funcionalidades más importantes y críticas que deben realizarse con mayor prioridad, la cual se representa a continuación:

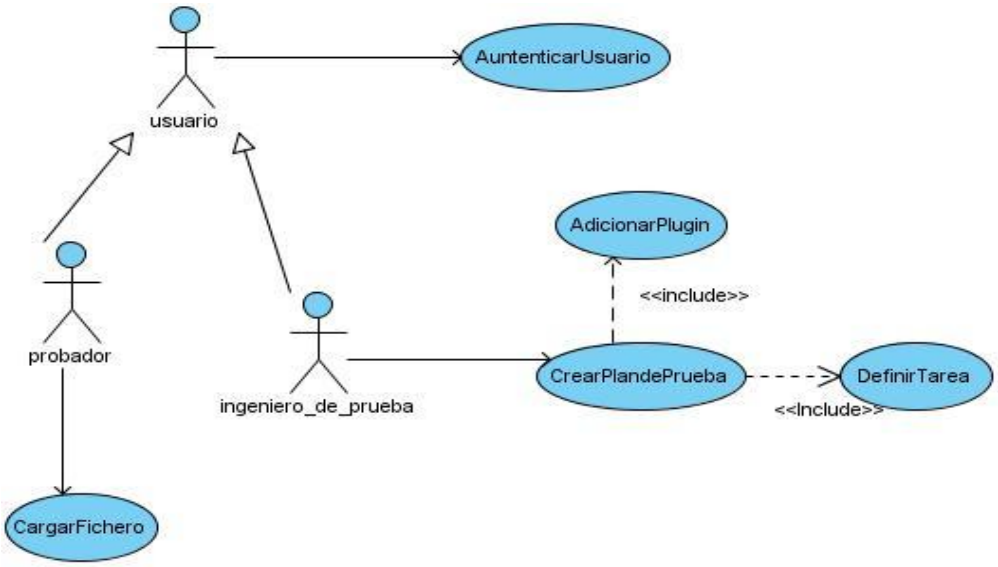

Figura 7. Vista de Caso de Uso

3.3.2 Propuesta de la vista lógica

La vista lógica, contiene las clases del diseño más importantes, organizadas por paquetes y subsistemas en capas de trabajo. Esta vista describe los paquetes en los que se dividirá la herramienta. En la figura siguiente, se muestra una propuesta de la vista lógica de la herramienta.

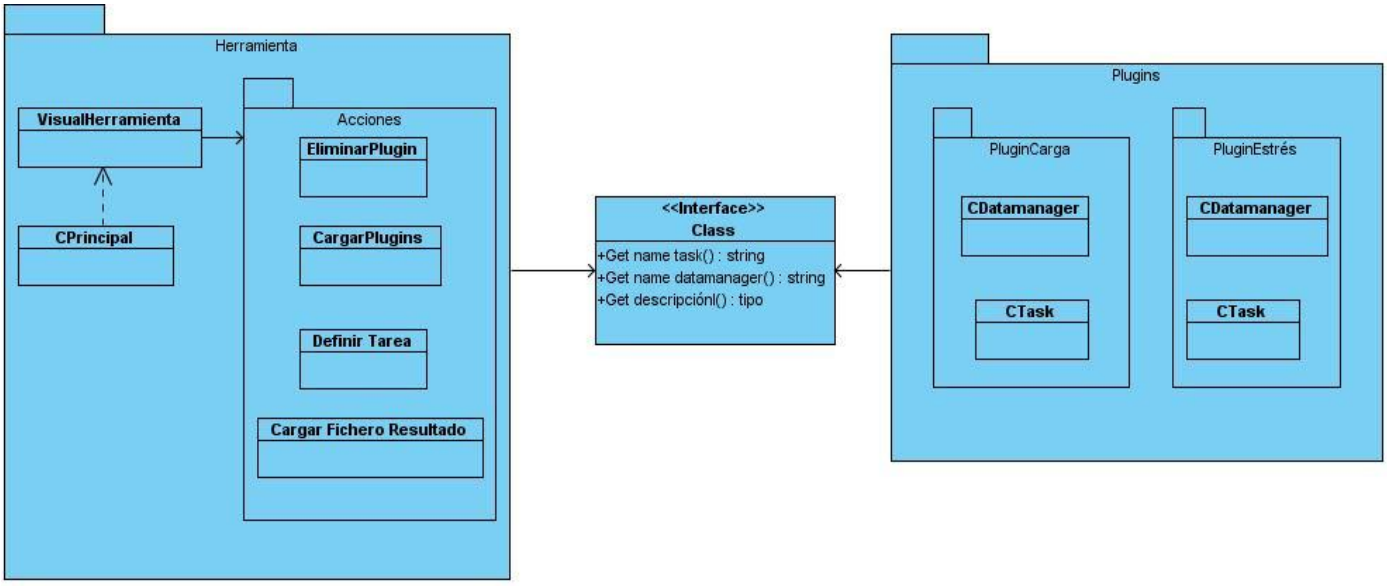

Figura 8. Propuesta de la vista lógica.

Herramienta: Este subsistema está compuesto por la clase *CPrincipal* que va a contener los métodos principales de la herramienta, esta clase va a estar relacionada con la clase *VisuaHerramienta* la cual va a contener las *acciones* fundamentales para que el sistema funcione.

Plug-ins: Este paquete está compuesto por varias clases que permiten la funcionalidad del plugin, entre ellas, están las clases principales que permitan la iteración con el sistema distribuido, estas clases son *CDatamanager* y *Ctask* que se van a implementar según el tipo de plug-ins que se adicione en la herramienta, inicialmente serán plugin de carga y estrés.

Interface: Esta clase va a funcionar de intermediaria entre la herramienta y los plug-ins. Va a contener los métodos comunes entre la herramienta y los plug-ins para enviar la tarea al sistema distribuido.

3.3.3 Propuesta de la vista de despliegue

La vista de despliegue describe los principales nodos físicos que se necesitan para desplegar el sistema y la relación entre ellos mediante los protocolos de comunicación. En la figura 9, se muestra una propuesta de la vista de despliegue de la herramienta.

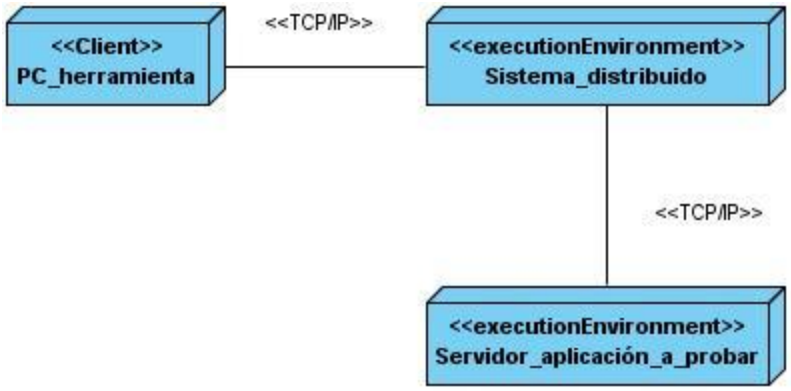

Figura 9. Propuesta de la vista de despliegue

PC herramienta

Se refiere a la PC cliente donde estará la herramienta que contendrá las pruebas, que serán enviadas al sistema distribuido.

Sistema_Distribuido

Es quien se encargará de ejecutar la tarea recibida por la herramienta.

Servidor_Aplicación_a_probar

Se refiere al servidor donde se encontrará el software que se desea probar.

Protocolo de Comunicación

Conexión TCP/IP: Es utilizado para establecer la conexión entre la PC herramienta y el servidor de sistemas distribuidos. Además de la conexión entre el sistema distribuido y el servidor de la aplicación a probar.

3.4 Estándares de codificación

Un estilo de codificación puede describirse, como reglas generales que indican como construir técnicamente un software, permitiendo una mejor comunicación entre los desarrolladores, diseñadores de bases de datos y programadores. Es importante que una vez definidos los estilos de codificación sean cumplidos y desarrollados, ya que así facilita la comprensión de los programas, disminuyendo el número de errores durante el desarrollo y mantenimiento de un software.

Las reglas que define el estilo de codificación adoptado en el presente trabajo se exponen íntegramente en el Anexo 1.

3.5 Aplicación del método de evaluación propuesto

Un método de evaluación, sirve de guía a los involucrados en el desarrollo del sistema, en la búsqueda de conflictos que puede presentar una arquitectura, y sus soluciones. Por esta razón, resulta conveniente evaluar la arquitectura de software propuesta hasta el momento, para ello se hace uso del Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM). Este método comprende nueve pasos, agrupados en cuatro fases (presentación, Investigación y Análisis, Prueba y reporte).

Fase1: Presentación

El equipo de evaluación presenta un panorama general de los pasos de ATAM, trata de establecer las expectativas, las técnicas utilizadas de los resultados del proceso y qué características de calidad se esperan lograr.

Paso 1: Presentación del método: El director de evaluación presenta el método ante los miembros del proyecto, se responden preguntas y se establecen las expectativas y el contexto sobre las actividades o tareas restantes. Este paso tiene como finalidad que todos los stakeholders comprendan la secuencia de los pasos a seguir, los instrumentos utilizados y las salidas del método y finalmente presentar la arquitectura que debe ser evaluada.

Paso 2: Presentación de las metas del negocio: Se realiza la descripción de las metas del negocio que motivan el esfuerzo, cada persona que posee una responsabilidad sobre la evaluación, ya sean representantes del proyecto o miembros del equipo, necesitan comprender el contexto del sistema y los aspectos primarios del contexto del negocio que motivan su desarrollo. En este paso, se explica a los stakeholders el contexto del sistema y los requerimientos del negocio dentro del cual va a funcionar el sistema. Algunos tópicos que debería contener esta presentación son: las funcionalidades más importantes del sistema, los objetivos del negocio y el contexto relacionado con el proyecto, las directrices arquitectónicas (Requerimientos de calidad a ser satisfechos por la arquitectura), entre otros.

En este paso, el líder del proyecto presenta el sistema desde la perspectiva del negocio, donde quedan expuestos como principales funcionalidades del sistema:

1. Autenticar usuario.
2. Gestionar usuario.
3. Crear plan de prueba
4. Definir tarea
5. Administrar tarea.
6. Cargar fichero
7. Adicionar plug-ins
8. Eliminar plug-ins

Como restricciones del sistema, relevantes arquitectónicamente, quedaron identificadas:

1. Debe estar instalada la máquina virtual de Java SDK 1.5.x.
2. Tiene que funcionar la plataforma de tareas distribuidas.

Como metas de los atributos de calidad que dan forma a la arquitectura:

1. Para poder acceder a la herramienta, se requiere de autenticarse en el sistema con usuario y contraseña válidos. Además, dependiendo de las credenciales del usuario se deshabilitarán o habilitarán opciones de la herramienta para proteger información de la misma.
2. Lograr la habilidad del sistema a mantenerse operativo a lo largo del tiempo.
3. Lograr la habilidad del sistema para ser ejecutado en diferentes ambientes computacionales.

Paso 3: Presentación de la arquitectura: En este paso, el arquitecto describe la arquitectura, enfocándose en como cumple con los objetivos del negocio. Este paso es desarrollado por los arquitectos e incluye toda la descripción arquitectónica en detalle, no se muestra en detalles la arquitectura que está siendo evaluada ya que anteriormente en el capítulo se mostró completamente.

Fase 2: Investigación y análisis

En esta fase, se identifican elementos de diseño que ayuden a analizar la arquitectura. Se identifican las propuestas arquitectónicas, además se crea el árbol de utilidad a partir de los atributos de calidad que se tuvieron en cuenta en relación con los escenarios definidos.

Paso4: Identificación de los enfoques arquitectónicos: A partir de aquí, quedan identificadas todas las propuestas arquitectónicas que serán analizadas, se especifica la tecnología que va a ser utilizada y el patrón arquitectónico, estos elementos no se muestran a continuación ya que se encuentran detallados en el capítulo1.

Paso5: Generación del Árbol de Utilidad: Se obtienen los atributos de calidad que engloban la "utilidad" del sistema especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
El árbol de utilidad se prioriza en dos dimensiones:
1. Por la importancia que cada escenario tiene para el éxito del sistema.
2. Por el grado de dificultad que posee el escenario para ser realizado, según la estimación del arquitecto.
Frecuentemente la escala utilizada para ambas dimensiones es High, Medium, Low, pero en este caso se utilizará Alta (A), Media (M) y Baja (B). Los escenarios son utilizados para probar el sistema, optimizando las oportunidades de que aparezcan riesgos en las decisiones arquitectónicas. Terminado el paso actual se obtuvo el siguiente Árbol de Utilidad:

Tabla 7. Árbol de utilidad.

Árbol de Utilidad			
Atributo de calidad	**Subcaracterística**	**Escenario**	**Prioridad**
Funcionalidad	Seguridad	Las funcionalidades del sistema se muestran de acuerdo al usuario que esté activo.	A,M
		El sistema debe restringir el acceso a datos.	A,M

		Tener protección contra ataques externos que puedan afectar la integridad de los datos almacenados.	A,A
		Autenticar los usuarios para permitir el uso adecuado de la herramienta.	A,B
	Adecuación	El sistema cumple con los RF y RNF.	A,B
	Interoperabilidad	La herramienta debe ser capaz de interactuar con el sistema distribuido.	A,M
Confiabilidad	Tolerancia a fallos & Recuperación	Si al enviarse una tarea al sistema distribuido ocurre un error el sistema muestra un mensaje al usuario que vuelva a enviar la tarea.	A,M
		Si al cargarse un plug-ins sobre la herramienta ocurre un error el sistema muestra un mensaje que vuelva a cargar el plug-ins.	A,M
Usabilidad	Entendibilidad, Capacidad de aprendizaje y Operabilidad	El sistema debe contar con una interfaz amigable, fácil de comprender, donde el usuario pueda orientarse fácilmente	M,B
		La herramienta tendrá un ambiente sencillo y será fácil de manejar para los usuarios.	M,B

		El sistema debe someterse a una etapa de adiestramiento en la que los usuarios se familiaricen con la herramienta y sean detectados los posibles errores para ser corregidos.	M,M
Eficiencia	Comportamiento de recursos	La carga de plug-ins sobre el sistema debe realizarse satisfactoriamente.	M,B
Mantenibilidad	Estabilidad Capacidad de pruebas	La herramienta debe estar configurada para adicionar plug-ins de carga y estrés.	A,M
Portabilidad	Adaptabilidad, Instalabilidad Reemplazabilidad	Desplegar la herramienta en los sistemas operativos Windows y GNU/Linux.	A,M

Paso6. Análisis de los enfoques arquitectónicos: En este paso se obtiene el Anexo 2, está basado en los escenarios identificados en el paso 5 y las propuestas arquitectónicas que cumplen los escenarios.

Fase 3: Pruebas

Paso 7. Tormenta de ideas y establecimiento de prioridad de escenarios: Con la participación de todos los involucrados, se analiza y se complementa el conjunto de escenarios y se le da prioridad a los escenarios sobre la base de su importancia.

Los escenarios creados en el árbol de utilidad fueron comparados con la lista de escenarios resultantes en la tormenta de ideas, recogidos en el Anexo 3, y al descubrir nuevos escenarios fueron adicionados al árbol de utilidad, obtenido en el paso 5. El nuevo árbol de utilidad que recoge todos los escenarios identificados durante el proceso de evaluación, constituye una de las salidas del método de evaluación, Anexo 4.

Paso 8. Análisis de los enfoques arquitectónicos: Este paso, repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento. Este paso tiene como objetivo documentar los nuevos escenarios obtenidos en el paso 7, representados en el Anexo 2.

Fase 4: Reporte

Paso 9. Presentación de los resultados: Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Finalmente, la información recogida durante el proceso de evaluación del ATAM se resume y presenta, las más importantes salidas son:

El conjunto de escenarios priorizados: Anexo 2.

El árbol de utilidad: Anexo 4.

Los riesgos descubiertos: Anexo 5.

Los no riesgos descubiertos: Anexo 6.

Los puntos de sensibilidad y de compromiso (trade-off) encontrados: Anexo 7.

3.8 Conclusiones

En este capítulo se define el modelo de representación arquitectónica describiendo la vista de casos de uso, una propuesta de la vista lógica y vista de despliegue de acuerdo a la importancia que tiene para la comprensión del sistema. Se concluye con la evaluación de la arquitectura, dando cumplimiento a las tareas propuestas. La evaluación de la arquitectura permitió descubrir las decisiones arquitectónicas que constituyen fortalezas o puntos débiles así como documentar detalladamente cómo estas decisiones responden a los atributos de calidad. Además, quedaron identificados y argumentados durante la evaluación, los riesgos, no riesgos, los puntos de sensibilidad y los puntos de compromiso (trade-off).

CONCLUSIONES GENERALES

Con el desarrollo del presente trabajo se llega a las siguientes conclusiones:

- ✓ Se realizó el análisis de una herramienta en java, para realizar pruebas de rendimiento a aplicaciones cliente-servidor utilizando La plataforma de Tareas Distribuidas.
- ✓ Se definió la propuesta de la arquitectura de la herramienta.
- ✓ Se realizó la evaluación de la propuesta de la arquitectura mediante el método de Análisis de Acuerdos de Arquitectura (ATAM), donde se obtuvo como resultado 11 puntos de sensibilidad, 3 riesgos, 9 no riesgos y 2 puntos de compromiso.

RECOMENDACIONES

- ✓ Se recomienda estudiar las decisiones arquitectónicas que constituyen riegos según el método de evaluación ATAM y valorar su sustitución por otras decisiones que no pongan en peligro la herramienta, como posible mejora a la solución dada en el presente trabajo.
- ✓ Se recomienda realizar el diseño e implementación de la herramienta.

REFERENCIAS BIBLIOGRÁFICAS

1. **Pressman, Robert S.** *Ingeniería del software. Un enfoque práctico.* 5ta edición. Capítulo 8.

2. **IEEE.** Computer Dictionary. Computer Society. 1990.

3. **Vázquez, Roberto Hugo.** Introducción a la calidad de software. [Online] http://gridtics.frm.utn.edu.ar/docs/Introduccion%20a%20la%20Calidad%20de%20Software%20Vazquez.pdf.

4. **Pressman, Roberto S.** *Ingeniería del software.Un enfoque práctico.* 5ta . Capítulo 18.

5. **Taset, Cesar Rául Jacas y Daniel Marino Miralles.** *T_arenal v2.0: Desarrollo del bak-end.* 2009. tesis. Disponible en: http://biblioteca.uci.cu/sbd/biuci/index.html.

6. **Hilliard, Rich.** *Recommended Practice for Architectural Description of Software-Intensive Systems.* 2000. Ubicado en: http://www.enterprise-architecture.info/Images/Documents/IEEE%201471-2000.pdf. IEEE-Std-1471.

7. wikipedia. [Online] http://es.wikipedia.org/wiki/Complemento_(inform%C3%A1tica.

8. **Gómez, Omar Salvador.** *Evaluando arquitecturas de software. Parte1.* 2007.

9. **Gustavo Andrés Brey, Gastón Escobar, Nicolas Passerini y Juan Arias.** *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.* Buenos Aires: Universidad Tecnológica Nacional. : s.n., 2005. Publicado en: http://apit.wdfiles.com/local--files/start/05 apit evaluacion.pdf.

10. **Brey, Gustavo Andrés, et al.** *Arquitectura de proyectos de IT.Evaluación de Arquitectura.* http://apit.wdfiles.com/local--files/start/05_apit_evaluacion.pdf.

11. **Kasman, Lawrence and Clements, Lawrence.** *Evaluating Software Architectures. Methods and case studies.* 2001.

12. **Pressman, Roger S.** *Ingeniería del software:Un enfoque práctico.* La habana : s.n., 2005. Parte 1.

13. **Rumbaugh, James and Boosh, Gary.** *E luenguaje unificado de modelado.*

BIBLIOGRAFÍA

1. **Pressman, Robert S.** *Ingeniería del software. Un enfoque práctico.* 5ta edición. Capítulo 8.

2. **IEEE.** Computer Dictionary. Computer Society. 1990.

3. **Vázquez, Roberto Hugo.** Introducción a la calidad de software. [Online] http://gridtics.frm.utn.edu.ar/docs/Introduccion%20a%20la%20Calidad%20de%20Software%20Vazquez.pdf.

4. **Pressman, Roberto S.** *Ingeniería del software.Un enfoque práctico.* 5ta . Capítulo 18.

5. **Taset, Cesar Rául Jacas y Daniel Marino Miralles.** *T_arenal v2.0: Desarrollo del bak-end.* 2009. tesis. Disponible en: http://biblioteca.uci.cu/sbd/biuci/index.html.

6. **Hilliard, Rich.** *Recommended Practice for Architectural Description of Software-Intensive Systems.* 2000. Ubicado en: http://www.enterprise-architecture.info/Images/Documents/IEEE%201471-2000.pdf. IEEE-Std-1471.

7. wikipedia. [Online] http://es.wikipedia.org/wiki/Complemento_(inform%C3%A1tica.

8. **Gómez, Omar Salvador.** *Evaluando arquitecturas de software. Parte1.* 2007.

9. **Gustavo Andrés Brey, Gastón Escobar, Nicolas Passerini y Juan Arias.** *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.* Buenos Aires: Universidad Tecnológica Nacional. : s.n., 2005. Publicado en: http://apit.wdfiles.com/local--files/start/05 apit evaluacion.pdf.

10. **Brey, Gustavo Andrés, et al.** *Arquitectura de proyectos de IT.Evaluación de Arquitectura.* http://apit.wdfiles.com/local--files/start/05_apit_evaluacion.pdf.

11. **Kasman, Lawrence and Clements, Lawrence.** *Evaluating Software Architectures. Methods and case studies.* 2001.

12. **Pressman, Roger S.** *Ingeniería del software:Un enfoque práctico.* La habana : s.n., 2005. Parte 1.

13. **Rumbaugh, James and Boosh, Gary.** *E luenguaje unificado de modelado.*

14. **Díaz, José.** *Introducción al proceso de pruebas.*

15. **Tuya, Javier.** *Las pruebas de software.* 2007.

16. **Raja Prado, Elena.** *Pruebas de software.* 2007. Vol. 1.

17. **Polo Prado, Macario.** *Pruebas de software.* Real : s.n.

18. **Clements, Paul.** *Comming Attractions in software architecture in practice.* 2da. 1996.

19. **Camacho, Erika, Cardeso, Fabio and Núñez, Gabriel.** *Arquitectura de software.* 2009. En linea en: Http://Prof.Usb.Ve/Lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.Pdf].

20. **Reinoso, Billy.** *Introducción a la arquitectura de software.*

21. **Cristiá, Maximiliano.** *Introducción a la arquitectura de software.* 2008.

22. **Gónzales, Alexander, et al.** *Método de evaluación de arquitecturas de software basadas en componentes(MECABIC).* 2005. Vol. 1, Publicado en: http://www.lisi.usb.ve/publicaciones/03%20evaluacion/evaluacion_09.pdf.

ANEXOS

Anexo1. Propuesta de estilos de codificación

Declaración de variables:

Codificar cada variable con un nombre que se va a relacionar con el contenido de la misma.

Las variables deben comenzar con minúscula, en caso de nombres compuestos deben estar separados por underscore (_).

En el caso de las variables que constituyen instancias de clases persistentes deben llamarse del mismo modo que la clase que representa empezando con minúscula.

Nombre de métodos:

El nombre debe describir lo más posible la funcionalidad que va a realizar.

Deben empezar con mayúscula y en caso de nombres compuestos debe estar separado por underscore (_).

Corrección de errores:

Si se encuentran errores, estos deben ser corregidos inmediatamente donde se encontraron y no luego de haber sido implementada la funcionalidad donde se encontró dicho error.

Regla:

Incluir comentarios para cada línea de código que no sea entendible por sí sola.

Realizar una programación orientada a objetos.

Líneas en blanco:

Entre declaración de métodos; declaración de la variable y declaración de primer método.

Nombre de clases:

El nombre de las clases debe empezar con la primera letra inicial mayúscula.

Anexo 2. Salidas del método de evaluación ATAM: Análisis de propuestas arquitectónicas.

Escenario#1	Las funcionalidades del sistema se muestran de acuerdo al

	usuario que esté activo.			
Atributo(s)	Funcionalidad-Seguridad			
Ambiente	Operación normal			
Estímulo	El usuario interactúa con sistema.			
Respuesta	Solamente se muestran las funcionalidades a las que tiene acceso según su tipo de usuario.			
Decisiones arquitectónicas	**Riesgo**	**Punto de sensibilidad**	**Punto de trade-off**	**No riesgo**
El usuario tiene que autenticarse en el sistema.		**S1**		**N1**
Explicación	Cada usuario dentro del sistema cumple un rol, y a partir de esto puede realizar funcionalidades independientes.			

Escenario#2	El sistema debe restringir el acceso a datos.			
Atributo(s)	Funcionalidad-seguridad			
Ambiente	Operación normal			
Estímulo	Un usuario determinado interactúa con el sistema.			
Respuesta	El sistema verifica que el usuario que interactúa tengo permiso para acceder a los datos.			
Decisiones arquitectónicas	**Riesgo**	**Punto de sensibilidad**	**Punto de trade-off**	**No riesgo**
Una vez autenticado el usuario, el sistema verifica que el usuario exista para acceder a los datos.		**S2**		**N2**
Explicación	El sistema lleva el control de todos los usuarios que interactúan con él y los permisos según el rol del usuario. Solamente tendrá acceso a todo tipo de datos el administrador de sistema, los demás usuarios tienen acceso restringido según la función que			

	desempeñen en él.

Escenario#3	Tener protección contra ataques externos que puedan afectar la integridad de los datos almacenados.			
Atributo(s)	Funcionalidad-seguridad			
Ambiente	Operación normal			
Estímulo	Un usuario externo decide piratear el sistema			
Respuesta	El sistema no se afecta ya que presenta todos los mecanismos de seguridad, para no ser atacado ante un intento de piratería.			
Decisiones arquitectónicas	**Riesgo**	**Punto de sensibilidad**	**Punto de trade-off**	**No riesgo**
Tener activados los mecanismos de seguridad como un antivirus y un corta fuegos para evitar ataques externos.		**S3**		**N3**
Explicación	La integridad de los datos almacenados es uno de los puntos más sensibles en la seguridad del sistema debido a la relevancia que tienen, ya que el sistema maneja, guarda y adapta los resultados de las pruebas para los usuarios relacionados.			

Escenario#4	Autenticar los usuarios para permitir el uso adecuado de la herramienta.
Atributo(s)	Funcionalidad-seguridad
Ambiente	Operación normal
Estímulo	Un usuario va a autenticarse en la herramienta
Respuesta	El sistema muestra una ventana para autenticarse.

Decisiones arquitectónicas	**Riesgo**	**Punto de sensibilidad**	**Punto de trade-off**	**No riesgo**
Autenticarse en el sistema		**S4**		**N4**
Explicación	El sistema tiene determinado la funcionalidad o funcionalidades distintivas según el tipo de usuario que desee autenticarse de forma tal que conozca al ser logueado un usuario y a que tipo pertenece.			

Escenario#5	El sistema cumple con los RF y RNF.
Atributo(s)	Funcionalidad-Adecuación
Ambiente	Operación normal
Estímulo	El usuario hace uso de las funcionalidades del sistema.
Respuesta	El sistema responde de acuerdo a las funcionalidades seleccionadas.

Decisiones arquitectónicas	**Riesgo**	**Punto de sensibilidad**	**Punto de trade-off**	**No riesgo**
Interactuar con el sistema.		**S5**		**N5**
Explicación	El sistema tiene determinado la funcionalidad o funcionalidades distintivas según el tipo de usuario que desee autenticarse de forma tal que conozca al ser logueado un usuario y a que tipo pertenece.			

Escenario#6	La herramienta debe ser capaz de interactuar con el sistema distribuido.
Atributo(s)	Funcionalidad-Interoperabilidad
Ambiente	Operación normal
Estímulo	Enviar información al sistema distribuido.
Respuesta	Se realiza el envío de la tarea, desde la herramienta propuesta

	al servidor distribuido de acuerdo a la arquitectura que se propuso para esta relación entre máquinas.			
Decisiones arquitectónicas	**Riesgo**	**Punto de sensibilidad**	**Punto de trade-off**	**No riesgo**
Protocolo TCP/IP	**R1**	**S6**		
Java		**S7**	**T1**	**N6**
Explicación	La seguridad de la información que es enviada a través de la red se garantiza con el uso del Protocolo TCP/IP. Este protocolo permitirá la transmisión de la tarea desde la herramienta al servidor distribuido. El lenguaje de programación a utilizar que se escoge es java siguiendo el mismo lenguaje de programación del sistema distribuido con el cual se va a interactuar.			

Las demás tablas del anexo 2 se encuentran en los documentos complementarios.

Anexo 3 Lista de escenarios priorizados de la tormenta de ideas.

Nro.	Escenario	Votos	Atributos de calidad
11	Capacidad de la herramienta de aceptar nuevos plug-ins sin alterar la estructura de la herramienta**.**	3	Funcionalidad
12	En caso de fallo, recuperar el sistema y los datos entrados sin enviar.	5	Confiabilidad

Anexo 4. Salidas del método de evaluación ATAM: Árbol de Utilidad.

Árbol de utilidad			
Atributo de calidad	**Subcaracterística**	**Escenario**	**Prioridad**
Funcionalidad	Seguridad	Las funcionalidades del sistema se muestran de acuerdo al usuario que esté activo.	A,M
		El sistema debe restringir el acceso a datos.	A,M
		Tener protección contra ataques externos que puedan afectar la integridad de los datos almacenados.	A,A
		Autenticar los usuarios para permitir el uso adecuado de la herramienta.	A,B
		Capacidad de la herramienta de aceptar nuevos plug-ins sin grandes costos, de tal modo que no alteren el orden y estructura de la herramienta**.**	A,M
	Adecuación	El sistema cumple con los requisitos funcionales y no funcionales solicitados por el cliente.	A,B
	Interoperabilidad	La herramienta debe ser capaz de interactuar con el	A,M

		sistema distribuido.	
Confiabilidad	Tolerancia a fallos & Recuperación	Si al enviarse una tarea al sistema distribuido ocurre un error el sistema muestra un mensaje al usuario que vuelva a enviar la tarea.	A,M
		Si al cargarse un plug-ins sobre la herramienta si ocurre un error el sistema muestra un mensaje que vuelva a cargar el plug-ins.	A,M
		En caso de fallo, recuperar el sistema y los datos entrados sin enviar.	A,A
Usabilidad	Entendibilidad, Capacidad de aprendizaje y Operabilidad	El sistema debe contar con una interfaz amigable, fácil de comprender, donde el usuario pueda orientarse fácilmente	M,B
		La herramienta tendrá un ambiente sencillo y será fácil de manejar para los usuarios.	M,B
		El sistema debe someterse a una etapa de adiestramiento en la que los usuarios se familiaricen con la herramienta y sean detectados los posibles errores para ser corregidos.	M,M

Eficiencia	Comportamiento de recursos	La carga de plug-ins sobre el sistema debe realizarse satisfactoriamente.	B,B
Mantenibilidad	Estabilidad Capacidad de pruebas	EL sistema debe permitir que se puedan adicionar nuevos plug-ins con sus funcionalidades propias.	A,M
Portabilidad	Adaptabilidad, Instalabilidad Reemplazabilidad	Desplegar la herramienta en los sistemas operativos Windows y GNU/Linux.	A,M

Anexo 5. Salidas del método de evaluación ATAM: Riesgos.

R1. Constituye un riesgo ya que puede suceder que al enviar la tarea al sistema distribuido ocurra un fallo provocado por el mismo, de tal modo que la tarea no sea enviada, al ser enviada la tarea estará fuera de las fronteras de la herramienta y esto trae consigo gran riesgo ya que no se podrá medir con seguridad la efectividad de la operación.

R2. Puede que al enviar la tarea al sistema distribuido la herramienta no detecte dicho error ya que está fuera de las fronteras de la herramienta, aunque se halla preparado el sistema ante algunos errores que ocurren al ejecutar una tarea, puede suceder que ocurra alguno que atente a la confiabilidad.

R3. Las decisiones arquitectónicas para la recuperación del sistema luego de un fallo externo ya sea al enviar una tarea al sistema distribuido como al recibirla no han sido definidas, esto podría ocasionar una perdida de la información al enviar o recibir una tarea y comprometer su confiabilidad.

Anexo 6. Salidas del método de evaluación ATAM: No Riesgos.

N1. Solamente se mostrará las funcionalidades deseadas según el usuario que esté activo en ese momento, no constituye un riesgo ya que las funciones que realice el administrador así como el probador

o el ingeniero de pruebas sólo estarán visibles para ellos mismos. Ningún usuario externo, no involucrado a la herramienta tendrá acceso a estas funcionalidades.

N2. El sistema lleva el control de todos los usuarios que interactúan con él, y le da permisos según el tipo de usuario que va a acceder a los datos. Sólo tendrá acceso a todo tipo de datos el administrador de sistema, los demás usuarios tienen acceso restringido según la función que desempeñen en él. Ningún usuario no involucrado con la herramienta tendrá acceso a los datos relacionados con la misma.

N3. La herramienta está protegida contra los ataques externos ya que cuenta con mecanismos de seguridad para la confiabilidad de los datos.

N4. Cada usuario para entrar a la herramienta debe autenticarse como medio de seguridad, así se tiene el control de cada usuario y que función está realizando en el momento que se loguea.

N5. El sistema cumple con los RF y RNF solicitados por el cliente. La arquitectura propuesta está dada para que satisfaga las necesidades funcionales y las cualidades requeridas, estará basada en satisfacer los requisitos funcionales y no funcionales.

N6. La herramienta va estar implementada en el lenguaje de programación java ya que además de ser un lenguaje multiplataforma es el mismo lenguaje que usa el sistema distribuido al cual se le brinda el servicio de realizar pruebas de rendimiento por medio de la herramienta propuesta.

N7. La herramienta va a estar configurada para que se puedan adicionar plug-ins sobre la misma, de forma que si ocurre algún error el sistema le haga saber al usuario que esté interactuando con el sistema.

N8. La herramienta estará basada inicialmente para realizar pruebas de carga y estrés pero estará adaptada y estructurada para adicionar otros plug-ins para realizar pruebas de rendimiento como, resistencia y pico definiendo así cada plug-ins sus funciones distintivas.

N9. La herramienta será implementada en java, por tanto no constituye un riesgo desplegar la herramienta en Linux o Windows ya que java es un lenguaje multiplataforma que permite realizar esta acción.

N10. La herramienta va a estar capacitada para adicionar nuevos plug-ins como se cita anteriormente (N7), no sólo de carga y estrés sino también de pico y resistencia. También tendrá como característica importante que al adicionar o eliminar un plug-ins no será un riesgo para la misma ya que va estar configurada para este tipo de funcionalidades con el objetivo que no se altere el orden y la estructura de la herramienta.

Anexo 7. Salidas del método de evaluación ATAM: Puntos de Sensibilidad y deTrade-off.

S1. Autenticarse en el sistema constituye un punto de sensibilidad ya que permite que una vez registrado el usuario tenga acceso a las funcionalidades de la herramienta en dependencia al tipo de usuario que sea.

S2. Es importante luego de autenticar un usuario verificar que este exista y que además tenga los permisos necesarios para acceder a los datos.

S3. Tener activados los mecanismos de seguridad como un antivirus y un corta fuegos constituyen un punto de sensibilidad ya que permiten tener protección en la herramienta contra ataques externos que pudieran surgir, además virus que puedan afectar el sistema.

S4. Autenticarse en el sistema es un punto de sensibilidad ya que permite tener el control sobre los usuarios que interactúan con el sistema.

S5. Interactuar con el sistema es un punto de sensibilidad ya que según el usuario que se registre podrá realizar sus funcionalidades propias para de esta forma darle cumplimiento a alguna funcionalidad que se requiera en el momento.

S6. Mediante el protocolo TCP/IP se asegura la interacción de la herramienta con el sistema distribuido al enviar una tarea.

S7. Usar Java como lenguaje de programación es un punto de sensibilidad ya que permitirá una mejor interacción entre ambos sistemas, además de poder contar con la ventaja de ser un lenguaje multiplataforma.

S8. El hecho de que el sistema detecte y erradique un error al enviar una tarea al sistema distribuido constituye un punto de sensibilidad ya que así se evidencia el buen funcionamiento del sistema.

S9. Detectar un error y corregirlo cuando se va a cargar un plug-ins es un punto de sensibilidad ya que se evidencia el correcto funcionamiento del sistema.

S10.Trabajar con el IDE Eclipse constituye un punto de sensibilidad ya que permite la integración de los plug-ins sobre la herramienta.

S11. Utilizar Java como lenguaje de programación va a constituir un punto de sensibilidad ya que es un lenguaje multiplataforma lo que va a permitir que la herramienta se pueda montar tanto en Linux como en Windows, además va a poder a interactuar eficientemente con el sistema distribuido que está escrito en el mismo lenguaje.

TI. Utilizar Java es un punto de compromiso ya que es una decisión que interviene en la respuesta de dos atributos de calidad diferentes. En el caso de la Interoperabilidad está dada por la facilidad de interactuar con el sistema distribuido y en caso de la portabilidad por ser un lenguaje multiplataforma.

T2. Utilizar IDE Eclipse en un punto de compromiso ya que es una decisión que interviene en la respuesta de dos atributos de calidad diferentes, Funcionalidad que está dada por la capacidad de la herramienta de aceptar plug-ins sobre la herramienta y Mantenibilidad que está dada por incorporar las funcionalidades del plug-ins sobre la herramienta.

GLOSARIO DE TÉRMINOS

DeSoft: Empresa Cubana Nacional de Software.

Softel: Empresa Cubana Productora de Software para la Técnica Electrónica.

Calisoft: Centro para la excelencia en el desarrollo de Proyectos Tecnológicos.

UCI: Universidad de las Ciencias Informáticas.

LIPS: Se refiere al Laboratorio Industrial de Pruebas de Software que existe en la Universidad.

QLOAD, JMeter, LoadRunner: Constituyen herramientas que se utilizan para medir el rendimiento en un sistema de software.

UML: Lenguaje de modelado.

CASE: Ingeniería de Software Asistida por Ordenador (por sus siglas en inglés Computer Aided Software Engineering).

IDE: Entorno de Desarrollo Integrado por sus siglas en inglés (Integrated Development Environment). Entorno de programación que ha sido empaquetado como un programa de aplicación, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica.

CU: Caso de uso.

Stakeholders: Personas u organizaciones que están activamente implicadas en el negocio, ya sea porque participan en él o porque sus intereses se ven afectados con los resultados del proyecto.

IEEE: Asociación técnico profesional mundial dedicada entre otros aspectos a la estandarización, su siglas en vienen dadas por su nombre en inglés The Institute of Electrical and Electronics Engineers.

AS: Arquitectura de Software

Plugin: Es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica.

C++: Potente lenguaje de programación para el desarrollo de aplicaciones.

Árbol de Utilidad: Es un 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.

Interfaz de Usuario: Es el medio que el usuario utiliza para comunicarse con una máquina, un equipo o una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo, normalmente suelen ser fáciles de entender y fáciles de accionar.

CRM: gestión de relaciones con los clientes y ambientes distribuidos

ERP (planificación de recursos empresariales): Son sistemas de información gerenciales que integran y manejan muchos de los negocios asociados con las operaciones de producción y de los aspectos de distribución de una compañía comprometida en la producción de bienes o servicios.

Código abierto: Código abierto u Open Source por sus siglas en ingles es un software que pone a disposición de cualquier usuario su código fuente, por tanto éste evoluciona, se desarrolla y mejora. Ya que los usuarios lo adaptan a sus necesidades y corrigen sus errores dando como resultado la producción de un mejor software.

Jakarta: El proyecto Jakarta es el que se encarga de crear y mantener todas las soluciones open source creadas para la plataforma Java.