UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
Facultad 5

TÍTULO: Herramienta para automatizar pruebas de rendimiento a conversores de protocolos.

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

AUTOR(ES): Yanet Reyes González
Yainier Reyna Mendoza

TUTOR(A): Ing. Marelis Virgen Pérez García

CO-TUTOR(ES): Ing. Suneidis Vinent Torres
Ing. José Manuel Batista Viltre

ASESOR: Ing. Antonio Cedeño Pozo

La Habana, <junio, 14, 2013>.
"Año 55 de la Revolución."

"No haremos el futuro grande que estamos buscando si no conocemos el pasado grande que tuvimos."

Hugo Chávez Frías

DEDICATORIA

De Yanet:

A mis padres por su infinito amor y su incondicionalidad en todos los momentos de mi vida.

A mi hermana por su cariño y por hacerme sentir que siempre puedo contar con ella.

A mis abuelos por estar tan presentes en todo momento de mi vida.

A mi Carlitos por llegar a mi vida y quedarse para siempre.

A mi familia en general por ser tan unida.

A Alejandro Clivillés por ser mi inspiración para cumplir este sueño que también fue el suyo.

De Yainier:

A mis padres Ceida y Jose y a mi abuelito Armando, por ser quienes me han enseñado todo en la vida, los que siempre me han apoyado y han confiado en mí.

A mi titi Danieska por tanto amor y dedicación, te quiero.

A mi hermano Yasmani que aunque no le conozca siento que pronto lo haremos.

Los quiero mucho.

AGRADECIMIENTOS

De Yanet:

Mis agradecimientos infinitos para mi mamá y mi papá por darme todo su amor, por hacerme sentir orgullosa de la educación que me dieron, por siempre ser mi apoyo en cada decisión que tomo.

A mi hermana por cuidarme tanto y ser un ejemplo para mí, aunque a veces parezca que ella es la más chiquita.

A mis 5 abuelos por brindarme todo su cariño y estar siempre tan presentes.

A mis tíos y primos por su cariño y por la ayuda que siempre me han ofrecido.

A mis suegros, cuñados y cuñadas por convertirse en mi segunda familia.

A mis grandísimas amigas, las brujis, Yanetsys y Nairys por ofrecerme su eterna amistad y confianza y por demostrarme que ni el tiempo ni la distancia logran romper una verdadera amistad.

A Lianet por ser como otra hermana para mí y a su familia por hacerme sentir parte de ellos.

A Lili por su amistad, por todos los buenos y malos momentos que hemos compartido durante estos 6 años y por siempre estar cuando la he necesitado.

A Yaimi, Liset, Yainury y el pelú de Batista por los momentos compartidos, ya sean de diversión o de estrés estudiando para alguna prueba.

A las amigas que hice en el IPI y que serán para toda la vida: Lilibeth, Anabel, Jetsabel e Hilda Rosa por esa amistad tan linda que hemos cultivado y mantenido a pesar de que todas elegimos caminos diferentes.

A mi loca amiga Ana María por alegrarme siempre con sus ocurrencias.

A todas mis amistades de las diferentes etapas de mi vida por los momentos compartidos.

A mi grupo de 1er año por ser el mejor grupo en que he estado y por compartir el mejor de los 5 años.

A mi compañero de tesis, Yainier por su paciencia conmigo y por su ayuda durante todos estos meses.

A mi niñito lindo, Carlitos por alentarme a superarme cada día, por toda la confianza que siempre me ha ofrecido, por aconsejarme cada vez que lo necesito, por ser una de las personas que más admiro, sencillamente gracias por formar parte de mi vida, te quiero.

A todos muchas gracias.

AGRADECIMIENTOS

De Yainier:

A mi mamita, por ser tan comprensiva y atenta, por el amor, el cariño y la dedicación que me ha brindado a lo largo de mi vida y por toda la confianza que ha depositado en mí. A ella que se ha convertido en mi inspiración y mi sentido de ser. Por ser todo para mí, te amo mamita.

A mi padre, porque así lo siento, gracias por formar parte de mi vida y hacer de mi la persona que soy hoy, a ti te debo muchas cosas y una de ellas es que te hayas ganado mi respeto, mi cariño y mi amor, te quiero.

A mi novia Danieska, por estar siempre a mi lado, por todos esos gratos momentos de amor y felicidad que me ha brindado, por su apoyo incondicional, a ella mi corazón y todo mi amor.

A mis amigos Yadier, Carlos, Adrián, Henry, la melliza por ser tan excelentes personas conmigo y por confiar siempre en mí.

A mis suegros Elsida y Andrés por ser prácticamente mis segundos papas y por saber depositar su confianza en mí como si fuese hijo de ellos también.

A toda mi familia que nombrarlos a todos sería imposible, que sé que me quieren mucho, a todos ellos muchas gracias por su confianza.

Al grupo de los hediondos mayores, él "mal vestío" de Ariel, el "no borra correos" de Reiniel, el "firstblood" de Daniel, y "élcome en parque" de Randiel, a todos ellos muchas gracias por compartir estos 6 años pues han sido sin duda los mejores.

A todos los amigos y amigas que he cosechado durante estos años de universidad, Bridón, Yadira, Lisday, Arleis, la China, Yanelis, Julio Cesar, Ángel, Yasmany, Alain, Dausbel, Yusemi, Vilche, Batista, Arianna, Carlitos, Yankiel.

A mi compañera de tesis, gracias por ser una excelente persona, fue un placer compartir contigo todo este tiempo. A Marlon por formar parte de nuestro equipo.

A mi tutora, co-tutores y a mi tribunal en general, por ser tan exigente conmigo provocando que me prepare más y así ayudándome a ser un mejor profesional.

A nuestro asesor Tony que fue nuestro guía durante todo el desarrollo de la aplicación, mi hermano a ti muchas gracias por todo.

Quisiera agradecerle a Wikipedia, Google, Monografías, El Basurero y el Rincón del Vago por permitirme ser Ingeniero.

A todos…gracias.

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 firmamos la presente a los 14 días del mes de junio del año 2013.

Yanet Reyes González

_____

Firma del Autor

Ing. Marelis Virgen Pérez García

_____

Firma del Tutor

Yainier Reyna Mendoza

_____

Firma del Autor

DATOS DE CONTACTO

Nombre y apellidos del tutor(a): Ing. Marelis Virgen Pérez García
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Ingeniero en Ciencias Informáticas.
e-mail: mvperez@uci.cu

Nombre y apellidos del co- tutor(a): Ing. Suneidis Vinent Torres
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Ingeniero en Ciencias Informáticas.
e-mail: svinent@uci.cu

Nombre y apellidos del co- tutor: Ing. José Manuel Batista Viltre
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Ingeniero en Ciencias Informáticas.
e-mail: jmbatista@uci.cu

Nombre y apellidos del asesor: Ing. Antonio Cedeño Pozo
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Ingeniero en Ciencias Informáticas.
e-mail: acedeno@uci.cu

RESUMEN

Actualmente en el Centro de Informática Industrial (CEDIN) de la Universidad de las Ciencias Informáticas, se encuentran en desarrollo un conjunto de conversores de protocolos, entre ellos se destaca un dispositivo que utiliza la red TETRA para el transporte de datos entre diferentes interfaces de red.

Para asegurar la calidad en este tipo de aplicaciones es necesaria la realización de determinadas pruebas que aporten mayor tranquilidad sobre las principales funcionalidades del producto, en el caso de los conversores de protocolo, sería el éxito en el transporte de los datos.

Una de las limitaciones identificadas en dicho centro es la inexistencia de un ambiente controlado en el que se puedan realizar pruebas de rendimiento a estos conversores de forma sencilla y eficaz.

A partir de la situación antes planteada se decide implementar una herramienta que permita la automatización de pruebas a conversores de protocolos. La herramienta está diseñada para operar en dos modos de ejecución que serían: Servidor y Cliente, además cuenta con dos módulos: el de comunicaciones y el de estadísticas, el primero es el encargado de gestionar las conexiones entre el cliente y el servidor, y el segundo es el encargado de hacer un análisis estadístico de los resultados obtenidos de las pruebas realizadas.

Con el propósito de resolver las limitaciones identificadas, en el presente trabajo se describen los principales resultados obtenidos como fruto de una investigación relacionada con las pruebas automatizadas, centrándose en el desarrollo de una herramienta para automatizar pruebas de rendimiento a conversores de protocolo.

PALABRAS CLAVE

Conversores de protocolos, pruebas automatizadas, Servidor, Cliente, conexiones

ÍNDICE

TABLAS Y FIGURAS

INTRODUCCIÓN

La expansión de la informática a todos los campos de la vida ha traído consigo un significativo aumento en la utilización de los ordenadores personales y en la necesidad de compartir información. Esta necesidad sería cubierta gracias a la evolución tecnológica y al amplio despliegue de las redes de acceso de banda ancha, sumada a la optimización de los mecanismos de compresión y transmisión. El surgimiento de estas nuevas tecnologías abrió la posibilidad de ofertar entre otros, servicios de voz y video en tiempo real a través de las redes.

Como parte de esta revolución tecnológica nace Internet, devenido medio de comunicación universal. La difusión de la información vía Internet ha crecido de manera explosiva y está cambiando su naturaleza. Actualmente se realizan múltiples operaciones desde los puestos de trabajo remotos, pero cada vez más a paso acelerado, las operaciones más complejas se salen del marco de la computadora personal, y se llevan a cabo en grandes centros de cómputo a través de prestadores de servicios externos.

Para realizar estas operaciones es necesario establecer comunicación entre dos computadoras, que en ocasiones usan protocolos diferentes. El protocolo, que viene siendo el idioma que habla la computadora, es un conjunto de reglas usadas por ordenadores para comunicarse entre sí a través de una red intercambiando información. Para que exista la comunicación se hace necesario que ambas estaciones hablen el mismo idioma.

A efectos de evitar la adquisición de equipos terminales específicos generalmente costosos, los conversores de protocolos permiten el diálogo entre terminales de datos que utilizan protocolos diferentes. El crecimiento e implantación de las redes, el desarrollo de técnicas avanzadas de digitalización de voz, protocolos de transmisión en tiempo real, mecanismos de control y priorización de tráfico ha provocado un aumento en el uso de estos dispositivos, encargados de facilitar la comunicación entre equipos que manejan lenguajes distintos.

Actualmente en el Centro de Informática Industrial (CEDIN) de la Universidad de las Ciencias Informáticas, se encuentran en desarrollo un conjunto de conversores de protocolos, entre los que se destaca un dispositivo que utiliza la red TETRA para el transporte de datos entre diferentes interfaces de red, dicho dispositivo se denomina TETSCADA y fue desarrollado para la empresa española Teltronic. Otros de los productos que se encuentran en fase de desarrollo son los conversores Ethernet-Serie y ModbusTCP-ModbusRTU.

Para asegurar la calidad en este tipo de aplicaciones es necesaria la realización de determinadas pruebas que aporten mayor tranquilidad sobre las principales funcionalidades del producto, en el caso particular de los conversores de protocolo, sería el éxito en el transporte de los datos. Las pruebas de software son una fase en el desarrollo de software consistente en probar las aplicaciones construidas, no sólo permiten depurar la aplicación, sino que han pasado a convertirse en un proceso integrador, automatizable y administrable que se realiza desde el momento del inicio de la codificación y se extiende hasta la entrega final al cliente.

Una de las limitaciones identificadas en dicho centro es la inexistencia de un ambiente controlado en el que se puedan realizar pruebas de rendimiento a estos conversores de forma sencilla y eficaz.

En la bibliografía consultada se identificaron un conjunto de herramientas que permiten de alguna manera automatizar un conjunto de pruebas, pero tienen importantes limitantes que impiden su empleo, por ejemplo:

- ✓ El pago de sus licencias, ya que las mismas pertenecen a la categoría de software privativo y el país no puede costear su pago.

- ✓ La imposibilidad de extensión que tienen las aplicaciones foráneas, que se alejan de principios de auto mantenimiento, muy ligados a la soberanía tecnológica por la que está apostando el país.

En el CEDIN se han desarrollado algunas aplicaciones para la realización de este tipo de pruebas, pero no son parametrizables ni permiten realizar análisis de tipo estadísticos que faciliten el estudio a posteriori del comportamiento de los conversores de protocolos. Por otra parte los aplicativos desarrollados son de difícil empleo, pues están destinados a una tarea muy específica, en ese caso se encuentran las aplicaciones TETSCADA-TCP-CLIENT y TETSCADA-TCP-SERVER.

Por todo lo antes descrito, se determina el siguiente **problema a resolver**: ¿Cómo realizar pruebas de rendimiento de forma automatizada a conversores de protocolos?

Teniendo como **objeto de estudio**: La etapa de prueba del proceso de desarrollo de software y las pruebas de software automatizadas.

Se establece como **objetivo general**: Desarrollar una herramienta que permita automatizar pruebas de rendimiento para conversores de protocolo.

Definiendo como **campo de acción**: Las pruebas automatizadas de rendimiento a conversores de protocolo.

Para dar cumplimiento al objetivo establecido se definen las siguientes **tareas investigativas:**

1. Definición del marco teórico de la investigación a partir de los conceptos fundamentales relacionados con las pruebas de rendimiento y las herramientas para la automatización de las mismas.

2. Captura de requisitos de software de la herramienta.

3. Selección de las tecnologías y herramientas adecuadas para el desarrollo.

4. Diseño de la herramienta:
   - Diseño de la interfaz gráfica.
   - Diseño del módulo de comunicaciones.
   - Diseño del módulo de análisis estadístico.

5. Implementación de la herramienta:
   - Implementación de la interfaz gráfica.
   - Implementación del módulo de comunicaciones.
   - Implementación del módulo de análisis estadístico.

6. Validación de la propuesta de solución a partir del diseño y ejecución de pruebas.

7. Evaluación de los resultados de las pruebas.

Para realizar las tareas se emplearon los siguientes **Métodos de investigación**:

Métodos teóricos:

Histórico – lógico: Permite realizar una investigación sobre los tipos de pruebas de rendimiento aplicadas en Cuba y el mundo.

Analítico – sintético: Permite realizar un estudio profundo sobre estrategias de pruebas de rendimiento de software, los tipos de pruebas referentes a los conversores de protocolo.

Métodos empíricos:

Entrevista: El método de la entrevista se utilizó para obtener los problemas y necesidades del producto. Además para obtener conocimiento de los proyectos que ya han realizado pruebas de rendimiento automatizadas.

CAPÍTULO 1: Análisis del estado del arte

1. Introducción

La fase dedicada a la realización de las pruebas generalmente es considerada costosa y molesta, sin embargo es sumamente necesaria ya que las mismas constituyen el instrumento más adecuado para determinar el estado de la calidad de un producto de software. En el presente capítulo, se describen los fundamentos teóricos que sustentan la investigación, haciendo una exposición detallada de los principales conceptos relacionados con las pruebas de software, se identifican las características imprescindibles de las herramientas automatizadas relacionadas con la aplicación de dichas pruebas y además se abordarán definiciones referentes a los dispositivos conocidos como conversores de protocolo.

1.1. Conversores de Protocolos

Una condición imprescindible para establecer comunicación entre dos computadoras es que ambas manejen el mismo protocolo, sin embargo esta condición no siempre se cumple y es en ese momento donde se hace necesario el empleo de un conversor de protocolo. Los conversores de protocolo son dispositivos que permiten el diálogo entre terminales de datos que utilizan protocolos diferentes. En muchos casos se trata de tarjetas inteligentes que se incorporan al hardware de uno de los equipos terminales, facilitando la comunicación entre ellos.

1.1.1. TETSCADA

Uno de los proyectos que actualmente se desarrolla en la línea de Automática Aplicada del CEDIN es el conversor de protocolo TETSCADA (Interfaz SCADA para acceso a radio TETRA (Radio Troncalizado Terrestre)), el cual proporciona un dispositivo pasarela (*gateway*) que sirve como transporte entre los equipos industriales y las aplicaciones de gestión SCADA. El dispositivo se conecta con los terminales de radio TETRA (interfaz RS-232) y con los dispositivos industriales (interfaces RS-232, RS-485 y Ethernet) y es gestionado por un *firmware* que implementa los mecanismos de comunicación de dichas interfaces. TETRA es un estándar global de comunicación de radio desarrollado para satisfacer las necesidades de los clientes que necesitan rápida conexión punto a punto y punto a múltiples puntos vía radio.

1.2. Pruebas de Software

El objetivo fundamental de todo proceso de desarrollo de software es obtener un producto final con la calidad adecuada, que cumpla y supere, las expectativas de los usuarios. En la actualidad las compañías industrializadas reconocen que la calidad del producto se traduce en ahorro de costos y en una mejora general. La industria de desarrollo de software no es la excepción, por lo que en los últimos años se han realizado intensos esfuerzos para aplicar los conceptos de calidad en este ámbito.

Entre las empresas productoras de software existe mucha competencia, pues a medida que pasa el tiempo las exigencias y pedidos de los clientes son mayores, más estrictos y mucho más complejos, de ahí que el término calidad sea frecuentemente usado en este mundo.

Acerca de esto, varios autores han dado sus definiciones. Por ejemplo algunos de ellos expresan que:

- ✓ La calidad de un producto de software se puede medir descomponiendo la calidad del producto en características y estas en criterios que pueden ser medidos mediante métricas. **(Grosso, 2006)**

- ✓ Es el grado con que un sistema, componente o proceso cumple los requisitos especificados y las necesidades o expectativas del cliente o usuario. **(Society Computer, 1994)**

- ✓ La calidad de software no es más que un conjunto de propiedades y de características de un producto o servicio, que le confieren aptitud para satisfacer unas necesidades explícitas o implícitas. **(Garcia, 2003)**

La realización de pruebas de software juega un papel fundamental dentro del proceso de desarrollo de un software, pues son las encargadas de garantizar en gran medida que el producto sea confiable y funcional, constituyen el instrumento adecuado para determinar el estado de la calidad de un producto de software. Su principal objetivo es medir el grado en que el software cumple con los requerimientos.

Las pruebas de software conforman el conjunto de actividades que originan los procesos capaces de detectar los errores en un producto. Cualquier aplicación que haya sido construida necesita de las pruebas de software. 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". **(Martínez Salazar, 2012)**

Las pruebas de software son guiadas por normas y principios que un diseñador de casos de pruebas siempre debe dominar y tener en cuenta para lograr la efectividad de este proceso.

Según **Pressman** estos **principios básicos** son:

- A todas las pruebas se les debe hacer un seguimiento hasta los requisitos del cliente.

- Las pruebas deben ser planificadas antes de comenzar.

- El principio de Pareto es aplicable a la prueba del software. El 80% de los errores está en el 20% de los módulos. Hay que identificar esos módulos y probarlos muy bien.

- Las pruebas deben comenzar por "lo pequeño" y progresar hacia "lo grande".

- No son posibles las pruebas exhaustivas.

Para ser más efectivas, las pruebas deben ser conducidas por un equipo independiente. **(Pressman, 2005)**

Existen dos formas de probar cualquier producto construido: 1) Si se conoce la función específica para la que se diseñó el producto, se aplican pruebas, que demuestren que cada función es plenamente operacional, mientras se buscan los errores de cada función; 2) si se conoce el funcionamiento interno del producto, se aplican pruebas para asegurarse de que todas las piezas encajan; es decir, que las operaciones internas se realizan de acuerdo con las especificaciones y que se han probado todos los componentes internos de manera adecuada. El primer término se refiere a las pruebas de caja negra y el segundo a las de caja blanca. A continuación se hace un análisis de cada una de estas pruebas. **(Pressman, 2005)**

- ✓ **Pruebas de caja blanca:** Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado. Requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código. **(Pressman, 2005)**

- ✓ **Prueba de caja negra:** Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene. **(Pressman, 2005)**

1.2.1. Niveles de las Pruebas de Software

Las pruebas se pueden agrupar en Pruebas de Bajo Nivel y Pruebas de Alto Nivel, a continuación se describen los tipos de pruebas que incluyen cada uno de estos grupos:

Pruebas de Bajo Nivel:

- **Pruebas de unidad:** Se centran en el módulo. Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. La prueba de unidad hace uso intensivo de las técnicas de prueba de caja blanca. **(Granada., 2009)**

Más adelante, cuando el módulo parece presentable, se entra en una fase de prueba sistemática. En esta etapa se empiezan a buscar fallos siguiendo algún criterio para que "no se escape nada". Los criterios más habituales son los denominados de caja negra y de caja blanca.

- **Prueba de integración:** El objetivo es seleccionar los módulos probados en la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. El objetivo principal es detectar las fallas de interacción entre las distintas clases que componen al sistema.

Se llama integración incremental cuando el programa se construye y se prueba en pequeños segmentos en los que los errores son más fáciles de aislar y corregir, es más probable que se pueda probar completamente las interfaces y se pueda aplicar un enfoque de prueba sistemática.

Pruebas de Alto Nivel:

- **Prueba del sistema:** Verifica que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. La prueba del sistema está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Algunas de estas pruebas son:

    - **Prueba de recuperación:** Fuerza un fallo del software y verifica que la recuperación se lleva a cabo apropiadamente.

    - **Prueba de seguridad:** Determina los niveles de permiso de usuarios, las operaciones de acceso al sistema y acceso a datos.

    - **Prueba de resistencia:** Determinan hasta donde puede soportar el programa determinadas condiciones extremas.

    - **Prueba de rendimiento:** Determinan los tiempos de respuesta, el espacio que ocupa el módulo en disco o en memoria, el flujo de datos que genera a través de un canal de comunicaciones.

    - **Prueba de instalación:** Se centra en asegurar que el sistema software desarrollado se puede instalar en diferentes configuraciones hardware y software y bajo condiciones excepcionales, por ejemplo con espacio de disco insuficiente o continuas interrupciones. **(Granada., 2009)**

- **Prueba de validación:** El software totalmente ensamblado se prueba como un todo para comprobar si cumple los requisitos funcionales y de rendimiento, facilidad de mantenimiento y recuperación de errores.

Las pruebas de validación en la ingeniería de software, son el proceso de revisión donde se verifica que el sistema producido cumple con las especificaciones señaladas. Es normalmente parte de las pruebas de software de un proyecto, que también utiliza técnicas tales como evaluaciones, inspecciones, y tutoriales. La validación es el proceso de comprobar que lo que se ha especificado es lo que el usuario realmente quiere.

1.2.2. Pruebas de rendimiento

Prueban el rendimiento del software en tiempo de ejecución. Determinan 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:** Se ejecuta para comprender el comportamiento de una aplicación ante una carga determinada. Esta carga puede ser el número de usuarios esperado ejecutando un número de transacciones durante un tiempo determinado. El resultado de esta prueba 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:** 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.

- ✓ **Pruebas de Resistencia (SOAK**): 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:** 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.

Las pruebas de rendimiento pueden servir para diferentes propósitos. Pueden demostrar que el sistema cumple los criterios de rendimiento. Pueden comparar dos de ellos para encontrar cual funciona mejor o medir que partes del mismo provocan que el conjunto ofrezca bajo rendimiento. Para su diagnóstico, los ingenieros de software utilizan herramientas que permiten monitorear y medir qué partes de un dispositivo o software contribuyen a disminuir el rendimiento.

1.2.3. Pruebas Manuales

Las pruebas manuales son el método de pruebas de software más antiguo y riguroso. Requieren a un probador que ejecute de manera manual, operaciones en la aplicación sin ayuda de herramientas de automatización. Estas pruebas ayudarán a descubrir cualquier problema relacionado con la funcionalidad del producto, especialmente defectos relacionados con la usabilidad y la interfaz gráfica de la aplicación.

Las herramientas de automatización ejecutan únicamente scripts diseñados para testear una funcionalidad o requisitos específicos y no poseen la habilidad de toma de decisiones, ni grabación de discrepancias no incluidas en el script.

1.2.4. Pruebas Automatizadas

La automatización de pruebas es la parte del ciclo de calidad, en la que el software de automatización es utilizado para controlar la ejecución de pruebas, comparar los resultados, preparar las condiciones y realizar informes. Las pruebas automatizadas son efectivas en entornos donde los cambios son frecuentes o en aplicaciones en las que se esperan versiones del producto (*releases*) críticos.

Existe una gran variedad de herramientas automatizadas de pruebas y *frameworks* disponibles. Esto, junto con las nuevas metodologías de desarrollo de aplicaciones, ha creado un reto a la hora de diseñar *frameworks* de pruebas que sean mantenibles y *suites* de pruebas destinadas a regresión.

1.3. Herramientas para automatizar pruebas de software.

Las pruebas manuales de software consumen gran cantidad de tiempo y recursos, lo que ocasiona muchas veces que algunas funcionalidades queden sin ser probadas y algunos

errores no puedan ser detectados. La automatización busca reducir el tiempo en el que se incurre al hacer las pruebas y además permite una mayor cobertura gracias a la creación de *scripts*, los cuales pueden ser ejecutados cada vez que haya cambios en la aplicación. En el mercado existen varias herramientas que son utilizadas para la automatización de pruebas, a continuación se mencionan algunas de ellas:

- ✓ **HP Quicktest Professional (QTP)**: Proporciona la capacidad de automatizar pruebas funcionales y pruebas de regresión para software y ambientes de prueba. Proporciona la capacidad de definir *scripts* de prueba y posee una interfaz gráfica que le permiten al usuario emular la funcionalidad que desea probar, incluyendo el uso de interfaces de usuario de las aplicaciones a probar. Incluye características como: Vista de experto, pruebas de procesos de negocio, grabado de pantalla (para captura de las evidencias de prueba), entre otras posibilidades.

- ✓ **Visual Studio Test Proffessional**: Conjunto de herramientas de pruebas integradas desarrolladas por Microsoft, que proporcionan soporte a todo el ciclo de planificación, ejecución y registro de pruebas, con facilidades de colaboración entre analistas de prueba (*testers*) y desarrolladores en la herramienta. Proporciona capacidad de realizar pruebas manuales, reutilización de pruebas manuales, integración con el *"team foundation server"*, gestión de ciclo de vida de aplicaciones.

- ✓ **Rational Functional Tester:** Herramienta de automatización de pruebas funcionales y de regresión. Proporciona capacidades de pruebas de interfaz gráfica, pruebas manejadas por datos (*Data Driven*), pruebas funcionales y puebas de regresión. Algunas de sus características son: Simplificación de creación y visualización de pruebas, pruebas de tipo *storyboards*, trazabilidad en todo el ciclo de vida, validación de data dinámica (por medio de un *wizard*), e inclusive capacidad de definir *scripts* (por medio de lenguajes de *Scripting*).

1.4. Metodologías de Desarrollo de Software

La tendencia de crear sistemas más sofisticados, adaptados a las nuevas tecnologías, a las necesidades de los usuarios que cambian constantemente, de mejorar los productos de una versión a otra, y de realizar todo este proceso de una forma más rápida se hace cada vez más complejo. Para lograr un producto de calidad que cumpla todas estas condiciones es imprescindible seguir las pautas de alguna metodología de desarrollo de software. La metodología constituye la columna vertebral en un proceso de desarrollo de software, ellas definen quien debe hacer que, cuando y como debe hacerlo. Dichas metodologías varían sus características, de acuerdo a las necesidades reales de los desarrolladores y clientes, del tiempo y recursos con los que se dispone, y principalmente del grado de complejidad del software y predictibilidad que se desea.

1.4.1. Metodologías Tradicionales

Las metodologías tradicionales definen una disciplina de trabajo sobre el proceso de desarrollo del software, con el objetivo de conseguir un software más eficiente y predecible, las mismas están pensadas para el uso exhaustivo de documentación durante todo el ciclo del proyecto. Hacen especial hincapié en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software. A continuación se ofrecen características de algunas de las metodologías tradicionales más usadas: RUP y MSF.

Metodología RUP (Proceso Unificado de Software): es un proceso formal que provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales (respetando cronograma y presupuesto). **(Figueroa, y otros, 2011)** Como lenguaje de modelado utiliza el UML y entre sus principales características se encuentran las siguientes:

- ✓ Dirigido por Casos de Uso: Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, constituyen la guía fundamental establecida para las actividades a realizar durante todo el proceso de desarrollo del sistema.

- ✓ Centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo.

- ✓ Iterativo e Incremental: RUP divide el proceso en cuatro fases de desarrollo, propone además que cada una de ellas se desarrolle en iteraciones, las cuáles aportan un incremento en el proceso de desarrollo y terminan con el cumplimiento del punto de control trazado en la fase. **(Fuentes, y otros, 2009)**

Metodología MSF (Microsoft Solution Framework): es un compendio de las mejores prácticas en cuanto a administración de proyectos se refiere. Más que una metodología rígida de administración de proyectos, es una serie de modelos que puede adaptarse a cualquier proyecto de tecnología de información. **(Figueroa, et al., 2011)**

MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. Entre sus características fundamentales se encuentran: **(Acuña, 2009)**

- ✓ Adaptable: Es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un lugar específico.

- ✓ Escalable: Puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.

- ✓ Flexible: Es utilizada en el ambiente de desarrollo de cualquier cliente.

- ✓ Tecnología Agnóstica: Porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

1.4.2. Metodologías Ágiles

Como contraparte de estas metodologías clásicas, han aparecido un nuevo conjunto denominadas metodologías ágiles, que se basan en dos aspectos puntuales, el retrasar las decisiones y la planificación adaptativa; permitiendo potenciar aún más el desarrollo de software a gran escala.

Aportan como novedad, nuevos métodos de trabajo que apuestan por una cantidad apropiada de proceso. Es decir, ni se pierden en una excesiva cantidad de cuestiones burocráticas, ni defienden tampoco la falta total de procesos. Buscan el equilibrio en la relación proceso/esfuerzo. Estas metodologías surgen como respuesta a problemas reales, se basan en el sentido común pero rompen con creencias arraigadas**. (Cáceres, 2012)**

Algunas de estas metodologías ágiles son:

- ✓ **Metodología Programación Extrema (XP o Extreme Programming):** Es la más destacada de los procesos ágiles de desarrollo de software, se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad, considera que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. **(Figueroa, y otros, 2011)**

  Utilizada para proyectos de corto término y equipo, cuyo plazo de entrega es inmediato. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

  Las características fundamentales de la metodología son:

  - **Desarrollo iterativo e incremental:** Pequeñas mejoras, unas tras otras.
  - **Pruebas unitarias continuas:** Frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
  - **Programación por parejas**: Se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata. **(Figueroa, y otros, 2011)**

- ✓ **Metodología SCRUM:** Es un proceso ágil y liviano que sirve para administrar y controlar el desarrollo de software. El desarrollo se realiza en forma iterativa e incremental (una iteración es un ciclo corto de construcción repetitivo). Cada ciclo o iteración termina con una pieza de software ejecutable que incorpora nueva funcionalidad. **(Figueroa, y otros, 2011)**Está recomendada para proyectos con un rápido cambio de requisitos.

  Sus principales características se pueden resumir en dos:

- ✓ El desarrollo de software se realiza mediante iteraciones, denominadas *sprint*, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente.

- ✓ La segunda característica importante son las reuniones a lo largo del proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

Para la presente investigación se decide emplear como metodología de desarrollo Programación Extrema (XP) ya que es muy adaptable a las necesidades. No sigue un régimen muy estricto para ponerla en uso, es el más destacado de los procesos ágiles de desarrollo de software. Es la más conocida y usada lo que se traduce en más probada. El cliente forma parte del equipo de desarrollo, logrando una mejor retroalimentación,

corrección de errores y así un producto que satisfaga todas las necesidades del mismo. La documentación de consulta para el aprendizaje de la misma es amplia y fácil de encontrar.

1.5. Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que permite organizar la complejidad de los sistemas a analizar o diseñar, se centra en su representación gráfica. Este lenguaje permite crear y leer los modelos, pero no dice cómo crearlos. Permite modelar sistemas utilizando tecnologías Orientada a Objetos. Las funciones fundamentales de UML son: **(Xavier Ferré Grau, 2012)**

- ✓ Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- ✓ Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- ✓ Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- ✓ Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

En la investigación se usará UML para el modelado, porque permite la representación gráfica de un sistema con tecnologías orientada a objetos y especifica cuáles serán las características antes de su construcción. Además a partir de los modelos especificados se pueden construir los sistemas diseñados y los propios elementos gráficos se utilizan como documentación del sistema desarrollado que pueden servir para su futura revisión. Una vez seleccionado el lenguaje de modelado visual se procede a seleccionar el lenguaje de programación para el desarrollo de la solución propuesta.

1.6. Lenguajes de Programación

El lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión. **(Prieto, 2007)**

En la actualidad existen varios lenguajes de programación, los cuales son empleados en dependencia del tipo de programa que se quiera construir, entre ellos se pueden mencionar: Java, C++, Delphi, Visual Basic, Visual C# .NET, y otros. Es importante tener en cuenta sus características para elegir el correcto para su uso en el desarrollo de aplicaciones informáticas.

1.6.1. Java

Java es un lenguaje de programación orientado a objetos. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. Este lenguaje está siendo adoptado por múltiples fabricantes como herramienta básica para el desarrollo de aplicaciones comerciales de gran repercusión

ya que es un lenguaje de desarrollo de propósito general, y como tal es válido para realizar todo tipo de aplicaciones profesionales. Presenta múltiples ventajas tales como: multiplataforma, robusto, de alto rendimiento y posee gran cantidad de herramientas gratuitas, es fácil de aprender para los programadores que conocen la programación orientada a objetos. Es un lenguaje bien estructurado, sin punteros y sin necesidad de tener que controlar la asignación de memoria a estructuras de datos u objetos. **(Ruiz, 2009)**

1.6.2. C++

C++, es un lenguaje de programación diseñado como extensión del lenguaje de programación C. La intención de su creación fue extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel. Una de las ventajas que ofrece es que es mucho más sencillo de aprender para los programadores que ya conocen el lenguaje C, sin embargo es a su vez uno de los menos automatizados, obliga a hacerlo casi todo manualmente al igual que C.

La definición "oficial" del lenguaje dice que C++ es de propósito general basado en C, al que se han añadido nuevos tipos de datos, clases, plantillas, mecanismo de excepciones, sistema de espacios de nombres, sobrecarga de operadores, referencias, operadores para manejo de memoria persistente, y algunas utilidades adicionales de librería, la librería Estándar C es un subconjunto de la librería C++. (Villoria, 2009)

Para la implementación de la herramienta se decide utilizar C++ como lenguaje de programación por sus múltiples ventajas, entre las que se encuentra la flexibilidad ya que el mismo permite relacionar los procedimientos que manipulan los datos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde estos datos aparezcan. Otra de sus grandes ventajas, que será muy útil en el desarrollo pues hará más fácil el trabajo con el Puerto Serie, es que es un lenguaje multi-nivel, es decir, se puede usar tanto para programar directamente el hardware, como para crear aplicaciones tipo Windows definidas todas por poseer una misma interfaz.

1.7. Entornos de Desarrollo Integrado

Los Entornos de Desarrollo Integrado (IDE por sus siglas en inglés) son un programa informático compuesto por un conjunto de herramientas de programación que proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación. En algunos lenguajes un IDE puede funcionar como un sistema en tiempo de ejecución donde se permite utilizar al lenguaje de programación en forma interactiva sin necesidad de trabajo orientado a archivos de texto. La correcta selección del mismo es de mucha importancia pues garantizará un óptimo desempeño del sistema. Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación.

1.7.1. Qt Creator

Es un entorno integrado de desarrollo (editor + compilador + depurador) completo, moderno, potente, fácil de manejar, eficiente, abierto y gratuito, que permite el desarrollo rápido de aplicaciones. Es un IDE multiplataforma muy completo, diseñado para hacer que el desarrollo en C++ de la aplicación Qt sea más rápido y fácil. **(varios, 2011)**

Entre sus principales características se encuentran las siguientes:

- ✓ Qt es utilizado en KDE, un Entorno de escritorio para sistemas como GNU/Linux o FreeBSD. Qt utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en otros lenguajes de programación.

- ✓ Funciona en todas las principales plataformas, y tiene un amplio apoyo. El API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y una multitud de otros para el manejo de ficheros, además de estructuras de datos tradicionales. **(varios, 2011)**

1.7.2. Eclipse

Es un entorno de desarrollo integrado de código abierto multiplataforma, es principalmente una plataforma de programación, usada para crear entornos integrados de desarrollo sin embargo, también se puede usar para construir aplicaciones cliente.

Entre sus principales características se encuentran las siguientes:

- ✓ Fue desarrollado originalmente por IBM como sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimos de lucro que fomenta una comunidad de código abierto. **(Baldur, 2012)**

- ✓ Eclipse así como sus *plugins* está desarrollado por completo en Java. Utiliza la biblioteca SWT (Standard Widget Toolkit) en sustitución de *Swing* para un mayor aprovechamiento de los componentes nativos del sistema donde se ejecuta. La contrapartida de esta librería es que es nativa, es decir, es necesario disponer de una librería SWT específica para cada sistema operativo. **(Autores, 2011)**

1.7.3. NetBeans

Es un entorno de desarrollo integrado multilenguaje, completo y modular. Es una plataforma gratis y de código abierto empleada para construir aplicaciones, una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas.

Entre sus principales características se encuentran las siguientes:

- ✓ Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Es un producto libre y gratuito sin restricciones de uso. El IDE NetBeans es un producto de código abierto. **(Corporation, 2012)**

- ✓ Provee todas las herramientas necesarias para crear aplicaciones de escritorio, de empresa, web y aplicaciones móviles con el lenguaje Java, C / C + +, e incluso lenguajes dinámicos como PHP, Java Script, Groovy, y Ruby. NetBeans IDE es fácil

de instalar y utilizar. Además, se ejecuta en muchas plataformas, incluyendo Windows, Linux, Mac OS X y Solaris.

Como IDE de desarrollo se decide emplear QT Creator en primer lugar por ser completamente gratuito, además sus herramientas, bibliotecas y clases están disponibles para casi todas las plataformas. Por otra parte posee extensas bibliotecas con clases y herramientas para la creación de ricas aplicaciones. Estas bibliotecas y clases están bien documentadas, son muy fáciles de usar y tienen una gran herencia de programación orientada a objetos lo cual hace de la programación de interfaces gráficas una tarea placentera.

1.8. Herramientas Case

Las Herramientas CASE (Ingeniería de Software Asistida por Ordenador por sus siglas en inglés Computer Aided Software Engineering) son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. Estas aplicaciones están destinadas a aumentar la productividad.

El proceso de construcción de un software requiere que las tareas sean organizadas y completadas en forma correcta y eficiente. Las Herramientas CASE fueron desarrolladas para automatizar esos procesos y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el ciclo de desarrollo de software. A continuación se describirán algunas de estas herramientas.

1.8.1. Visual Paradigm

Visual Paradigm (VP) es una herramienta profesional, que soporta el ciclo de vida completo de desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. A pesar de que no es software libre, posee una versión *freeware* (programa de libre acceso o programa con derechos de autor que se puede utilizar sin pago alguno) llamada Visual Paradigm para la edición de la comunidad UML.

1.8.2. Rational Rose Enterprise Edition

Es una herramienta para modelado visual, diseñada para ayudar a los desarrolladores a crear soluciones eficientes para el modelado de problemas del mundo real. La versión completa de esta herramienta de Rational Rose, permite que mediante el modelado del sistema total, la herramienta obtenga el código a partir de los diagramas desarrollados en ella, o de forma inversa, que la herramienta realice reingeniería, o sea a partir del código del sistema se puede obtener todo el modelado. Esta herramienta además permite: especificar, analizar y diseñar el sistema antes de codificarlo, mantiene la consistencia de los modelos del sistema software, chequeo de la sintaxis UML y generación de la documentación. **(Varios, 2007)**

1.8.3. Enterprise Architect

Es una herramienta comprensible de análisis y diseño UML, cubriendo el desarrollo de software desde el paso de los requerimientos a través de las etapas del análisis, modelos de

diseño, pruebas y mantenimiento. Es una herramienta multi-usuario, basada en Windows, diseñada para ayudar a construir software robusto y fácil de mantener. Ofrece salida de documentación flexible y de alta calidad. El Lenguaje Unificado de Modelado provee beneficios significativos para construir modelos de sistemas de software rigurosos y donde es posible mantener la trazabilidad de manera consistente. Enterprise Architect soporta este proceso en un ambiente fácil de usar, rápido y flexible, permite ingeniería de Código Directa e Inversa (ediciones Corporativa y Profesional solamente) y Soporte para Java, C#, C++, Delphi, Visual Basic, Python y PHP. **(Systems, 2007)**

Para la investigación se decide utilizar Visual Paradigm como herramienta para el modelado ya que la misma ofrece un completo conjunto de herramientas de los equipos de desarrollo de software necesarios para la captura de requisitos, la planificación de programas, la planificación de controles, la clase de modelado así como el modelado de datos.

1.9. Conclusiones parciales

A través de la realización de este capítulo se construyó el basamento teórico con vistas al desarrollo de la aplicación mediante un estudio del estado del arte de las principales metodologías y herramientas que se utilizan en el desarrollo de la aplicación que propone el presente trabajo de diploma. Además se enuncian varios conceptos para hacer más entendible esta investigación así como algunos ejemplos de este tipo de aplicaciones existentes en el mercado. Luego de la investigación se decide trabajar con:

- ✓ La metodología de desarrollo de software XP para la elaboración de la herramienta, pues permite centrarse en los individuos y las iteraciones por encima de los procesos y las herramientas. Además no requiere de la generación de una documentación exhaustiva y se adapta con facilidad a los cambios de requisitos, tecnología o equipamiento.

- ✓ El Lenguaje de Modelado UML porque posibilita la representación gráfica de un sistema con tecnologías orientada a objetos.

- ✓ La herramienta CASE Visual Paradigm ya que soporta UML y posibilita realizar ingeniería inversa, generar documentación correspondiente en cada flujo de trabajo y además la UCI cuenta con la licencia que autoriza su uso.

- ✓ Lenguaje de programación C++, para todo el desarrollo, con su IDE de desarrollo Qt Creator, por ser un lenguaje multiplataforma y muy completo, diseñado para hacer que el desarrollo en Qt de la aplicación sea más rápido y fácil.

CAPÍTULO 2: Descripción de la solución.

2. Introducción

El presente capítulo tiene como objetivo fundamental describir la solución que propone el presente trabajo, además de definir las fases por las que transitará la aplicación desarrollada según la metodología de desarrollo de software seleccionada. Se describirán además las necesidades del sistema a través de las historias de usuarios. Se enunciarán tanto los requisitos funcionales como los no funcionales, así como las clases definidas en la implementación.

2.1. Análisis y diseño de la herramienta.

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

El ciclo de vida ideal de XP consiste de seis fases:

I. Exploración

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

II. Planificación de la Entrega (*Release*)

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

III. Iteraciones

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

IV. Producción

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase. Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento).

V. Mantenimiento

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

VI. Muerte del Proyecto

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

Las metodologías ágiles ofrecen una solución casi a medida para una gran cantidad de proyectos, sin embargo toda metodología debe ser adaptada al contexto del proyecto (recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema), en el caso del presente trabajo su ciclo de vida solo llegará hasta la fase de producción.

Programación Extrema propone varios roles a desempeñar a lo largo de un proyecto, a continuación se describirán solo los empleados para el desarrollo del presente software:

Programador: Escribe las pruebas unitarias y produce el código del sistema.

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

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

2.1.1. Propuesta del sistema.

Se implementará una herramienta que permita la automatización de pruebas a conversores de protocolos. La herramienta está diseñada para operar en dos modos de ejecución que serían como servidor en un caso y como cliente en el otro, además cuenta con dos módulos principales que son el módulo de comunicaciones y el de estadísticas, donde el primero es el encargado de gestionar las conexiones entre el cliente y el servidor y el segundo como su

nombre lo indica es el encargado de hacer un análisis estadístico de los resultados obtenidos de las pruebas antes realizadas. Debido a que la herramienta tiene dos comportamientos asociados se hace necesario separar por cada uno de estos sus funcionalidades específicas.

Servidor: En este modo de ejecución pueden trabajar dos tipos de servidores al mismo tiempo, TCP/IP y Puerto Serie. En el caso del protocolo de comunicación TCP/IP el servidor cuenta con un parámetro de configuración que no es nada más que el puerto por el cual el usuario quiere aceptar peticiones, una vez hecho esto la herramienta utiliza la dirección IP física de la computadora en la que se está ejecutando y el puerto especificado y se pone en escucha a las peticiones de los clientes. Al iniciar se espera a que lleguen las solicitudes de los clientes a través de las interfaces antes mencionadas, una vez recibida la solicitud si todos los parámetros de conexión están correctos envía una respuesta al cliente que sería la conexión creada entre ellos, la comunicación con el o los clientes queda en espera de recibir algún tipo de información o simplemente puede enviar datos al cliente deseado.

Cliente: Es quien inicia solicitudes o peticiones, tienen por tanto un papel activo en la comunicación con el servidor, una vez que envía la solicitud de conexión espera a recibir respuesta del servidor de lo contrario notifica al usuario que no se puede aceptar su petición. En este modo se pueden manejar varios clientes concurrentes tanto por interfaz TCP/IP como Puerto Serie. En el caso de los clientes con interfaz TCP/IP tienen como parámetros de configuración el IP y el puerto del servidor al que se quiere conectar, al instante en que se establece la conexión se procede a realizar el envío de datos al servidor.

2.1.2. Actores relacionados con el sistema.

Los actores relacionados con el sistema son aquellos que interactúen con la aplicación y que obtienen un resultado de valor de uno o varios procesos que se ejecutan en el mismo. La presente aplicación consta de un actor: el probador, el cual hace uso de la aplicación, beneficiándose de sus funcionalidades. Es el responsable de gestionar las conexiones y de supervisar el comportamiento de las pruebas.

2.1.3. Historias de usuarios.

Las historias de usuario (HU) son la técnica utilizada en XP para especificar los requisitos del software. Tienen el mismo propósito que los casos de uso, con la excepción de que no se limitan a la descripción de la interfaz de usuario. Se utilizan en la creación de las pruebas, para verificar si el programa cumple con lo que especifica cada HU. La principal diferencia entre estas y la tradicional especificación de requisitos, es el nivel de detalle. Las HU solamente proporcionan los detalles sobre la estimación del riesgo y cuánto tiempo llevará la implementación de dicha HU. A continuación se describe la HU Gestionar Cliente, las restantes pueden ser consultadas en el Anexo 1.

Tabla 1: HU Gestionar cliente

Historias de usuario	
Número: 1	**Nombre Historia de Usuario:** Crear cliente.

Actor: Probador	**Iteración Asignada:** 1
Prioridad de Negocio: Alta	**Puntos Estimados:** 3
Riesgo en desarrollo: Medio	**Puntos Reales:** 3
Descripción: El actor podrá crear el o los clientes que se van a conectar al servidor.	
Observaciones: El actor deberá haber iniciado la aplicación en el modo de ejecución Cliente.	

Para describir los requisitos no funcionales XP también proporciona las HU, a continuación se describen las que se realizaron para el desarrollo del presente trabajo:

Tabla 2: HU Requisitos de software

Historias de usuario	
Número: 12	**Nombre Historia de Usuario:** Requisitos de software
Descripción: Se debe tener instalado una computadora con el sistema operativo GNU/Linux Ubuntu 12.04 con los siguientes paquetes instalados: • Qt versión 4.8 ó superior	
Observaciones:	

Tabla 3: HU Requisitos de Hardware

Historias de usuario	
Número: 13	**Nombre Historia de Usuario:** Requisitos de Hardware
Descripción: Para la ejecución de la aplicación se debe tener una computadora con las siguientes prestaciones: • Procesador Intel Pentium IV	

• Frecuencia del CPU: 2.4GHz • Memoria RAM:>= 512MB
Observaciones:

Tabla 4: HU Restricciones en el diseño e implementación

Historias de usuario	
Número: 14	**Nombre Historia de Usuario:** Restricciones en el diseño e implementación.
Descripción: Para el desarrollo de la solución se definen una serie de restricciones: • Lenguaje de programación: C++ • Entorno integrado de desarrollo (IDE): Qt Creator	
Observaciones:	

Tabla 5: HU Requisitos de apariencia o interfaz externa

Historias de usuario	
Número: 15	**Nombre Historia de Usuario:** Requisitos de apariencia o interfaz externa.
Descripción: La interfaz debe ser sencilla, de fácil uso, e intuitiva en cuanto a los pasos a seguir a la hora de probar.	
Observaciones:	

2.2. Diseño del sistema.

La metodología XP no requiere la descripción del sistema por medio de diagramas de clase, sino que se guía por técnicas como las tarjetas CRC (Contenido, Responsabilidad y Colaboración) que identifican y organizan las clases orientadas a objetos que son relevantes para el incremento actual de software. Esto no implica que no se utilicen los diagramas para obtener una mejor visión y comunicación entre el equipo de trabajo, siempre y cuando su complejidad no sea alta y defina información importante.

2.2.1. Tarjetas CRC (Contenido o Clase, Responsabilidad y Colaboración).

Las tarjetas CRC son el único producto de trabajo de diseño que se generan como parte del proceso XP, esta metodología estimula el uso de estas tarjetas como un mecanismo eficaz para pensar en el software en un contexto orientado a objetos. Las tarjetas CRC representan objetos; la clase a la que pertenece el objeto se escribe en la parte superior de la tarjeta, en una columna a la izquierda se escriben las responsabilidades u objetivos que debe cumplir el objeto y a la derecha, las clases que colaboran con cada responsabilidad.

La aplicación a desarrollar consta de las siguientes clases:

- MainWindow
- Server
- ClienteControler
- Cliente
- ClienteTCP
- ClienteSerie

A continuación la tarjeta CRC correspondiente a la clase MainWindow, para consultar las restantes dirigirse al Anexo 2.

Tabla 6: Tarjeta CRC de la clase MainWindow

MainWindow	
Responsabilidades	**Colaboradores**
-Es la clase que controla el modo de ejecución. -Según el modo de ejecución seleccionado es el objeto con el que trabaja.	• Server • ClienteControler

2.2.2. Arquitectura del sistema

La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Esta arquitectura consiste básicamente en un cliente que realiza peticiones a otro programa (el servidor) que le da respuesta. Aunque esta idea se puede aplicar a programas que se ejecutan sobre una sola computadora es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras. La interacción cliente-servidor es el soporte de la mayor parte de la comunicación por redes.

El servidor debe negociar con su sistema operativo un puerto donde esperar las solicitudes. El servidor espera pasivamente las peticiones en un puerto bien conocido que ha sido reservado para el servicio que ofrece. El cliente también solicita, a su sistema operativo, un puerto no usado desde el cual enviar su solicitud y esperar respuesta. Un cliente ubica un puerto arbitrario, no utilizado y no reservado, para su comunicación.

En una interacción se necesita reservar solo uno de los dos puertos, asignados a un identificador único de puerto para cada servicio, se facilita la construcción de clientes y servidores.

Los servidores por lo general son más difíciles de construir que los clientes, pues aunque se implantan como programas de aplicación deben manejar peticiones concurrentes, así como reforzar todos los procedimientos de acceso y protección del sistema computacional en el que corren, y protegerse contra todos los errores posibles. El cliente y el servidor pueden interactuar en la misma máquina.

2.2.3. Patrones de diseño

Los patrones de diseño constituyen la base en la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño puede resultar una solución a un problema de diseño. Para considerar una solución como patrón debe poseer ciertas características: debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores y ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

El ***Simple Factory*** es un patrón de creación que se encarga de crear instancias de objetos de manera que ya no se tendrán que instanciar directamente proporcionando a los programas una mayor flexibilidad para decidir qué objetos usar. Dicho patrón será empleado en la clase MainWindow, pues la misma se encargará de crear y devolver el objeto que se define en su implementación, es decir crearía Server y ClienteControler.

El ***Singleton*** es un patrón que garantiza que solamente se cree una instancia de la clase y provee un punto de acceso global a él. Todos los objetos que utilizan una instancia de esa clase usan la misma instancia. Este patrón será empleado en la clase ClienteControler para restringir la creación de objetos pertenecientes a esa clase.

2.3. Planificación y entrega

Para XP la planificación es una fase corta en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las historias de usuario

así como las entregas asociadas a éstas. Comúnmente esta fase consiste en una o varias reuniones grupales de planificación. El resultado de dicha fase es un Plan de Entregas.

Los programadores realizan una estimación del esfuerzo necesario de cada una de las HU en dependencia de la prioridad establecida por el cliente. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente.

2.3.1. Plan de entrega

Una vez culminada la confección de las HU por parte del cliente se inicia la elaboración del Plan de Entregas, el principal propósito de dicho plan es darles a los programadores una estimación de dichas historias en cuanto al nivel de detalle para conocer el tiempo que se puede tardar en la implementación de cada una.

Dicho plan agrupa las historias de usuarios en el orden que serán realizadas hasta conformar una entrega. El plan es confeccionado en base de las estimaciones de esfuerzos de desarrollo realizadas por los programadores. Las estimaciones de esfuerzos asociadas a la implementación de las historias tendrán como medida el punto. Un punto, se corresponde a una semana de programación ideal. Las historias deben poder ser programadas en un tiempo entre una y tres semanas.

La planificación se puede realizar basándose en el tiempo o en el alcance. La velocidad del proyecto es aprovechada para establecer cuántas historias de usuario se pueden hacer antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias.

Tabla 7: Plan de entrega

Historia de usuario	Final 1ra Iteración 23/03/2012	Final 2da Iteración 18/05/2012
Iniciar Servidor Detener Servidor Guardar Configuración Cargar Configuración Conectar cliente Desconectar cliente	8	Finalizado
Enviar Datos Recibir Datos Crear Cliente	No empezado	12

Modificar Cliente Eliminar Cliente Enviar fichero Enviar datos durante x tiempo Analizar envío de datos por cliente Analizar envío de datos de todos los clientes		

Tabla 8: Estimación del esfuerzo por historia de usuarios

Historias de usuario	Tiempo estimado	Iteración	Tiempo real
Iniciar Servidor Detener Servidor Guardar Configuración Cargar Configuración Conectar cliente Desconectar cliente	8	1	8
Enviar Datos Recibir Datos Crear Cliente Modificar Cliente Eliminar Cliente Enviar fichero Enviar datos durante x tiempo Analizar envío de datos por cliente Analizar envío de datos de todos los clientes	12	2	12

2.3.2. Plan de iteraciones

Teniendo el esfuerzo estimado con que serán desarrolladas las historias de usuarios y el plan de entregas elaborado, se continúa a planificar la etapa de implementación del sistema. El Plan de Iteraciones recoge las historias de usuario que serán implementadas en cada iteración, de acuerdo al orden preestablecido. De acuerdo con lo antes mencionado se decide realizar el sistema en dos iteraciones, las cuales se especifican a continuación:

Iteración 1

Esta primera iteración tiene como meta implementar las historias de usuarios con prioridad media para el cliente, las cuales son: iniciar servidor, detener servidor, guardar configuración, cargar configuración. Estas historias de usuario constituyen la estructura básica del sistema. Con la finalización de esta iteración se obtendrá la primera entrega del sistema con el propósito de mostrar al cliente lo realizado y recibir retroalimentación del mismo.

Iteración 2

En esta iteración se implementan las historias de usuario que tienen prioridad alta para el cliente, las cuales son: conectar cliente, desconectar cliente, enviar datos, recibir datos, gestionar cliente. La versión de prueba referente a esta iteración junto con las implementaciones anteriores, serán mostradas al cliente con el objetivo de realizar cambios en base a la opinión del mismo. Al finalizar esta iteración, luego de que haya pasado satisfactoriamente por la etapa de pruebas se dispondrá de la aplicación con todas las funcionalidades descritas por el cliente.

2.3.3. Plan de duración de las iteraciones

Programación Extrema propone, para una mayor organización del trabajo y como parte del ciclo de vida de un proyecto, la creación de un plan de duración de cada una de las iteraciones cuya finalidad es reflejar la duración de cada iteración, así como el orden en que serán implementadas las HU dentro de cada una de las mismas.

Tabla 9: Plan de duración de las iteraciones

Iteraciones	Orden de las HU a implementar	Duración total de las iteraciones
Iteración 1	Iniciar Servidor Detener Servidor Guardar Configuración Cargar Configuración Conectar cliente Desconectar cliente	8 semanas

Iteración 2	Enviar Datos Recibir Datos Crear Cliente Modificar Cliente Eliminar Cliente Enviar fichero Enviar datos durante x tiempo Analizar envío de datos por cliente Analizar envío de datos de todos los clientes	12 semanas

2.3.4. Historias de usuario divididas en tareas

Las historias de usuario están implícitas dentro de tareas de programación, que serán definidas en cada iteración. Estas tareas son escritas en fichas como las historias de usuario, pero en el lenguaje de los programadores. Cada tarea debe ser estimada en un período de uno a tres días de desarrollo.

2.3.4.1. Iteración 1

Tabla 10: Tareas abordadas en la primera iteración

Historia de usuario	**Tareas**
Iniciar Servidor	- Insertar parámetros de configuración. - Validar parámetros de configuración.
Detener Servidor	-Validar estado del servidor.
Guardar Configuración	- Crear fichero de configuración.
Cargar Configuración	- Seleccionar fichero de configuración. -Mostrar datos.
Conectar cliente	- Validar los parámetros de configuración

	- Conectar con el servidor
Desconectar cliente	- Validar estado del cliente - Desconectar el servidor

2.3.4.2. Iteración 2

Tabla 11: Tareas abordadas en la segunda iteración

Historia de usuario	**Tareas**
Enviar Datos	- Validar datos. -Verificar estado del cliente.
Recibir Datos	-Verificar estado del servidor.
Crear Cliente	-Definir y desarrollar la interfaz. -Validar datos de entrada. -Probar la aplicación.
Modificar Cliente	-Definir y desarrollar la interfaz. -Validar datos de entrada. -Probar la aplicación.
Eliminar Cliente	-Validar datos de entrada. -Probar la aplicación.
Enviar fichero	- Seleccionar y leer el fichero.
Enviar datos durante x tiempo	- Verificar el tiempo de envío.
Analizar envío de datos por cliente	- Definir y desarrollar la interfaz Analizar envío de datos por cliente del módulo de estadísticas.

	- Monitorizar los datos enviados por el cliente. - Presentar los datos al probador.
Analizar envío de datos de todos los clientes	- Definir y desarrollar la interfaz Analizar envío. - Monitorizar los datos enviados por todos los clientes. - Presentar los datos al probador.

2.4. Conclusiones parciales

En el presente capítulo quedó descrita la propuesta del sistema a implementar, además se obtuvo el Plan de Entrega correspondiente a la fase de planificación. Se definieron las iteraciones por las que transitará la aplicación especificando qué historias de usuarios serán implementadas en cada una. Las clases del sistema fueron representadas mediante las tarjetas CRC como parte del diseño del mismo. Se decidió utilizar Cliente-Servidor como arquitectura del sistema, así como el empleo de los patrones de diseño de software *Singleton* y *Simple Factory*.

CAPITULO 3: Implementación y prueba

3. Introducción

Dentro del proceso de desarrollo de software la etapa de implementación, es el proceso de convertir una especificación del sistema en un sistema ejecutable. La misma implica los procesos de transformar las clases y objetos en ficheros fuente, binarios y ejecutables. En la etapa de pruebas el resultado final, o sea el ejecutable, es evaluado en cuanto a su calidad y desempeño como producto de software.

En el presente capítulo se tratarán los temas referentes a la implementación y pruebas que se le realizarán a la aplicación. Serán definidos además los estándares de codificación empleados para una mayor organización, limpieza y claridad en la escritura del código; y se describirán los artefactos generados durante estas etapas.

3.1. Estándares de codificación

Un estándar de codificación es una tecnología, formato o método desarrollado, adoptado a través de proceso abierto de consenso, con la ventaja de facilitar que el código fuente de cualquier aplicación sea legible, se le pueda dar mantenimiento, sea interoperable y pueda ser distribuido. Los estándares abiertos son capaces de soportar y administrar en forma más sencilla y menos costosa la creciente complejidad de los sistemas, constituyendo una especie de "garantía universal" de compatibilidad, ajena a cualquier proveedor de la industria por sí mismo.

3.1.1. Definiciones generales

Las definiciones se realizan de manera descriptiva, tratando de ser lo más explícito posible, evitando las abreviaturas y los nombres cortos.

Clases:

Las clases deben comenzar con mayúscula y en caso de estar conformada por palabras compuestas, la definición debe ser continua y cada palabra debe iniciar con mayúscula siguiendo el estilo determinado.

Ejemplo:

```
class ClienteControler
```

Métodos:

Los métodos deben empezar con minúsculas, escribiéndose de manera seguida en caso de estar conformados por palabras compuestas. Se escoge iniciar las funcionalidades con minúsculas para diferenciarlas de las clases, pero en caso de ser compuestas, a partir de la segunda palabra debe comenzar con letra mayúscula.

Ejemplo:

```
int buscarClienteById(QString id);
QString getId();
```

Variables:

Las variables comienzan con minúsculas, aquellas que sean compuestas están separadas por un guión bajo.

Ejemplo:

```
QList<Cliente *> * lista_de_clientes;
ClienteControler*  instancia;
```

3.1.2. Comentarios

Se utilizan comentarios para dar un mejor entendimiento a clases o funciones. La utilización de los mismos estará presente en casi todo el código ya que la herramienta estará sujeta a realización de nuevas versiones por lo que se hace necesario la correcta explicación de la misma.

Clases y funciones:

Aquellas clases o funciones en las que se utilicen comentarios para especificar su papel en el código, deben tener dicha aclaración seguida de su definición, buscando una mayor interrelación con el código y evitando que se confunda con el código que le antecede. Para un mejor entendimiento se utiliza de forma diferenciada en el caso de las clases y las funciones.

Para describir las clases se utiliza el formato siguiente:

```
/**
@class: aqui se especifica el nombre de la clase
@brief: se describe el funcionamiento de la clase
@author: se especifican el o los autores que implementan la clase
*/
```

Ejemplo:

```
class ClienteControler
{
    /*
    @class ClienteControler
    @brief Clase que contendrá los principales valores de un
    cliente así como que cumplirá el rol de padre en la relación
    de herencia con las clases ClienteSerie y ClienteTcp.
    @author Yainier Reyna Mendoza | Yanet Reyes Gonzalez
    */

    //resto de la clase
};
```

Para describir las funciones se utiliza el formato siguiente:

```
/**
 * @brief Función para agregar un cliente
 * @param cliente Cliente que se va a agregar a la lista de clientes
 * @return Verdadero en caso que se agregue el cliente, falso en caso
 *contrario
 */
bool addCliente(Cliente * cliente);
```

En caso de que la función no tenga parámetros de entrada se omite **@param**, además si la función es *void* que no devuelve ningún valor se omite **@return**.

Ejemplo:

```
/**
 *@brief Función para cambiar la paridad
 *@param paridad Nueva paridad para el cliente
 */
void setParidad(QString paridad)
{
  //implementación de la función
}
```

Variables:

Se utilizará para dar una pequeña descripción del uso y la función o funciones que desempeña la variable.

Ejemplo:

```
QString id;          //identificador de cada cliente
QString historial;   //historial de mensajes con el servidor
bool conectado;      //variable para saber el estado de la conexión
```

3.2. Diagrama de despliegue

El diagrama de despliegue es un tipo de diagrama del Lenguaje Unificado de Modelado que se utiliza para describir los principales nodos físicos que se necesitan para desplegar el sistema y la relación entre ellos mediante los protocolos de comunicación. En la Figura 1, se muestra una propuesta de la vista de despliegue de la herramienta.

Figura 1: Diagrama de despliegue

A continuación se expone una breve descripción de los nodos y sus conexiones:

Cliente: Este nodo representa todos los clientes que se puede hacer peticiones al servidor. La conexión entre los mismos puede ser mediante el protocolo TCP/IP o el protocolo Puerto Serie.

TETSCADA: Este nodo representa al conversor de protocolo que sirve como transporte a las peticiones.

Servidor: Este nodo representa a los dos tipos de servidores que pueden ser iniciados. El protocolo empleado para responder las solicitudes estará en dependencia del empelado por el cliente, o sea TCP/IP o Puerto Serie.

3.3. Implementación

Programación Extrema plantea que la implementación de un software se hace de forma iterativa, obteniendo al final de cada iteración un producto funcional que debe ser probado y mostrado al cliente para que los desarrolladores trabajen conociendo las opiniones dadas sobre el mismo.

Durante el transcurso de las iteraciones se realiza la implementación de las historias de usuario que fueron definidas anteriormente.

A continuación se describen las tareas de programación referentes a las HU que se implementarán en cada iteración:

Iteración 1

Tabla 12: Tarea 1 de la HU Iniciar Servidor

Número tarea: 1	**Historia de Usuario # 4**
Nombre tarea: Definición y desarrollo de la interfaz iniciar servidor.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 19/11/2012	**Fecha fin:** 23/11/2012
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza el diseño de la pantalla iniciar servidor. Se crea el diseño realizando una correspondencia entre los tipos de datos a entrar, los datos soportados y los componentes visuales que más se adapten a estos.	

Tabla 13: Tarea 2 de la HU Iniciar Servidor

Número tarea: 2	**Historia de Usuario # 4**
Nombre tarea: Validación de la entrada de datos	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 26/11/2012	**Fecha fin:** 30/11/2012
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza la validación de datos teniendo presente los tipos de datos soportados y los posibles a entrar por los usuarios, así como los campos obligatorios a llenar.	

Tabla 14: Tarea 3 de la HU Iniciar Servidor

Número tarea: 3	**Historia de Usuario # 4**
Nombre tarea: Prueba de la aplicación iniciar servidor.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 03/11/02012	**Fecha fin:** 07/12/2012
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se le realizan una serie de pruebas funcionales a la aplicación, con la finalidad de obtener posibles fallas al realizar esta tarea. Para ello se le introducen una serie de	

datos (correctos e incorrectos), para así comparar los resultados devueltos y los esperados.

Tabla 15: Tarea 1 de la HU Detener Servidor

Número tarea: 1	**Historia de Usuario # 5**
Nombre tarea: Validación del estado del servidor	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 10/12/2012	**Fecha fin:** 14/12/2012
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza la validación del estado del servidor, el mismo debe estar iniciado para poder ejecutar la tarea.	

Tabla 16: Tarea 1 de la HU Guardar Configuración

Número tarea: 1	**Historia de Usuario # 10**
Nombre tarea: Selección de los datos de configuración a guardar	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 17/12/2012	**Fecha fin:** 18/12/2012
Programador responsable: Yanet Reyes González	
Descripción: Se realiza la selección de los datos a guardar en el archivo de configuración.	

Tabla 17: Tarea 2 de la HU Guardar Configuración

Número tarea: 2	**Historia de Usuario # 10**
Nombre tarea: Creación del fichero de configuración	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 19/12/2012	**Fecha fin:** 21/12/2012
Programador responsable: Yanet Reyes González	
Descripción: Una vez obtenidos todos los datos a guardar se crea el fichero que contendrá toda la información necesaria del sistema.	

Tabla 18 Tarea 1 de la HU Cargar Configuración

Número tarea: 1	**Historia de Usuario # 11**
Nombre tarea: Selección del fichero de configuración.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 07/01/2013	**Fecha fin:** 08/01/2013
Programador responsable: Yanet Reyes González	
Descripción: Se selecciona el fichero que contiene la configuración que se quiere cargar.	

Tabla 19: Tarea 2 de la HU Cargar Configuración

Número tarea: 2	**Historia de Usuario # 11**
Nombre tarea: Mostrar datos de configuración	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 09/01/2013	**Fecha fin:** 11/01/2013
Programador responsable: Yanet Reyes González	
Descripción: Se muestran todos los datos obtenidos del fichero de configuración.	

Tabla 20: Tarea 1 de la HU Conectar cliente

Número tarea: 1	**Historia de Usuario # 8**
Nombre tarea: Validación de los parámetros de configuración.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio:14/01/02013	**Fecha fin:** 15/01/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza la validación de los parámetros de configuración para garantizar la conexión del cliente con el servidor, para ello se hace necesaria la verificación de todos los datos entrados por el usuario.	

Tabla 21: Tarea 2 de la HU Conectar cliente

Número tarea: 2	**Historia de Usuario # 8**
Nombre tarea: Conexión con el servidor.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 16/01/2013	**Fecha fin:**18/01/2013

Programador responsable: Yainier Reyna Mendoza
Descripción: Se realiza la conexión con el servidor.

Tabla 22: Tarea 1 de la HU Desconectar cliente

Número tarea: 1	**Historia de Usuario # 9**
Nombre tarea: Validación del estado del cliente	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 21/01/2013	**Fecha fin:** 23/01/2013
Programador responsable: Yanet Reyes González	
Descripción: Se realiza la verificación del estado del cliente el cual debe ser "Conectado"	

Tabla 23: Tarea 2 de la HU Desconectar cliente

Número tarea: 2	**Historia de Usuario # 9**
Nombre tarea: Desconexión con el servidor.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 24/01/2013	**Fecha fin:** 25/01/2013
Programador responsable: Yanet Reyes González	
Descripción: Se realiza la desconexión con el servidor.	

Iteración 2

Tabla 24: Tarea 1 de la HU Enviar Datos

Número tarea: 1	**Historia de Usuario # 6**
Nombre tarea: Verificación del estado de conexión.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 30/01/2013	**Fecha fin:** 01/02/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se verifica el estado del cliente que debe ser "Conectado" para poder proceder al envío de los datos al servidor.	

Tabla 25: Tarea 1 de la HU Recibir Datos

Número tarea: 1	**Historia de Usuario # 7**
Nombre tarea: Verificación del estado del servidor	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 04/02/2013	**Fecha fin:** 06/02/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza la comprobación del estado del servidor que debe ser "Online" para poder recibir los datos.	

Tabla 26: Tarea 2 de la HU Recibir Datos

Número tarea: 2	**Historia de Usuario # 7**
Nombre tarea: Recepción de los datos	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 07/02/2013	**Fecha fin:** 08/02/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza la recepción de los datos enviados desde el cliente.	

Tabla 27: Tarea 1 de la HU Crear cliente

Número tarea: 1	**Historia de Usuario # 1**
Nombre tarea: Definición y desarrollo de la interfaz crear cliente.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 11/02/2013	**Fecha fin:** 15/02/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza el diseño de la pantalla crear cliente. Se crea el diseño realizando una correspondencia entre los tipos de datos a entrar, los datos soportados y los componentes visuales que más se adapten a estos.	

Tabla 28: Tarea 2 de la HU Crear cliente

Número tarea: 2	**Historia de Usuario # 1**
Nombre tarea: Validación de la entrada de datos	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1

Fecha inicio: 18/02/2013	**Fecha fin:** 22/02/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza la validación de datos teniendo presente los tipos de datos soportados y los posibles a entrar por los usuarios, así como los campos obligatorios a llenar.	

Tabla 29: Tarea 3 de la HU Crear cliente

Número tarea: 3	**Historia de Usuario # 1**
Nombre tarea: Creación de un cliente	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 25/02/2013	**Fecha fin:** 27/02/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza la creación de un cliente, el mismo puede ser un Cliente TCP o un Cliente Serie.	

Tabla 30: Tarea 4 de la HU Crear cliente

Número tarea: 4	**Historia de Usuario # 1**
Nombre tarea: Prueba de la aplicación crear cliente.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 28/02/2013	**Fecha fin:**01/03/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se le realizan una serie de pruebas funcionales a la aplicación, con la finalidad de obtener posibles fallas al realizar esta tarea. Para ello se le introducen una serie de datos (correctos e incorrectos), para así comparar los resultados devueltos y los esperados.	

Tabla 31: Tarea 1 de la HU Modificar cliente

Número tarea: 1	**Historia de Usuario # 2**
Nombre tarea: Definición y desarrollo de la interfaz modificar cliente.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 04/03/2013	**Fecha fin:**08/03/2013

Programador responsable: Yainier Reyna Mendoza
Descripción: Se realiza el diseño de la pantalla modificar cliente. Se crea el diseño realizando una correspondencia entre los tipos de datos a modificar, los datos soportados y los componentes visuales que más se adapten a estos.

Tabla 32: Tarea 2 de la HU Modificar cliente

Número tarea: 2	**Historia de Usuario # 2**
Nombre tarea: Validación de la entrada de datos	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 11/03/2013	**Fecha fin:** 12/03/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza la validación de datos teniendo presente los tipos de datos soportados y los posibles a entrar por los usuarios, así como los campos obligatorios a llenar.	

Tabla 33: Tarea 3 de la HU Modificar cliente

Número tarea: 3	**Historia de Usuario # 2**
Nombre tarea: Modificación del cliente	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 13/03/2013	**Fecha fin:** 15/03/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza la modificación del cliente y se muestran los cambios en el visual de la aplicación.	

Tabla 34: Tarea 1 de la HU Eliminar cliente

Número tarea: 1	**Historia de Usuario # 3**
Nombre tarea: Validación de los datos de entrada	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 18/03/2013	**Fecha fin:** 19/03/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza la validación de los datos de entrada teniendo en cuenta	

que el cliente que se va a eliminar no esté conectado al servidor.

Tabla 35: Tarea 2 de la HU Eliminar cliente

Número tarea: 2	**Historia de Usuario # 3**
Nombre tarea: Eliminación del cliente	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 20/03/2013	**Fecha fin:** 22/03/2013
Programador responsable: Yainier Reyna Mendoza	
Descripción: Se realiza la eliminación del cliente y se muestran los cambios en el visual de la aplicación.	

Las tareas restantes se encuentran descritas en el Anexo 3.

3.4. Vistas de la aplicación

Al ejecutarse la aplicación se muestra una ventana donde el usuario escogerá el modo de ejecución en que va a trabajar (Figura 2). En dependencia del modo de ejecución seleccionado se muestra el ambiente de trabajo Cliente (Figura 3) o Servidor (Figura 4). En el modo Cliente tendrá la opción de crear conexiones tipo TCP y conexiones tipo Serie haciendo clic derecho sobre el tipo que desee. Aparecerá una nueva ventana con los elementos visuales correspondientes al tipo de cliente: TCP (Figura 5) o Serie (Figura 6). En el modo Servidor podrá Iniciar o Detener el mismo con el botón correspondiente, trabajar en este modo le permitirá conocer si el servidor fue iniciado o detenido, si se ha establecido una nueva conexión con el mismo así como recibir los mensajes enviados por los clientes (Figura 7). En la Figura 8 se muestra la ventana del módulo de Estadísticas.

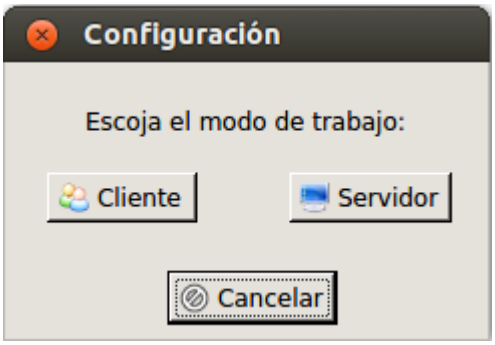

Figura 2: Modo de Ejecución

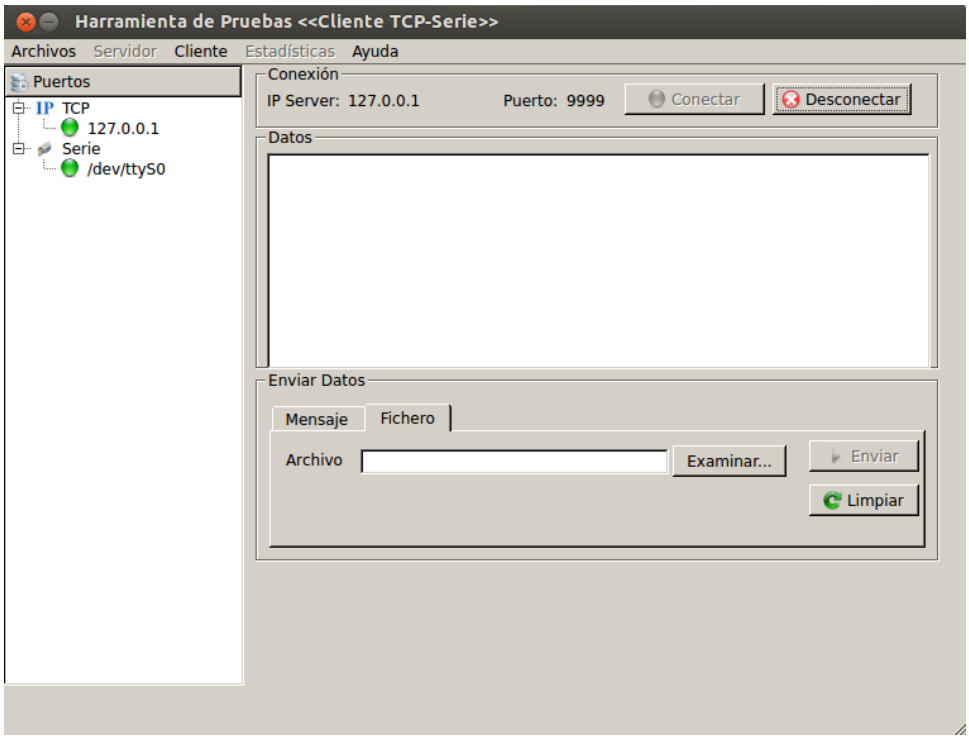

Figura 3: Modo Cliente

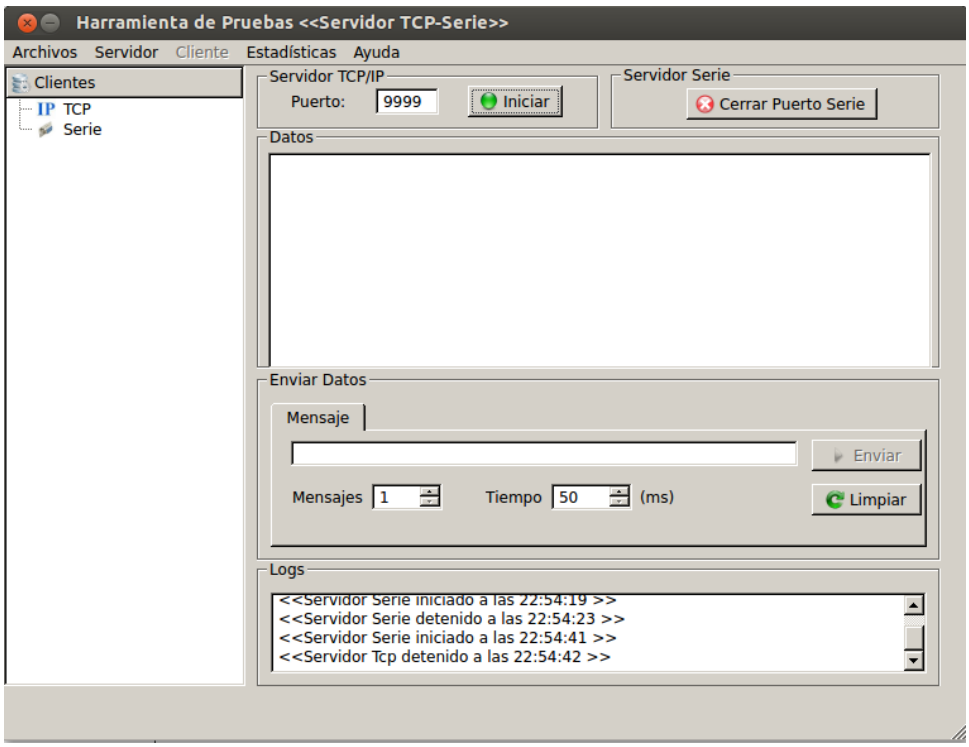

Figura 4: Modo Servidor

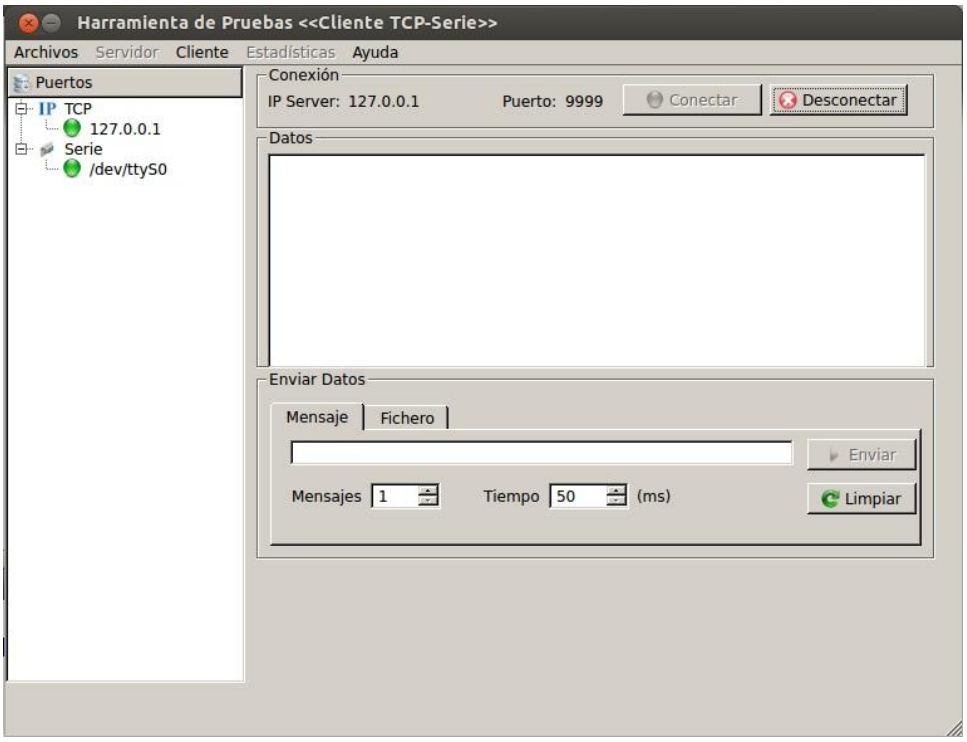

Figura 5: Cliente TCP

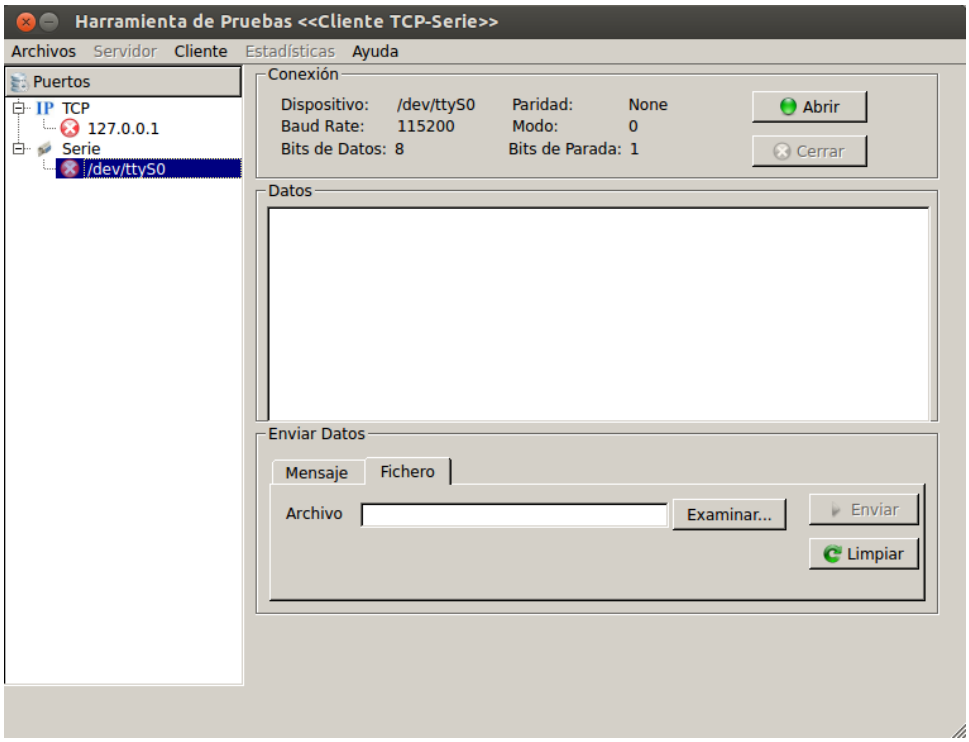

Figura 6: Cliente Serie

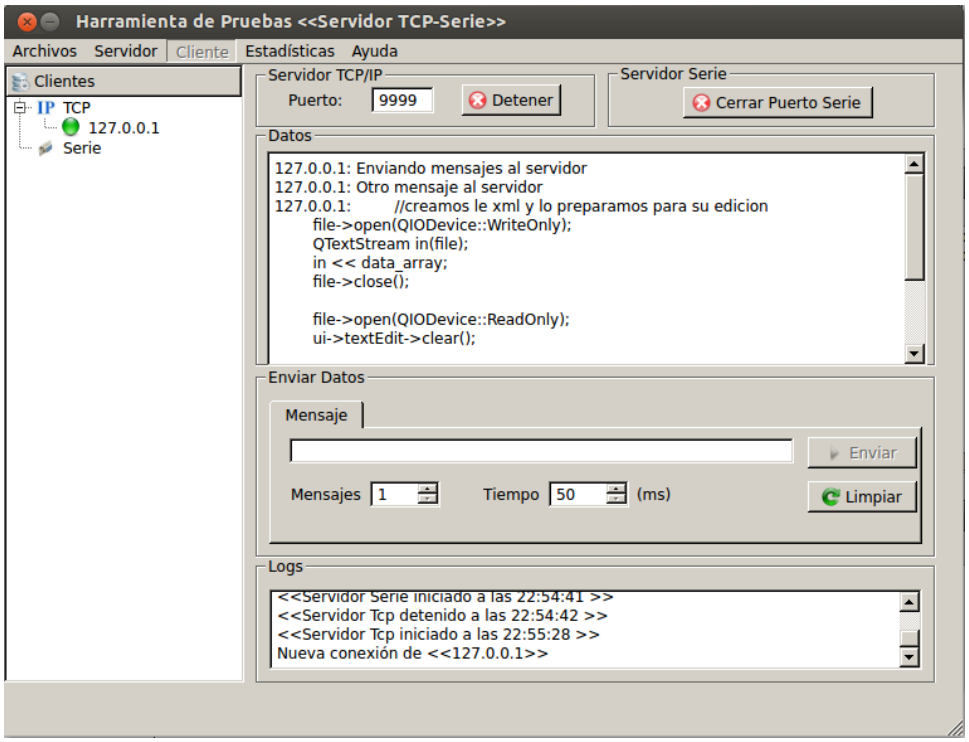

Figura 7: Servidor Iniciado

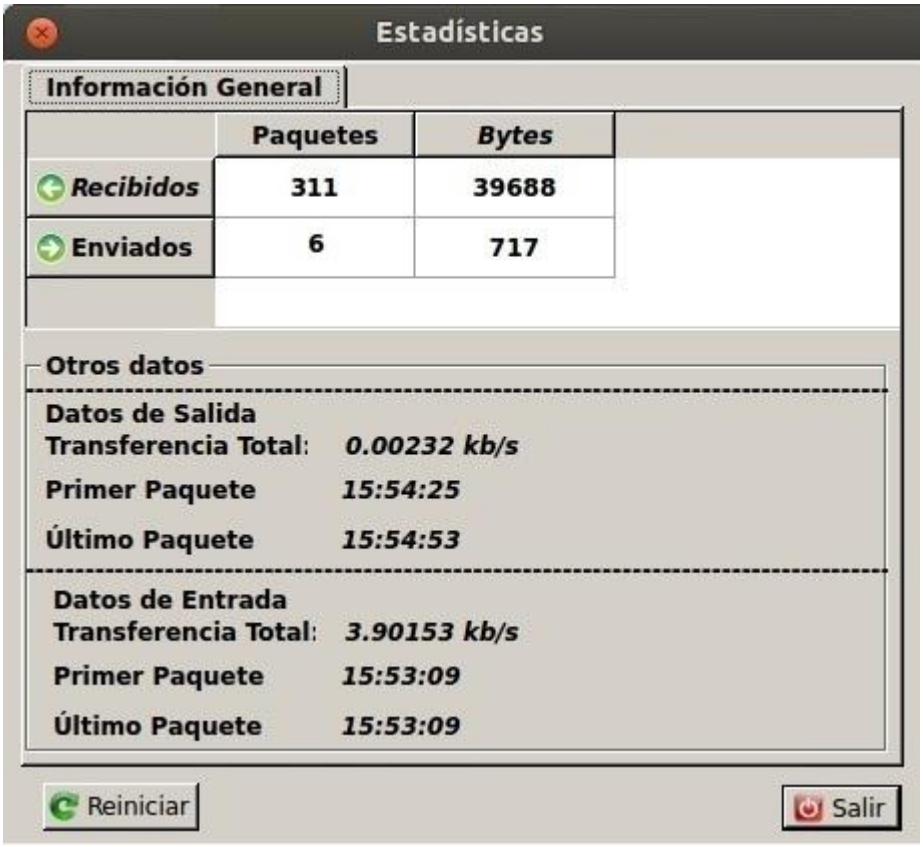

Figura 8: Módulo Estadístico

3.5. Prueba

Una de las fases más importantes del ciclo de vida antes de entregar un software para su explotación es la fase de pruebas. Antes de que el software se le entregue al usuario final es necesario realizar pruebas con el objetivo de detectar errores de la aplicación y la documentación; este proceso resulta de gran importancia ya que da una medida de la calidad del producto siempre que se lleve a cabo de forma apropiada.

Las pruebas son una de las prácticas fundamentales en las cuales se basa XP, a las cuales divide en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñadas por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñada por el cliente final.

3.5.1. Pruebas de aceptación

Las pruebas de aceptación en XP, también llamadas pruebas del cliente son especificadas por el cliente y se centran en las características y funcionalidades generales del sistema que son visibles y revisables por parte del cliente. Se definen para cada historia de usuario y los clientes son responsables de verificar que los resultados de estas pruebas sean

correctos. Durante una iteración la historia de usuario seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación.

Las pruebas de aceptación funcionan como una caja negra, cada una de ellas representa una salida esperada del sistema. Cada historia de usuario tiene tantas pruebas de aceptación como sean necesarias para garantizar su correcto funcionamiento, y no se considera completa hasta que no supera todas sus pruebas de aceptación. El cliente con ayuda del probador define las pruebas de aceptación para cada historia de usuario a principio de cada iteración.

A continuación se detallan las pruebas de aceptación propuestas por iteración para una mayor organización de las mismas.

3.5.1.1. Iteración 1

Tabla 36: DCP Modificar Cliente

Caso de prueba de aceptación	
Código: HU2_P1	**Historia de usuario:** 2
Nombre: Modificar cliente.	
Descripción: Esta funcionalidad es la que permite modificar los clientes que hayan sido creados.	
Condiciones de ejecución: El actor deberá haber iniciado la aplicación en el modo de ejecución Cliente.	
Entradas/Pasos de ejecución: Clic derecho en el cliente que se va a modificar opción “Editar” y se insertan los nuevos datos.	
Resultado esperado: Si los datos son correctos se modifican los datos del cliente, en caso contrario muestra un mensaje indicando que los datos son incorrectos y muestra un ejemplo correcto.	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 37: DCP Eliminar Cliente

Caso de prueba de aceptación	
Código: HU3_P1	**Historia de usuario:** 3
Nombre: Eliminar cliente.	
Descripción: Esta funcionalidad es la que permite eliminar los clientes que hayan sido creados.	
Condiciones de ejecución: Debe existir algún cliente.	
Entradas/Pasos de ejecución: Clic derecho encima del cliente a eliminar opción “Eliminar” y se confirma la operación.	
Resultado esperado:	

El cliente es eliminado.
Evaluación de la prueba: Prueba satisfactoria

Tabla 38: DCP Iniciar Servidor

Caso de prueba de aceptación	
Código: HU4_P1	**Historia de usuario:** 4
Nombre: Iniciar servidor.	
Descripción: Esta funcionalidad es la que permite iniciar los servicios del servidor.	
Condiciones de ejecución: El actor deberá haber iniciado la aplicación en el modo de ejecución Servidor y especificado el puerto de conexión.	
Entradas/Pasos de ejecución: En el menú Servidor seleccionar el servidor que se va a iniciar, clic en “Iniciar”.	
Resultado esperado: Se inician los servicios del servidor, en el caso del servidor Puerto Serie si no encuentra el dispositivo aparece un mensaje indicándolo.	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 39: DCP Detener Servidor

Caso de prueba de aceptación	
Código: HU5_P1	**Historia de usuario:** 5
Nombre: Detener servidor.	
Descripción: Esta funcionalidad es la que permite detener los servicios del servidor.	
Condiciones de ejecución: El actor deberá haber iniciado la aplicación en el modo de ejecución Servidor y especificado el puerto de conexión.	
Entradas/Pasos de ejecución: En el menú Servidor seleccionar el servidor que se va a detener y hacer clic en Detener.	
Resultado esperado: El servidor es detenido.	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 40: DCP Conectar cliente

Caso de prueba de aceptación	
Código: HU8_P1	**Historia de usuario: 8**
Nombre: Conectar cliente.	
Descripción: Esta funcionalidad es la que permite conectar el cliente al servidor.	
Condiciones de ejecución: El actor deberá haber especificado IP y puerto de conexión.	

Entradas/Pasos de ejecución: Se selecciona el cliente que se va a conectar y se hace clic en el botón Conectar.
Resultado esperado: El cliente se conecta de forma satisfactoria y se muestra un log indicándolo.
Evaluación de la prueba: Prueba satisfactoria

Tabla 41: DCP Desconectar cliente

Caso de prueba de aceptación	
Código: HU9_P1	**Historia de usuario: 9**
Nombre: Desconectar cliente.	
Descripción: Esta funcionalidad es la que permite desconectar el cliente del servidor.	
Condiciones de ejecución: El actor deberá haber conectado el cliente al servidor.	
Entradas/Pasos de ejecución: Se selecciona el cliente que se va a desconectar y se hace clic en el botón Desconectar.	
Resultado esperado: El cliente se desconecta de forma satisfactoria y se muestra un log indicándolo.	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 42: DCP Guardar Configuración

Caso de prueba de aceptación	
Código: HU10_P1	**Historia de usuario: 10**
Nombre: Guardar configuración.	
Descripción: Esta funcionalidad es la que permite guardar la configuración del modo de ejecución en el que se encuentre.	
Condiciones de ejecución: El actor deberá haber creado alguna configuración.	
Entradas/Pasos de ejecución: En el menú Archivo escoger opción Guardar y especificar la dirección donde desea guardar la configuración.	
Resultado esperado: Se guarda la configuración actual de forma satisfactoria.	
Evaluación de la prueba: Prueba satisfactoria	

3.5.1.2. Iteración 2

Tabla 43: DCP Enviar Datos

Caso de prueba de aceptación

Código: HU6_P1	**Historia de usuario:** 6
Nombre: Enviar datos.	
Descripción: Esta funcionalidad es la que permite enviar datos en ambos modos de ejecución.	
Condiciones de ejecución: El actor deberá haber establecido la conexión entre el servidor y el cliente.	
Entradas/Pasos de ejecución: Se escribe el mensaje o se selecciona el archivo que desean enviar y luego clic en el botón Enviar.	
Resultado esperado: El mensaje es enviado satisfactoriamente, en caso contrario muestra un mensaje indicando que el mismo no pudo ser enviado.	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 44: DCP Recibir Datos

Caso de prueba de aceptación	
Código: HU7_P1	**Historia de usuario: 7**
Nombre: Recibir datos.	
Descripción: Esta funcionalidad es la que permite recibir datos en ambos modos de ejecución.	
Condiciones de ejecución: El actor deberá haber establecido la conexión entre el servidor y el cliente.	
Entradas/Pasos de ejecución: Esperar la llegada de los datos.	
Resultado esperado: Los datos son recibidos de manera satisfactoria.	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 45: DCP Cargar Configuración

Caso de prueba de aceptación	
Código: HU11_P1	**Historia de usuario: 11**
Nombre: Cargar configuración.	
Descripción: Esta funcionalidad es la que permite cargar alguna configuración del modo de ejecución en el que se encuentre.	
Condiciones de ejecución: El actor deberá haber guardado alguna configuración.	
Entradas/Pasos de ejecución: En el menú Archivo escoger opción Cargar y seleccionar el archivo que desea cargar.	
Resultado esperado: Al seleccionar la opción Cargar se muestra un mensaje de alerta preguntando si desea reemplazar la configuración actual, en caso de aceptar le da la opción de seleccionar el archivo de la nueva configuración, en caso contrario permanece en	

la misma configuración.
Evaluación de la prueba: Prueba satisfactoria

Tabla 46: DCP Enviar fichero

Caso de prueba de aceptación	
Código: HU12_P1	**Historia de usuario: 12**
Nombre: Enviar fichero.	
Descripción: Esta funcionalidad es la que permite enviar un fichero de texto al servidor.	
Condiciones de ejecución: El actor debe seleccionar un fichero de texto.	
Entradas/Pasos de ejecución: Clic en “Examinar” seleccionar el fichero que desea enviar, clic en “Enviar”.	
Resultado esperado: El fichero es enviado de forma correcta.	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 47: DCP Enviar datos durante x tiempo

Caso de prueba de aceptación	
Código: HU13_P1	**Historia de usuario: 13**
Nombre: Enviar datos durante x tiempo.	
Descripción: Esta funcionalidad es la que permite enviar datos al servidor durante un tiempo determinado.	
Condiciones de ejecución: El actor especifica el tiempo que estará el cliente enviando datos al servidor.	
Entradas/Pasos de ejecución: En la pestaña Mensaje se especifica un tiempo mayor a 50 ms y clic en “Enviar”.	
Resultado esperado: Se estará enviando datos al servidor durante el tiempo especificado.	
Evaluación de la prueba: Prueba satisfactoria	

3.6. Resultado de las pruebas

A lo largo del desarrollo del sistema, y como propone la metodología XP, se realizaron pruebas unitarias desarrolladas por los programadores y encargadas de verificar el código de forma automática; además se diseñaron y aplicaron 13 casos de prueba basados en las historias de usuario con el objetivo de comprobar el cumplimiento de los requisitos funcionales planteados por el cliente.

Al finalizar con todas las pruebas se detectaron errores funcionales en la aplicación los cuales fueron corregidos adecuadamente en el momento de su detección, logrando con esto que un error no persistiera y afectara de alguna manera el funcionamiento general del software.

A continuación se muestra un gráfico con las no conformidades (NC) detectadas y resueltas en cada iteración del proceso de desarrollo del software:

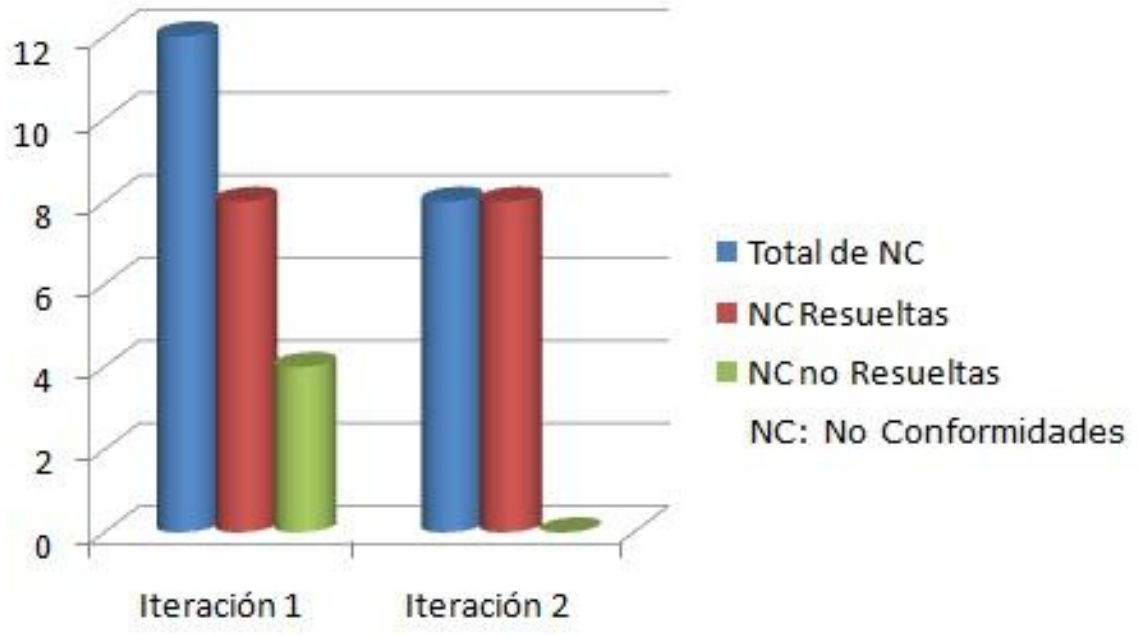

Gráfico 1. No Conformidades detectadas por Iteración.

3.7. Conclusiones parciales

En el presente capítulo:

- ✓ Se mostró una breve descripción de los estándares de codificación que favorecerán la legibilidad y el mantenimiento del software para futuros cambios.
- ✓ Quedaron plasmados los procesos llevados a cabo para la implementación de la herramienta.
- ✓ Se empleó un diagrama de despliegue para un mejor entendimiento del sistema.
- ✓ Se realizaron las pruebas de aceptación a cada historia de usuario, las cuales ofrecerán al cliente conformidad y seguridad ante las funcionalidades del sistema.

CONCLUSIONES

Una vez concluida la investigación y la implementación de la herramienta propuesta en el presente trabajo de diploma, se le han dado cumplimiento a los objetivos planteados y se obtuvieron los resultados que a continuación se mencionan:

- ✓ A partir de la selección y uso de XP como metodología de desarrollo, se generaron un conjunto de artefactos que guiaron el desarrollo de la solución y sirven como base para incorporar nuevas funcionalidades a la herramienta.

- ✓ Se desarrolló una herramienta denominada “Test-Soft” que permite la realización de pruebas automatizadas a una amplia gama de conversores de protocolo.

- ✓ Se realizaron un conjunto de pruebas de aceptación, cuyos resultados permiten afirmar que la herramienta cumple satisfactoriamente con los requerimientos especificados por el cliente.

RECOMENDACIONES

Al finalizar la investigación se recomienda:

- Enriquecer el módulo Estadístico e incorporar una Base de Datos que almacene los datos del módulo Estadístico.
- Lograr adaptar la herramienta Test-Soft para otros sistemas operativos.
- Orientar la implementación al trabajo con flujo.
- Especificar en la aplicación los tamaños de los mensajes enviados.
- Incorporar el Manual de usuario a la ayuda de la aplicación.
- Hacer el empaquetado de la herramienta.

REFERENCIAS BIBLIOGRÁFICAS

1. Acuña, K.B. 2009. ***SELECCIÓN DE METODOLOGÍAS DE DESARROLLO PARA APLICACIONES WEB.*** **2009.**
2. Autores, Varios. 2011. **El proyecto Eclipse. [En línea] 2011. http://es.scribd.com/doc/35236225/Eclipse-Java-en-Espanol..**
3. Baldur. 2012. ***¿Qué es Eclipse? .*** **2012.**
4. Cáceres, P. 2012. ***Procesos Ágiles para el Desarrollo de Aplicaciones Web.*** **2012.**
5. Corporation, O. 2012. **Bienvenido a NetBeans. [En línea] 2012. http://netbeans.org/index_es.html..**
6. Figueroa, Roberth G., Solís, Camilo J. y Cabrera, Armando A. 2011. ***METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.*** **Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación: s.n., 2011.**
7. Fuentes, Ernesto Miguel Figueredo y Lorenzo, Yanay Viera. 2009. ***Proceso de capacitación que propone RUP según normas de calidad.*** **2009.**
8. Garcia, Joaquin. 2003. ***Ingenieros Software.*** **2003.**
9. Granada., Profesores del dpto de lenguajes y sistemas informáticos de la universidad de. 2009. **Aporia.** ***Aporia.*** **[En línea] 15 de Noviembre de 2009. http://aporia.ugr.es/lsi/.**
10. 2006**Grid TICs***Grid TICs***http://gridtics.frm.utn.edu.ar**
11. Martínez Salazar, Eduardo. 2012. **E-GOV.** ***E-GOV.*** **[En línea] 8 de Octubre de 2012. http://www.egov.ufsc.br/portal/conteudo/propuesta-de-procedimiento-para-realizar-pruebas-de-caja-blanca-las-aplicaciones-que-se-des.**
12. Penadés, Patricio Letelier y Mª Carmen. 2004. **Métodologías Ágiles para el desarrollo de software. eXtreme Programming (XP. [En línea] 2004. http://www.willydev.net/descargas/masyxp.pdf.**
13. Pressman, Roger S. 2005. ***Ingeniería de Software: Un enfoque Práctico.*** **2005.**
14. Prieto, Paloma. 2007. **Observatorio Tecnológico. [En línea] Gobierno de España, 8 de Octubre de 2007. http://recursostic.educacion.es/observatorio/web/en/component/content/article/19-monograficos/502-monografico-lenguajes-de-programacion.**
15. Rodríguez, A.F.G. 2012. ***LEGUAJES DE PROGRAMACIÓN.*** **2012.**
16. Ruiz, Alberto. 2009. **Observatorio Tecnológico. [En línea] Gobierno de España, 21 de Agosto de 2009. http://recursostic.educacion.es/observatorio/web/en/software/programacion/911-monografico-java.**
17. Society Computer, IEEE. 1994. ***Software Engineering Standards.*** **1994. 2.**
18. Systems, S. Enterprise Architect. 2007. **Herramienta de diseño UML. . [En línea] 2007. http://www.sparxsystems.com.ar/products/ea.html.**
19. varios, Autores. 2011. ***Inicio Rápido con Qt 4.*** **2011.**

20. Varios, Autores. 2007. ***Paradigma visual para UML (Plataforma Java) (Visual Paradigm for UML.*** **2007.**

21. Villoria, Cristina. 2009. **Observatorio Tecnológico. [En línea] Gobierno de España, 21 de Diciembre de 2009. http://recursostic.educacion.es/observatorio/web/en/software/programacion/745-introduccion-a-la-programacion-con-el-lenguaje-c.**

22. Xavier Ferré Grau, M.I.S.S. 2012. ***Desarrollo Orientado a Objetos con UML.*** **2012.**

ANEXOS

Anexo 1: Historias de usuario

Tabla 48: HU Modificar Cliente

Historias de usuario	
Número: 2	**Nombre Historia de Usuario:** Modificar Cliente.
Actor: Probador	**Iteración Asignada:** 1
Prioridad de Negocio: Alta	**Puntos Estimados:** 2
Riesgo en desarrollo: Medio	**Puntos Reales:** 2
Descripción: El actor podrá modificar los clientes que se van a conectar al servidor.	
Observaciones: El actor deberá haber iniciado la aplicación en el modo de ejecución Cliente.	

Tabla 49: HU Eliminar Cliente

Historias de usuario	
Número: 3	**Nombre Historia de Usuario:** Eliminar Cliente.
Actor: Probador	**Iteración Asignada:** 1
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El actor podrá eliminar los clientes que se van a conectar al servidor.	
Observaciones: El actor deberá haber iniciado la aplicación en el modo de ejecución Cliente.	

Tabla 50: HU Iniciar Servidor

Historias de usuario	
Número: 4	**Nombre Historia de Usuario:** Iniciar servidor.
Actor: Probador	**Iteración Asignada:** 1
Prioridad de Negocio: Media	**Puntos Estimados:** 3
Riesgo en desarrollo: Medio	**Puntos Reales:** 3
Descripción: El actor podrá iniciar el servidor.	

Observaciones: El actor deberá haber iniciado la aplicación en el modo de ejecución Servidor y especificado el puerto de conexión.

Tabla 51: HU Detener Servidor

Historias de usuario	
Número: 5	**Nombre Historia de Usuario:** Detener servidor.
Actor: Probador	**Iteración Asignada:** 1
Prioridad de Negocio: Media	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El actor podrá detener el servidor.	
Observaciones: El actor deberá haber iniciado la aplicación en el modo de ejecución Servidor.	

Tabla 52: HU Enviar Datos

Historias de usuario	
Número: 6	**Nombre Historia de Usuario:** Enviar datos.
Actor: Probador	**Iteración Asignada:** 2
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El actor podrá enviar datos en ambos modos de ejecución.	
Observaciones: El actor deberá haber establecido la conexión entre el servidor y el cliente.	

Tabla 53: HU Recibir Datos

Historias de usuario	
Número: 7	**Nombre Historia de Usuario:** Recibir datos.
Actor: Probador	**Iteración Asignada:** 2
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El actor podrá recibir datos en ambos modos de ejecución.	
Observaciones: El actor deberá haber establecido la conexión entre el servidor y el cliente.	

Tabla 54: HU Conectar Cliente

Historias de usuario	
Número: 8	**Nombre Historia de Usuario:** Conectar cliente.
Actor: Probador	**Iteración Asignada:** 1
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El actor podrá conectar el cliente al servidor.	
Observaciones: El actor deberá haber especificado IP y puerto de conexión.	

Tabla 55: HU Desconectar Cliente

Historias de usuario	
Número: 9	**Nombre Historia de Usuario:** Desconectar cliente.
Actor: Probador	**Iteración Asignada:** 1
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El actor podrá desconectar el cliente del servidor.	
Observaciones: El actor deberá haber conectado el cliente al servidor.	

Tabla 56: HU Guardar Configuración

Historias de usuario	
Número: 10	**Nombre Historia de Usuario:** Guardar configuración.
Actor: Probador	**Iteración Asignada:** 1
Prioridad de Negocio: Media	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El actor podrá guardar la configuración del modo de ejecución en el que se encuentre.	
Observaciones: El actor deberá haber creado alguna configuración.	

Tabla 57: HU Cargar Configuración

Historias de usuario	
Número: 11	**Nombre Historia de Usuario:** Cargar configuración.
Actor: Probador	**Iteración Asignada:** 2

Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales**: 1
Descripción: El actor podrá cargar alguna configuración del modo de ejecución en el que se encuentre.	
Observaciones: El actor deberá haber guardado alguna configuración.	

Tabla 58: HU Enviar fichero

Historias de usuario	
Número: 12	**Nombre Historia de Usuario:** Enviar fichero.
Actor: Probador	**Iteración Asignada:** 2
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El actor podrá enviar un fichero de texto al servidor.	
Observaciones: El actor debe seleccionar un fichero de texto.	

Tabla 59: HU Enviar datos durante x tiempo

Historias de usuario	
Número: 13	**Nombre Historia de Usuario:** Enviar datos durante x tiempo.
Actor: Probador	**Iteración Asignada:** 2
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El actor podrá enviar datos al servidor durante un tiempo determinado.	
Observaciones: El actor especifica el tiempo que estará el cliente enviando datos al servidor.	

Tabla 60: HU Analizar envío de datos por cliente

Historias de usuario	
Número: 14	**Nombre Historia de Usuario:** Analizar envío de datos por cliente.
Actor: Probador	**Iteración Asignada:** 2
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El actor podrá visualizar un resumen de los datos que ha enviado desde un cliente al servidor.	

Observaciones: El actor selecciona de cual cliente quiere visualizar la información.

Tabla 61: HU Analizar envío de datos de todos los clientes

Historias de usuario	
Número: 15	**Nombre Historia de Usuario:** Analizar envío de datos de todos los clientes.
Actor: Probador	**Iteración Asignada:** 2
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El actor podrá visualizar un resumen de los datos que se han enviado desde todos los clientes al servidor.	
Observaciones: El actor selecciona la opción para visualizar todos los datos de los clientes.	

Anexo 2: Tarjetas Clase Responsabilidad Colaboración

Tabla 62: Tarjeta CRC de la clase Server

Server	
Responsabilidades	Colaboradores
-Es la clase que maneja todos los clientes.	• Cliente

Tabla 63: Tarjeta CRC de la clase ClienteControler

ClienteControler	
Responsabilidades	Colaboradores
-Es la clase que maneja las conexiones.	• Cliente

Tabla 64: Tarjeta CRC de la clase Cliente

Cliente	
Responsabilidades	Colaboradores
-Es la clase que contiene la información necesaria para generar un nuevo cliente.	

Tabla 65: Tarjeta CRC de la clase ClienteTCP

ClienteTCP	
Responsabilidades	Colaboradores
-Es la clase que contiene la información necesaria para generar un nuevo cliente del tipo TCP.	• Cliente

Tabla 66: Tarjeta CRC de la clase ClienteSerie

ClienteSerie	
Responsabilidades	Colaboradores
-Es la clase que contiene la información necesaria para generar un nuevo cliente del tipo Serie.	• Cliente

Anexo 3: Descripción de las tareas de programación

Tabla 67: Tarea 1 de la HU Enviar Fichero

Número tarea: 1	**Historia de Usuario # 12**
Nombre tarea: Selección y lectura del fichero.	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 25/03/2013	**Fecha fin:** 29/03/2013

Programador responsable: Yanet Reyes González
Descripción: Se realiza la selección del fichero a enviar y posteriormente la lectura de su contenido para ser enviada al servidor.

Tabla 68: Tarea 1 de la HU Enviar datos durante x tiempo

Número tarea: 1	**Historia de Usuario # 13**
Nombre tarea: Verificación del tiempo de envío.	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 01/04/2013	**Fecha fin:** 05/04/2013
Programador responsable: Yanet Reyes González	
Descripción: Se realiza la verificación del tiempo que el probador escoja en la aplicación, si este tiempo es mayor de 50 milisegundos entonces se ejecutará esta tarea.	

Tabla 69: Tarea 1 de la HU Analizar envío de datos por cliente

Número tarea: 1	**Historia de Usuario # 14**
Nombre tarea: Definición y desarrollo de la interfaz Analizar envío de datos por cliente del módulo de estadísticas.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.4**
Fecha inicio: 08/04/2013	**Fecha fin:** 10/04/2013
Programador responsable: Yanet Reyes González	
Descripción: Se realiza el diseño de la interfaz teniendo en cuenta los elementos visuales que más se ajusten a la solución.	

Tabla 70: Tarea 2 de la HU Analizar envío de datos por cliente

Número tarea: 2	**Historia de Usuario # 14**
Nombre tarea: Monitorización de los datos enviados por el cliente.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.3**
Fecha inicio: 11/04/2013	**Fecha fin:** 11/04/2013
Programador responsable: Yanet Reyes González	
Descripción: Se realiza la monitorización de todos los datos que entran al servidor mediante las conexiones TCP o Serie.	

Tabla 71: Tarea 3 de la HU Analizar envío de datos por cliente

Número tarea: 3	**Historia de Usuario # 14**
Nombre tarea: Presentación de los datos al probador.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.3**
Fecha inicio: 12/04/2013	**Fecha fin:** 12/04/2013
Programador responsable: Yanet Reyes González	
Descripción: Se realiza la muestra de los datos al probador en el visual de la aplicación.	

Tabla 72: Tarea 1 de la HU Analizar envío de datos de todos los cliente

Número tarea: 1	**Historia de Usuario # 15**
Nombre tarea: Definición y desarrollo de la interfaz Analizar envío de datos de todos los clientes del módulo de estadísticas.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.4**
Fecha inicio: 15/04/2013	**Fecha fin:** 17/04/2013
Programador responsable: Yanet Reyes González	
Descripción: Se realiza el diseño de la interfaz teniendo en cuenta los elementos visuales que más se ajusten a la solución.	

Tabla 73: Tarea 2 de la HU Analizar envío de datos de todos los cliente

Número tarea: 2	**Historia de Usuario # 15**
Nombre tarea: Monitorización de los datos enviados por todos los clientes.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.3**
Fecha inicio: 18/04/2013	**Fecha fin:** 18/04/2013
Programador responsable: Yanet Reyes González	
Descripción: Se realiza la monitorización de todos los datos que entran al servidor mediante las conexiones TCP o Serie.	

Tabla 74: Tarea 3 de la HU Analizar envío de datos de todos los cliente

Número tarea: 3	**Historia de Usuario # 15**
Nombre tarea: Presentación de los datos al probador.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.3**

Fecha inicio: 19/04/2013	**Fecha fin:** 19/04/2013
Programador responsable: Yanet Reyes González	
Descripción: Se realiza la muestra de los datos al probador en el visual de la aplicación.	

GLOSARIO

TETSCADA: Interfaz SCADA para acceso a radio TETRA.

SCADA: Sistema de Supervisión, Control y Adquisición de Datos.

TETRA: Radio Troncalizado Terrestre.

Conversores de Protocolos: son dispositivos que permiten el diálogo entre terminales de datos que utilizan protocolos diferentes.

Calidad: Es la aptitud además de un conjunto de características y propiedades de un servicio, producto o proceso, las cuales satisfacen las necesidades establecidas por un usuario.

CEDIN: Centro de Desarrollo de Informática Industrial.

XP (Programación Extrema): Metodología de desarrollo de software ágil.

Caso de prueba (Test case): Un caso de prueba solamente prueba una sola funcionalidad, es decir por cada prueba que se quiera hacer se describe un caso de prueba. Los Caso de prueba se pueden definir como una entrada y un resultado esperado.

Framework (Marco de Trabajo): En el desarrollo de software es un esquema o patrón el cual se usa para el desarrollo y la implementación de una aplicación, se puede decir además que es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. También se puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Software: Son procedimientos, programas y datos almacenados en un ordenador, es decir estos tienen la tarea de que el hardware (la máquina) cumpla con la realización de todas las tareas que se le orientan.