ESCUELA POLITÉCNICA DEL EJÉRCITO

DISEÑO DE UN MODELO PARA EVALUACION/PRUEBAS DEL SOFTWARE EN BASE A INGENIERIA DE PRUEBAS APLICANDO EL ESTANDAR ISO/IEC 29119 EN LA EMPRESA OMNISOFT DE LA CIUDAD DE QUITO

Tesis de Maestría en Ingeniería de Software

Ing. José Napoleón Páez Escobar.

Director: Máster Ing. Marco Jarrín López

Diciembre 15, 2011

Latacunga – Ecuador

Agradecimientos

En primer lugar, gracias a Dios todo poderoso por este regalo que me ha dado, que es la capacidad de aprender y crecer, sé que durante toda mi vida él ha estado atento y pendiente de mis problemas y dificultades, que sin su bendición no hubiese podido salir adelante. Gracias Señor.

A mi familia, mi madre que apoyo y la confianza que deposito en mi para alcanzar esta meta.
A mi esposa Maritza que fue un apoyo en los momentos difíciles. Gracias por hacer de nuestra casa un hogar.
A mí querida hija que es mi motivo de superación.
A mi abuela que es una persona que me apoyado en momentos difíciles.
A mis tíos y primos por el apoyo brindado.

Así también al ingeniero Marco Jarrín por todo el apoyo como amigo y director de este trabajo.
Al ingeniero Jorge Alarcón por el apoyo como asesor de esta tesis.

Y de manera especial a mis amigos por el apoyo y contribución para el presente documento.

A todos muchas gracias,

José Páez

Índice *Pág.*

ANEXOS

ANEXO A

Ciclo de vida del software

ANEXO B

Elementos Notacionales en UML.

ANEXO C

Manual del modelo de pruebas ISO/IEC 29119

Índice de Figuras

Índice de Tablas.

Índice de Gráficos.

Resumen

Se presenta en este artículo una estrategia para la evaluación/pruebas del software en base a ingeniería de pruebas aplicando el estándar ISO/IEC 29119 de un producto de software. La estrategia define el alcance y la agenda de las pruebas a partir del análisis de riesgo del producto, combinando los casos de prueba con diseño previo y el testing exploratorio. Se definen los ciclos de prueba en función del plan de desarrollo del producto y se elabora una planificación global a partir de sus funcionalidades, la que es revisada y refinada al comenzar cada ciclo de prueba.

El testing cumple un papel fundamental en la estrategia. Por un lado, ayuda a mitigar la posibilidad de equivocarse al realizar el análisis de riesgo del producto, dejando de lado funcionalidades importantes para el negocio. Si en una sesión de testing se encuentra un incidente crítico para el negocio que no fue detectado por los casos de prueba diseñados, el análisis de riesgo debe ser revisado. Por otro lado, complementa la prueba con diseño previo cuando no se dispone del tiempo suficiente en un ciclo como para generar los casos de prueba que cubran las funcionalidades requeridas.

Palabras clave: Ingeniería de software, pruebas, gestión de las pruebas, PruebasFuncionales.

Abstract

Is presented in this paper a strategy for evaluation /testing of software engineering based on tests using the ISO /IEC29119 of a software product. The strategy defines the scope and agenda of the evidence from the product risk analysis, test cases by combining the previous design and exploratory testing. Cycles are defined test based on the product development plan and comprehensive planning is made based on their functionality, which is reviewed andrefinedat the beginning ofeach test cycle.

The testing plays a fundamental role in the strategy. On the one hand, it helps mitigate the possibility of error when performing the risk analysis of the product, leaving out important business functions. If a testing session is a critical incident to the business that was not detected by the test cases designed, the risk analysis should be revised. On the other hand, complements the previous design test when there is not enough time in a cycle to generate test cases that cover the functionality required.

KeyWords: Software engineering, testing, test management, functional testing.

Introducción

Actualmente es una práctica común en la actividad de desarrollo de software. Situaciones como la competencia en el mercado para generar nuevos productos o actualizar versiones, han propiciado que muchos desarrolladores busquen nuevas opciones para generar software de calidad en tiempos muy cortos. Incluso, en ambientes de desarrollo no comercial, la actividad de proporcionar un software.

La creación de nuevas metodologías de desarrollo, cuyas características han permitido alcanzar en diversos grados la rápida construcción de programas, ha considerado a la norma ISO/IEC 29119 que cubrirá el ciclo de vida completo, a través del análisis, diseño, implementación y mantenimiento de las pruebas software, contribuyendo así a que esta se convierta en una alternativa atractiva en la construcción de software. Este modelo permite mejorar la calidad del software notablemente, ya que la prueba de software es una actividad que de una u otra manera es llevada a cabo en algún momento al menos por su desarrollador original. Para ello este puede planear y realizar un proceso de prueba enfocado a un ambiente en el que espera operará el producto.

Actualmente son pocas las herramientas que permiten evaluar al software.Cuando alguien requiere integrar un elemento de software para construir una aplicación osimplemente para utilizarla de forma aislada, la evaluación que se realiza radica en la mayoríade los casos en una prueba manual, que en el peor de los casos no es nada sistematizada nifundamentada. Si el usuario requiriera realizar una prueba más exhaustiva, esta prueba seríamás amplia y formal, pero en raras ocasiones automatizada, por lo que para llevarla a cabo serequiere una cantidad de tiempo considerable. El propósito de este trabajo es presentar una alternativa. Esto se llevará a cabo mediante la implementación de una herramienta quepermita realizar pruebas a sistemas desarrollados.

CAPITULO I

1.-Estado del Arte.

1.1-Definición de Software.

“Se conoce como software *al equipamiento lógico o soporte lógico* de una computadora digital; comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos, que son llamados hardware**”** (López, 2007, p. 89).

1.2.-Historia del Software.

La página web **http://www.tiposdesoftware.com/historia-del-software.htm**, menciona que**:**

> La primera teoría sobre el software fue propuesta por Alan Turing en su ensayo de 1935 sobre números computables, con una aplicación destinada a la toma de decisiones. El término "software" fue utilizado por primera vez de forma escrita por John W. Tukey en 1958.] El estudio de los campos académicos sobre el software se divide en informática y la ingeniería de software.

> Como los programas cada vez entraban más en el reino de firmware y el hardware por si sólo se hacía más pequeño, más barato y más rápido debido a la ley de Moore, los elementos de la computación que primero se consideraban software, pasan a ser hardware. La mayoría de las compañías de hardware hoy en día tienen más programadores de software en nómina que diseñadores de hardware, ya que las

herramientas de software han automatizado muchas de las tareas de los ingenieros de circuitos. Al igual que la industria automotriz, la industria del software ha crecido de unos pocos visionarios que operaban en su garaje con sus prototipos. Steve Jobs y Bill Gates fueron los Henry Ford y Chevrolet Luis de sus tiempos. En el caso del desarrollo de software, el despegue final es generalmente aceptado que se produce con la publicación en la década de 1980 de las especificaciones para el IBM Personal Computer. Hoy su movimiento sería visto como un tipo de público-sourcing. (párr. 1.2.3)

Esta página web **http://www.tiposdesoftware.com/historia-del-software.htm** manifiesta:

Hasta ese momento, el software se incluye con el hardware de los fabricantes de equipos originales (OEM), tales como Data General, Digital Equipment y de IBM. Cuando un cliente compra una minicomputadora, esta incluye el software que es instalado por los ingenieros empleados por el OEM. Las empresas de informática de hardware, no sólo incluyen sus paquetes de software, sino que también asesoren sobre la ubicación de los equipos normalmente en un espacio refrigerado llamado sala de ordenadores. La mayoría de las empresas tenían su software en su contabilidad valorados 0 ya que no podían venderlo. Cuando Data General introdujo su software Data General Nova, una compañía llamada Digidyne intentó instalar este software que ya había adquirido en un equipo distinto. Data General se negó a darle una licencia para poder hacerlo y fueron a los Tribunales. La Corte Suprema dijo que si Digidyne había pagado era propietaria de ese software debía poder instalarlo en el equipo que quisiese, lo que se llamó Digidyne v. Poco después IBM publicó los registros de DOS y nació Microsoft. La decisión de la Corte Suprema permitió valorar el software, patentarlo y comerciar con él. Es difícil imaginar hoy que una vez la gente sentía que el software no valía nada sin una máquina. Hay muchas empresas de éxito hoy en día que venden sólo productos de software, aunque todavía hay muchos problemas comunes de concesión de licencias de software debido a la complejidad de los diseños y documentación, lo que lleva a los trolls de patentes.

Con las especificaciones de software de código abierto y la posibilidad de concesión de licencias de software, nuevas oportunidades se levantaron de herramientas de software que luego se convirtieron en el estándar de facto, como DOS para los sistemas operativos, sino también diversos programas de procesamiento de texto y hojas de cálculo. En un patrón de crecimiento similar, los métodos de desarrollo de propiedad se convirtieron en la metodología estándar de desarrollo de software. (párr. 6,7)

1.3.-Clasificación del software.

En relación a la clasificación del software, García López (2007) afirma que:

Si bien esta distinción es, en cierto modo, arbitraria, y a veces confusa, a los fines prácticos se puede clasificar al software en tres grandes tipos:

Software de sistema: Su objetivo es desvincular adecuadamente al usuario y al programador de los detalles de la computadora en particular que se use, aislándolo especialmente del procesamiento referido a las características internas de: memoria, discos, puertos y dispositivos de comunicaciones, impresoras, pantallas, teclados, etc. El software de sistema le procura al usuario y programador adecuadas interfaces de alto nivel, herramientas y utilidades de apoyo que permiten su mantenimiento. Incluye entre otros:

- Sistemas operativos
- Controladores de dispositivos
- Herramientas de diagnóstico
- Herramientas de Corrección y Optimización

- Servidores
- Utilidades

Software de programación: Es el conjunto de herramientas que permiten al programador desarrollar programas informáticos, usando diferentes alternativas y lenguajes de programación, de una manera práctica. Incluye entre otros:

- Editores de texto
- Compiladores
- Intérpretes
- Enlazadores
- Depuradores

Entornos de Desarrollo Integrados (IDE): Agrupan las anteriores herramientas, usualmente en un entorno visual, de forma tal que el programador no necesite introducir múltiples comandos para compilar, interpretar, depurar, etc. Habitualmente cuentan con una avanzada interfaz gráfica de usuario (GUI).

Software de aplicación: Es aquel que permite a los usuarios llevar a cabo una o varias tareas específicas, en cualquier campo de actividad susceptible de ser automatizado o asistido, con especial énfasis en los negocios. Incluye entre otros:

- Aplicaciones para Control de sistemas y automatización industrial
- Aplicaciones ofimáticas
- Software educativo
- Software empresarial
- Bases de datos
- Telecomunicaciones (por ejemplo Internet y toda su estructura lógica)

- Videojuegos
- Software médico
- Software de Cálculo Numérico y simbólico.
- Software de Diseño Asistido (CAD)
- Software de Control Numérico (CAM)

Los componentes lógicos incluyen, entre muchos otros, las aplicaciones informáticas; tales como el procesador de texto, que permite al usuario realizar todas las tareas concernientes a la edición de textos; el software de sistema, tal como el sistema operativo, que, básicamente, permite al resto de los programas funcionar adecuadamente, facilitando también la interacción entre los componentes físicos y el resto de las aplicaciones, y proporcionando una interfaz con el usuario.. (pág. 90, 91,92).

1.4.- Tipos de Software.

En la industria informática existen diferentes categorías que enmarcan la naturaleza del software, depende del tipo de licencia que emplee para el mismo. Cada una de ellas está sujeta a diferentes características o restricciones de distribución y uso tal como se enuncia a continuación:

1.4.1.- Software libre.

Este tipo de software, permite la libre adquisición, modificación y distribución de los programas. Se distribuye bajo la licencia GPL (General Public License), la cual se mantiene, aun cuando el usuario haya realizado modificaciones al mismo. Esta distribución incluye tanto el programa como el respectivo código fuente.

De acuerdo con la página web **http://www.gnu.org/philosophy/free-sw.es.html** nos dice que **SOFTWARE LIBRE** es: **"**Una cuestión de la libertad de los usuarios de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software**".** (párr. 1).

Se refiere a cuatro tipos de libertades para los usuarios del software:

- La libertad de ejecutar el programa, para cualquier propósito (libertad 0).
- La libertad de estudiar cómo trabaja el programa, y adaptarlo a sus necesidades (libertad 1). El acceso al código fuente es una condición necesaria.
- La libertad de redistribuir copias para que pueda ayudar al prójimo (libertad 2).
- La libertad de mejorar el programa y publicar sus mejoras, y versiones modificadas en general, para que se beneficie toda la comunidad (libertad 3). El acceso al código fuente es una condición necesaria.

Un programa es software libre si los usuarios tienen todas esas libertades. Entonces, debería ser libre de redistribuir copias, con o sin modificaciones, ya sea gratis o a cambio de una tarifa por distribución, a cualquiera y en cualquier parte. El ser libre de hacer esto significa, entre otras cosas, que no tiene que pedir o pagar el permiso.

1.4.2.- Freeware.

Este tipo de software, se adquiere de manera gratuita pero está limitado tanto el tiempo de uso como la funcionalidad de la herramienta. No permite la modificación del código y contempla algunas restricciones para permitir su redistribución.

El término **freeware** define un tipo de software privativo que se distribuye sin costo, disponible para su uso y por tiempo ilimitado, siendo una variante gratuita del **shareware**, en el que la meta es lograr que un usuario pruebe el producto durante un tiempo ("trial") limitado, y si le satisface, pague por él, habilitando toda su funcionalidad. A veces se incluye el código fuente pero no es lo usual.

Freeware suele incluir una licencia de uso, que permite su redistribución pero con algunas restricciones, como no modificar la aplicación en sí, ni venderla, y dar cuenta de su autor. También puede desautorizar el uso en una compañía con fines comerciales o en una entidad gubernamental, o bien, requerir pagos si se le va a dar uso comercial. Todo esto depende del tipo de licencia en concreto a la que se acoge el software.

Se denomina **shareware** a una modalidad de distribución de software, tanto videojuegos como videos X, en la que el usuario puede evaluar de forma gratuita el producto, pero con limitaciones en el tiempo de uso o en algunas de las formas de uso o con restricciones en las capacidades finales.

Para adquirir una licencia de software que permita el uso del software de manera completa se requiere de un pago (muchas veces modesto) aunque también existe el llamado "shareware de precio cero", pero esta modalidad es poco común.

1.4.3.- Software Propietario.

Hace referencia al software sobre el que una persona o compañía conserva los derechos de autor, restringiendo el uso, modificación y adquisición de la herramienta, así como el acceso al código fuente.

1.4.4.- Plataformas de software libre.

Según la **Free Software Foundation**, "Es la denominación del software que respeta la libertad de los usuarios sobre su producto adquirido y por tanto, una vez obtenido puede ser usado, copiado, estudiado, cambiado y redistribuido libremente. El software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software; de modo más preciso, se refiere a cuatro libertades de los usuarios del software: la libertad de usar el programa, con cualquier propósito; de estudiar el funcionamiento del programa, y adaptarlo a las necesidades; de distribuir copias, con lo cual se puede ayudar a otros, y de mejorar el programa y hacer públicas las mejoras, de modo que toda la comunidad se beneficie (para la segunda y última libertad mencionadas, el acceso al código fuente es un requisito previo)".(párr. 1).

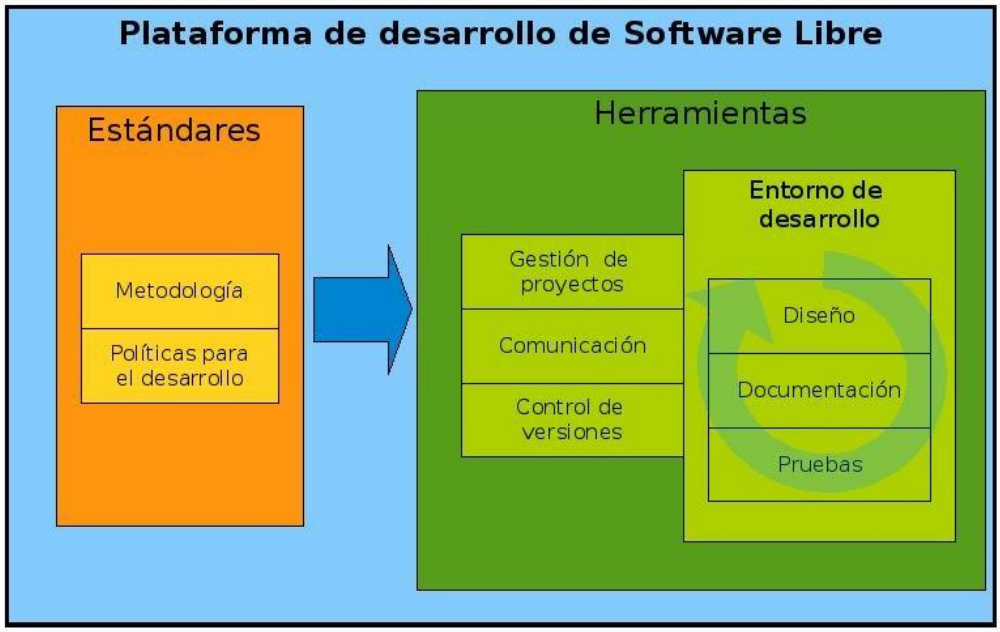


Figura 1. Plataformas de desarrollo de Software libre.

1.5.- Problemas por no Implementar Pruebas de Software.

Antes de que pueda ser usado el sistema de información debe ser probado. Durante este proceso se debe poner en práctica todas las estrategias posibles para garantizar que el usuario inicial del sistema se encuentre libre de problemas.

La implementación es la última fase del desarrollo de sistemas. Es el proceso de instalar equipos o software nuevo, como resultado de un análisis y diseño previo como resultado de la situación o mejoramiento de la forma de llevar a cabo un proceso automatizado. Al

implementar un sistema lo primero que debemos hacer es asegurarnos que el sistema sea operacional o que funcione de acuerdo a los requerimientos del análisis y permitir que los usuarios puedan operarlos.

Uno de los aspectos más descuidados en este gran proceso de desarrollo es pues el de las pruebas, a pesar de ser el medio por el cual se puede asegurar la calidad del producto de software, cumplir con los requisitos del usuario en la mayoría de los casos no es suficiente, para ello hace falta definir un proceso de pruebas serio, uniforme para todos los proyectos, es decir, estandarizado y que incluya **las mejores prácticas de pruebas** siempre bajo un modelo que asegure la mejora continua.

Una alternativa para alcanzar competitividad en la industria del software requiere desarrollar y aplicar un modelo basado en metodologías o procedimientos estándares para el planeamiento, especificación y ejecución de pruebas de software, por eso se ha visto la necesidad de implementar uno de los estándares como es la ISO/IEC 29119 EN LA EMPRESA OMNISOFT DE LA CIUDAD DE QUITO

.

En algunas empresas las actividades de pruebas de software pasan casi desapercibidas, pues no se crean casos de pruebas que permitan garantizar la calidad del software, la ausencia de una orientación clara en la planificación del proyecto y de políticas organizacionales que apoyen este proceso debido al desconocimiento o inaplicabilidad de algunos modelos de calidad aumentan el riesgo de producir software que inmediatamente reporta continuos errores o fallas graves. No se trata de sólo invertir más tiempo o contratar más personal, lo que se necesita es una metodología que defina actividades y responsabilidades, naturalmente esta tarea no se puede realizar de manera improvisada sino que es un proceso gradual.

1.6 Ingeniería de Software.

1.6.1.- Definición.

Según la página web: **http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software** menciona que:

> Ingeniería de software es el área de la ingeniería que ofrece métodos y técnicas para desarrollar y mantener software.
>
> Esta ingeniería trata con áreas muy diversas de la informática y de las ciencias de la computación, tales como construcción de compiladores, sistemas operativos, o desarrollos Intranet/Internet, abordando todas las fases del ciclo de vida del desarrollo de cualquier tipo de sistemas de información y aplicables a infinidad de áreas: negocios, investigación científica, medicina, producción, logística, banca, control de tráfico, meteorología, derecho, Internet, Intranet, etc.
> Una definición precisa aún no ha sido contemplada en los diccionarios, sin embargo se pueden citar las enunciadas por algunos de los más prestigiosos autores:
>
> Ingeniería de software es el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas software (Zelkovitz, 1978)
> Ingeniería de software es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como desarrollo de software o producción de software (Bohem, 1976).
>
> Ingeniería de software trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable, que sea fiable y trabaje en máquinas reales (Bauer, 1972).

> Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software; es decir, la aplicación de la ingeniería al software (IEEE, 1993) (párr. 1, 2, 3, 4).

1.6.2.- Proceso de creación del software.

Jacobson (2008) encontró que "Se define como Proceso al conjunto ordenado de pasos a seguir para llegar a la solución de un problema u obtención de un producto, en este caso particular, para lograr la obtención de un producto software que resuelva un problema" (p.56).

Pressman (2009) encontró lo siguiente:

> El proceso de creación de software puede llegar a ser muy complejo, dependiendo de su porte, características y criticidad del mismo. Por ejemplo, la creación de un sistema operativo es una tarea que requiere proyecto, gestión, numerosos recursos y todo un equipo disciplinado de trabajo. En el otro extremo, si se trata de un sencillo programa (por ejemplo, la resolución de una ecuación de segundo orden), éste puede ser realizado por un solo programador (incluso aficionado) fácilmente. Es así que normalmente se dividen en tres categorías según su tamaño (líneas de código) o costo: de Pequeño, Mediano y Gran porte. Existen varias metodologías para **estimarlo**, una de las más populares es el sistema COCOMO que provee métodos y un software (programa) que calcula y provee una estimación de todos los costos de producción en un «proyecto software» (relación horas/hombre, costo monetario, cantidad de líneas fuente de acuerdo a lenguaje usado, etc.).

> Considerando los de gran porte, es necesario realizar complejas tareas, tanto técnicas como de gerencia, una fuerte gestión y análisis diversos (entre otras cosas), por lo cual se ha desarrollado una ingeniería para su estudio y realización: es conocida como Ingeniería de Software.
>
> En tanto que en los de mediano porte, pequeños equipos de trabajo (incluso un avezado analista-programador solitario) pueden realizar la tarea. Aunque, siempre en casos de mediano y gran porte (y a veces también en algunos de pequeño porte, según su complejidad), se deben seguir ciertas etapas que son necesarias para la construcción del software. Tales etapas, si bien deben existir, son flexibles en su forma de aplicación, de acuerdo a la metodología o Proceso de Desarrollo escogido y utilizado por el equipo de desarrollo o por el analista-programador solitario (si fuere el caso). (pág. 125,127-129).

Haeberer (2008) manifestó lo siguiente:

> Los **«procesos de desarrollo de software»** poseen reglas preestablecidas, y deben ser aplicados en la creación del software de mediano y gran porte, ya que en caso contrario lo más seguro es que el proyecto o no logre concluir o termine sin cumplir los objetivos previstos, y con variedad de fallos inaceptables (fracasan, en pocas palabras). Entre tales «procesos» los hay ágiles o livianos (ejemplo XP), pesados y lentos (ejemplo RUP) y variantes intermedias; y normalmente se aplican de acuerdo al tipo y porte del software a desarrollar, a criterio del líder (si lo hay) del equipo de desarrollo. Algunos de esos procesos son Programación Extrema (en inglés eXtremeProgramming o XP), Proceso Unificado de Rational (en inglés RationalUnifiedProcess o RUP), FeatureDrivenDevelopment (FDD), etc. (p. 95)

Sommerville (2009) manifiesto que:

> Cualquiera sea el «proceso» utilizado y aplicado al desarrollo del software (RUP, FDD, etc), y casi independientemente de él, siempre se debe aplicar un «modelo de ciclo de vida». Se estima que, del total de proyectos software grandes emprendidos, un 28% fracasan, un 46% caen en severas modificaciones que lo retrasan y un 26% son totalmente exitosos. Cuando un proyecto fracasa, rara vez es debido a fallas técnicas, la principal causa de fallos y fracasos es la falta de aplicación de una buena metodología o proceso de desarrollo. Entre otras, una fuerte tendencia, desde hace pocas décadas, es mejorar las metodologías o procesos de desarrollo, o crear nuevas y concientizar a los profesionales en su utilización adecuada. Normalmente los especialistas en el estudio y desarrollo de estas áreas (metodologías) y afines (tales como modelos y hasta la gestión misma de los proyectos) son los Ingenieros en Software, es su orientación. Los especialistas en cualquier otra área de desarrollo informático (analista, programador, Lic. en Informática, Ingeniero en Informática, Ingeniero de Sistemas, etc.) normalmente aplican sus conocimientos especializados pero utilizando modelos, paradigmas y procesos ya elaborados. (pág. 124, 125).

Sommerville (2009) manifiesto que:

> El proceso de desarrollo puede involucrar numerosas y variadas tareas, desde lo administrativo, pasando por lo técnico y hasta la gestión y el gerenciamiento. Pero casi rigurosamente siempre se cumplen ciertas etapas mínimas; las que se pueden resumir como sigue:
>
> - ✓ Captura, E licitación , Especificación y Análisis de requisitos (ERS)
> - ✓ Diseño
> - ✓ Codificación
> - ✓ Pruebas (unitarias y de integración)
> - ✓ Instalación y paso a Producción
> - ✓ Mantenimiento

En las anteriores etapas pueden variar ligeramente sus nombres, o ser más globales, o contrariamente, ser más refinadas; por ejemplo indicar como una única fase (a los fines documentales e interpretativos) de «análisis y diseño»; o indicar como «implementación» lo que está dicho como «codificación»; pero en rigor, todas existen e incluyen, básicamente, las mismas tareas específicas. (pág. 127)

1.7. Metodologías.

1.7.1.- Definición.

Según la página web **http://www.misrespuestas.com/que-es-una-metodologia.html**nos dice: "Una metodología es aquella guía que se sigue a fin realizar las acciones propias de una investigación. En términos más sencillos se trata de la guía que nos va indicando qué hacer y cómo actuar cuando se quiere obtener algún tipo de investigación. Es posible definir una metodología como aquel enfoque que permite observar un problema de una forma total, sistemática, disciplinada y con cierta disciplina.

Al intentar comprender la definición que se hace de lo que es una metodología, resulta de suma importancia tener en cuenta que una metodología no es lo mismo que la técnica de investigación. Las técnicas son parte de una metodología, y se define como aquellos procedimientos que se utilizan para llevar a cabo la metodología, por lo tanto, como es posible intuir, es uno de los muchos elementos que incluye" (párr. 1, 2).

1.7.2.- Etapas del Proceso.

1.7.2.1.-Análisis de requerimientos.

Según esta página web

http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software menciona que:

> Extraer los requisitos y requerimientos de un producto de software es la primera etapa para crearlo. Mientras que los clientes piensan que ellos saben lo que el software tiene que hacer, se requiere de habilidad y experiencia en la ingeniería de software para reconocer requerimientos incompletos, ambiguos o contradictorios. El resultado del análisis de requerimientos con el cliente se plasma en el documento ERS, *Especificación de Requerimientos del Sistema*, cuya estructura puede venir definida por varios estándares, tales como CMMI. Asimismo, se define un diagrama de Entidad/Relación, en el que se plasman las principales entidades que participarán en el desarrollo del software.
>
> La captura, análisis y especificación de requerimientos (incluso pruebas de ellos), es una parte crucial; de esta etapa depende en gran medida el logro de los objetivos finales. Se han ideado modelos y diversos procesos de trabajo para estos fines. Aunque aún no está formalizada, ya se habla de la Ingeniería de requerimientos, por ejemplo en dos capítulos del libro de Sommerville "Ingeniería del software" titulados "Requerimientos del software" y "Procesos de la Ingeniería de Requerimientos".
>
> La IEEE Std. 830-1998 normaliza la creación de las especificaciones de requerimientos de software (Software Requirements Specification) (párr. 1, 2,3).

1.7.2.2.- Especificación.

Según la página web

http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software menciona que:

> La especificación de requisitos describe el comportamiento esperado en el software una vez desarrollado. Gran parte del éxito de un proyecto de software radicará en la identificación de las necesidades del negocio (definidas por la alta dirección), así como la interacción con los usuarios funcionales para la recolección, clasificación, identificación, priorización y especificación de los requisitos del software.
>
> Entre las técnicas utilizadas para la especificación de requisitos se encuentran:
>
> - Caso de uso,
> - Historias de usuario,
>
> Siendo los primeros más rigurosas y formales, los segundas más ágiles e informales (párr. 4).

1.7.2.3.-Arquitectura.

Según la página web

http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software menciona que:

> La integración de infraestructura, desarrollo de aplicaciones, bases de datos y herramientas gerenciales, requieren de capacidad y liderazgo para poder ser conceptualizados y proyectados a futuro, solucionando los problemas de hoy. El rol en el cual se delegan todas estas actividades es el del Arquitecto.
>
> El arquitecto de software es la persona que añade valor a los procesos de negocios gracias a su valioso aporte de soluciones tecnológicas.
>
> La arquitectura de sistemas en general, es una actividad de planeación, ya sea a nivel de infraestructura de red y hardware, o de software.
>
> La arquitectura de software consiste en el diseño de componentes de una aplicación (entidades del negocio), generalmente utilizando patrones de arquitectura. El diseño arquitectónico debe permitir visualizar la interacción entre las entidades del negocio y además poder ser validado, por ejemplo por medio de diagramas de secuencia. Un diseño arquitectónico describe en general el cómo se construirá una aplicación de software. Para ello se documenta utilizando diagramas, por ejemplo:

- Diagramas de clases
- Diagramas de base de datos
- Diagramas de despliegue plegados
- Diagramas de secuencia multidireccional

Siendo los dos primeros los mínimos necesarios para describir la arquitectura de un proyecto que iniciará a ser codificado. Depende del alcance del proyecto, complejidad y necesidades, el arquitecto elige que diagramas elaborar. Entre las herramientas para diseñar arquitecturas de software se encuentran:

- Enterprise Architect
- Microsoft Visio for Enterprise Architects (párr. 5, 6, 7, 8).

1.7.2.4.- Programación.

Según la página web

http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software menciona que: "Reducir un diseño a código puede ser la parte más obvia del trabajo de ingeniería de software, pero no necesariamente es la que demanda mayor trabajo y ni la más complicada. La complejidad y la duración de esta etapa está íntimamente relacionada al o a los lenguajes de programación utilizados, así como al diseño previamente realizado" (párr. 9).

1.7.2.5.- Prueba.

Según la página web

http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software menciona que:

Consiste en comprobar que el software realice correctamente las tareas indicadas en la especificación del problema. Una técnica de prueba es probar por separado cada módulo del software, y luego probarlo de forma integral, para así llegar al objetivo. Se considera una buena práctica el que las pruebas sean efectuadas por alguien distinto al desarrollador que la programó, idealmente un área de pruebas; sin perjuicio de lo anterior el programador debe hacer sus propias pruebas. En general hay dos grandes formas de organizar un área de pruebas, la primera es que esté compuesta por personal inexperto y que desconozca el tema de pruebas, de esta forma se evalúa que la documentación entregada sea de calidad, que los procesos descritos son tan claros que cualquiera puede entenderlos y el software hace las cosas tal y como están descritas. El segundo enfoque es tener un área de pruebas conformada por programadores con experiencia, personas que saben sin mayores indicaciones en qué condiciones puede fallar una aplicación y que pueden poner atención en detalles que personal inexperto no consideraría (párr. 10).

1.7.2.6.- Documentación.

Según la página web

http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software menciona que: "Todo lo concerniente a la documentación del propio desarrollo del software y de la gestión del proyecto, pasando por modelaciones (UML), diagramas de casos de uso, pruebas, manuales de usuario, manuales técnicos, etc; todo con el propósito de eventuales correcciones, usabilidad, mantenimiento futuro y ampliaciones al sistema" (párr. 11).

1.7.2.7.- Mantenimiento.

Según la página web

http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software menciona que:

> Mantener y mejorar el software para enfrentar errores descubiertos y nuevos requisitos. Esto puede llevar más tiempo incluso que el desarrollo inicial del software. Alrededor de 2/3 del ciclo de vida de un proyecto. Una pequeña parte de este trabajo consiste en arreglar errores, o *bugs*. La mayor parte consiste en extender el sistema para hacer nuevas funcionalidades y hacer frente a su evolución (párr. 12).

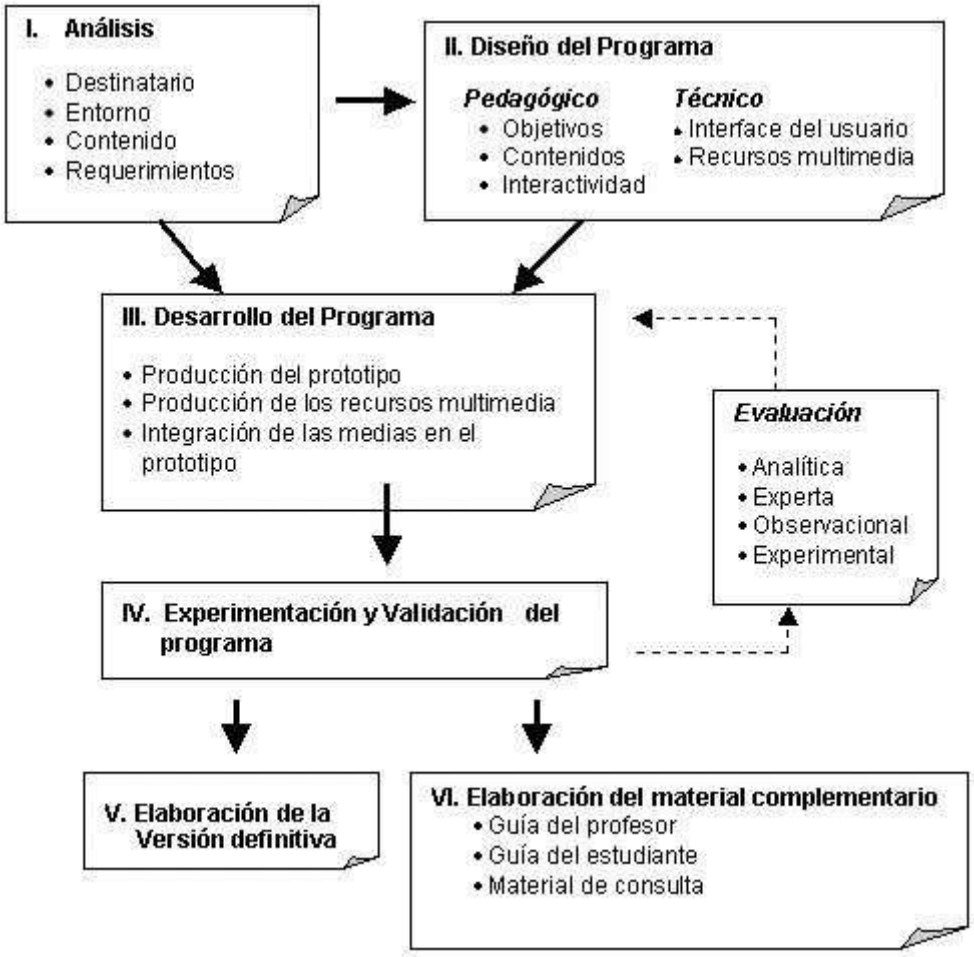


Figura 2.Etapas en el desarrollo del software.

1.8. Tipos de Metodologías de Desarrollo de Software

Un proceso de software detallado y completo suele denominarse "Metodología". Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, evolutivo, incremental, etc.). Adicionalmente una metodología debería definir con precisión los artefactos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología al proyecto, guías para uso de herramientas de apoyo, etc. Habitualmente se utiliza el término "método" para referirse a técnicas, notaciones y guías

asociadas, que son aplicables a una (o algunas) actividades del proceso de desarrollo, por ejemplo, suele hablarse de métodos de análisis y/o diseño.

La comparación y/o clasificación de metodologías no es una tarea sencilla debido a la diversidad de propuestas y diferencias en el grado de detalle, información disponible y alcance de cada una de ellas. A grandes rasgos, si tomamos como criterio las notaciones utilizadas para especificar artefactos producidos en actividades de análisis y diseño, podemos clasificar las metodologías en dos grupos: Metodologías Estructuradas y Metodologías Orientadas a Objetos. Por otra parte, considerando su filosofía de desarrollo, aquellas metodologías con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de Metodologías Tradicionales (o peyorativamente denominada Metodologías Pesadas, o Peso Pesado). Otras metodologías, denominadas Metodologías Ágiles, están más orientadas a la generación de código con ciclos muy cortos de desarrollo, se dirigen a equipos de desarrollo pequeños, hacen especial hincapié en aspectos humanos asociados al trabajo en equipo e involucran activamente al cliente en el proceso. A continuación se revisan brevemente cada una de estas categorías de metodologías.

1.8.1.-Metodologías estructuradas.

Según la página web **http://metodologiasestructuradas.blogspot.com/2009/02/metodologias-estructuradas.html** menciona que:

> Tiene como objetivo emplear las metodologías de análisis y diseño estructurado para su uso con herramientas CASE, incrementando la productividad en el desarrollo e implantación de sistemas de información y entre ellas podemos encontrar a Kendall & Kendall entre otras.

Crea los modelos de forma descendente. Son las orientadas a procesos, a datos y las mixtas. Intentan aplicar formas ingenieriles para solucionar problemas técnicos al obtener un sistema de información, proponen la creación de modelos, flujos y estructuras mediante un top-down.

Es la primera aproximación al problema. Está orientada a procesos, es decir, se centra en especificar y descomponer la funcionalidad del sistema. Se utilizan varias herramientas:

Diagramas de flujo de datos (DFD): Representan la forma en la que los datos se mueven y se transforman. Incluye:
Procesos
Flujos de datos
Almacenes de datos (párr. 1, 2, 3).

1.8.2.-Metodologías orientadas a objetos.

Según la página web **http://html.rincondelvago.com/metodologia-de-analisis-y-diseno-orientado-a-objetos.html** menciona que:

Su historia va unida a la evolución de los lenguajes de programación orientada a objeto, los más representativos: a fines de los 60's SIMULA, a fines de los 70's Smalltalk-80, la primera versión de C++ por BjarneStroustrup en 1981 y actualmente Java o C# de Microsoft. A fines de los 80's comenzaron a consolidarse algunos métodos Orientadas a Objeto.

En 1995 Booch y Rumbaugh proponen el Método Unificado con la ambiciosa idea de conseguir una unificación de sus métodos y notaciones, que posteriormente se reorienta a un objetivo más modesto, para dar lugar al Unified Modeling Language (UML), la notación OO más popular en la actualidad.

Algunos métodos OO con notaciones predecesoras de UML son: OOAD (Booch), OOSE (Jacobson), Coad&Yourdon, Shaler&Mellor y OMT (Rumbaugh).Algunas metodologías orientadas a objetos que utilizan la notación UML son: Rational Unified Process (RUP), OPEN, MÉTRICA (que también soporta la notación estructurada) (párr. 1, 2, 3).

La metodología de desarrollo de software orientada a objetos es cada día más usada, pues permite desarrollar software fácilmente extensible y reusable. Esto último es sólo posible si los desarrolladores conocen muy bien los fundamentos en que está basada esta metodología. Por eso, este curso revisa los conceptos más importantes que se encuentran en las distintas etapas del desarrollo de software orientado a objetos.

1.8.3.-Metodologías tradicionales (no ágiles)

Según la página web **http://www.eumed.net/libros/2009c/584/Metodologias%20tradicionales%20y%20metodologias%20agiles.htm** se dice que:

> Las metodologías no ágiles son aquellas que están guiadas por una fuerte planificación durante todo el proceso de desarrollo; llamadas también metodologías tradicionales o clásicas, donde se realiza una intensa etapa de análisis y diseño antes de la construcción del sistema.

> Todas las propuestas metodológicas antes indicadas pueden considerarse como metodologías tradicionales. Aunque en el caso particular de RUP, por el especial énfasis que presenta en cuanto a su adaptación a las condiciones del proyecto (mediante su configuración previa a aplicarse), realizando una configuración adecuada, podría considerarse ágil (párr. 1, 2).

Estas metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis 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. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada.

1.8.4.-Metodologías ágiles

La página web **www.willydev.net/descargas/prev/TodoAgil.Pdf** menciona que

> Un proceso es ágil cuando el desarrollo de software es incremental (entregas pequeñas de software, con ciclos rápidos), cooperativo (cliente y desarrolladores trabajan juntos constantemente con una cercana comunicación), sencillo (el método en sí mismo es

fácil de aprender y modificar, bien documentado), y adaptable (permite realizar cambios de último momento).

Entre las metodologías ágiles identificadas en:

- Extreme Programming.
- Scrum.
- Familia de Metodologías Crystal.
- Feature Driven Development.
- Proceso Unificado Rational, una configuración ágil.
- Dynamic Systems Development Method.
- Adaptive Software Development.
- Open Source Software Development (párr. 1, 2)

Las metodologías ágiles proporcionan una serie de pautas y principios junto a técnicas pragmáticas que puede que no curen todos los males pero harán la entrega del proyecto menos complicada y más satisfactoria tanto para los clientes como para los equipos de entrega.

1.9.- Ciclo de vida del Software.

Según la página web www.csicsif.es/andalucia/modules/mod_ense/.../Carlos_Caballero.pdf *menciona que:*

"El ciclo de vida cuando se elabora software o proyecto informático es el conjunto de etapas y estados por los que pasa el desarrollo desde que se plantea como necesidad o problema, por parte de un cliente, hasta que se da por terminado y se considera como una solución completa, correcta y estable(que resuelve el problema inicial)" (párr. 7).

Las etapas o fases principales del ciclo de vida de una aplicación son las siguientes:

1.9.1.-Análisis.

Es la etapa inicial, en esta etapa se persigue extraer el conocimiento del problema que se desea resolver. Extrayendo sus características, detalles, limitaciones,… En la etapa de análisis se descompone el problema en partes para obtener un conjunto de sub problemas que sean comprendidos y de fácil resolución.

En resumen se establece qué se quiere desarrollar en función de las necesidades del usuario y las distintas restricciones del desarrollo como pueden ser factores tecnológicos, financieros o de recursos humanos.

El análisis del sistema se realiza teniendo presente los objetivos de:

1.9.1.1.- Identificar las necesidades del cliente.

En esta etapa se hace necesario una comunicación fluida entre el usuario o cliente y el analista para poder identificar las necesidades del mismo. Esta tarea se lleva a cabo a través de entrevistas y cuestionarios.

1.9.1.2.- Evaluar la viabilidad del proyecto.

En esta etapa se determina la posibilidad de realizar con garantías del proyecto, encontramos cuatro áreas de interés:

- ✓ **Viabilidad económica.** Estimar los costes de desarrollo y los beneficios finales.
- ✓ **Viabilidad técnica.** Posibilidad de realizar el proyecto con las herramientas que se disponen.
- ✓ **Viabilidad legal.** No incurrir en violaciones legales en el desarrollo del proyecto.

1.9.2.- Diseño.

El objetivo de esta etapa es decidir cómo resolver cada uno de los sub problemas identificados en la etapa de análisis y cómo integrar todas las soluciones diseñadas en una solución global.

1.9.3.-Implementación.

En esta fase se realiza la codificación, se realiza la programación de la solución diseñada, en el lenguaje de programación y plataforma elegida a tal efecto teniendo en cuenta las restricciones obtenidas en la etapa de análisis.

1.9.4.-Pruebas.

El objetivo de esta fase es garantizar el correcto funcionamiento de las aplicaciones desarrolladas en la etapa de implementación, así como su adecuación a los requisitos y necesidades expresadas por el cliente o usuario en la etapa de análisis.

Estos objetivos se plasman en dos aspectos que son complementarios entre sí

- ✓ **La verificación.** Consiste en comprobar el correcto funcionamiento del código programado.

- ✓ **La validación.** Consiste en asegurar que la aplicación que se ha obtenido es el producto correcto.

Se realizan dos tipos de pruebas distintas.

- **Pruebas unitarias.** Para comprobar que cada módulo hace lo que tiene que hacer sujeto a las restricciones.

- **Pruebas de integración.** Para comprobar que los módulos funcionan de manera correcta entre sí.

1.9.5.-Implantación.

En esta fase se implanta el proyecto en el entorno que se va a emplear esta etapa consiste en:

- ➢ Instalación
- ➢ Transición desde el sistema anterior
- ➢ Eliminación del sistema anterior

- Formación de los usuarios del sistema

1.9.6.- Mantenimiento.

Aunque terminada la fase de implantación se cierra el proyecto, es necesaria la fase de mantenimiento del software durante un tiempo. No obstante tradicionalmente se llevan a cabo las siguientes etapas:

- **Correctivo.** Corrige los errores no detectados en la fase de construcción.
- **Adaptativo.** Adapta el programa a nuevas características y/o cambios en las normativas.
- **Perfectivo.** Añade nuevas características al software.
- **Preventivo.** Se realizan cambios en el software para facilitar el mantenimiento de futuras funcionalidades.

Como se puede observar existen una gran cantidad de etapas en el ciclo de vida y estas consumen un gran tiempo en el desarrollo de software de calidad. No obstante, paralelamente se realizan actividades que complementan a cada etapa y ayudan a garantizar la integridad y coherencia en el proceso de este modo se consigue ganar calidad en el desarrollo de software.

Las tareas paralelas que se desarrollan se pueden clasificar en las siguientes:

1.9.6.1.-Gestión de cambios.

Cuando en el desarrollo se produce un cambio, éste se propaga a lo largo de las distintas etapas del ciclo de vida. Para controlar estos cambios y las repercusiones que estos producen en los módulos y en el sistema se debe realizar un seguimiento y control de cambios.

1.9.6.2.-Gestión de configuración.

Al comienzo del ciclo de vida se define una configuración inicial o básica de los recursos necesarios (Hardware y software). Esta configuración va evolucionando a lo largo del desarrollo, por lo que se deben gestionar los cambios que se produzcan en ellos.

1.9.6.3.-Gestión de la documentación.

Agrupa a las actividades dedicadas a planificar, diseñar, editar, producir, distribuir y mantener los documentos necesarios para los desarrolladores y los usuarios.

1.10.- Modelos del ciclo de vida.

Según la página web http://www.tiposdesoftware.com/historia-del-software.htm manifiesta que:

"Para cada una de las fases o etapas listadas en el ítem anterior, existen sub-etapas (o tareas). El modelo de proceso o modelo de ciclo de vida utilizado para el desarrollo define el orden para las tareas o actividades involucradas también definen la coordinación entre ellas, enlace y realimentación entre las mencionadas etapas. Entre los más conocidos se puede mencionar: modelo en cascada o secuencial, modelo espiral, modelo iterativo incremental. De los

antedichos hay a su vez algunas variantes o alternativas, más o menos atractivas según sea la aplicación requerida y sus requisitos.”(párr. 15).

1.10.1.- Modelo cascada

Según la página web http://www.tiposdesoftware.com/historia-del-software.htm manifiesta que:

> Aunque es más comúnmente conocido como modelo en cascada es también llamado «modelo clásico», «modelo tradicional» o «modelo lineal secuencial».

El modelo en cascada puro difícilmente se utiliza tal cual, pues esto implicaría un previo y absoluto conocimiento de los requisitos, la no volatilidad de los mismos (o rigidez) y etapas subsiguientes libres de errores; ello sólo podría ser aplicable a escasos y pequeños desarrollos de sistemas. En estas circunstancias, el paso de una etapa a otra de las mencionadas sería sin retorno, por ejemplo pasar del Diseño a la Codificación implicaría un diseño exacto y sin errores ni probable modificación o evolución: «codifique lo diseñado que no habrán en absoluto variantes ni errores». Esto es utópico; ya que intrínsecamente el software es de carácter evolutivo, cambiante y difícilmente libre de errores, tanto durante su desarrollo como durante su vida operativa. (párr. 16).

Según la página web http://www.tiposdesoftware.com/historia-del-software.htm menciona que:

El modelo lineal o en cascada es el paradigma más antiguo y extensamente utilizado, sin embargo las críticas a él (ver desventajas) han puesto en duda su eficacia. Pese a todo,

tiene un lugar muy importante en la Ingeniería de software y continúa siendo el más utilizado; y siempre es mejor que un enfoque al azar.

Desventajas del modelo cascada:

- ✓ Los cambios introducidos durante el desarrollo pueden confundir al equipo profesional en las etapas tempranas del proyecto. Si los cambios se producen en etapa madura (codificación o prueba) pueden ser catastróficos para un proyecto grande.
- ✓ No es frecuente que el cliente o usuario final explicite clara y completamente los requisitos (etapa de inicio); y el modelo lineal lo requiere. La incertidumbre natural en los comienzos es luego difícil de acomodar.
- ✓ El cliente debe tener paciencia ya que el software no estará disponible hasta muy avanzado el proyecto. Un error detectado por el cliente (en fase de operación) puede ser desastroso, implicando reinicio del proyecto, con altos costos. (párr. 17, 18,19)

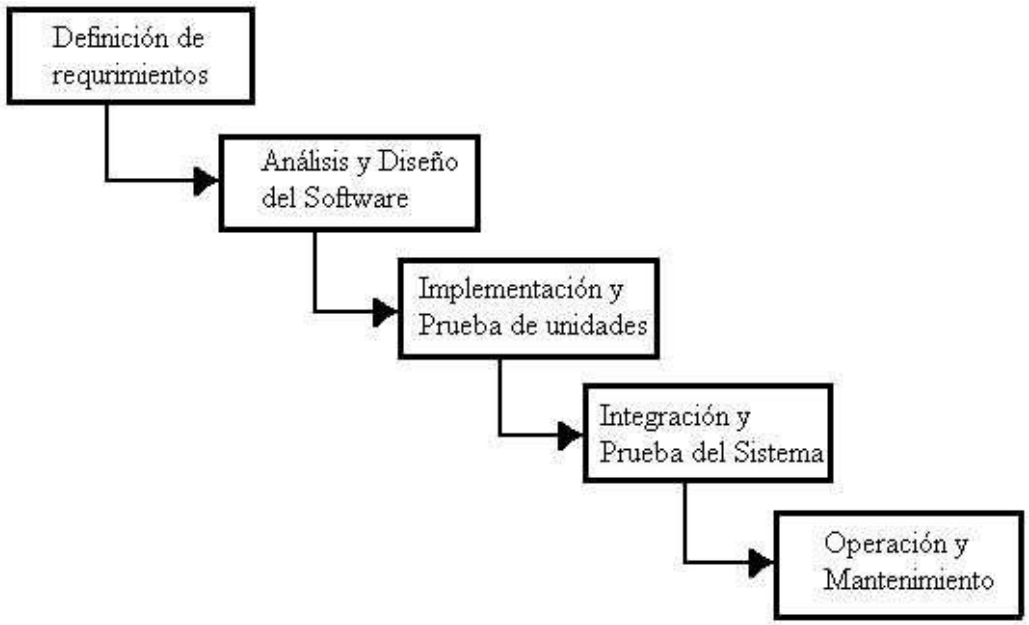


Figura 3.Modelo cascada puro o secuencial para el ciclo de vida del software.

1.10.2.- Modelos evolutivos.

Kendall Sccott(2009) menciona que:

> El software evoluciona con el tiempo. Los requisitos del usuario y del producto suelen cambiar conforme se desarrolla el mismo. Las fechas de mercado y la competencia hacen que no sea posible esperar a poner en el mercado un producto absolutamente completo, por lo que se debe introducir una versión funcional limitada de alguna forma para aliviar las presiones competitivas.
>
> En esas u otras situaciones similares los desarrolladores necesitan modelos de progreso que estén diseñados para acomodarse a una evolución temporal o progresiva, donde los requisitos centrales son conocidos de antemano, aunque no estén bien definidos a nivel detalle.
>
> En el modelo cascada y cascada realimentado no se tiene en cuenta la naturaleza evolutiva del software, se plantea como estático con requisitos bien conocidos y definidos desde el inicio.
>
> Los evolutivos son modelos iterativos, permiten desarrollar versiones cada vez más completas y complejas, hasta llegar al objetivo final deseado; incluso evolucionar más allá, durante la fase de operación.
>
> Los modelos iterativos incrementales y espirales (entre otros) son dos de los más conocidos y utilizados del tipo evolutivo. (pág. 255).

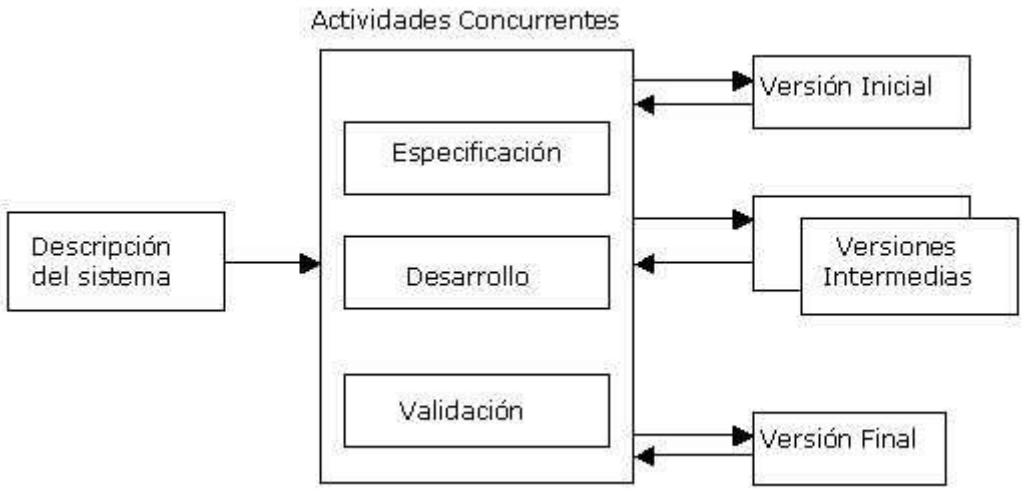


Figura 4. Diagrama genérico del desarrollo evolutivo incremental.

1.10.3.- Modelo iterativo incremental.

Kendall Sccott(2009) menciona que: "En términos generales, podemos distinguir, en la figura 4, los pasos generales que sigue el proceso de desarrollo de un producto software. En el modelo de ciclo de vida seleccionado, se identifican claramente dichos pasos. La Descripción del Sistema es esencial para especificar y confeccionar los distintos incrementos hasta llegar al Producto global y final. Las actividades concurrentes (Especificación, Desarrollo y Validación) sintetizan el desarrollo pormenorizado de los incrementos, que se hará posteriormente"(pág. 255).

Kendall Sccott (2009) manifiesta que:

> El modelo es aconsejable para el desarrollo de software en el cual se observe, en su etapa inicial de análisis, que posee áreas bastante bien definidas a cubrir, con suficiente independencia como para ser desarrolladas en etapas sucesivas. Tales áreas a cubrir suelen tener distintos grados de apremio por lo cual las mismas se deben priorizar en un análisis previo, es decir, definir cuál será la primera, la segunda, y así sucesivamente; esto se conoce como definición de los incrementos con base en priorización. Pueden no existir prioridades funcionales por parte del cliente, pero el desarrollador debe fijarlas de todos modos y con algún criterio, ya que basándose en ellas se desarrollarán y entregarán los distintos incrementos (pág. 256).

En resumen, un modelo incremental lleva a pensar en un desarrollo modular, con entregas parciales del producto software denominados **incrementos del sistema**, que son escogidos según prioridades predefinidas de algún modo. El modelo permite una implementación con refinamientos sucesivos (ampliación o mejora). Con cada incremento se agrega nueva funcionalidad o se cubren nuevos requisitos o bien se mejora la versión previamente implementada del producto software.

Este modelo brinda cierta flexibilidad para que durante el desarrollo se incluyan cambios en los requisitos por parte del usuario, un cambio de requisitos propuesto y aprobado puede analizarse e implementarse como un nuevo incremento o, eventualmente, podrá constituir una mejora/adecuación de uno ya planeado. Aunque si se produce un cambio de requisitos por parte del cliente que afecte incrementos previos ya terminados (detección/incorporación tardía) se debe evaluar la factibilidad y realizar un acuerdo con el cliente, ya que puede impactar fuertemente en los costos.

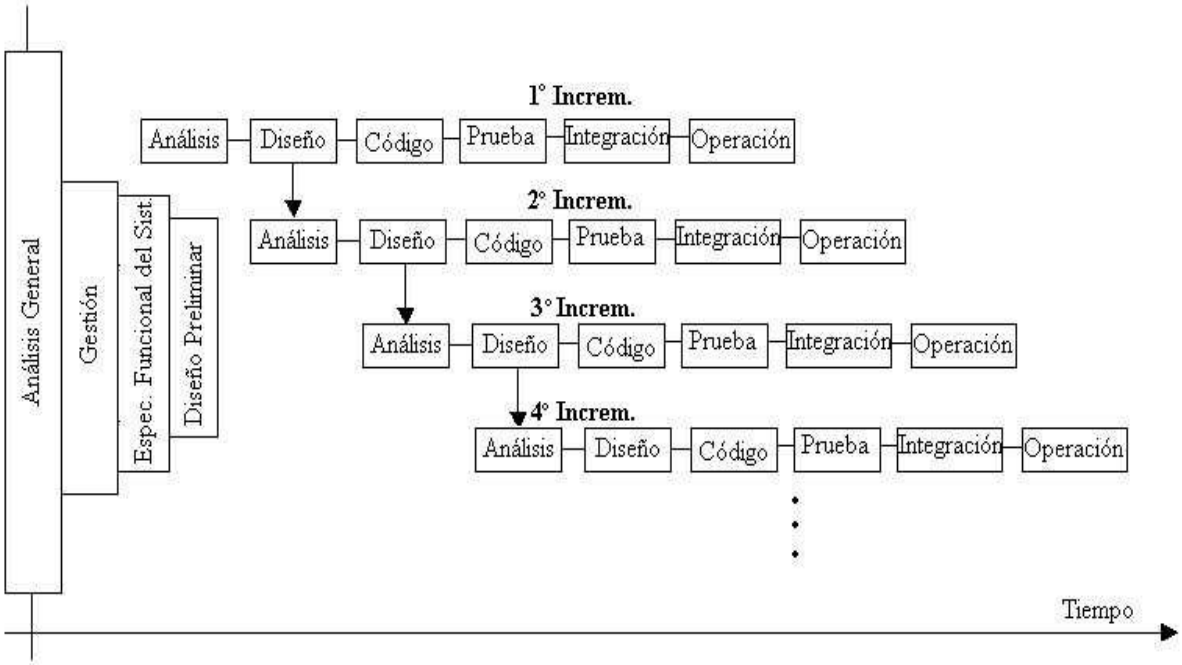


Figura 5. Modelo iterativo incremental para el ciclo de vida del software.

1.10.4.-Modelo espiral.

Weitzenfeld (2007) menciona que: "El modelo espiral fue propuesto inicialmente por Barry Boehm. Es un modelo evolutivo que conjuga la naturaleza iterativa del modelo MCP con los aspectos controlados y sistemáticos del Modelo Cascada. Proporciona potencial para desarrollo rápido de versiones incrementales. En el modelo espiral el software se construye en una serie de versiones incrementales. En las primeras iteraciones la versión incremental podría ser un modelo en papel o bien un prototipo. En las últimas iteraciones se producen versiones cada vez más completas del sistema diseñado" (pág. 109).

Weitzenfeld (2007) menciona que: "El modelo se divide en un número de Actividades de marco de trabajo, llamadas «**regiones de tareas**». En general existen entre tres y seis regiones

de tareas (hay variantes del modelo). En la figura 4 se muestra el esquema de un Modelo Espiral con seis regiones. En este caso se explica una variante del modelo original de Boehm, expuesto en su tratado de 1988; en 1998 expuso un tratado más reciente" (pág. 109).

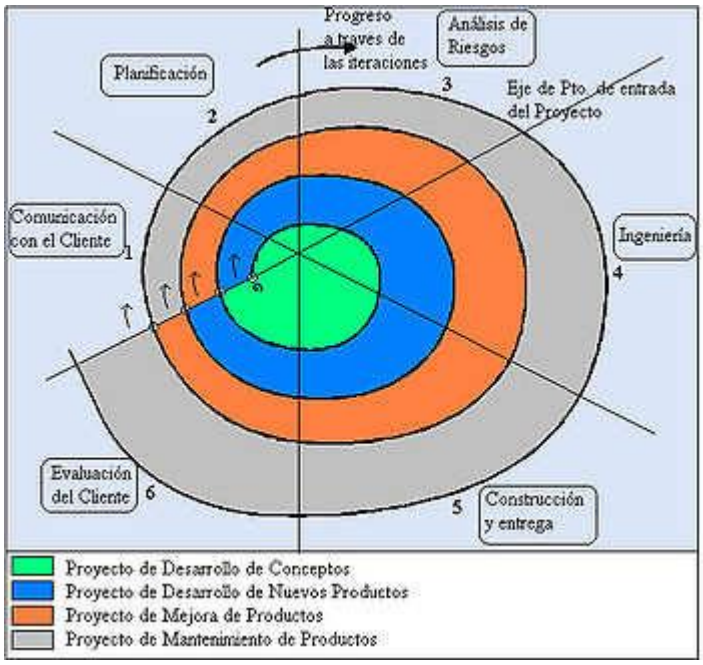


Figura 6. Modelo espiral para el ciclo de vida del software.

Las regiones definidas en el modelo de la figura son:

- **Región 1 -** Tareas requeridas para establecer la comunicación entre el cliente y el desarrollador.
- **Región 2 -** Tareas inherentes a la definición de los recursos, tiempo y otra información relacionada con el proyecto.
- **Región 3 -** Tareas necesarias para evaluar los riesgos técnicos y de gestión del proyecto.
- **Región 4 -** Tareas para construir una o más representaciones de la aplicación software.
- **Región 5 -** Tareas para construir la aplicación, instalarla, probarla y proporcionar soporte al usuario o cliente (Ej. documentación y práctica).

- **Región 6 -** Tareas para obtener la reacción del cliente, según la evaluación de lo creado e instalado en los ciclos anteriores.

El modelo espiral da un enfoque realista, que evoluciona igual que el software; se adapta muy bien para desarrollos a gran escala. El espiral utiliza el MCP para reducir riesgos y permite aplicarlo en cualquier etapa de la evolución. Mantiene el enfoque clásico (cascada) pero incorpora un marco de trabajo iterativo que refleja mejor la realidad.

Este modelo requiere considerar riesgos técnicos en todas las etapas del proyecto; aplicado adecuadamente debe reducirlos antes de que sean un verdadero problema. El Modelo evolutivo como el Espiral es particularmente apto para el desarrollo de Sistemas Operativos (complejos); también en sistemas de altos riesgos o críticos (Ej. navegadores y controladores aeronáuticos) y en todos aquellos en que sea necesaria una fuerte gestión del proyecto y sus riesgos, técnicos o de gestión.

Desventajas importantes:

- Requiere mucha experiencia y habilidad para la evaluación de los riesgos, lo cual es requisito para el éxito del proyecto.
- Es difícil convencer a los grandes clientes que se podrá controlar este enfoque evolutivo.

Este modelo no se ha usado tanto, como el Cascada (Incremental) o MCP, por lo que no se tiene bien medida su eficacia, es un paradigma relativamente nuevo y difícil de implementar y controlar.

1.10.5.-Modelo espiral Win&Win.

Según la página web **http://www.hanantek.com/win-win**:

> Esta es una adaptación del modelo de espiral que se hace hincapié explícitamente situados en la participación del cliente en un proceso de negociación en la génesis del desarrollo de productos. Idealmente, el desarrollador simplemente preguntar al cliente lo que se requiere y el cliente proporcionaría el suficiente detalle para proceder. Por desgracia esto rara vez sucede y negociaciones significativas entre ambas partes son necesarias para equilibrar la funcionalidad, rendimiento, etc. con los costos y de salida al mercado razones de tiempo.

> El modelo “win-win”. Deriva su nombre del objetivo de estas negociaciones, es decir, de "ganar-ganar". El cliente recibe el producto que satisface la mayoría de sus necesidades, y el desarrollador trabaja para alcanzar presupuestos y fechas de entrega. Para lograr este objetivo, el modelo define un conjunto de actividades de negociación al principio de cada paso alrededor de la espiral. En vez de una sola actividad de los clientes de comunicación las actividades se definen los siguientes:
>
> • Identificación de los actores del sistema.
>
> • Determinación de las partes interesadas de "ganar" condiciones
>
> • Las negociaciones de la victoria de las condiciones de las partes interesadas a que reconciliarlos en un conjunto de condiciones de ganar-ganar para todos los interesados (párr. 1,2).

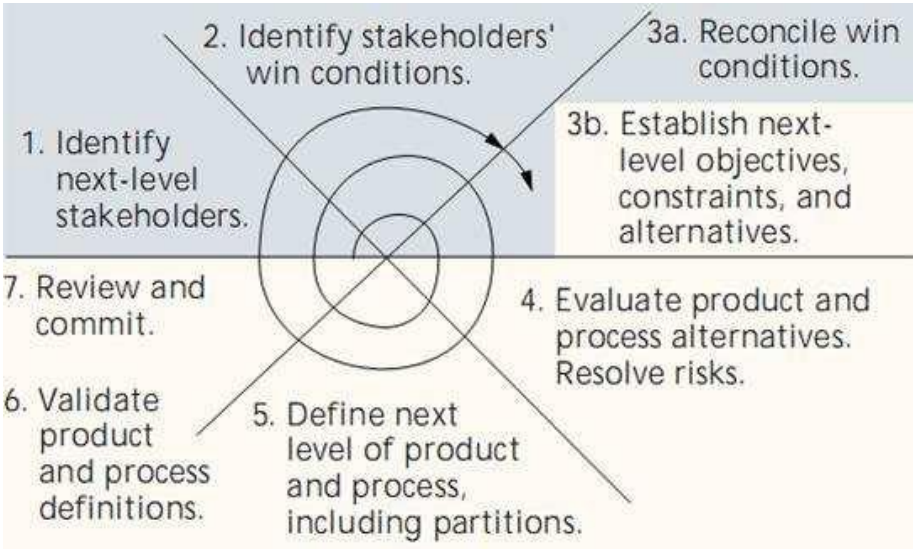


Figura 7.El modelo "win-win".

Además del énfasis inicial puesto en la condición de ganar-ganar, el modelo también presenta tres etapas del proceso (puntos de anclaje), que ayudan a establecer la realización de un ciclo alrededor de la espiral y proporcionar los hitos de decisión antes de que el proyecto de producto de software. Estos son:

Objetivos del Ciclo de Vida (Life Cycle Objectives - LCO) - Define una serie de objetivos para cada actividad de software más importantes (un conjunto de objetivos relacionados con la definición de los principales requisitos de nivel de producto).

Arquitectura del Ciclo de Vida (Life Cycle Architecture - LCA) - Establece los objetivos que deben cumplirse como la arquitectura de software se define.

Initial Operational Capability (Initial Operational Capability - IOC) representa un conjunto de objetivos asociados a la preparación del software para la instalación y distribución, la preparación previa a las instalaciones del sitio, asistencia requerida por todas las partes que utilizará o soporte técnico del software.

Con esta breve descripción se puede declarar que el software de producción más rápido puede facilitarse a través de la participación colaborativa de las partes interesadas pertinentes además de ser barato a través de reproceso y mantenimiento reducciones (párr. 3-7).

1.11. Ingeniería de Pruebas.

La página web **http://es.wikipedia.org/wiki/Pruebas_de_software** encontramos que:

Las **pruebas de software**, en inglés *testing* son los procesos que permiten verificar y revelar la calidad de un producto software. Son utilizadas para identificar posibles fallos de implementación, calidad, o usabilidad de un programa de ordenador o videojuego. Básicamente es una fase en el desarrollo de software consistente en probar las aplicaciones construidas.

Las pruebas de software se integran dentro de las diferentes fases del ciclo del software dentro de la Ingeniería de software. Así se ejecuta un programa y mediante técnicas experimentales se trata de descubrir que errores tiene.

Para determinar el nivel de calidad se deben efectuar unas medidas o pruebas que permitan comprobar el grado de cumplimiento respecto de las especificaciones iniciales del sistema (párr. 1, 2, 3).

1.11.1.- Pruebas unitarias.

Según la página web **http://lsi.ugr.es/~ig1/docis/pruso.pdf**:

> En programación, una **prueba unitaria** es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las pruebas de integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión.
>
> La idea es escribir casos de prueba para cada función no trivial o método en el módulo de forma que cada caso sea independiente del resto (párr. 1).

En la dirección **web http://es.wikipedia.org/wiki/Prueba_unitaria** se menciona:

> **Características**
>
> Para que una prueba unitaria sea buena se deben cumplir los siguientes requisitos:
>
> - **Automatizable**: no debería requerirse una intervención manual. Esto es especialmente útil para integración continua.
> - **Completas**: deben cubrir la mayor cantidad de código.
> - **Repetibles o Reutilizables**: no se deben crear pruebas que sólo puedan ser ejecutadas una sola vez. También es útil para integración continua.
> - **Independientes**: la ejecución de una prueba no debe afectar a la ejecución de otra.
> - **Profesionales**: las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc. (párr. 2).

Aunque estos requisitos no tienen que ser cumplidos al pie de la letra, se recomienda seguirlos o de lo contrario las pruebas pierden parte de su función.

La página **web http://es.wikipedia.org/wiki/Prueba_unitaria** se detalla lo siguiente:

Ventajas

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

1. **Fomentan el cambio**: Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (lo que se ha dado en llamar refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.

2. **Simplifica la integración**: Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente. De esta manera se facilitan las pruebas de integración.

3. **Documenta el código**: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.

4. **Separación de la interfaz y la implementación**: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro, a veces usando objetos mock (mockobject) para simular el comportamiento de objetos complejos.

5. **Los errores están más acotados y son más fáciles de localizar**: dado que tenemos pruebas unitarias que pueden desenmascararlos (párr. 1, 2, 3, 4 ,5 ,6).

Limitaciones

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

1.11.2.- Pruebas funcionales.

Según esta página **http://www.calidadysoftware.com/testing/pruebas_funcionales.php**:

> Se denominan pruebas funcionales o Functional Testing, a las pruebas de software que tienen por objetivo probar que los sistemas desarrollados, cumplan con las funciones específicas para los cuales han sido creados, es común que este tipo de pruebas sean desarrolladas por analistas de pruebas con apoyo de algunos usuarios finales, esta etapa suele ser la última etapa de pruebas y al dar conformidad sobre esta el paso siguiente es el pase a producción.
>
> A este tipo de pruebas se les denomina también pruebas de comportamiento o pruebas de caja negra, ya que los testers o analistas de pruebas, no enfocan su atención a como se generan las respuestas del sistema, básicamente el enfoque de este tipo de prueba se basa en el análisis de los datos de entrada y en los de salida, esto generalmente se define en los casos de prueba preparados antes del inicio de las pruebas (párr. 1, 2).

Una **prueba funcional** es una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. La pruebas funcionales se hacen

mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el paquete informático.

1.11.3.- Pruebas de Integración.

En la dirección web **http://es.wikipedia.org/wiki/Pruebas_de_Integraci%C3%B3n** se manifiesta que:

> **Pruebas integrales** o **pruebas de integración** son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias. Únicamente se refieren a la prueba o pruebas de todos los elementos unitarios que componen un proceso, hecha en conjunto, de una sola vez.

> Consiste en realizar pruebas para verificar que un gran conjunto de partes de software funcionan juntos.

> Las pruebas de integración (algunas veces llamadas integración y testeo I&t) es la fase de prueba de software en la cual módulos individuales de software son combinados y probados como un grupo. Son las pruebas posteriores a las pruebas unitarias y preceden a las pruebas del sistema (párr. 1, 2, 3).

El objetivo de las pruebas de integración es verificar el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente con el fin de comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la

funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes.

1.11.4.- Pruebas de validación.

En la página web **http://es.wikipedia.org/wiki/Pruebas_de_validaci%C3%B3n** se dice que:

> Las **pruebas de validación** en la ingeniería de software son el proceso de revisión que el sistema de software producido cumple con las especificaciones y que cumple su cometido. Es normalmente una parte del proceso de pruebas de software de un proyecto, que también utiliza técnicas tales como evaluaciones, inspecciones, y tutoriales. La validación es el proceso de comprobar lo que se ha especificado es lo que el usuario realmente quería.
>
> Se trata de evaluar el sistema o parte de este durante o al final del desarrollo para determinar si satisface los requisitos iniciales. La pregunta a realizarse es: ¿Es esto lo que el cliente quiere?.
>
> **Enfoques a la verificación**
>
> - Dinámica de verificación, también conocido como ensayos o experimentación.
> - Estática de verificación, también conocido como análisis.

Tipos

- Pruebas de aceptación: desarrolladas por el cliente.
- Pruebas alfa realizadas por el usuario con el desarrollador como observador en un entorno controlado (simulación de un entorno de producción).
- Pruebas beta: realizadas por el usuario en su entorno de trabajo y sin observadores.

Características

Comprobar que se satisfacen los requisitos:

- Se usan la mismas técnicas, pero con otro objetivo.
- No hay programas de prueba, sino sólo el código final de la aplicación.
- Se prueba el programa completo.
- Uno o varios casos de prueba por cada requisito o caso de uso especificado.
- Se prueba también rendimiento, capacidad, etc. (y no sólo resultados correctos).
- Pruebas alfa (desarrolladores) y beta (usuarios) (párr. 1, 2, 3, 4, 5).

El objetivo de estas pruebas es obtener información útil para la validación de la implementación de los algoritmos estudiados. Se asume para esta parte que el software ha cumplido la etapa de verificación, por lo tanto está libre de errores de tiempo de ejecución, lo que no significa que esté libre de errores lógicos (diferencias entre la estrategia propuesta y la implementada).

1.11.5.- Pruebas de sistema.

1.11.5.1.- Caja blanca (sistemas).

Según la página web **http://es.wikipedia.org/wiki/Pruebas_de_caja_blanca**:

> Las **pruebas de caja blanca** (también conocidas como *pruebas de caja de cristal* o *pruebas estructurales*) se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. El testeador escoge distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados.
>
> Al estar basadas en una implementación concreta, si ésta se modifica, por regla general las pruebas también deberán rediseñarse.
>
> Aunque las pruebas de caja blanca son aplicables a varios niveles unidad, integración y sistema, habitualmente se aplican a las unidades de software. Su cometido es comprobar los flujos de ejecución dentro de cada unidad (función, clase, módulo, etc.) pero también pueden testear los flujos entre unidades durante la integración, e incluso entre subsistemas, durante las pruebas de sistema.
>
> A pesar de que este enfoque permite diseñar pruebas que cubran una amplia variedad de casos de prueba, podría pasar por alto partes incompletas de la

especificación o requisitos faltantes, pese a garantizar la prueba exhaustiva de todos los flujos de ejecución del código analizado.

Las principales técnicas de diseño de pruebas de caja blanca son:

- Pruebas de flujo de control
- Pruebas de flujo de datos
- Pruebas de bifurcación (*branchtesting*)
- Pruebas de caminos básicos (párr. 1, 2, 3, 4).

Éstas se basan en el conocimiento de la lógica interna del código del sistema. Las pruebas contemplan los distintos caminos que se pueden generar gracias a las estructuras condicionales, a los distintos estados del mismo.

1.11.5.2.- Caja negra (sistemas).

En la página **http://es.wikipedia.org/wiki/Caja_negra(sistemas)** se dice que:

> En teoría de sistemas y física, se denomina **caja negra** a aquel elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. En otras palabras, de una caja negra nos interesará su forma de interactuar con el medio que le rodea (en ocasiones, otros elementos que también podrían ser cajas negras) entendiendo **qué es lo que hace**, pero sin dar importancia a **cómo lo hace**. Por tanto, de una caja negra deben estar muy bien definidas sus entradas y salidas, es decir, su interfaz; en cambio, no se precisa definir ni conocer los detalles internos de su funcionamiento (párr. 1).

El sistema de pruebas de caja negra no considera la codificación dentro de sus parámetros a evaluar, es decir, que no están basadas en el conocimiento del diseño interno del programa. Estas pruebas se enfocan en los requerimientos establecidos y en la funcionalidad del sistema.

1.11.6.- Pruebas de aceptación

La página web

http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node55.html se dice que:

> El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento.
>
> Las pruebas de aceptación son definidas por el usuario del sistema y preparadas por el equipo de desarrollo, aunque la ejecución y aprobación final corresponden al usuario.
>
> La validación del sistema se consigue mediante la realización de pruebas de caja negra que demuestran la conformidad con los requisitos y que se recogen en el plan de pruebas, el cual define las verificaciones a realizar y los casos de prueba asociados. Dicho plan está diseñado para asegurar que se satisfacen todos los requisitos funcionales especificados por el usuario teniendo en cuenta también los requisitos no funcionales relacionados con el rendimiento, seguridad de acceso al sistema, a los datos y procesos, así como a los distintos recursos del sistema (párr. 1).

1.11.6.1.- Pruebas de regresión.

En la página web **http://es.wikipedia.org/wiki/Pruebas_de_regresion** se menciona que:

> Se denominan **Pruebas de regresión** a cualquier tipo de pruebas de software que intentan descubrir las causas de nuevos errores (bugs), carencias de funcionalidad, o divergencias funcionales con respecto al comportamiento esperado del software, inducidos por cambios recientemente realizados en partes de la aplicación que anteriormente al citado cambio no eran propensas a este tipo de error. Esto implica que el error tratado se reproduce como consecuencia inesperada del citado cambio en el programa.
>
> Este tipo de cambio puede ser debido a prácticas no adecuadas de control de versiones, falta de consideración acerca del ámbito o contexto de producción final y extensibilidad del error que fue corregido (fragilidad de la corrección), o simplemente una consecuencia del rediseño de la aplicación.
>
> Por lo tanto, en la mayoría de las situaciones del desarrollo de software se considera una buena práctica que cuando se localiza y corrige un bug, se grabe una prueba que exponga el bug y se vuelvan a probar regularmente después de los cambios subsiguientes que experimente el programa.
>
> Existen herramientas de software que permiten detectar este tipo de errores de manera parcial o totalmente automatizada, la práctica habitual en programación extrema es que este tipo de pruebas se ejecuten en cada uno de los pasos del ciclo de vida del desarrollo del software (párr. 1, 2, 3, 4).

El objetivo de las pruebas de regresión es comprobar que los cambios sobre un componente de un sistema de información, no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.

Las pruebas de regresión se deben llevar a cabo cada vez que se hace un cambio en el sistema, tanto para corregir un error como para realizar una mejora. No es suficiente probar sólo los componentes modificados o añadidos, o las funciones que en ellos se realizan, sino que también es necesario controlar que las modificaciones no produzcan efectos negativos sobre el mismo u otros componentes.

Las pruebas de regresión pueden incluir:

- La repetición de los casos de pruebas que se han realizado anteriormente y están directamente relacionados con la parte del sistema modificada.
- La revisión de los procedimientos manuales preparados antes del cambio, para asegurar que permanecen correctamente.
- La obtención impresa del diccionario de datos de forma que se compruebe que los elementos de datos que han sufrido algún cambio son correctos.

1.11.7.- Pruebas de carga.

La página web **http://es.wikipedia.org/wiki/Pruebas_de_rendimiento_del_software** dice que:

> Este es el tipo más sencillo de pruebas de rendimiento. Una prueba de carga se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de

peticiones esperada. Esta carga puede ser el número esperado de usuarios concurrentes utilizando la aplicación y que realizan un número específico de transacciones durante el tiempo que dura la carga. Esta prueba puede mostrar los tiempos de respuesta de todas las transacciones importantes de la aplicación. Si la base de datos, el servidor de aplicaciones, etc. también se monitorizan, entonces esta prueba puede mostrar el cuello de botella en la aplicación (párr. 1).

El propósito principal de una prueba de carga es simular el acceso de muchos usuarios a un servidor al mismo tiempo.

1.11.8.- Pruebas de prestaciones.

La página web **http://www.morales-vazquez.com/pdfs/prestaciones.pdf** se refiere a:

"Las pruebas de prestaciones no arrojan nunca resultados absolutos. Devuelven valores aproximados y con un cierto margen de error. Minimizar ese error es nuestra tarea y para ello se hace imprescindible un amplio conocimiento de todos los conceptos relacionados en esta tarea y seguir una metodología clara y precisa" (párr. 1).

La principal necesidad de realizar unas pruebas de prestaciones es obvia y más aún en los sistemas Web: habitualmente los principales problemas que nos encontramos en nuestra utilización diaria de este tipo de aplicaciones no son ni caídas del sistema ni anomalías en su funcionamiento, sino problemas de rendimiento y degradación de recursos.

La finalidad de realizar pruebas de prestaciones es simular la normal utilización del sistema antes de su paso a explotación para predecir anticipadamente este tipo de situaciones y facilitar su corrección.

1.11.9.- Pruebas de recorrido.

Según la página web **http://www.slideshare.net/LuisDomingo/la-auditora-de-software-7728198**:

> Nunca se puede asegurar que una aplicación funcionará correctamente siempre, pues es inviable económicamente realizar pruebas de todos los componentes individuales y de todos los componentes como conjunto.
>
> Las pruebas deben ser diseñadas para descubrir fallos y no para demostrar que el software funciona. Siendo más razonable diseñar pruebas de aquellas partes en donde la probabilidad de fallo es mayor. Las pruebas deben ser diseñadas por personas distintas a las personas que desarrollan el componente que se desea probar (párr. 1, 2).

1.11.10.- Pruebas de mutación.

La página web **http://www.slideshare.net/LuisDomingo/la-auditora-de-software-7728198** se dice que: "Esta prueba está basada en la introducción deliberada de diferentes códigos externos al programa (bugs) para reexaminar si estos bugs pueden ser detectados. Requiere gran disponibilidad de recursos de computación" (párr. 1).

La página web **http://www.willydev.net/descargas/prev/MutaJUnit.PDF** manifiesta que: "Las pruebas mediante mutación fueron propuestas originalmente por Hamlet (1977) y DeMillo et al. (1978), y han sido desde entonces muy utilizadas. Dado un programa porque se va a probar, se genera un conjunto de copias de P, pero introduciendo en cada copia

una pequeña alteración, normalmente sintáctica, mediante la aplicación de un operador de mutación". A estas copias levemente alteradas se las llama "mutantes de P. (párr. 1)

1.11.11.- Pruebas concurrentes.

La página web **http://www.slideshare.net/LuisDomingo/la-auditora-de-software-7728198** menciona que: "Esto viene al caso de que comúnmente uno define los escenarios de Performance o de pruebas de Carga de la manera: "La aplicación debe soportar un máximo de 250 usuarios concurrentes" poniendo implícitamente la cantidad e usuarios concurrentes como parámetro de entrada para las pruebas (párr. 15).

En la cadena de valor del desarrollo de un software específico, el proceso de prueba es clave a la hora de detectar errores o fallas. Conceptos como estabilidad, escalabilidad, eficiencia y seguridad se relacionan a la calidad de un producto bien desarrollado. Las aplicaciones de software han crecido en complejidad y tamaño, y por consiguiente también en costos. Hoy en día es crucial verificar y evaluar la calidad de lo construido de modo de minimizar el costo de su reparación. Mientras antes se detecte una falla, más barata es su corrección.

El proceso de prueba es un proceso técnico especializado de investigación que requiere de profesionales altamente capacitados en lenguajes de desarrollo, métodos y técnicas de pruebas y herramientas especializadas. El conocimiento que debe manejar un ingeniero de prueba es muchas veces superior al del desarrollador de software.

1.12. Calidad de Software.

La calidad del Software es un conjunto de cualidades medibles y específicas que varía de un sistema a otro, dependiendo de tipo de software que se va a desarrollar, para determinar su utilidad y existencia. El desarrollo de software se ha convertido en uno de los principales problemas que tiene que afrontar la Ingeniería de Software. Tanto las comercializadoras de software y los investigadores, por esto se hace indispensables realizar una análisis de los modelos de calidad como son : Norma ISO/IEC, integración del modelo de maduración de la capacidad (CMMI) y (IT MARK) modelo de calidad para PYMES, para determinar los beneficios y sus inconvenientes que presenta para el desarrollo de Software con calidad.

La **calidad del software** es una preocupación a la que se dedican muchos esfuerzos. Sin embargo, el software casi nunca es perfecto. Todo proyecto tiene como objetivo producir software de la mejor calidad posible, que cumpla, y si puede supere las expectativas de los usuarios.

En la **web** **http://bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm** se dice que:

> La calidad del software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia. La calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad.
>
> La calidad del software es medible y varía de un sistema a otro o de un programa a otro. Un software elaborado para el control de naves espaciales debe ser confiable al nivel de "cero fallas"; un software hecho para ejecutarse una sola vez no requiere el mismo nivel de calidad; mientras que un producto de software para ser explotado durante un largo período (10 años o más), necesita ser confiable, mantenible y flexible para disminuir los costos de mantenimiento y perfeccionamiento durante el tiempo de explotación (párr. 1, 2).

1.12.1.- Norma ISO/IEC 12007.

Según la página **http://es.wikipedia.org/wiki/ISO/IEC_12207**:

> ISO/IEC 12207 establece un proceso de ciclo de vida para el software que incluye procesos y actividades que se aplican desde la definición de requisitos, pasando por la adquisición y configuración de los servicios del sistema, hasta la finalización de su uso. Este estándar tiene como objetivo principal proporcionar una estructura común para que compradores, proveedores, desarrolladores, personal de mantenimiento, operadores, gestores y técnicos involucrados en el desarrollo de software usen un lenguaje común. Este lenguaje común se establece en forma de procesos bien definidos (párr. 1, 2).

1.12.1.1.-Procesos.

Los procesos se clasifican en tres tipos: Principales, de soporte y de la organización. Los procesos de soporte y de organización deben existir independientemente de la organización y del proyecto ejecutado. Los procesos principales se instancian de acuerdo con la situación particular.

- Procesos principales.
  - Adquisición.
  - Suministro.
  - Desarrollo.
  - Operación.
  - Mantenimiento.

- Procesos de soporte.
  - Documentación

- Gestión de la configuración.
- Aseguramiento de calidad.
- Verificación.
- Validación.
- Revisión conjunta.
- Auditoría.
- Resolución de problemas.

- Procesos de la organización.

  - Gestión.
  - Infraestructura.
  - Mejora.
  - Recursos Humanos.

En la siguiente gráfica se muestra la dependencia entre Procesos, Actividades y Tareas.

Versiones

- ISO/IEC 12207:1995. Primera publicación.
- ISO/IEC 12207:1995/Amd 1:2002. Primera modificación.
- ISO/IEC 12207:1995/Amd 2:2004. Segunda modificación.
- ISO/IEC 12207:2008.

1.12.2.- Norma ISO/IEC 29119.

Según la página **http://www.javiergarzas.com/2009/03/isoiec-29119-hacia-un-nueva-norma-para.html**:

> En mayo de 2007 cuando ISO creó un grupo de trabajo (WG26) para desarrollar un nuevo "Estándar de Pruebas Software". El objetivo principal de esta norma, actualmente en desarrollo, es que sirva de referente internacional en el ámbito de las pruebas software y que permita tanto eliminar las inconsistencias existentes entre las actuales normas, así como cubrir aquellas áreas del "testing" que simplemente no habían sido tratadas hasta ahora en el resto de normas publicadas. Esta futura norma se conoce como "ISO/IEC 29119 Software Testing", cuya publicación en versión FIS (Final International Standard) está prevista para mediados del año 2012.

> La ISO/IEC 29119 cubrirá el ciclo de vida completo, a través del análisis, diseño, implementación y mantenimiento de las pruebas software. Y se basa en las principales normas que en la actualidad son referentes en pruebas del software, modelos de procesos y ciclos de vida, principalmente:

- BSI (British Standards Institution): BS 7925-1, Software Testing: Part 1-Vocabulary y BS 7925-2, Software Testing: Part 2-Software Component Testing.
- IEEE: IEEE Std. 829, Software Test Documentation y IEEE Std 1008, Software Unit Testing, IEEE Std 1012-1998 Software Verification and Validation y IEEE Std 1028-1997 Software Reviews.

- ISO/IEC: ISO/IEC 12207, Software Life Cycle Processes, ISO/IEC 15289, System and Software Life Cycle Process Information Products y ISO/IEC TR 19759, Guide to the Software Engineering Body of Knowledge (párr. 1, 2, 3)

Según la página **web** **http://in2test.lsi.uniovi.es/gt26/** menciona que: "La estructura de ISO/IEC 29119 consta de cuatro partes:

1. Conceptos y Vocabulario
2. Proceso de Pruebas
3. Documentación de Pruebas
4. Técnicas de Prueba

De acuerdo con el plan de trabajo aprobado por ISO en una primera fase se elaboran las partes 2 y 3, y posteriormente las 1 y 4" (párr. 1).

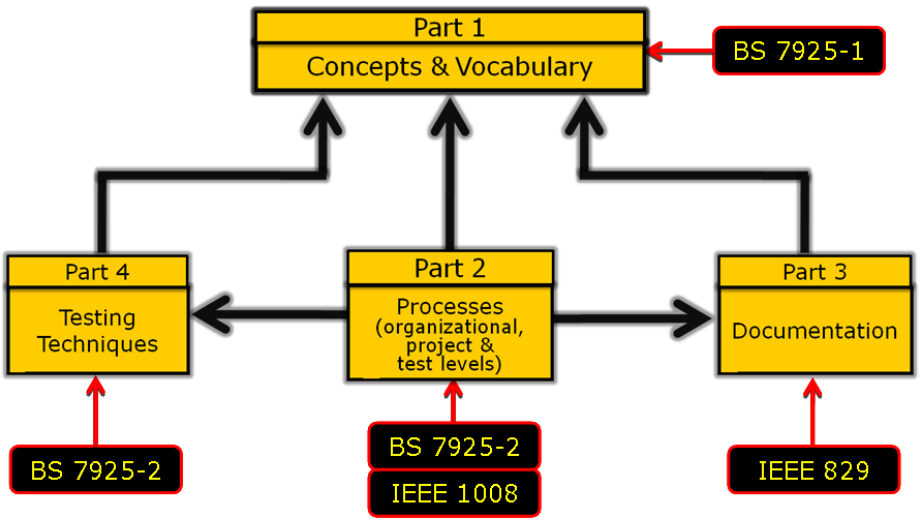


Figura 8. Estructura de ISO/IEC 29119

1.13. Herramientas para pruebas Software.

En el mercado actual la automatización de procesos de pruebas de software se ha convertido en extremadamente importante. Empresas de desarrollo de software utilizan herramientas de automatización de pruebas de software para colmar las lagunas en los sistemas recientemente desarrollados para que el usuario final reciba un producto de calidad que está libre de errores.

1.13.1.- Asistente de QA Pro 2011.

Asistente de QA Pro es un software automatizado herramienta prueba desarrollada por Seapine Software. Seapine Software es un proveedor líder de herramientas de desarrollo de producto para las organizaciones de TI en todo el mundo. Mientras que la empresa ofrece una variedad de diferentes herramientas informáticas diseñado para equipar a los clientes con tecnologías inteligentes. Diseñado para proporcionar pruebas funcionales automatizadas y la prueba de carga, Asistente de QA Pro reducirá su empresa de inversión en software y tiempo de formación.

1.13.2.- Oracle Enterprise Manager

Oracle es bien conocida en la industria de TI; la compañía ha estado proporcionando aplicaciones de software de negocios integrada por décadas. Oracle Enterprise Manager está diseñado para aplicaciones empresariales, y ofrece control extrema de las empresas de sus recursos de TI sin una etiqueta de precio extrema. Enterprise Manager ofrece herramientas de pruebas rigurosas para garantizar una gestión simplificada de prueba, promoviendo así la máxima eficiencia.

1.13.3.- Pruebas en cualquier lugar

Pruebas en cualquier lugar son un software fácil de utilizar pruebas de aplicación que puede dar a los usuarios las herramientas que necesitan para participar en pruebas automatizadas en cualquier lugar. Con cinco formas diferentes para automatizar, junto con algunas características de presentación de informes, los usuarios pueden reducir su costo total de propiedad y reducir la cantidad de tiempo necesario para probar programas hasta un 60%. Esta aplicación ofrece automatización inteligente fiable y tecnologías revolucionarias para secuencias de comandos y pruebas grabadas.

1.13.4.- AppLabs

AppLabs pretende ser la mayor calidad y pruebas gestión empresa de software en el mundo. Sus herramientas de prueba de automatización avanzadas permiten a los clientes efectivamente lograr el rendimiento óptimo de su inversión en software.

1.13.5.- SmarteSoft

SmarteSoft hace prueba de automatización fácil y asequible. Esta solución administrar y supervisar los requisitos del proyecto y presentará un informe defectos y casos de prueba en un entorno cómodo. Si desea pruebas de regresión y acceso a un centro de control unificado con herramientas de prueba manuales y automáticas.

1.13.6.- Sonar

Sonar es una herramienta de software libre y gratuito que permite gestionar la calidad del código fuente. Al instalarla tendremos un servicio para recopilar, analizar, y visualizar métricas del código fuente. Como resultado la empresa podrá visualizar informes estáticos con resumen de las métricas del código fuente y analizar la evolución temporal de los productos software que se encuentran en proceso de desarrollo o mantenimiento.

1.13.7.- JMETER

Herramienta para la realización de pruebas de rendimiento. Simula el ingreso de usuarios periódicos y permite hacer pruebas de carga y de stress. Lleva a cabo simulaciones sobre cualquier recurso de Software.

1.13.8.- TestLink

TestLink es una herramienta que nos permite fácilmente crear y gestionar casos de prueba, así como organizarlos en planes de pruebas. Estos planes de pruebas permiten a los miembros del equipo ejecutar casos de prueba y realizar un seguimiento de los resultados de la pruebas de forma dinámica, generar informes, trazabilidad de requisitos de software, priorizar y asignar tareas.

Es una herramienta Web basada en PHP y compatible con MySQL, PostgreSQL y MS SQLServer, así mismo dispone de licencia GPL; que es desarrollada y mantenida por la "Open Community Testers" por lo que los propios desarrolladores entienden y conocen las necesidades de los equipos de calidad.

CAPITULO II

2.- Descripción, análisis e interpretación de resultados.

2.2. Antecedentes.

Un desarrollo de software podrá estar marcado por características variables como sus dimensiones, tecnología, sector o criticidad, pero todos ellos comparten una característica en

común, lo desarrollan personas, mismas que cometen errores y cambian de ideas, por eso hay que verificar las veces que se requiera las piezas de software.

Iniciemos un recorrido por todo el proceso seguido en el desarrollo de un software, desde que se concibe como idea hasta su puesta en marcha.

Hay que mencionar que el desarrollo de software es un proceso creativo sistematizado por la Ingeniería del Software con el fin de acotar el riesgo al fracaso en la consecución del objetivo creativo por medio de diversas técnicas que se han demostrado de manera adecuada en base a la experiencia previa.

Las pruebas representan una actividad fundamental en el desarrollo de software y, en muchos casos, supone prácticamente el único medio empleado en los proyectos para la verificación y validación del software.

2.2.- Definición del Problema.

2.2.1. Planteamiento del Problema.

Uno de los aspectos más descuidados en este gran proceso de desarrollo de software son las pruebas y su gestión, a pesar de ser el medio por el cual se puede asegurar la calidad del producto de software, cumplir con los requisitos del usuario en la mayoría de los casos no es suficiente, para ello hace falta definir un proceso de pruebas estandarizado para todos los proyectos, asegurando siempre la mejora continua.

En las universidades que ofertan Ingeniería Informática o carreras afines, las actividades de pruebas de software pasan casi desapercibidas, pues no se crean casos de pruebas que permitan garantizar la calidad del software, la ausencia de una orientación clara en la planificación del proyecto y de políticas organizacionales que apoyen este proceso debido al desconocimiento o inaplicabilidad aumentan el riesgo de producir software que inmediatamente reporta continuos errores o fallas graves.

No se trata únicamente de invertir más tiempo o contratar más personal, lo que se necesita es un modelo gestionable y estandarizado bajo normas internacionales.

Existen herramientas que soportan las pruebas del software en todas las etapas de un proyecto. Algunas ofrecen una serie integrada que soporta las pruebas y el desarrollo a lo largo de la vida de un proyecto, desde la reunión de los requisitos hasta el soporte del funcionamiento en vivo del sistema. Algunas herramientas se enfocan en una sola parte del ciclo de vida. La base de conocimientos de las herramientas de prueba de software proporciona los criterios funcionales que se esperan de una eficaz herramienta.

Debido a la poca importancia que habitualmente se le da al proceso de gestión de pruebas y el precio elevado de las herramientas para esta gestión, dan como resultado un software bajo en calidad y operatividad.

El mercado actual no cuenta con una herramienta que permita controlar y gestionar de manera adecuada las pruebas que realizan durante todo el ciclo de vida del software, tampoco implementan el estándar ISO/IEC 29119 en las actividades de evaluación y de calidad. Por ende solo realiza algunas pruebas y solo al código al final, días antes de entregar el producto al cliente.

Anteriormente, la empresa llevaba toda la gestión de pruebas de manera manual y poco ortodoxo; en la actualidad, los registra en archivos de Excel sin base en ningún estándar lo mismo que conlleva a confusión y no tener claridad en la situación actual de un proyecto de software.

2.2.2.- Formulación del Problema.

¿Cómo superar las deficiencias del proceso de pruebas de software aplicando el estándar ISO/IEC 29119 para la gestión y control de pruebas en el aseguramiento de calidad de software?

2.2.3.- Sistematización del Problema.

¿Qué conceptos están relacionados con el proceso de pruebas de software?

¿Cómo analizar el proceso de pruebas de software como medio de aseguramiento de calidad por excelencia?

¿Cuáles son los estándares y modelos más usados y sus deficiencias para con los procesos de pruebas?

¿Qué es el estándar ISO/IEC 29119 para pruebas de software?

2.3.-Alcance.

Este estándar, cuya elaboración comenzó en el 2007 tiene como objetivo cubrir todo el ciclo de vida de las pruebas de sistemas software incluyendo los aspectos relativos a la organización, gestión, diseño y ejecución de las pruebas, para remplazar varios estándares IEEE y BSI sobre pruebas de software.

La estructura de ISO/IEC 29119 consta de cuatro partes:

1. Conceptos y Vocabulario

2. Proceso de Pruebas
3. Documentación de Pruebas
4. Técnicas de Prueba

De acuerdo con el plan de trabajo aprobado por ISO en una primera fase se elaboran las partes 2 y 3, y posteriormente la 1 y 4.

Con la implementación del modelo de software planteado se obtendrá una herramienta de gestión, control y seguimiento basado en la ISO/IEC 29119 para el proceso de pruebas de software como medio de aseguramiento de la calidad.

2.4.-Objetivo General.

- Diseñar una herramienta que permita llevar el control y la gestión de la evaluación, pruebas durante el desarrollo del software con el estándar ISO/IEC19119

2.5.-Objetivos Específicos.

- Encontrar y documentar en el sistema propuesto a desarrollar los defectos que puedan afectar la calidad del software.
- Almacenar el diseño de pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo

- Identificar, encontrar y asegurar que los defectos encontrados sean corregidos antes de entregar el software.
- Aplicar las fases del estándar ISO/IEC19119 dentro del software propuesto.

2.6.- Justificación de la investigación.

La empresa Omnisoft como entidad desarrolladora de software tiene la necesidad de garantizar la alta calidad de software, debiendo dedicar tiempo de desarrollo similar al de programación y desde luego implica un impacto en el costo; siendo el proceso de pruebas una herramienta esencial para obtener un software con menor riesgo a fallos y mejorando el nivel de calidad para sus clientes. Esta fase que muchos de los involucrados a través de la presente investigación consideran implementar.

La importancia del tema a investigar está relacionada con un problema actual que es contar con productos de software, libre de fallas y errores. Aplicable de tal forma que sus resultados aporten con estándares de calidad en el área de la Ingeniería de software y que la sociedad tan dependiente de las tecnologías en sus tareas cotidianas cuenten con productos que ayuden al aseguramiento de su calidad y mejora continua. Es por esto que la intención es rescatar la importancia de los estándares y metodologías para las pruebas de software que van a garantizar calidad.

En el mercado actual existe una variedad de herramientas y normas que permiten tener un proceso de pruebas pero todas son particulares, como vimos en el primer capítulo tenemos normas pero la que más enfoque tiene al proceso de software es la ISO/IEC 29119 la misma que permite cubrir todo el ciclo de vida completo, a través del análisis, diseño, implementación y mantenimiento de las pruebas software, pero no existe una herramienta que permita tener una gestión como nos dice la norma ISO, es por ello que se ve necesario el desarrollo de una herramienta, enfocándola para nuestro entorno pero con bases internacionales con el fin de obtener un mejor producto y una mejor calidad.

Análisis de las herramientas:

	Gestiona		Monitorea		Evalúa		Base Norma ISO		Permite toma decisiones	
Herramientas	**Si**	**No**	**Si**	**No**	**Si**	**No**	**Si**	**No**	**SI**	**No**
Asistente de QA Pro 2011	X		X			x		x		x
Oracle Enterprise Manager	X		X			x		x		x
Pruebas en cualquier lugar	X		X			x		x		x
AppLabs	X		X			x		x		x
SmarteSoft	X		X			x		x		x
Sonar	X		X			x		x		x
JMETER	X		X			x		x		x
TestLink	X		X			x		x		x

Resultado: Se puede apreciar que ninguna herramienta es completa, tienen un enfoque de gestión pero sobre puntos específicos del ciclo de vida. No se basan en normas ISO para pruebas de software y no permite una clara visión para la toma de decisiones.

2.7.-Hipótesis.

¿EL DISEÑO DE UN MODELO CON ESTÁNDAR ISO/IEC 29119 PERMITIRÁ MEJORAR LA EVALUACION DE PRUEBAS DE SOFTWARE?

2.8.-Aspectos metodológicos de la investigación.

La realización de las pruebas tiene por función controlar la calidad del software durante todo el proceso de desarrollo. Aplican las pruebas que se diseñan para medir la confiabilidad del software tanto desde el punto de vista conceptual como de su utilización.

Mediante la presente investigación se pretende llegar a sistematizar el estándar ISO/IEC29119 que permita la gestión, control y seguimiento de las pruebas de desarrollo de software con lo cual se mejorará notablemente la calidad del producto final, consiguiendo así mayor confiabilidad a la empresa cuando entregue el producto al cliente.

Es necesaria una comprobación sistemática para buscar los posibles errores durante todo el ciclo de vida; se debe velar por el cumplimiento satisfactorio de los objetivos relacionados con la confiabilidad, usabilidad del software desde los puntos de vista conceptual de la utilización, representación y codificación**.**

2.9.-Método de investigación.

Es el procedimiento riguroso formulado de una manera secuencial y lógica que el investigador debe seguir en la adquisición del conocimiento.

2.9.1.-Método Inductivo.

Proceso de conocimiento que se inicia con la observación de fenómenos particulares para llegar a conclusiones y premisas de carácter general que pueden ser aplicadas a situaciones similares a la observación.

2.10.-Tipo de investigación.

2.10.1.-Investigación de campo.

La investigación de campo se presenta mediante la manipulación de una variable externa no comprobada en condiciones rigurosamente controladas, con el fin de describir de qué modo o por qué causas se produce una situación o acontecimiento particular.

Podríamos definirla diciendo que es el proceso que, utilizando el método científico, permite obtener nuevos conocimientos en el campo de la realidad social (investigación pura), o bien estudiar una situación para diagnosticar necesidades y problemas a efectos de aplicar los conocimientos con fines prácticos (investigación aplicada).

Este tipo de investigación es también conocida como investigación in situ ya que se realiza en el propio sitio donde se encuentra el objeto de estudio. Ello permite el conocimiento más a fondo del investigador, puede manejar los datos con más seguridad y podrá soportarse en diseños exploratorios, descriptivos y experimentales, creando una situación de control en la cual manipula sobre una o más variables dependientes (efectos).

2.10.2.-Investigación Bibliográfica.

La investigación bibliográfica es aquella etapa de la investigación científica donde se explora que se ha escrito en la comunidad científica sobre un determinado tema o problema. ¿Qué hay que consultar, y cómo hacerlo?

La bibliografía está constituida por un método científico y por el objetivo de la ciencia, formándose entre el conocimiento vulgar y el científico.

Utiliza un método histórico que consta de apoyo teórico y metodología científica

2.10.3. Investigación experimental.

La investigación experimental está integrada por un conjunto de actividades metódicas y técnicas que se realizan para recabar la información y datos necesarios sobre el tema a investigar y el problema a resolver, se presenta mediante la manipulación de una variable experimental no comprobada, en condiciones rigurosamente controladas, con el fin de describir de qué modo o por qué causa se produce una situación o acontecimiento particular.

Su diferencia con los otros tipos de investigación es que el objetivo de estudio y su tratamiento dependen completamente del investigador, de las decisiones que tome para manejar su experimento.

El experimento es una situación provocada por el investigador para introducir determinadas variables de estudio manipuladas por él, para controlar el aumento o disminución de esas variables y su efecto en las conductas observadas.

Esto da lugar al desarrollo de investigaciones conocidas como cuantitativas, las cuales se apoyan en las pruebas estadísticas tradicionales. Pero especialmente en el ámbito de las ciencias sociales se observan fenómenos complejos y que no pueden ser alcanzados ser observados a menos que se realicen esfuerzos con alto grado de subjetividad y orientados hacia las cualidades más que a la cantidad. Así se originan diversas metodologías para la recolección y análisis de datos (no necesariamente numéricos) con los cuales se realiza la investigación conocida con el nombre de Cualitativa.

2.11.- Entorno de la empresa Omnisoft de la ciudad de Quito.

2.11.1 Antecedentes.

OMNISOFT es una empresa consultora cuyo objetivo básico es ofrecer soluciones informáticas integrales, que se adapten a las necesidades de sus clientes, basando el desarrollo tecnológico en su experiencia profesional, técnica y de negocio.

OMNISOFT inicia su actividad en 1999 y colabora con las principales entidades financieras e industriales. Actualmente, cuenta con más de 200 empleados, distribuidos entre sus oficinas y los centros de sus clientes.

2.11.2 Funciones.

OMNISOFT es una compañía limitada cuyo objeto social es la prestación de servicios de consultoría y asesoría en las áreas de informática, ingeniería y telecomunicaciones;

identificando, planificando, elaborando, evaluando y supervisando proyectos de desarrollo en sus niveles de pre factibilidad, factibilidad, diseño u operación.

Además, la empresa brinda servicios en la elaboración de estudios económicos, financieros, de organización, administración, auditoria e investigación.

2.11.3. Objetivos de la empresa Omnisoft.

- ✓ Ejecutar una estrategia completa en la automatización de organizaciones para elevar su nivel competitivo hacia la excelencia. Darnos a conocer por nuestra capacidad y potencialidad en el área de la tecnología dentro de las organizaciones e instituciones más importantes del país.

2.11.4. Misión.

Nuestra misión es ser una herramienta clave e indispensable para las grandes empresas en donde nuestros productos sean líderes en el mercado. Buscamos constantemente la superioridad en todo lo que hacemos y sobre todo dedicamos tiempo a los clientes para satisfacerlos. Aprovechamos el talento creativo latente de cada uno de nuestros trabajadores para conseguir la mejora continua, sirviendo a la comunidad con productos de excelente calidad.

2.11.5.-Visión.

Queremos llegar a tener una renovación continua y aprovechar el poder creativo en todo lo que realiza la compañía: en ideas, en calidad, en satisfacción del cliente. Tratar a cada empleado con dignidad y respeto; siempre trabajando con honradez, integridad y ética en todos los aspectos del negocio, originando excelencia en reputación y siempre velando por la satisfacción al cliente.

2.11.6.-Estructura organizacional.

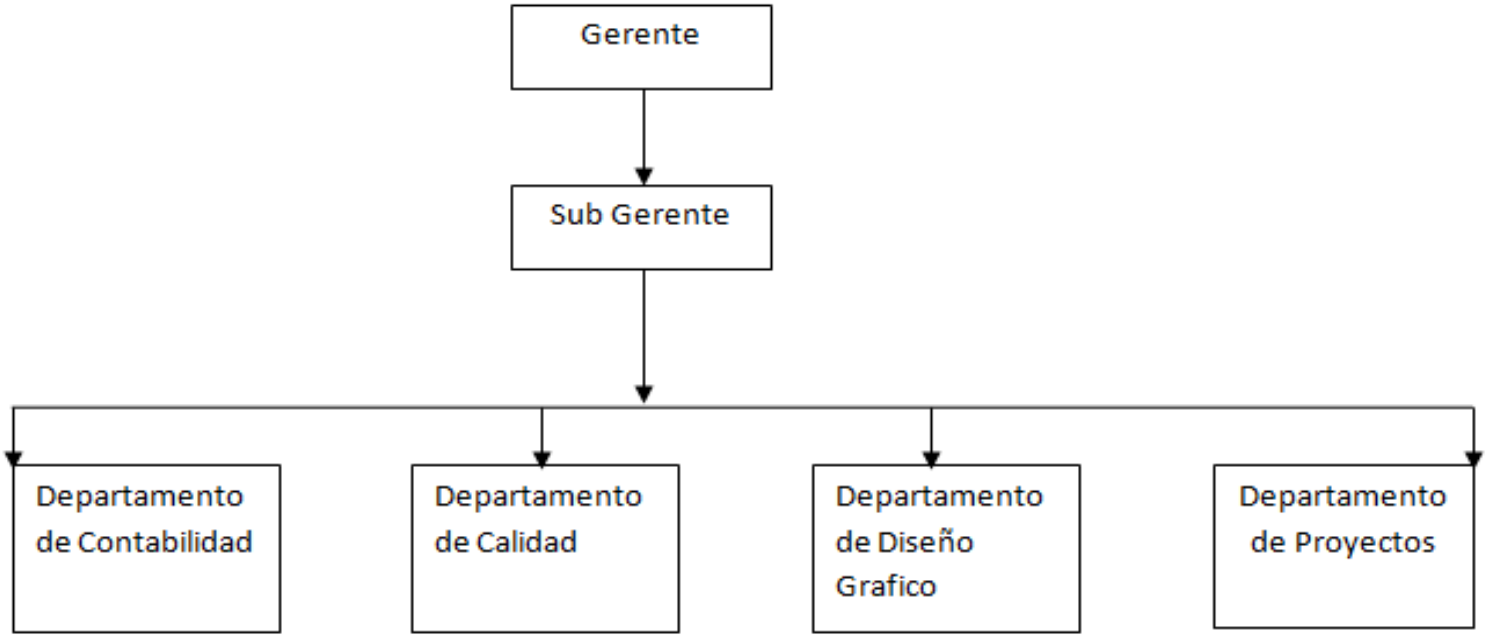


Figura 9. Estructura Organizacional.

2.11.7.-Productos de la empresa Omnisoft.

OMNISOFT ha desarrollado varios productos en diferentes áreas de negocio, tales como: Finanzas, Administración de Empresas, Gestión de Proyectos, Educación, Medicina, Hotelería, Floricultura, Comunicaciones e Internet.

Todos los productos son desarrollados con tecnología WEB, lo que permite a la empresa tener en sus sistemas de información las ventajas de los estándares empleados en el Internet, localmente en sus redes locales e Intranets.

2.11.7.1.-Área financiera contable.

Omnisoft posee los siguientes productos:

a) Sistema Financiero Contable Cooperativo

b) Sistema Administrativo Financiero

c) Sistema Académico Financiero

d) Sistema de Comercialización de Agua Potable

e) Sistema de Nómina de Personal y Recursos Humanos

f) Sistema de Gestión y Presupuesto

g) Sistema de Administración de Tesorería

2.11.7.2.-Área de educación.

Omnisoft ha desarrollado una solución académica, administrativa y financiera orientada a cubrir las necesidades del sector educativo. Ponemos a su disposición los siguientes productos:

a) Red Virtual de Educación

2.11.7.3.-Área de medicina.

Con la asesoría de varios médicos especialistas en diferentes áreas, se ha desarrollado dos sistemas de información encaminados a facilitar las tareas diarias de los médicos.

a) Sistema de Control de Pacientes.

b) Red Virtual de Medicina.

2.11.7.4.-Área de comunicaciones e internet.

Omnisoft ha desarrollado soluciones informáticas enfocadas a facilitar la administración, facturación y control de acceso de los usuarios de los proveedores de acceso a Internet (ISP's).

a) Sistema de Administración de ISP's

b) Filtros de Acceso a Internet.

2.11.7.5.-Área de gestión empresarial.

Con el objeto de optimizar la gestión de una empresa, Omnisoft desarrolló un sistema para el seguimiento, control, evaluación y atención de clientes, proveedores y empleados. Ponemos a su disposición nuestro sistema:

a) Sistema de Gestión Empresarial (CRM)

2.11.7.6.-Área de hotelería.

Omnisoft ha desarrollado algunos web sites (sitios web) para hoteles, agencias de viajes y empresas dedicadas al turismo. Además, tenemos dos sistemas de información orientados a la administración de hoteles y a la gestión de eventos y congresos.

a) Sistema de Administración Hotelera.

b) Sistema de gestión de eventos y congresos

2.11.7.7.- Área de floricultura.

Tomando en cuenta el crecimiento de exportaciones de flores, Omnisoft desarrolló un sistema de información orientado a facilitar la comercialización de flores hacia el extranjero, con un sistema multi-idioma, las solicitudes de compra, el seguimiento al envió del pedido y la descarga del inventario se automatizan optimizando recursos. La empresa pone a su disposición:

a) Sistema de Comercialización de Flores

2.12.- Análisis de los resultados de la encuesta aplicada al personal de Omnisoft.

La investigación propuesta se realizará a los administradores y al personal de los diferentes departamentos de la empresa, estarán enfocadas a un gerente general, tres jefes de proyectos, doce programadores, un DBA (Administrador Base Datos) y un arquitecto de software, obteniendo así un total de dieciocho personas involucradas para la recolección de datos.

En base a las preguntas planteadas se ha podido rescatar los siguientes resultados que se constituyen en fuente confiable para poder implementar cambios, teniendo como principal alternativa alcanzar la competitividad en la industria del software, para esto se requiere desarrollar y aplicar un modelo basado en metodologías o procedimientos estándares para el planeamiento, especificación y ejecución de pruebas de software.

Se ha realizado las siguientes preguntas que nos han permitido tener una visión más clara y concisa del entorno de la empresa.

1.- • ¿Qué conceptos están relacionados con el proceso de pruebas de software?

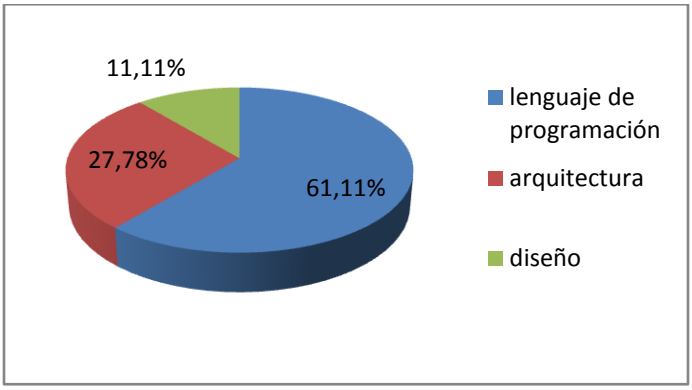
Tabla nº1.

Alternativa	Valor	Porcentaje
lenguaje de programación	11	61,11%
arquitectura	5	27,78%
diseño	2	11,11%
Total	**18**	**100%**

Fuente: Encuesta

Realizado por: Investigador

Gráfico nº1.



Fuente: Encuesta

Realizado por: Investigador

Después de haber obtenido los datos de la población, podemos darnos cuenta que un 61% de las personas encuestadas consideran que los conceptos de programación están relacionados con el proceso de pruebas de software, mientras que un 27% de los encuestados manifestaron que la arquitectura de software y un 11% el diseño.

2.-• ¿Cómo analizar el proceso de pruebas de software como medio de aseguramiento de calidad?

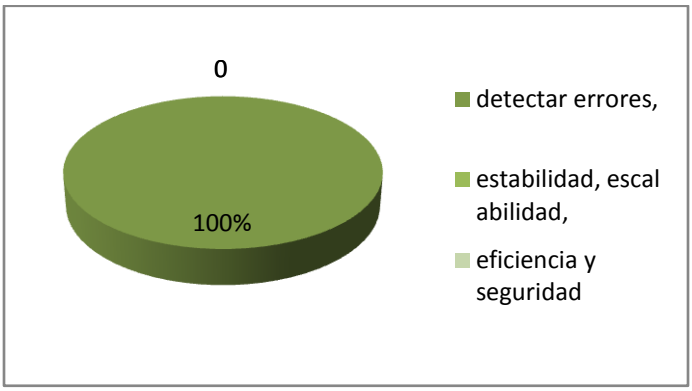
Tabla nº2.

Alternativa	Valor	Porcentaje
detectar errores, estabilidad, escalabilidad, eficiencia y seguridad	18	10,00,%
Total	**18**	**100%**

Fuente: Encuesta

Realizado por: Investigador

GRÁFICO Nº2.- BENEFICIOS DEL SOFTWARE LIBRE.



Fuente: Encuesta

Realizado por: Investigador

De las personas encuestadas opinan un 100% que para analizar el proceso de pruebas de software como medio de aseguramiento de calidad se deben detectar errores, tener una estabilidad, escalabilidad, eficiencia y seguridad

3. • ¿Cuáles son los estándares ISO e ISO/IEC relacionados con las pruebas del software?

Tabla nº3

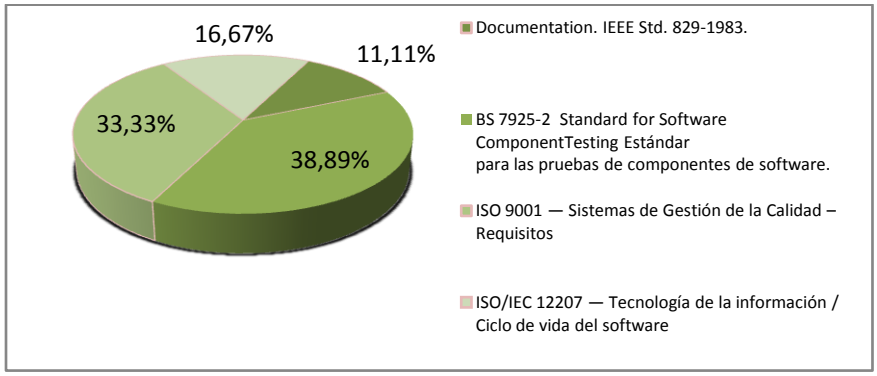
Respuestas	Valor	Porcentaje
Documentation. IEEE Std. 829-1983.	2	11,11%
BS 7925-2 Standard for Software Component Testing Estándar para las pruebas de componentes de software.	7	38,89%

ISO 9001 — Sistemas de Gestión de la Calidad – Requisitos	6	33,33%
ISO/IEC 12207 — Tecnología de la información / Ciclo de vida del software	3	16,67
Total	**18**	**100%**

Fuente: Encuesta

Realizado por: Investigador

Gráfico nº3.



Fuente: Encuesta

Realizado por: Investigador

En la pregunta aplicada, a los encuestados mencionan que los estándares ISO e ISO/IEC relacionados con las pruebas del software son la BS 7925-2 un 38%, la ISO 9001 un33%, la ISO/IEC 12207 un 16% y la IEEE Std 829-1983 un 16%

4. ¿Hay procesos de ingeniería de pruebas estándares aplicables en nuestro entorno?

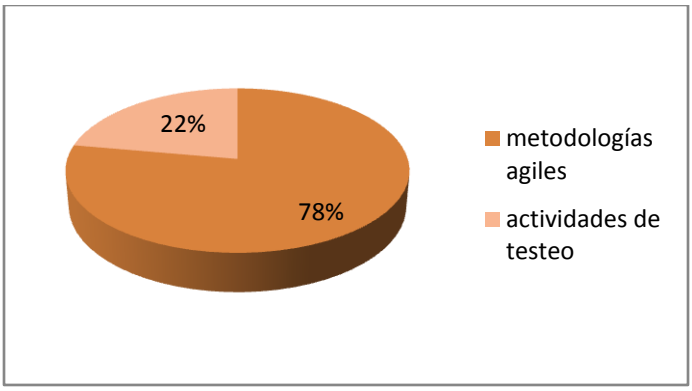
Tabla nº4

Respuesta	Valor	Porcentaje
metodologías agiles	14	77,78%
actividades de testeo	4	22,22%
Total	**18**	**100%**

Fuente: Encuesta

Realizado por: Investigador

Gráfico nº4.



Fuente: Encuesta

Realizado por: Investigador

Después de haber obtenido los datos de la encuesta realizada, podemos evidenciar que hay procesos de ingeniería de pruebas estándares aplicables en nuestro entorno como las metodologías ágiles representado por un 77% y las actividades de testeo representado por un 22%.

5. ¿Qué modelos de pruebas existen y se gestionan a través de herramientas de gestión?

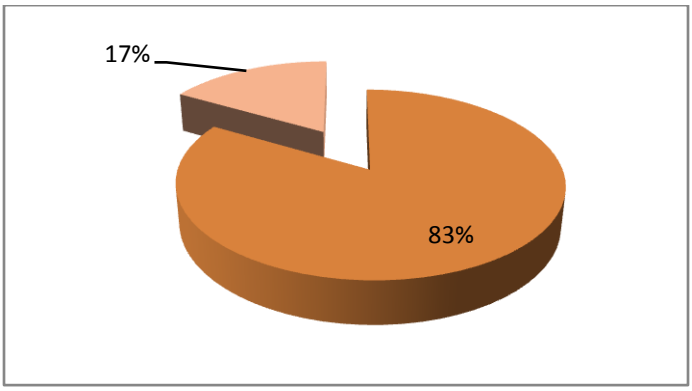
Tabla nº5.

Respuesta	Valor	Porcentaje
No existe una herramienta de gestión de las pruebas	15	83,33%
Existen Herramientas que realizan seguimiento de los casos de pruebas	3	16,67%
Total	**18**	**100%**

Fuente: Encuesta

Realizado por: Investigador

Gráfico nº 5.



Fuente: Encuesta

Realizado por: Investigador

Al respecto, la mayoría de encuestados, esto es el 83%, manifiestan que no existe una herramienta de gestión de las pruebas, mientras que un 16% menciona que existen herramientas que realizan seguimiento de los casos de pruebas.

6. ¿Cómo mejoramos la calidad con un entorno de gestión de pruebas?

Tabla nº6.

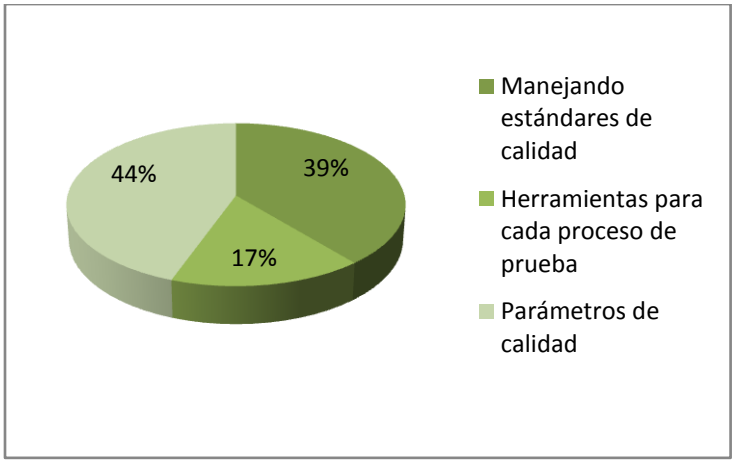
Respuesta	**Valor**	**Porcentaje**
Manejando estándares de calidad	7	38,89%
Herramientas para cada proceso de prueba	3	16,67%
Parámetros de calidad	8	44,44%
Total	**18**	**100%**

Fuente: Encuesta

Realizado por: Investigador

Gráfico nº6.

.



Fuente: Encuesta

Realizado por: Investigador

De los resultados obtenidos sobre esta interrogante como mejoramos la calidad con un entorno de gestión de pruebas, se pudo evidenciar que un 38% de los encuestados consideran que manejando estándares de calidad mientras que un 16% mencionan que con herramientas para cada proceso de prueba y un 44% manejando estándares de calidad.

7. ¿Cree usted que la gestión y el control ayudaría a mejorar la calidad del producto software?

Tabla nº7.

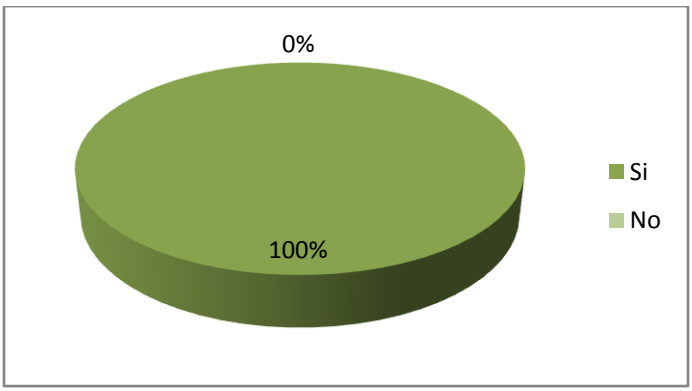
Alternativa	**Valor**	**Porcentaje**
SI	18	100%
NO	0	0%
Total	**18**	**100%**

Fuente: Encuesta

Realizado por: Investigador

Gráfico nº7.

.



Fuente: Encuesta

Realizado por: Investigador

De los resultados obtenidos de la encuesta realizada podemos evidenciar que un contundente 100% manifiestan que la gestión y el control ayudarían a mejorar la calidad del producto software, mientras que un 0% mencionan que NO.

8. ¿Cree usted que la herramienta de un modelo para evaluación/pruebas del software en base a ingeniería de pruebas aplicando el estándar ISO/IEC 29119 se pueda implementar en otras instituciones, empresas públicas y privadas, tiene algún enfoque centralista?

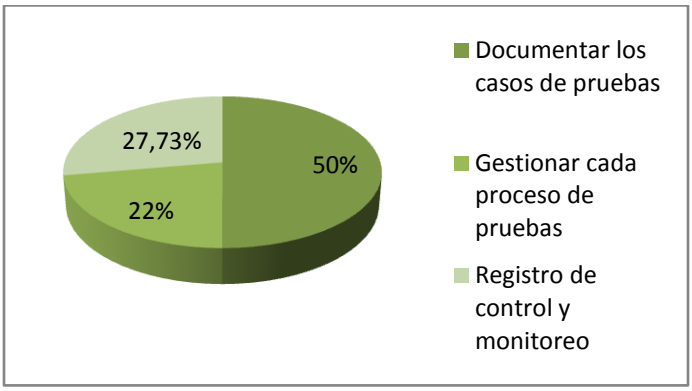
Tabla n8.

Respuesta	**Valor**	**Porcentaje**
Documentar los casos de pruebas	9	50,00%
Gestionar cada proceso de pruebas	4	22,22%
Registro de control y monitoreo	5	27,73%
Total	**18**	**100%**

Fuente: Encuesta

Realizado por: Investigador

Gráfico nº8.



Fuente: Encuesta

Realizado por: Investigador

En la pregunta realizada podemos evidenciar que un 50% de los encuestados mencionan que hay que documentar los casos de pruebas, un 22% que se deben gestionar los casos de pruebas, y un 27% que mencionan que hay que llevar un registro de control y monitoreo.

CAPITULO III

3.- Diseño de un modelo para evaluación/pruebas del software en base a ingeniería de pruebas aplicando el estándar ISO/IEC 29119 en la empresa Omnisoft de la ciudad de Quito.

3.1.- Introducción.

Actualmente, en el área de la informática la calidad de software se ha convertido en uno de los principales problemas que tiene que afrontar la Ingeniería de Software. Tanto las comercializadoras de software y los investigadores han venido realizando una gran cantidad de investigaciones sobre: ¿Cómo lograr software de calidad?, ¿Cómo evaluar el software de calidad? Estos dos grandes interrogantes han dado una serie de respuestas, donde estas

están estrechamente relacionadas. Pero para esto primero debemos definir **¿Qué es la Calidad de Software?** Según ISO "El conjunto de características de una entidad que le confieren su aptitud para satisfacer las necesidades expresadas y las implícitas". ISO 8402 (UNE 66-001-92).

La calidad es sinónimo de eficiencia, flexibilidad, portabilidad, usabilidad, seguridad e integridad.

La calidad del software es el conjunto de cualidades medibles y específicas que varía de un sistema a otro, dependiendo del tipo de software que se va a desarrollar, para determinar su utilidad y existencia. Este desarrollo debe ser confiable, mantenible y flexible para disminuirlos costos de mantenimiento y perfeccionamiento durante el tiempo de utilización y durante las etapas del ciclo de vida del software.

Para lograr el éxito en el desarrollo de software es necesario hacerlo con eficiencia y demostrar su buena usabilidad. Esto sólo es posible con la implantación de un Sistema para el Aseguramiento de la Calidad del Software con la definición internacional ISO de calidad ISO 15504 (SPICE), ampliamente aceptada, y por los estándares del grupo ISO Norma ISO/IEC 12007, Modelo de maduración de la Capacidad (CMMI).

En este proyecto de tesis vamos a realizar un **Diseño de un modelo para evaluación/pruebas del software en base a ingeniería de pruebas aplicando el estándar ISO/IEC 29119** especificando sus ventajas, desventajas y su aplicación ya que estos modelos son una serie de buenas técnicas para garantizar el ciclo de vida del software, orientados a los procesos de gestión y desarrollo de proyectos, para cualquier tipo de organización.

3.2.- Presentación.

En todo desarrollo de sistemas de software es de suma importancia el seguir alguna especificación que permita a los desarrolladores el tener una disciplina que haga que todas las etapas del desarrollo del sistema, desde la concepción de la idea inicial, requerimientos hasta la implementación del sistema en el ambiente de producción, sean no solo más coherentes sino también más formales.

El desarrollo de software que este proyecto propone ser una herramienta que pretende tener aplicación dentro del contexto de un problema real, tiene que seguir un proceso de análisis y diseño que proporcione los cimientos bajo los cuales se va a desarrollar la aplicación conjuntamente con normas de calidad en la creación de software. Es por esto que en este capítulo se detallan los procesos de ingeniería de software, análisis, diseño que se involucran durante el desarrollo de una aplicación de software.

El capítulo en sí proporciona una pequeña introducción a lo que es la disciplina de la ingeniería de software, y posteriormente detallará los procesos y principios de análisis y diseño del software que sustentan este proyecto. También se especifican las técnicas de documentación del software que son utilizadas para complementar el desarrollo del sistema que se propone. Aunque el área de estudio y de aplicación de la ingeniería de software abarca también las etapas más complejas de desarrollo y pruebas del software.

3.3. Implementación del Modelo de Evaluación/Pruebas de Software en base a ingeniería de pruebas aplicando el estándar ISO/IEC 29119.

En el presente capitulo, se describirán las complicaciones que se tuvieron para el desarrollo del modelo de evaluación. Dentro del proceso de pruebas se simularon concurrencia y funcionalidad, las pruebas de concurrencia fueron simuladas desde diferentes computadoras accediendo al mismo sitio donde se encuentra alojado el programa informático en una intranet.

La implementación de este software constituye una de las mejores alternativas para la empresa Omnisoft, porque permite llevar el control y la gestión de las pruebas durante el desarrollo de un producto de software con base a un estándar internacional como es la ISO/IEC 29119.

Esta herramienta desarrollada es fundamental, en especial para el gerente de software o jefe de proyecto para una rápida toma de decisiones y acciones correctivas, a tiempo y que no retrase la planificación del desarrollo.

De esta manera podemos dar seguimiento a los proyectos durante todo el ciclo de vida, por lo cual garantizará que todo software desarrollado este monitoreado y óptimo para la implementación en un ambiente de pre-producción o producción. En consecuencia se mejorará de manera implícita el proceso de gestión de las pruebas dando como resultado una mejor calidad del producto software.

El tiempo para elaborar una cotización de un producto software puede reducirse de manera significativa ya que cuenta con una base histórica de proyectos de software realizados.

La Ingeniería de Software es el análisis, diseño, construcción, verificación y gestión de entidades técnicas. En general, todo proceso de ingeniería debe comenzar por contestar las siguientes preguntas:

¿Cuál es el problema a resolver?,
¿Cuáles son las características de la entidad que se utiliza para resolver el problema?,
¿Cómo se realizará la entidad (y la solución)?,
¿Cómo se construirá la entidad?,
¿Cómo va a probarse la entidad?, y
¿Cómo se apoyará la entidad cuando los usuarios finales soliciten correcciones y adaptaciones a la entidad?

Para los fines que se desarrolla el software propuesto dentro de este proyecto, podemos contestar estas preguntas en una primera instancia desde un punto de vista global y sin considerar detalles específicos, de tal manera que se pueden establecer los siguientes puntos:

- Desarrollar una aplicación de software que permita gestionar y evaluar la calidad del software.

- La aplicación de software debe tener características tales que cumplan con el objetivo del proyecto, es decir, que el software este perfectamente orientado a sus usuarios y que realmente puede ser aplicado al área del problema.

El software se realizará bajo la siguiente premisa:

Herramientas de codificación.

- ✓ Phpnetbeans.
- ✓ starUML.
- ✓ Mysql 5
- ✓ Servidor Apache

Componentes.

- ✓ Greybox.
- ✓ Loginbox.
- ✓ Fpdf.
- ✓ Graphico.
- ✓ Jquery.
- ✓ Motools.

• La aplicación de software deberá ser probada en un intervalo de tiempo adecuado y lo suficientemente amplio como para poder obtener retroalimentación por parte de los usuarios y hacer las correcciones pertinentes.

• El software deberá estar documentado adecuadamente para facilitar futuros procesos tales como expansiones o adaptaciones a nuevas exigencias por parte de los usuarios finales.

Existen diferentes modelos de procesos para la Ingeniería de Software. Cada uno de estos modelos pretende de una manera u otra proporcionar un entorno más ordenado y menos complicado en el proceso de desarrollo de software. Para el caso de esta tesis es necesario apegarse lo más posible a uno de estos modelos con el fin de tener una organización de actividades que se planean en base a una serie de etapas lógicas e interconectadas entre sí.

El modelo de ingeniería de software que esta tesis sigue es el modelo lineal secuencial, que será descrito a continuación.

3.4.-Ciclo de vida (Modelo en cascada).

El modelo lineal secuencial, también conocido como modelo en cascada, se basa en un enfoque sistemático y secuencial del desarrollo del software que comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas, y mantenimiento (**Anexo A**). La siguiente figura ilustra el modelo lineal secuencial para la ingeniería de software.

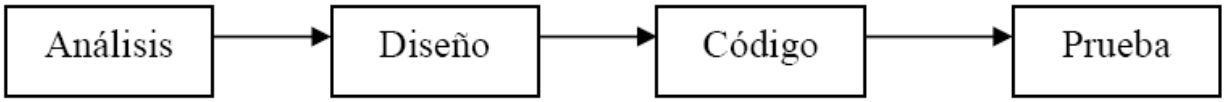


Figura 10. Modelo Lineal Secuencial.

El modelo lineal secuencial contempla seis actividades que deben llevarse a cabo. A continuación se describen estas actividades, y se centra en cada una de ella para los fines de este proyecto:

3.4.1.- Ingeniería y modelado de Sistemas.

El software siempre forma parte de un contexto más grande, que puede ir desde una empresa hasta un sistema. El trabajo comienza estableciendo requisitos de todos los elementos del sistema, y asignando al software algún subgrupo de estos requisitos. En el

caso de la herramienta de software que este proyecto propone, pertenece al contexto de las metodologías de tratamiento y es por esto que tienen que establecerse requerimientos funcionales y no funcionales que permitan que el software desarrollado pueda ubicarse exitosamente dentro de este contexto. En el capítulo 1, Estado del Arte, ya se establecieron algunos requerimientos generales que un ambiente virtual debe contemplar en sus primeras etapas de desarrollo, más no se han especificado los requerimientos particulares para la aplicación de software propuesta dentro de este proyecto. Estos requerimientos serán considerados posteriormente dentro de este capítulo.

3.4.2.-Análisis de los requisitos del software.

El proceso de reunión de requisitos se intensifica y se centra especialmente en el software. Dentro del proceso de análisis es fundamental que a través de una colección de requerimientos funcionales y no funcionales, el desarrollador o desarrolladores del software logren comprender completamente la naturaleza del entorno que deben construirse en la base para desarrollar la aplicación, la función requerida, comportamiento, rendimiento e interconexión. En el caso de este proyecto, el proceso de análisis y de obtención de requerimientos se lleva a cabo a través de trabajar conjuntamente con todas las personas que intervienen en la empresa Omnisoft, quienes proporciona los parámetros bajo los cuales la aplicación debe desarrollarse y de esta manera cumplir con los objetivos planteados.

En el proceso de desarrollo de un sistema, sea o no para la web, el equipo de desarrollo que enfrenta el problema de la identificación de requisitos. La definición de las necesidades del sistema es un proceso complejo, pues en él hay que identificar los requisitos que el sistema debe cumplir para satisfacer las necesidades de los usuarios finales y de los clientes. Para realizar este proceso, no existe una técnica única estandarizada y estructurada que ofrezca

un marco de desarrollo que garantice la calidad del resultado. Existe en cambio un conjunto de técnicas, cuyo uso proponen las diferentes metodologías para el desarrollo de aplicaciones web. Se debe tener en cuenta que la selección de las técnicas y el éxito de los resultados que se obtengan, depende en gran medida tanto del equipo de análisis y desarrollo, como de los propios clientes o usuarios que en ella participen.

El proceso de especificación de requisitos se puede dividir en tres grandes actividades (Lowe& Hall, 1999):

1.- Captura de requisitos

2.- Definición de requisitos

3.- Validación de requisitos.

En la figura se presenta el proceso de ingeniería de requisitos que incluye estas tres actividades. Para la representación se ha usado la notación de diagrama de actividades propuesta en UML.

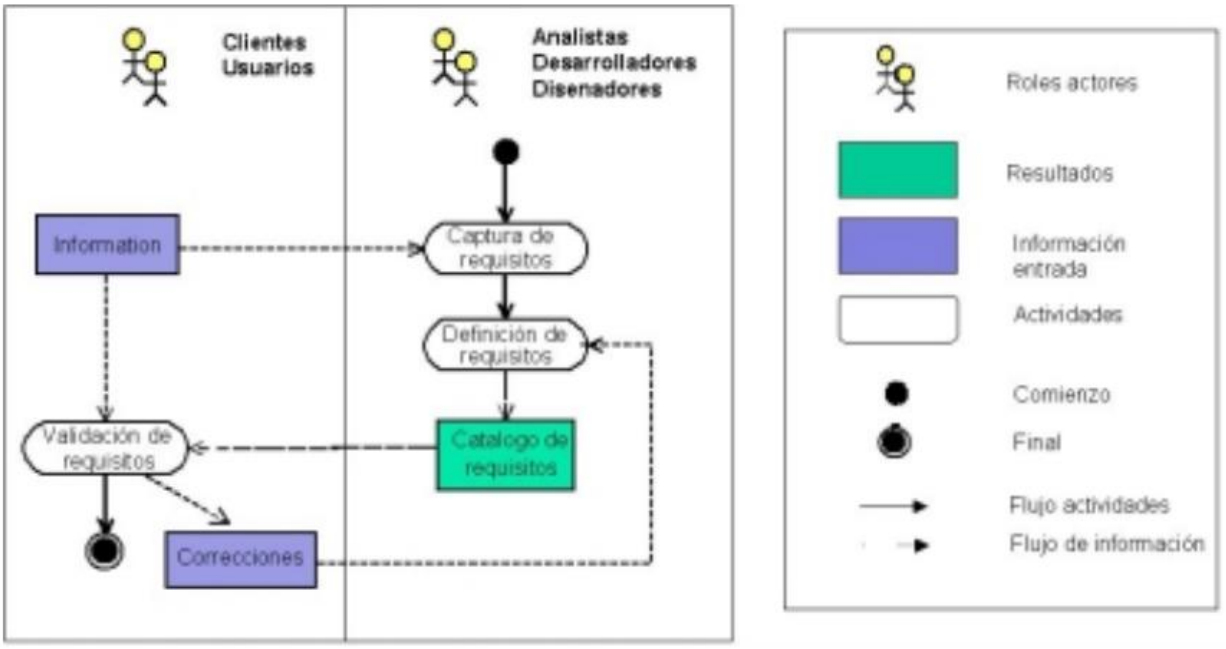


Figura 11.Diagrama de actividades.

El proceso comienza con la realización de la captura de requisitos, el grupo de técnicos toma la información suministrada por los usuarios y clientes. Esta información puede provenir de fuentes muy diversas: documentos, aplicaciones existentes, a través de entrevistas, etc. En base a esta información, el equipo de desarrollo elabora el catálogo de requisitos. Finalmente con la validación de requisitos se realiza la valoración de los mismos, comprobando si existen inconsistencias, errores o si faltan requisitos por definir. El proceso de definición-validación es iterativo y en algunos proyectos complejos resulta necesario ejecutarlo varias veces.

En los siguientes apartados se presentan brevemente algunas técnicas clásicas para realizar las actividades de captura, definición y validación de requisitos. Estas técnicas pueden resultar más o menos apropiadas para la ingeniería de requisitos para aplicaciones en el entorno web. Resulta muy difícil establecer criterios para seleccionar técnicas apropiadas. Entre estos criterios pueden ser considerados la facilidad de aprendizaje y de uso, la estabilidad, el costo, la calidad y completitud de los resultados y el tiempo requerido para aplicar las técnicas. Así podemos decir que el uso de lenguajes naturales produce resultados más imprecisos que una descripción con casos de uso y ésta a su vez es más imprecisa que requisitos descriptos formalmente. Los casos de uso son apropiados tanto para pequeños como grandes sistemas, mientras que el uso de plantillas resulta menos apto para grandes sistemas. Así mismo, técnicas como JAD (Joint Application Design) son más difíciles de usar y consumen mucho más tiempo que entrevistas, permitiendo en cambio obtener resultados de mayor calidad.

3.4.3.-Diseño.

El diseño del software es realmente un proceso de muchos pasos pero que se clasifican dentro de uno mismo. En general, la actividad del diseño se refiere al establecimiento de las estructuras de datos, la arquitectura general del software, representaciones de interfaz y algoritmos. El proceso de diseño traduce requisitos en una representación de software que se describe a continuación.

3.4.3.1.-Analisis.

Diagrama de Casos de Uso

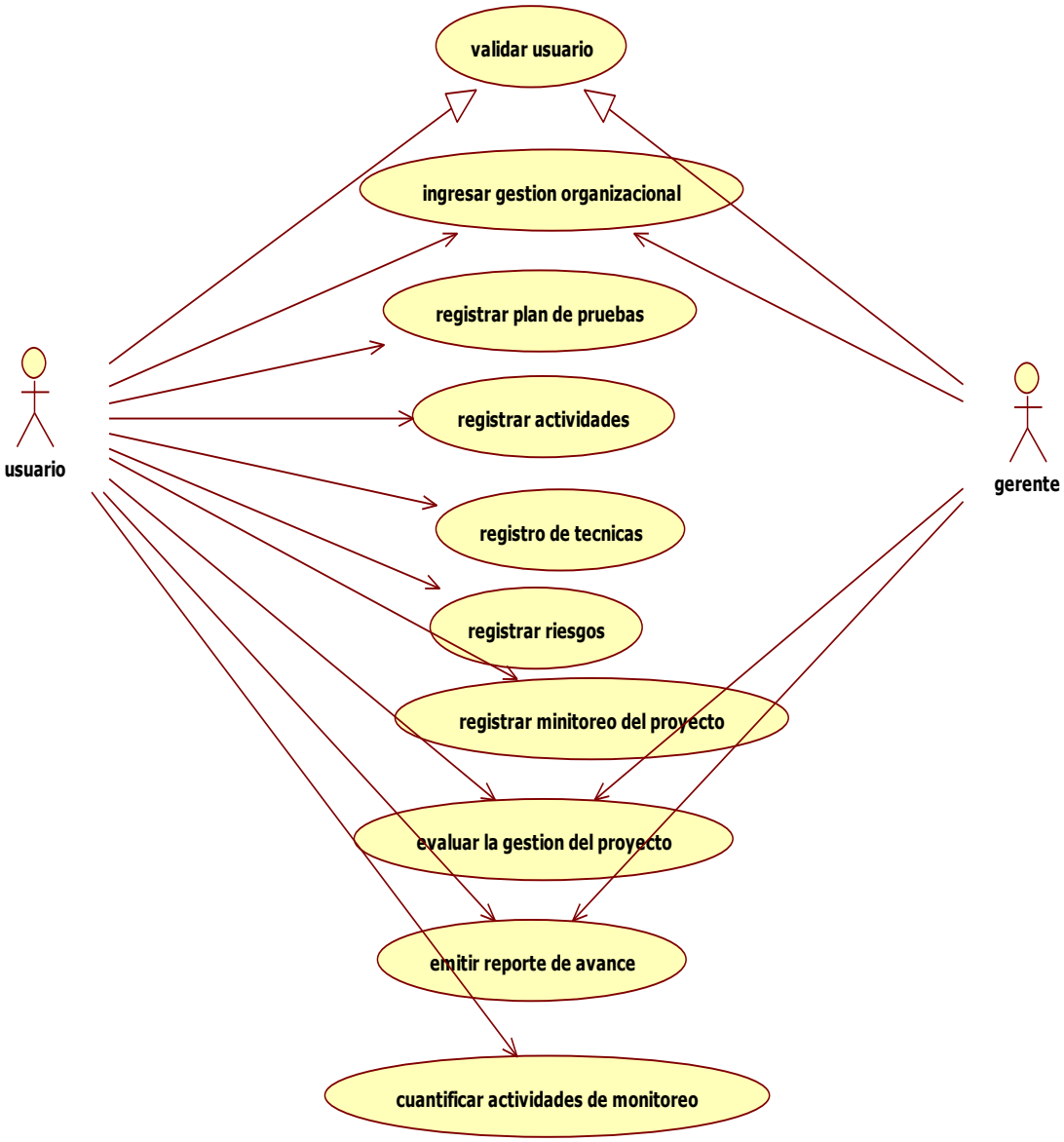


Diagrama de clases

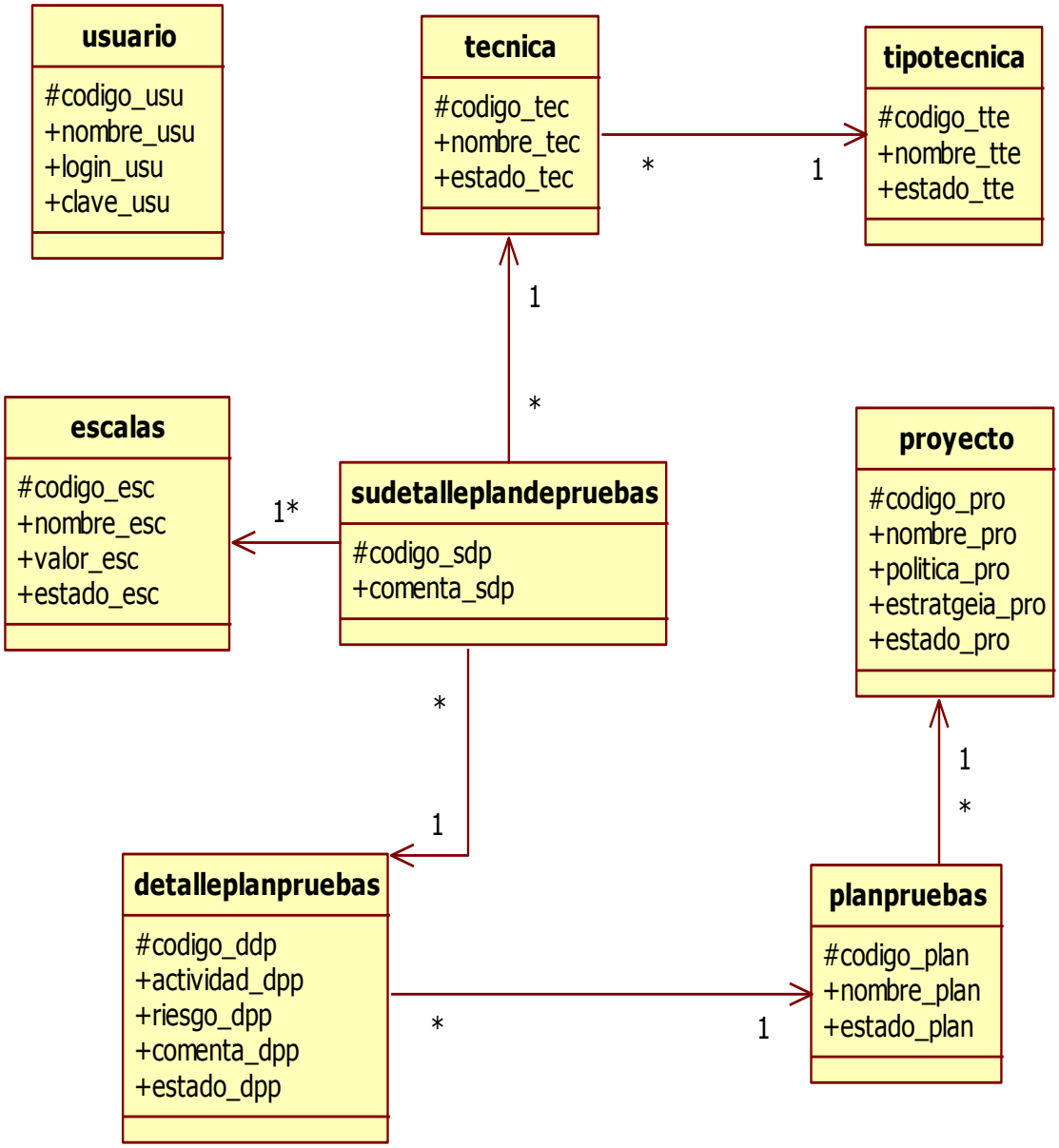

usuario
#codigo_usu
+nombre_usu
+login_usu
+clave_usu
tecnica
#codigo_tec
+nombre_tec
+estado_tec
tipotecnica
#codigo_tte
+nombre_tte
+estado_tte
escalas
#codigo_esc
+nombre_esc
+valor_esc
+estado_esc
sudetalleplandepruebas
#codigo_sdp
+comenta_sdp
proyecto
#codigo_pro
+nombre_pro
+politica_pro
+estratgeia_pro
+estado_pro
detalleplanpruebas
#codigo_ddp
+actividad_dpp
+riesgo_dpp
+comenta_dpp
+estado_dpp
planpruebas
#codigo_plan
+nombre_plan
+estado_plan
*
1
1
*
1*
*
1
*
1
1
*

Diagrama de Secuencia

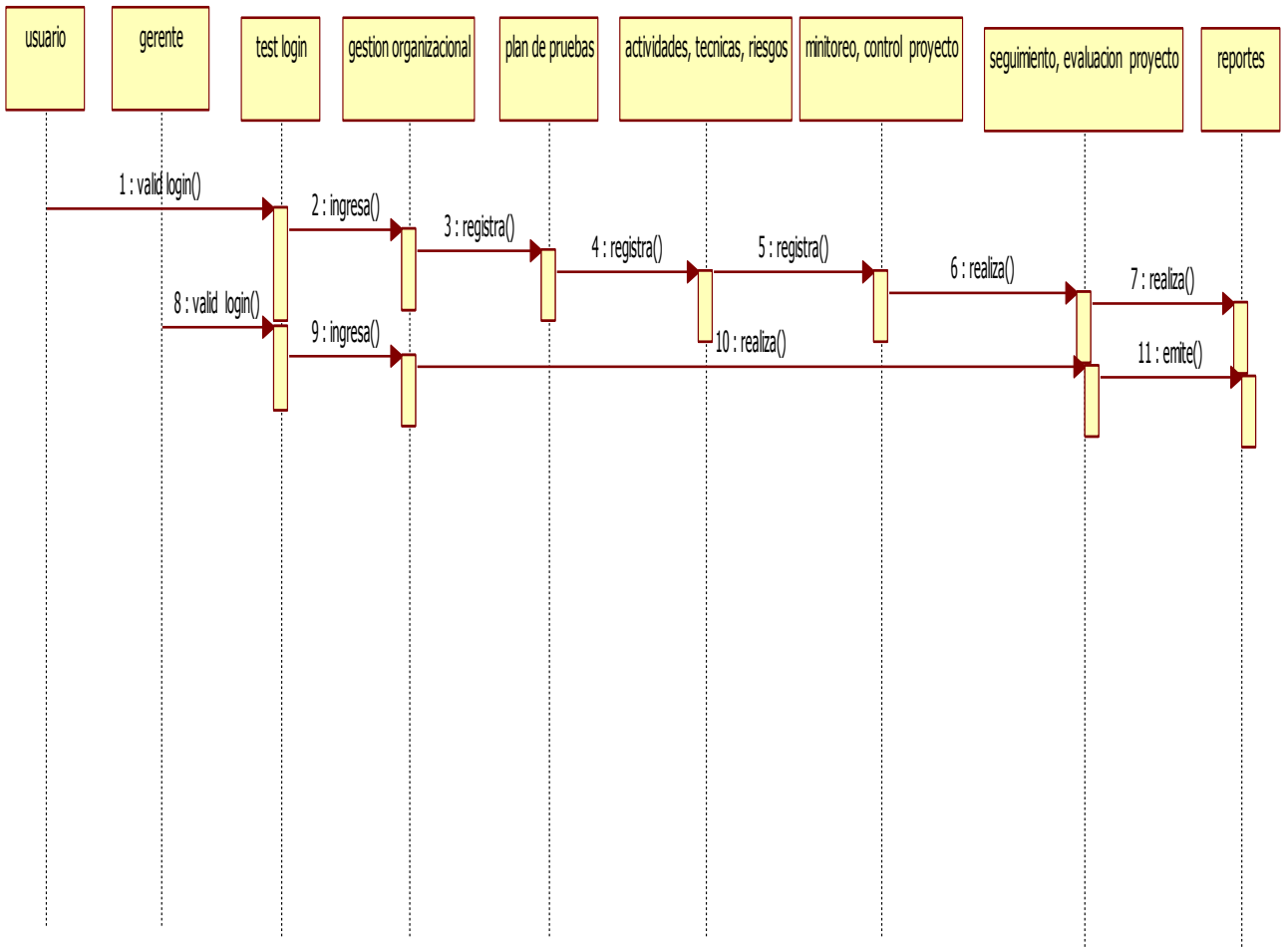


Diagrama de estados

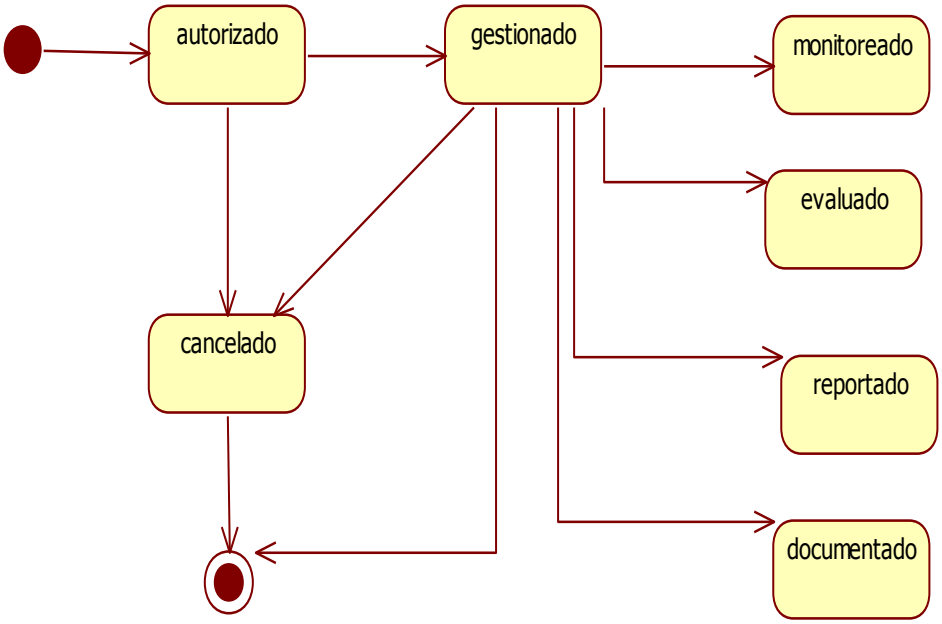


Diagrama de Actividades

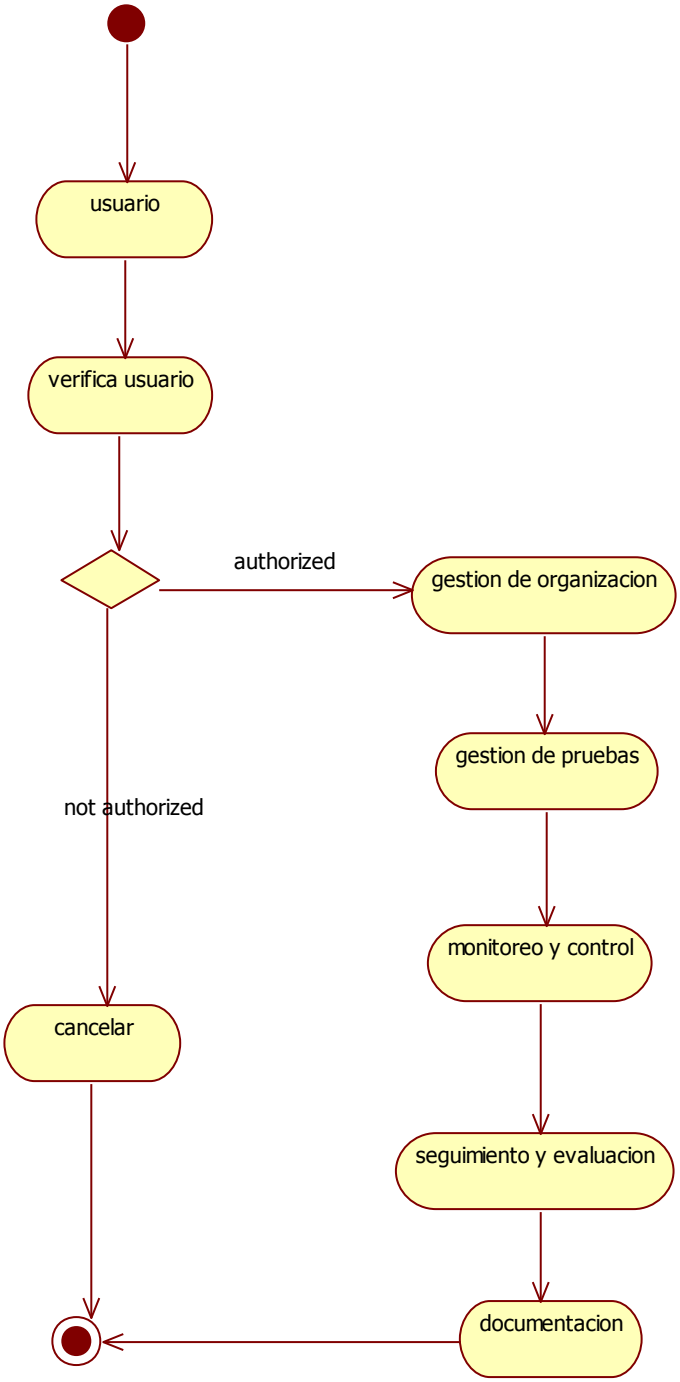
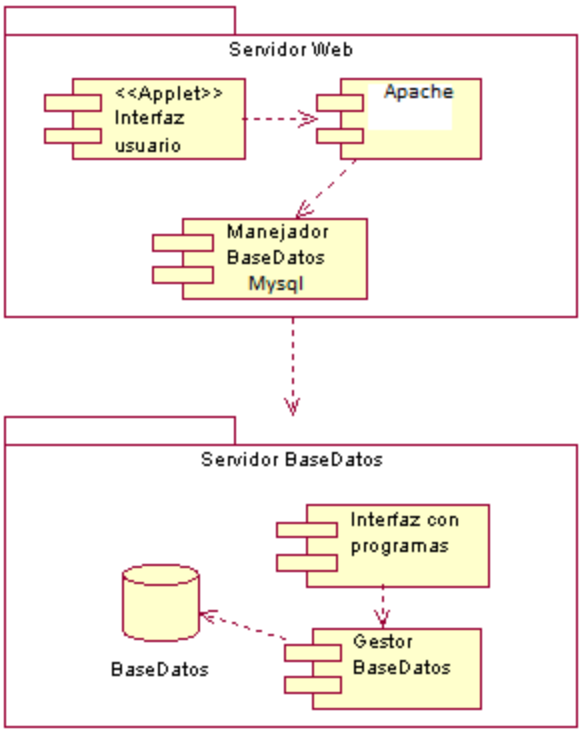
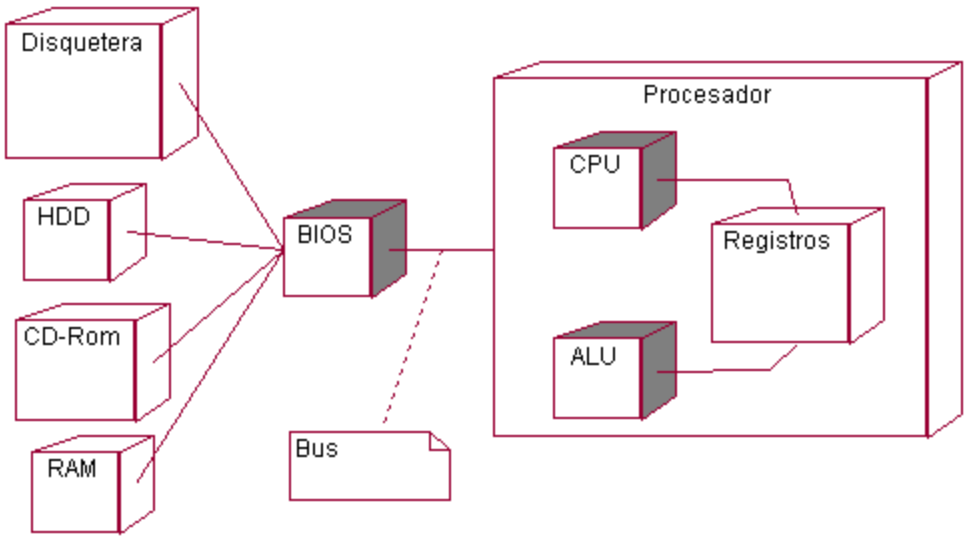


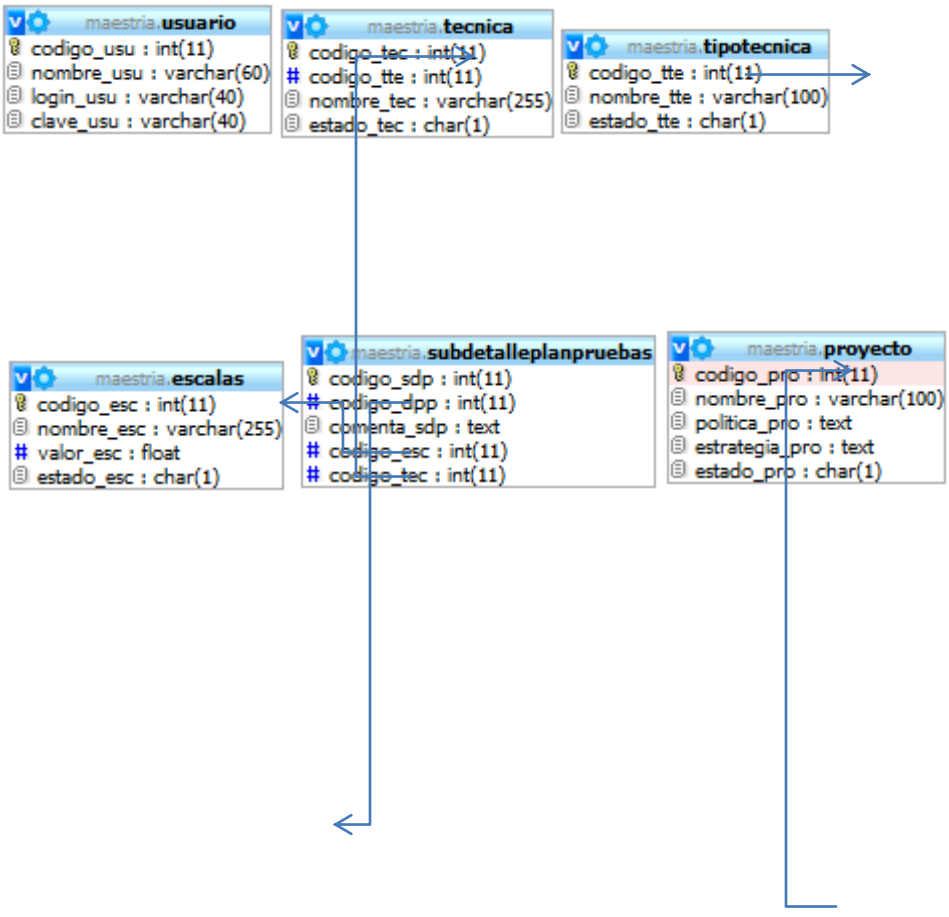
Diagrama de Componentes

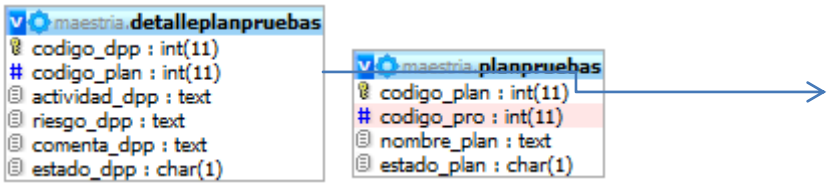


Diagramas de Despliegue

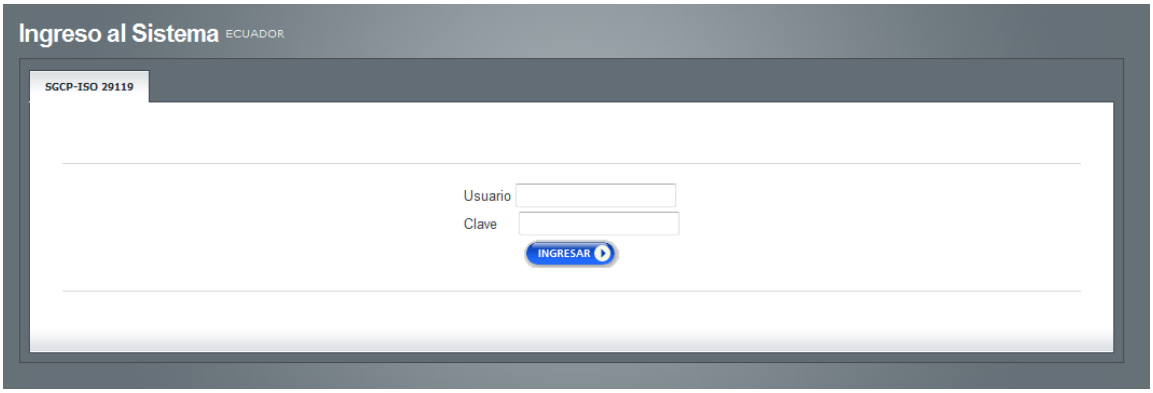


3.4.3.2.-Modelo de la base de Datos.

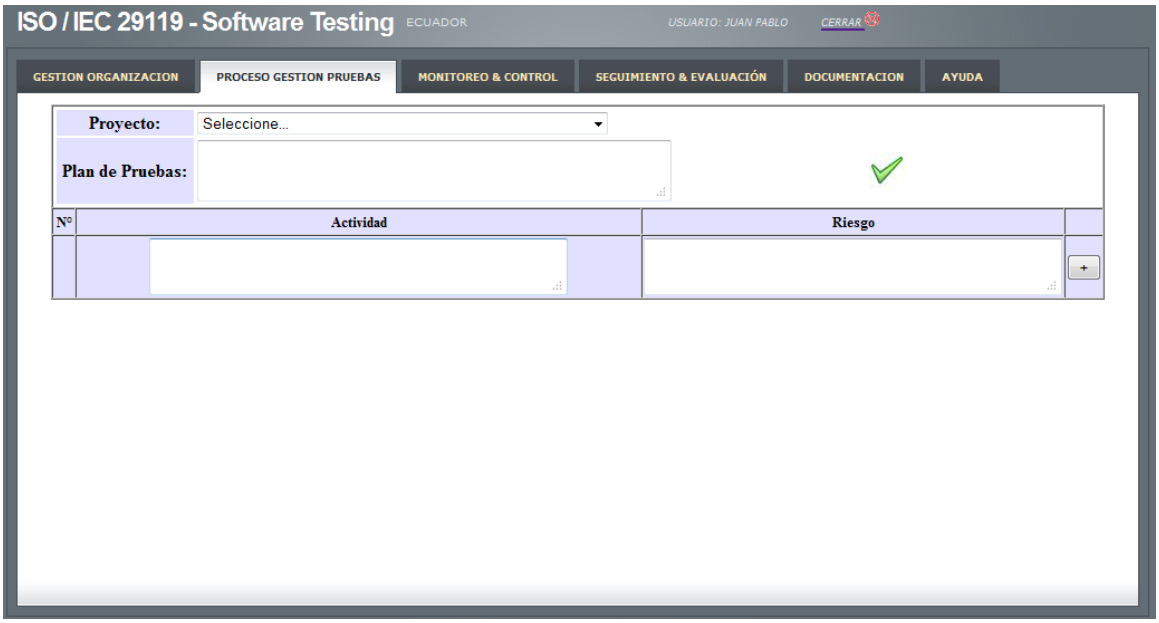




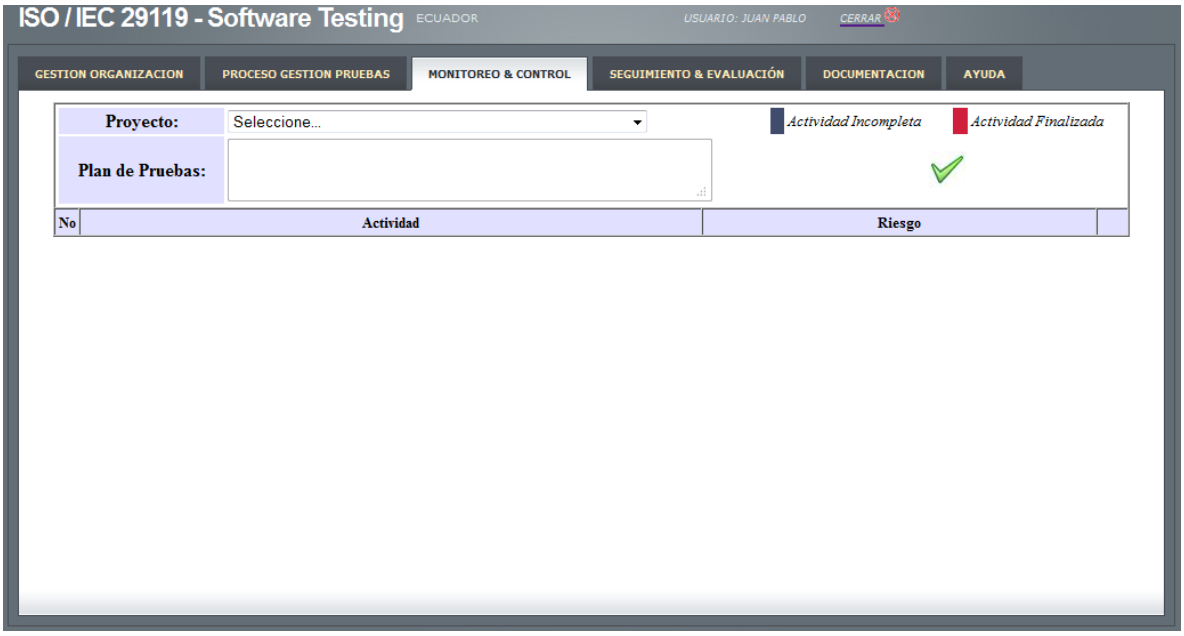
3.4.3.3.-Interfaz de Ingreso del Sistema.



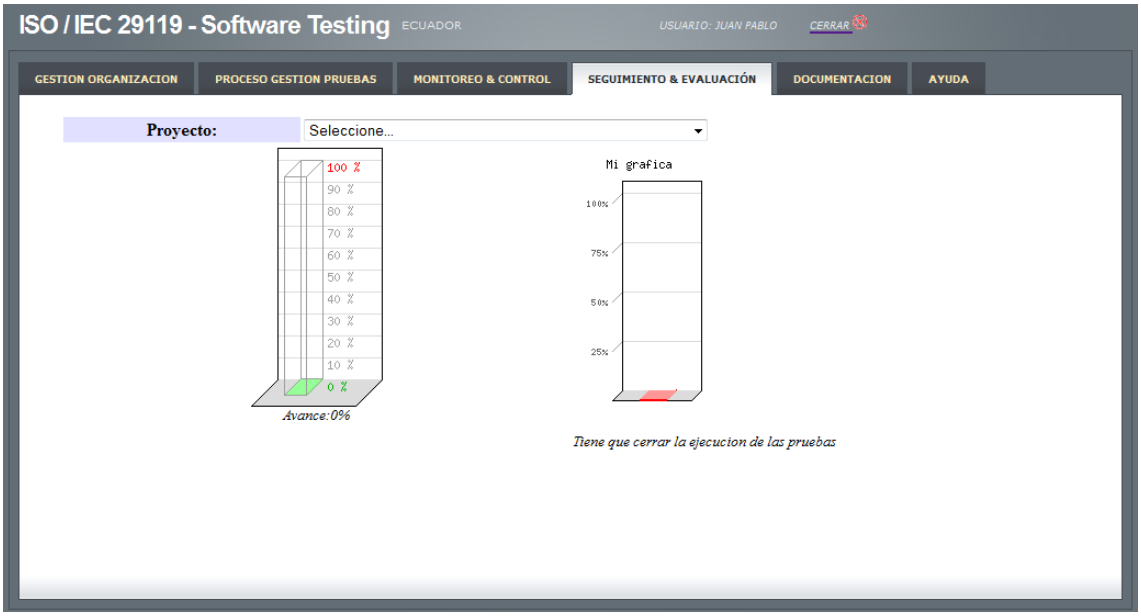
3.4.3.4.-Proceso de Gestión de Pruebas.



3.4.3.5.-Interfaz de Monitoreo y control.



3.4.3.6.-Interfaz de Seguimiento y evaluación.



3.4.4.-Generación de Código.

Esta actividad consiste en traducir el diseño a un lenguaje legible por la máquina. En el caso de la aplicación de este proyecto de software, la generación de código se refiere tanto a la parte de generación de los ambientes virtuales, como parte importante se añadirá elementos interactivos que permitirá observar el comportamiento en entornos especiales. El lenguaje de programación PHP, Java Script, Ajax, starUML son lenguaje de codificación y diseño en 3D mediante los cuales se podrá especificar en porcentaje las características del software que se van agregando al proceso de gestión de pruebas.

Los proyectos en su gran mayoría ayudan a formar el área de pruebas en las organizaciones y/o mejorar su metodología de pruebas. La estrategia que se presenta en este artículo fue concebida inicialmente para las pruebas de testing con el estándar ISO/IEC 29119, donde un equipo o persona es controlado para probar un producto de software dentro de un intervalo de tiempo definido. Dicha estrategia luego también ha sido utilizada en consultorías, donde las organizaciones adoptan esta estrategia para probar sus propios productos, la estrategia de gestión se adapta muy bien a este contexto. Las áreas de prueba internas de las organizaciones pueden utilizar esta estrategia para planificar sus pruebas de testing y mantener las actividades de pruebas controladas.

3.4.5.-Pruebas.

Para muchas de las actividades que lleva a cabo el ser humano, la prueba es una actividad necesaria que permite determinar la calidad de estas. Hoy por ejemplo, se realizan pruebas para determinar resistencia de determinados materiales, para determinar la vida útil de

diversas maquinarias y equipos, o bien para verificar que ciertas tareas se realicen de forma correcta.

Definitivamente la prueba es una actividad fundamental en muchos procesos de desarrollo, incluyendo el del software. De manera general, se puede decir que la prueba de software permite al desarrollador determinar si el producto generado satisface las especificaciones establecidas. Así mismo, una prueba de software permite detectar la presencia de errores que pudieran generar salidas o comportamientos inapropiados durante su ejecución.

De acuerdo a la **IEEE [IEEE90]** el concepto de prueba (testing) 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".

Cuando se habla de condiciones específicas en la definición anterior, se puede suponer la presencia de una especie de ambiente de operación de la prueba, para el cual deben existir determinados valores para las entradas y las salidas, así como también ciertas condiciones que delimitan a dicho ambiente de operación. Formalmente esto es conocido como caso de prueba.

La IEEE [IEEE90] define un caso de prueba como: "Un conjunto de entradas, condiciones de ejecución y resultados esperados diseñados para un objetivo particular".

Una vez que se ha generado código, comienzan las pruebas del software que se ha desarrollado. El proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, y en los procesos externos

funcionales, es decir, la realización de las prueba para la detección de errores. En el caso de una herramienta de software es necesario tener etapas de pruebas tanto para la parte funcional del software, como para la parte aplicativa del mismo. Se requiere probar el software con sujetos reales que puedan evaluar el comportamiento del software con el fin de proporcionar una retroalimentación a los desarrolladores. Es sumamente importante que durante el proceso de desarrollo no se pierda el contacto con los interesados o solicitantes del desarrollo de software, de esta manera los objetivos de proyecto se mantendrán vigentes y se tendrá una idea clara de los aspectos que tienen que probarse durante el periodo de pruebas.

Términos como falla, equivocación y error, pueden considerarse como sinónimos, sin embargo, dentro del contexto de prueba de software no es prudente realizar esta suposición.

Con el propósito evitar confusiones y presentar al lector conceptos básicos en materia de pruebas, se presentan estas definiciones tomadas de (IEEE90):

a) Equivocación (mistake):

Acción del ser humano que produce un resultado incorrecto.

b) Defecto o falta (fault):

Un paso, proceso o definición de dato incorrecto en un programa de computadora. El resultado de una equivocación.

c) Falla (failure):

Resultado incorrecto. El resultado de una falla.

d) Error (error):

Magnitud por la que el resultado es incorrecto.

Las pruebas de unidad se concentran en verificar si las funcionalidades descritas en las especificaciones o en los requisitos iniciales corresponden a las que se presentan en el producto final. En esta área, al igual que la de pruebas de integración, se han generado pocos trabajos, por lo que se emplean muchos de los métodos tradicionales.

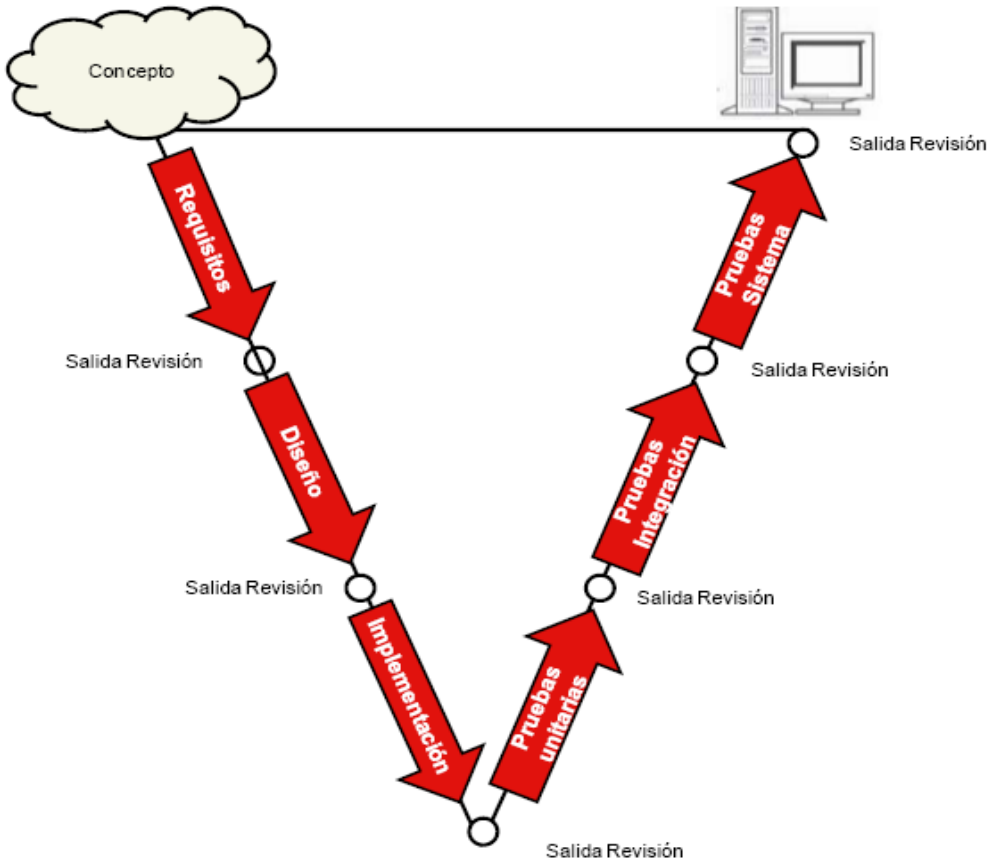


Figura 12.Revisiones durante el desarrollo de un producto.

3.4.5.1.-Prueba de función.

La prueba de función es llevada a cabo por el grupo de personas que desarrollaron el producto. Este enfoque se orienta a confirmar que la aplicación alcanza los requerimientos y la funcionalidad especificadas por el usuario.

3.4.5.2.-Pruebas de aceptación (beta).

En este tipo de pruebas, versiones que aún no han sido liberadas en el mercado, son ofrecidas a ciertos grupos de usuarios con el propósito de que las utilicen. El propósito de esto es que los usuarios reporten defectos que pudieran presentarse.

3.4.5.3.-Prueba bajo stress.

Para realizar esta prueba, el sistema somete a condiciones extremas de trabajo, como pueden ser un alto volumen de transacciones o un gran número de usuarios. Aplicando este enfoque, se puede verificar si el sistema se comporta como se espera aún ante este tipo de escenarios.

3.4.5.4.-Pruebas de caja blanca.

Las pruebas de caja blanca enfocan su atención a los detalles procedimentales del software, por ello la implementación de estas pruebas depende fuertemente de la disponibilidad de código fuente. Este tipo de pruebas, permiten generar casos para ejercitar y validar los caminos de cada módulo, las condiciones lógicas, los bucles y sus límites, así como también para las estructuras de datos. Las pruebas de caja blanca también son conocidas como pruebas de caja de cristal o pruebas estructurales.

3.4.5.5.-Pruebas de caja negra.

Este tipo de pruebas, conocidas también como pruebas funcionales o pruebas de comportamiento, concentran la atención en generar casos de prueba que permitan ejercitar los requisitos funcionales de un programa. A diferencia de las pruebas de caja blanca, que se basan en la lógica interna del software, este tipo de pruebas se concentran en su funcionalidad, por lo que mucho del trabajo se realiza interactuando con la interfaz del software. Los casos de prueba generados en este enfoque, se diseñan a partir de valores entrada y salida. De esta forma, se puede determinar la validez de una salida para un conjunto de entradas proporcionadas.

La aplicación de pruebas de caja negra permite detectar errores como funciones incorrectas o ausentes, errores en estructuras de datos, errores de rendimiento, así como errores de inicialización y terminación.

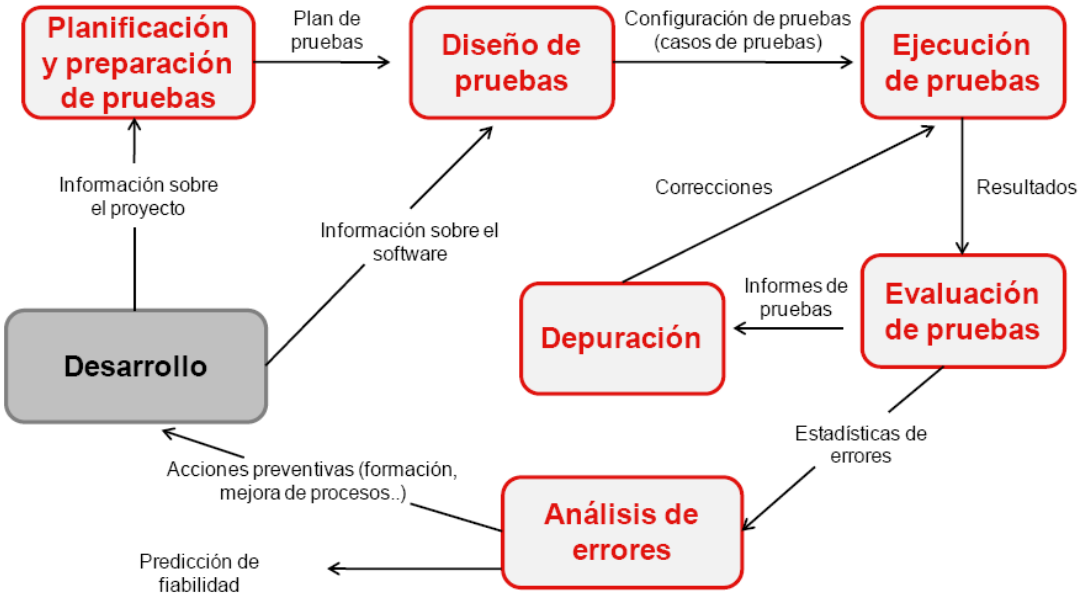


Figura 13.Flujo del Proceso de pruebas

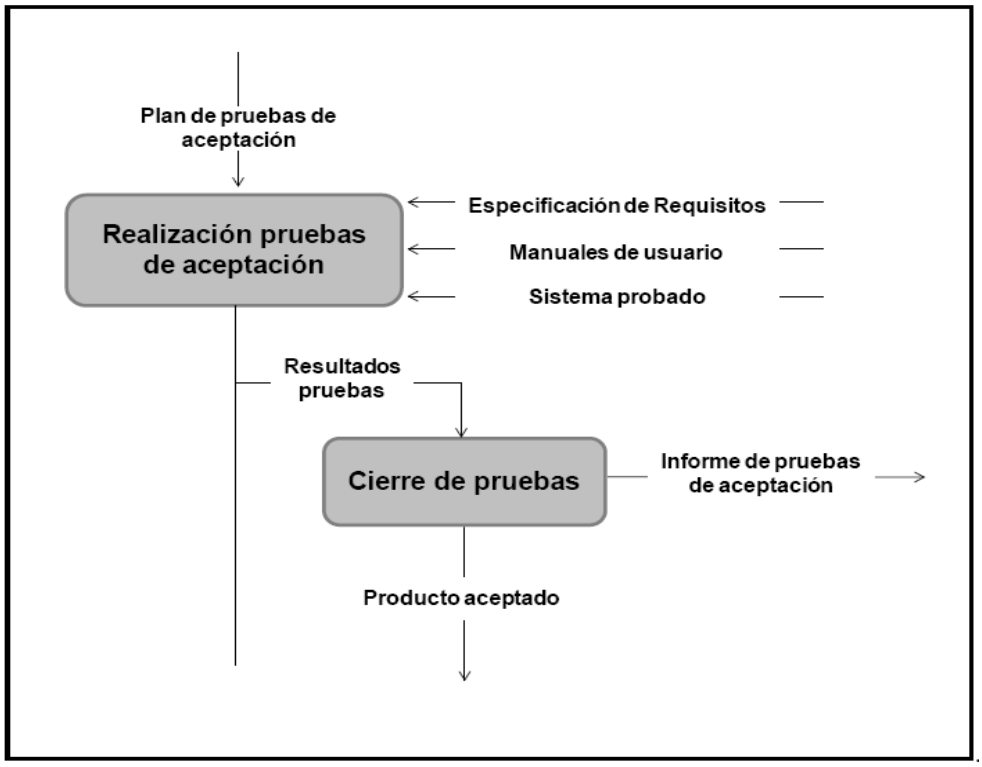
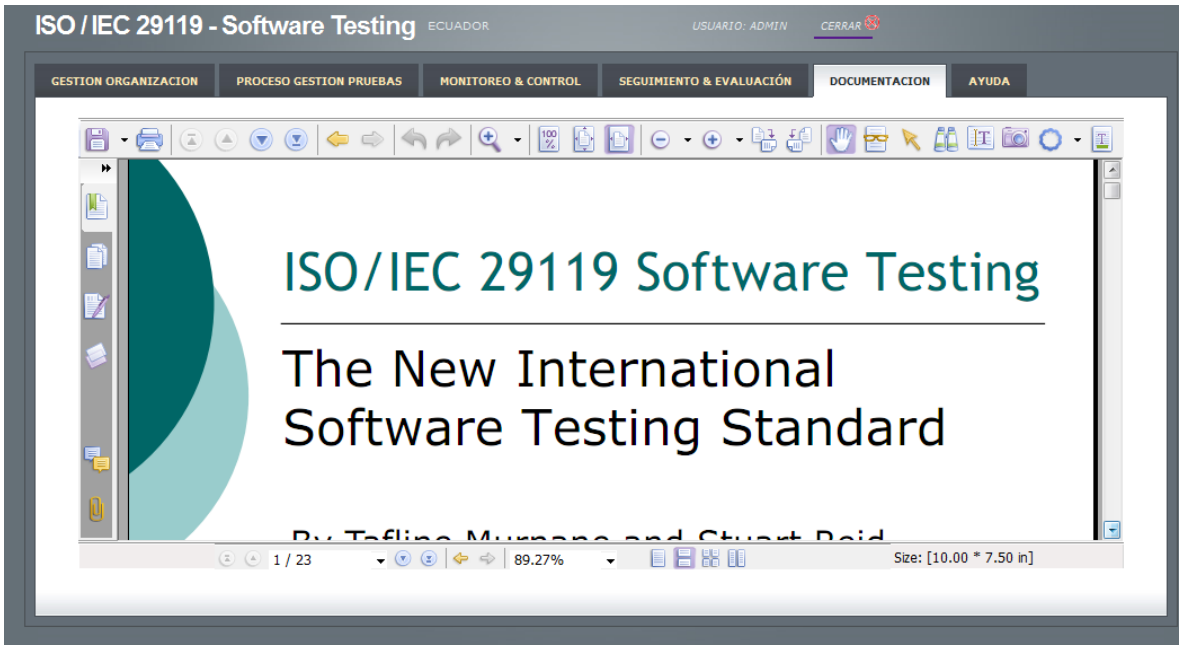


Figura14. Flujo de control pruebas de aceptación.

3.4.6.-Mantenimiento.

El software indudablemente sufrirá cambios, y habrá que hacer algunas modificaciones a su funcionalidad. Es de suma importancia que el software de calidad pueda adaptarse con fines de acoplarse a los cambios de su entorno externo. Una de las secciones posteriores de este documento se refiere específicamente a posibles expansiones de este proyecto, y por medio de la documentación apropiada y atinada del software se pueden presentar las vías para el mantenimiento y modificaciones al mismo. En este capítulo, se describe paso a paso la manera en que se elaboran las escenas virtuales propuestas por este proyecto, de tal manera que este mismo documento puede utilizarse posteriormente como referencia para nuevos proyectos de desarrollo de aplicaciones de software y que sigan la metodología de trabajo que se propone.



En cada una de las etapas de un modelo de ciclo de vida, se pueden establecer una serie de objetivos, tareas y actividades que lo caracterizan. Haremos un repaso y una pequeña descripción de cada una de las etapas del ciclo de vida del software; una vez conocidas las etapas, tendremos que analizar cómo abordarlas en su conjunto. Existen distintos modelos de ciclo de vida, y la elección de un modelo para un determinado tipo de proyecto es realmente importante; el orden de las etapas es uno de estos puntos importantes, Si elegimos el modelo de cascada el cual la validación se realiza al final del proyecto, y luego debemos retomar etapas previas.

- ✓ **Expresión de necesidades:** esta etapa tiene como objetivo el armado de un documento en el cual se reflejan los requerimientos y funcionalidades que ofrecerá al usuario el sistema a implementar (qué, y no cómo, se va a implementar).

- ✓ **Especificaciones:** formalizamos los requerimientos; el documento obtenido en la etapa anterior se tomará como punto de partida para esta etapa.

- ✓ **Análisis:** determinamos los elementos que intervienen en el sistema a desarrollar, su estructura, relaciones, evolución temporal, funcionalidades, tendremos una descripción clara de qué producto vamos a construir, qué funcionalidades aportará y qué comportamiento tendrá.

- ✓ **Diseño:** ya sabemos qué hacer, ahora tenemos que determinar cómo debemos hacerlo (¿cómo debe ser construido el sistema en cuestión?; definimos en detalle

entidades y relaciones de las bases de datos, seleccionamos el lenguaje que vamos a utilizar, el Sistema Gestor de Bases de Datos, etc.).

- ✓ **Implementación:** empezamos a codificar algoritmos y estructuras de datos, definidos en las etapas anteriores, en el correspondiente lenguaje de programación o para un determinado sistema gestor de bases de datos. En muchos proyectos se pasa directamente a esta etapa; son proyectos muy arriesgados que adoptan un modelo de ciclo de vida de code&fix (codificar y corregir) donde se eliminan las etapas de especificaciones, análisis y diseño con la consiguiente pérdida de control sobre la gestión del proyecto.

- ✓ **Debugging:** el objetivo de esta etapa es garantizar que nuestro programa no contiene errores de diseño o codificación. En esta etapa no deseamos saber si nuestro programa realiza lo que solicitó el usuario, esa tarea le corresponde a la etapa de implementación. En ésta deseamos encontrar la mayor cantidad de errores. Todos los programas contienen errores: encontrarlos es cuestión de tiempo. Lo ideal es encontrar la mayoría, si no todos, en esta etapa. También se pueden agregar testeos de performance.

- ✓ **Validación:** esta etapa tiene como objetivo la verificación de que el sistema desarrollado cumple con los requerimientos expresados inicialmente por el cliente y que han dado lugar al presente proyecto. En muchos proyectos las etapas de validación y debugging se realizan en paralelo por la estrecha relación que llevan. Sin embargo, tenemos que evitar la confusión: podemos realizarlos en paralelo, pero no como una única etapa.

- ✓ **Evolución:** en la mayoría de los proyectos se considera esta etapa como mantenimiento y evolución, y se le asigna, no sólo el agregado de nuevas funcionalidades (evolución); sino la corrección de errores que surgen (mantenimiento). En la práctica esta denominación no es del todo errónea, ya que es

posible que aun luego de una etapa de debugging y validación exhaustiva, se filtren errores.

3.5.-Metodología para el desarrollo de Software.

La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito. Esta sistematización nos indica cómo dividiremos un gran proyecto en módulos más pequeños llamados etapas, y las acciones que corresponden en cada una de ellas, nos ayuda a definir entradas y salidas para cada una de las etapas y, sobretodo, normaliza el modo en que administraremos el proyecto. Entonces, una metodología para el desarrollo de software son los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software desde que surge la necesidad del producto hasta que cumplimos el objetivo por el cual fue creado.

Desde un punto de vista general puede considerarse que el ciclo de vida de un software tiene tres etapas claramente diferenciadas, las cuales se detallan a continuación:

- **Planificación:** idearemos un planeamiento detallado que guíe la gestión del proyecto, temporal y económicamente.

- **Implementación:** acordaremos el conjunto de actividades que componen la realización del producto.

- **Puesta en producción:** nuestro proyecto entra en la etapa de definición, allí donde se lo presentamos al cliente o usuario final, sabiendo que funciona correctamente y responde a los requerimientos solicitados en su momento. Esta etapa es muy importante no sólo por representar la aceptación o no del proyecto por parte del cliente o usuario final sino por las múltiples dificultades que suele presentar en la práctica, alargándose excesivamente y provocando costos no previstos.

A estas tres grandes etapas es conveniente añadir otras dos que, si bien pudieron enunciarse junto a las otras, es conveniente hacer una diferenciación ya que se tiende a no darles la importancia que requieren.

Para asegurar el éxito durante el desarrollo de software no es suficiente contar con notaciones de modelado y herramientas, hace falta un elemento importante: la metodología de desarrollo, la cual nos provee de una dirección a seguir para la correcta aplicación de los demás elementos.

Generalmente el proceso de desarrollo llevaba asociado un marcado énfasis en el control y la definición de roles, actividades, artefactos incluyendo el modelado y la documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos) y poco reajustables, donde por lo general se exige un alto grado de mesura en el proceso. Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

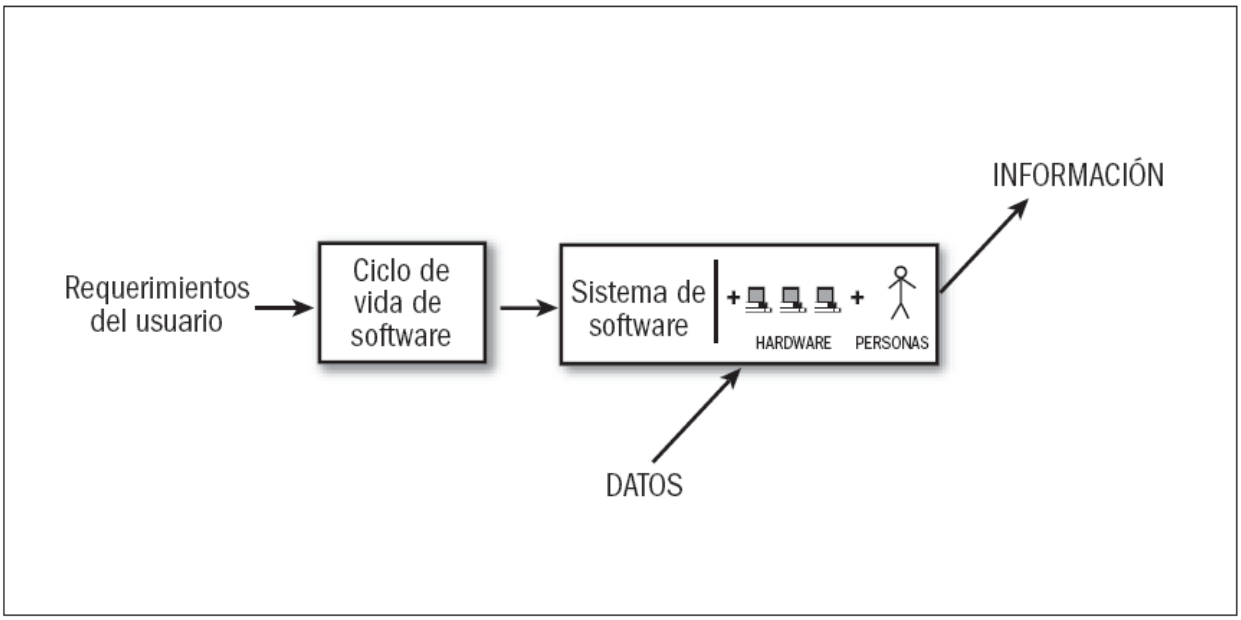


Figura 15. El ciclo de vida de un producto software se desarrolla fuera del ámbito productivo, aunque debemos conocer el entorno (environment) en el que será ejecutado.

- ✓ **Inicio:** éste es el nacimiento de la idea. Aquí definimos los objetivos del proyecto y los recursos necesarios para su ejecución. Hacia dónde queremos ir, y no cómo queremos ir. Las características implícitas o explícitas de cada proyecto hacen necesaria una etapa previa destinada a obtener el objetivo por el cual se escribirán miles o cientos de miles de líneas de código. Un alto porcentaje del éxito de nuestro proyecto se definirá en estas etapas que, al igual que la etapa de debugging, muchos líderes de proyecto subestiman.

- ✓ **Control en producción**: control del producto, analizando cómo el proceso difiere o no de los requerimientos originales e iniciando las acciones correctivas si fuesen necesarias. Cuando decimos que hay que corregir el producto, hacemos referencia a

pequeñas desviaciones de los requerimientos originales que puedan llegar a surgir en el ambiente productivo. Si nuestro programa no realiza la tarea para lo cual fue creada, esta etapa no es la adecuada para el rediseño. Incluimos también en esta etapa el liderazgo, documentación y capacitación, proporcionando directivas a los recursos humanos, para que hagan su trabajo en forma correcta y efectiva.

Lo que buscamos guiándonos con una metodología es prolijidad, corrección y control en cada etapa del desarrollo de un programa, lo que nos permitirá una forma sistemática de poder obtener un producto correcto y libre de errores.

3.5.1.-Metodología RAD.

"Aproximación al desarrollo de sistemas que incorpora una variedad de herramientas de diseño automatizadas. Desarrollada por el 'gurú' de la industria, James Martin, está centrada tanto en la administración humana y en la participación del usuario, como en la tecnología". (Freedman, 1993).

"Metodología para el desarrollo de sistemas, fue creada para disminuir radicalmente el tiempo necesario para diseñar e implementar Sistemas de Información. El RAD cuenta con una participación intensa del usuario, sesiones JAD, prototipaje, herramientas CSE integradas y generadores de código".(Valacich et al., 2001)

Principios básicos:

- Objetivo clave es para un rápido desarrollo permitiendo una alta calidad en el software con un costo relativamente bajo en inversión.
- Intenta reducir los riesgos inherentes del proyecto partiéndolo en segmentos más pequeños y proporcionar más facilidad de cambio durante el proceso de desarrollo.
- Orientación dedicada a producir sistemas de alta calidad con rapidez, mediante el uso de iteración por prototipos (en cualquier etapa de desarrollo), promueve la participación de los usuarios y el uso de herramientas de desarrollo computarizadas. Estas herramientas pueden incluir constructores de Interfaz gráfica de usuario (GUI), Computer Aided Software Engineering (CASE), los sistemas de gestión de bases de datos (DBMS), lenguajes de programación de cuarta generación, generadores de código, y técnicas orientada a objetos.
- Hace especial hincapié en el cumplimiento de la necesidad comercial, mientras que la parte de ingeniería software es de menor importancia.
- Control de proyecto implica el desarrollo de las prioridades y la definición de los plazos de entrega. Si el proyecto empieza a aplazarse, se hace hincapié en la reducción de requisitos para el ajuste, no en el aumento de la fecha límite.
- En general incluye Joint application development (JAD), donde los usuarios están intensamente participando en el diseño del sistema, ya sea a través de la creación de consenso estructurado en talleres, o por vía electrónica.
- La participación activa de los usuarios es imprescindible.
- Iterativamente se realiza la producción de software, en lugar de enfocarse en un prototipo.
- Produce la documentación necesaria para facilitar el futuro desarrollo y mantenimiento.

El RAD requiere de cuatro (4) faces esenciales: gerencia, gente, metodologías y herramientas.

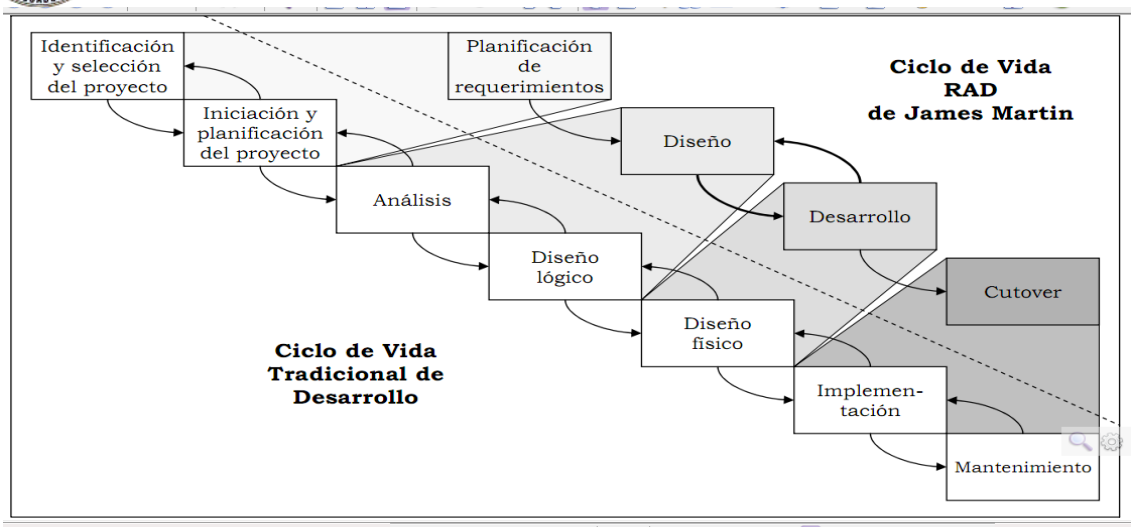


Figura 16. Ciclo de vida RAD.

Muchas firmas han adoptado el RAD como uno de sus enfoques para el desarrollo de sistemas. Entre éstas, esta Cambridge Technology Partners (CTP), especificando un ciclo de vida para el RAD que consiste en cinco (5) pasos:

1) Enfoque
2) Taller de soluciones rápidas.
3) Diseño.
4) Desarrollo.
5) Rollout

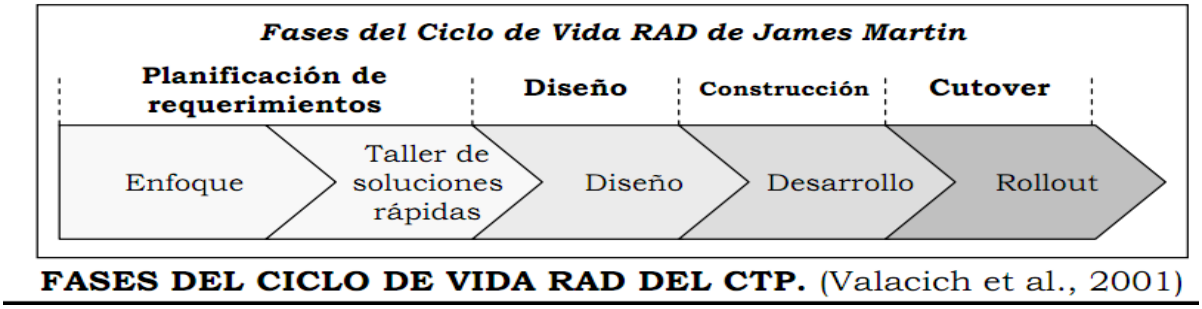


Figura 17.Fases del ciclo de vida RAD.

Ventajas:

- Enfatiza ciclos de desarrollo extremadamente cortos.
- Tiene las ventajas del modelo clásico.
- Se asegura de que el producto entregado cumple las necesidades del cliente
- Ahorro dramático de tiempo durante el desarrollo del sistema.
- Puede ahorrarse tiempo, dinero y esfuerzo humano.
- Estrecha correspondencia entre los requerimientos del usuario y las especificaciones del sistema.
- Trabaja muy bien cuando la velocidad de desarrollo es importante (cambios rápidos de las condiciones del negocio), o cuando los sistemas pueden capitalizarse en oportunidades estratégicas.
- Permite cambiar rápidamente el diseño de los sistemas cuando los usuarios lo demandan.
- Los sistemas son optimizados por los usuarios involucrados en el proceso del RAD.
- Se concentra en los elementos esenciales del sistema, desde el punto de vista del usuario.
- El usuario se compromete y se hace propietario del sistema.

Desventajas:

- Solo se puede aplicar si el sistema se puede modularizar de forma que permita completarse cada una de las funciones principales en menos de tres meses.
- Para proyectos grandes puede requerir muchos equipos de trabajo distintos.
- Requiere clientes y desarrolladores comprometidos en las rápidas actividades necesarias.
- No resulta adecuado cuando los riesgos técnicos son elevados.
- Se pueden tener problemas con la aceptación del prototipo.

3.5.1.1.- Planificación de requerimientos (RAD).

Para que el desarrollo de un proyecto de software concluya con éxito, es de suma importancia que antes de empezar a codificar los programas que constituirán la aplicación de software completa, se tenga una completa y plena comprensión de los requisitos del software.

Esta etapa requiere que usuarios con un vasto conocimiento de los procesos de la compañía determinen cuáles serán las funciones del sistema. Debe darse una discusión estructurada sobre los problemas de la compañía que necesitan solución. Por lo general, esta etapa se completa rápidamente cuando se crean equipos que envuelven usuarios y ejecutivos con un conocimiento amplio sobre las necesidades de la institución la planificación de los requisitos se da en modalidad de taller conocido como Junta de Planificación de Requisitos (JRP por sus siglas en inglés).

La tarea del análisis de requisitos es un proceso de descubrimiento, refinamiento, modelado y especificación. Se refina en detalle el ámbito del software, y se crean modelos de los requisitos de datos, flujo de información y control, y del comportamiento operativo. Se analizan soluciones alternativas y se asignan a diferentes elementos del software. El análisis de requisitos permite al desarrollador o desarrolladores especificar la función y el rendimiento del software, indica la interfaz del software con otros elementos del sistema y establece las restricciones que debe cumplir el software.

3.5.1.2.-Diseño (RAD).

El diseño del software desarrolla un modelo de instrumentación o implantación basado en los modelos conceptuales desarrollados durante el análisis del sistema. Implica diseñar la decisión sobre la distribución de datos y procesos

El diseño es la primera de las tres actividades técnicas que implica un proceso de ingeniería de software; estas etapas son diseño, codificación (en el caso de este proyecto Desarrollo e Implementación) y pruebas. Generalmente la fase de diseño produce un diseño de datos, un diseño arquitectónico, un diseño de interfaz, y un diseño procedimental.

Esta consiste de un análisis detallado de las actividades de la compañía en relación al sistema propuesto. Los usuarios participan activamente en talleres bajo la tutela de profesionales de la informática. En ellos descomponen funciones y definen entidades asociadas con el sistema. Una vez se completa el análisis se crean los diagramas que definen las alteraciones entre los procesos y la data. Al finalizar el análisis se traza el diseño del sistema. Se desarrollan los procedimientos y los esquemas de pantallas. Los prototipos de procedimientos críticos se construyen y se repasan en un plan de implementación del sistema.

3.5.1.3.-Construccion (Desarrollo)-(RAD).

En la etapa de construcción el equipo de desarrolladores, trabajando de cerca con los usuarios, finaliza el diseño y la construcción del sistema. La construcción de la aplicación consiste de una serie de pasos donde los usuarios tienen la oportunidad de afirmar los requisitos y repasar los resultados. Las pruebas al sistema se llevan a cabo durante esta etapa. También se crea la documentación y las instrucciones necesarias para manejar la nueva aplicación, rutinas y procedimientos para operar el sistema.

3.5.1.4.-Cutover (Rollout).

Esta etapa envuelve la implementación del nuevo producto y el manejo del cambio del viejo al nuevo sistema. Se hacen pruebas comprensivas y se adiestran los usuarios. Los cambios organizacionales y la operación del nuevo sistema se hacen en paralelo con el viejo sistema hasta que el nuevo se establezca completamente.

3.6.-Ciclo de prueba.

Durante el ciclo de vida de un producto, sin importar cuál sea el proceso de desarrollo, se van generando distintas versiones de la aplicación. Las actividades de la prueba se realizan para una determinada versión del producto, sobre la cual se ejecutan las pruebas y se reportan los incidentes encontrados. Las pruebas que serán ejecutadas sobre una versión son planificadas con anticipación y deberían ser ejecutadas, a menos que las prioridades cambien.

En un ciclo de prueba se puede ejecutar una, alguna o todas las pruebas planificadas para el producto. Uno de los principales desafíos desde el punto de vista de la prueba independiente es estimar cuantos ciclos de prueba se requieren, ya que no todas las versiones que genera desarrollo llegan a ser aprobadas por el equipo de prueba, entre dos ciclos de prueba podrían existir más de dos versiones del producto generadas por el desarrollador.

3.7.-Casos de estudio: experiencias en empresas reales.

El alcance de las pruebas indica qué funcionalidades se van a probar y cuáles no. Para poder definir el alcance, se divide el sistema en módulos, componentes o subsistemas, no todos los componentes serán probados con la misma importancia y pueden existir componentes que queden fuera del alcance de las pruebas. Cada componente agrupa varias funcionalidades, se dividen las funcionalidades hasta un nivel en el que sea posible definir el alcance. Luego de esto, se analizan las funcionalidades, dando como resultado un Inventario de Pruebas. El inventario es una lista de las funcionalidades del producto de software.

3.8.- Planificación de cada ciclo de prueba.

Al comienzo del proyecto, se planifican las pruebas a realizar del producto en cada versión que se genere del mismo. Esta planificación debe ser revisada al comenzar cada nuevo ciclo de prueba, ya que los supuestos sobre los que se definió la planificación probablemente hayan cambiado. Por ejemplo, al transcurrir el tiempo pueden cambiar las prioridades de las pruebas debido a cambios en las prioridades del negocio o a la confianza adquirida en el

producto como resultado de la realización de las pruebas en ciclos anteriores. También en cada ciclo se deben planificar las pruebas de regresión. Al obtener una nueva versión donde se corrigieron incidentes, se deben ejecutar nuevamente los casos de prueba que encontraron esos incidentes. Como a prioridad no se conoce cuales ni cuantos serán esos incidentes, al comenzar cada ciclo, deben ser consideradas las pruebas de regresión.

3.9.-Roles y responsabilidades.

Los participantes de cualquier tipo de revisión formal deberían tener conocimiento sobre el proceso de revisión. Un factor crítico de éxito de una inspección o revisión técnica es formar a los participantes que van a tomar parte en la misma.
Las mejores revisiones formales surgen de equipos bien organizados, y guiados por un moderador formado. Dentro del equipo de revisión, se distinguen 5 tipos de participantes:

- **Moderador.** El moderador de la revisión dirige el proceso de revisión. En cooperación con el autor del documento, el moderador determina el tipo de revisión que se va a realizar y la composición del equipo de trabajo. El moderador realiza una validación de entrada y un seguimiento del trabajo realizado con el objetivo de controlar la calidad del proceso de revisión.

- **Autor.** Es la persona que ha creado el documento bajo revisión. El objetivo principal del autor es mejorar la calidad del documento y su propia habilidad para escribir futuros documentos.

- **Documentador.** Es la persona encargada de tomar nota de cada defecto mencionado y de cualquier sugerencia acerca de la mejora del proceso aunque, en la práctica, es el autor quien juega este papel.

- **Revisor.** La tarea del revisor, también llamado validador o inspector, es la de validar cualquier material con posibles defectos antes de realizar la propia reunión de revisión. El nivel de minuciosidad, el conocimiento del revisor sobre el dominio o su propia habilidad técnica dependen del tipo de revisión.

- **Supervisor.** Es quien decide destinar un tiempo de la planificación del proyecto para realizar las revisiones, quien determina si se han cumplido los objetivos de la revisión y, por supuesto, quien decide ejecutarla. El supervisor también se ocupa de las solicitudes de formación de los participantes en la revisión.

Programadores (Programmer)

José Napoleón Páez Escobar.

- Responsable de Decisiones Técnicas.
- Responsable de construir el Sistema.
- Sin distinción entre analistas, diseñadores o codificadores.

Cliente (Customer)

Master Ing. Marco Jarrín López

- Son parte del equipo
- Determinan qué construir y cuándo.

Asesor (Manager)

Master Ing. Marco Jarrín López

- El líder del equipo - toma las decisiones importantes
- Principal responsable del proceso
- Tiende a estar en un segundo plano a medida que el equipo toma consistencia.

Rastreador (Tracker)

- **José Napoleón Páez Escobar.**
- MetricMan
- Observa sin molestar
- Conserva datos históricos

Probador (Tester)

- **José Napoleón Páez Escobar.**
- Ayuda al cliente con las pruebas funcionales
- Se asegura de que los tests funcionales se ejecutan

3.10.- Resultado Obtenidos.

Luego de la implementación del modelo para evaluación/pruebas del software en base a ingeniería de pruebas aplicando el estándar ISO/IEC 29119. Se obtuvo los siguientes resultados.

Actividades realizadas:

- Inducción al personal sobre el estándar ISO/IEC 29119.
- Capacitación en el uso de la herramienta de gestión al personal encargado de pruebas.
- Registro de la información de los proyectos.
- Seguimiento a las actividades de prueba.
- Control de las actividades de prueba.
- Toma de decisiones por parte de dirección y gerencia

Logros obtenidos:

- ✓ Automatización en la gestión del proceso de pruebas.
- ✓ Mayor confiabilidad en el estado de pruebas del proyecto.
- ✓ Aumento de la calidad en el producto de software final.
- ✓ Rapidez en la toma de decisiones.
- ✓ Identificación de la fase de mayor complejidad y de mayor cuidado para futuros proyectos.
- ✓ Medio de estimación más asertiva de costo y tiempo con fundamento en una base histórica.
- ✓ Mayor garantía en los productos desarrollados.
- ✓ Mayor prestigio institucional.

3.11.- Caso Práctico

El nombre del proyecto: Sistema Informático para registro único de aliados. (RUA)

La política: Desarrollo de un sistema fiable para la implementación en el Ministerio de Coordinación y Desarrollo Social con base en la unificación de la información de los aliados en los diferentes ministerios coordinados.

La estrategia: Desarrollo e implementación de un software que permita tener una base única del personal coordinado por parte del súper ministerio.

Estado del proyecto: Esta en activo.

Usuario	*Proceso*
Juan Pablo	Gestión Organización
Juan Pablo	Proceso Gestión Pruebas
Santiago Rosales	Monitoreo & Control
Marco Rosillo	Monitoreo & Control
Marco Jarrín	Seguimiento & Evaluación
José Páez	Seguimiento & Evaluación

Resultado de la gestión e identificación del proyecto: Caso exitoso.

Comentarios: De fácil ingreso de la información inicial.

Una vez que se identificó el proyecto, la política, estrategia se pasa al proceso de gestión de pruebas.

Define nombre del plan de pruebas: Plan de pruebas para el sistema RUA

Identificación Actividades:

1 Especificación de Requerimientos

2 Especificaciones Funcionales

3 Revisión de Caso de Uso

4 Especificaciones de Diseño

5 Prototipo

6 Modelo o Flujo del Negocio

7 Modelo o Flujo de Datos

8 Construcción del software

9 Puesta en ambiente de preproducción

10 Ambiente producción

11 Revisión del Manual

Riesgos:

1 Mala especificación de los requerimientos

2 No especificación funcional

3 Duplicidad en los casos de uso

4 Un entorno poco moderno

5 Proyección baja calidad

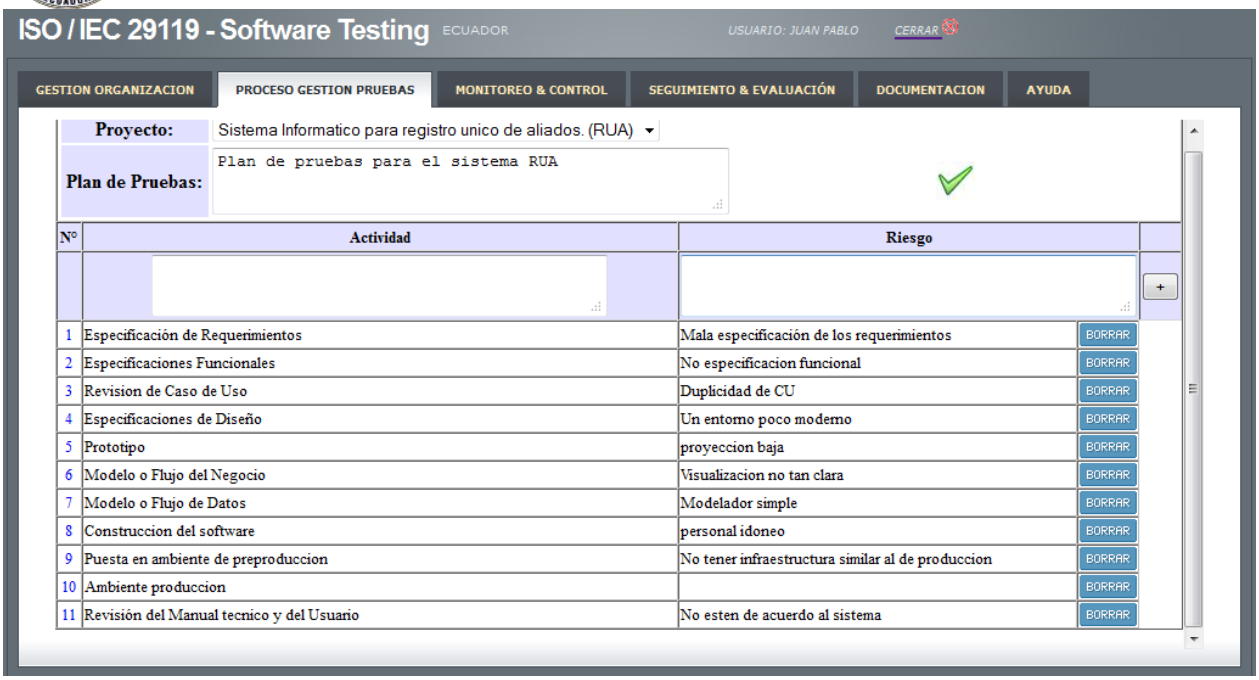
6 Visualización no tan clara

7 Modelador simple

8 Personal no idóneo

9 No tener infraestructura similar al de producción

10 Manual técnico y del Usuario no están de acuerdo al sistema



Una vez definida el proceso que se va gestionar se procede al siguiente nivel de monitoreo y control.

1 Especificación de Requerimientos

Clase	***Técnica***	***Puntaje***	***Comentario Especifico***
Estática	Inspecciones	Aceptable	Revisión física y documental del proceso de registro de aliados
Estática	Tutoriales	Aceptable	Análisis del entorno político al cual va dirigido
Estática	Inspecciones	Aceptable	Comunicación con los diferentes usuarios involucrados
Estática	Inspecciones	Aceptable	Recopilación de la mejor forma de llegar al

			usuario sin mayor complejidad
Estática	Inspecciones	Aceptable	Análisis de la información recopilada
Estática	Inspecciones	Aceptable	Validación de los requerimientos con gerencia
Estática	Inspecciones	Aceptable	Aceptación de los requerimientos
Estática	Inspecciones	Aceptable	Compromiso y mutuo acuerdo en el análisis

2 Especificaciones Funcionales

Clase	***Técnica***	***Puntaje***	***Comentario Especifico***
Estática	Inspecciones	Aceptable	Se probó el flujo de navegación con enfoque

			web 2.0
Estática	Inspecciones	Aceptable	Se inspeccionó la estructura de la información
Estática	Inspecciones	Aceptable	Se revisó el número de clics por pantalla para acceso a datos
Estática	Inspecciones	Aceptable	Revisión de menús de navegación horizontal y vertical

3 Revisión de Caso de Uso.

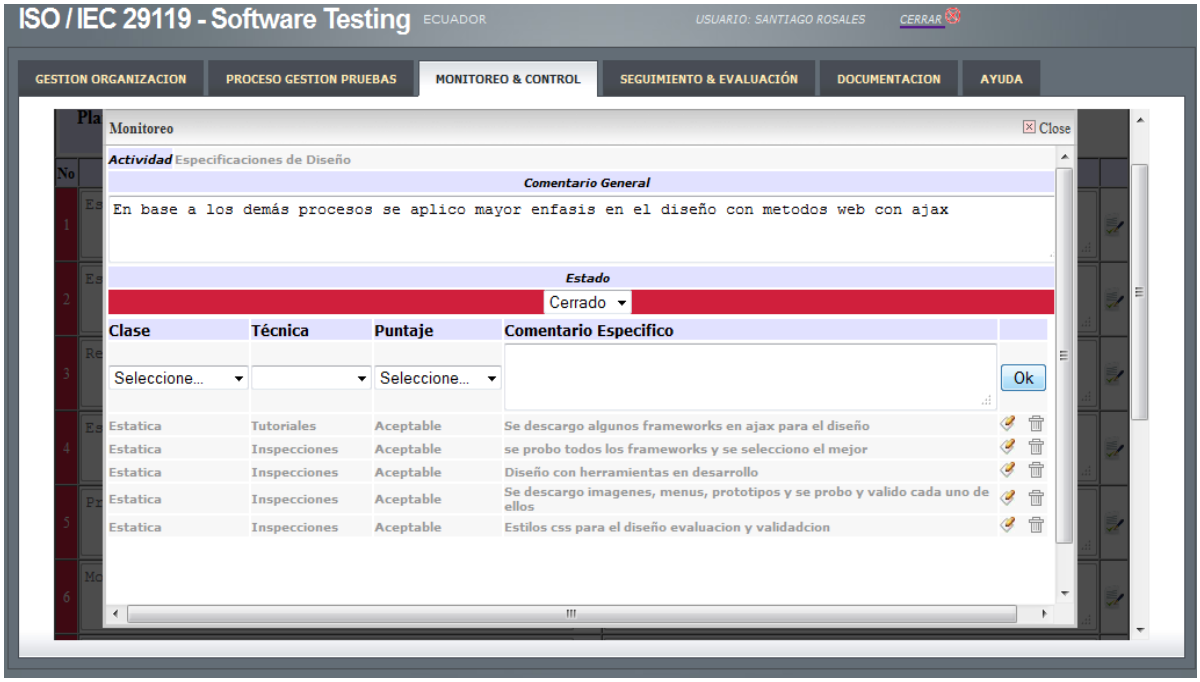
Clase	***Técnica***	***Puntaje***	***Comentario Especifico***
Estática	Inspecciones	Aceptable	Revisión de cada caso de uso
Estática	Tutoriales	Aceptable	Se revisó procesos que se manejó en sistemas anteriores y se validó

Estática	Inspecciones	Aceptable	Identificación de actores, procesos
Estática	Inspecciones	Aceptable	Revisión y aprobación de los diagramas

4 Especificaciones de Diseño

Clase	***Técnica***	***Puntaje***	***Comentario Especifico***
Estática	Tutoriales	Aceptable	Se descargó algunos frameworks en ajax para el diseño
Estática	Inspecciones	Aceptable	se probó todos los frameworks y se seleccionó el mejor
Estática	Inspecciones	Aceptable	Diseño con herramientas en desarrollo
Estática	Inspecciones	Aceptable	Se descargó imágenes,

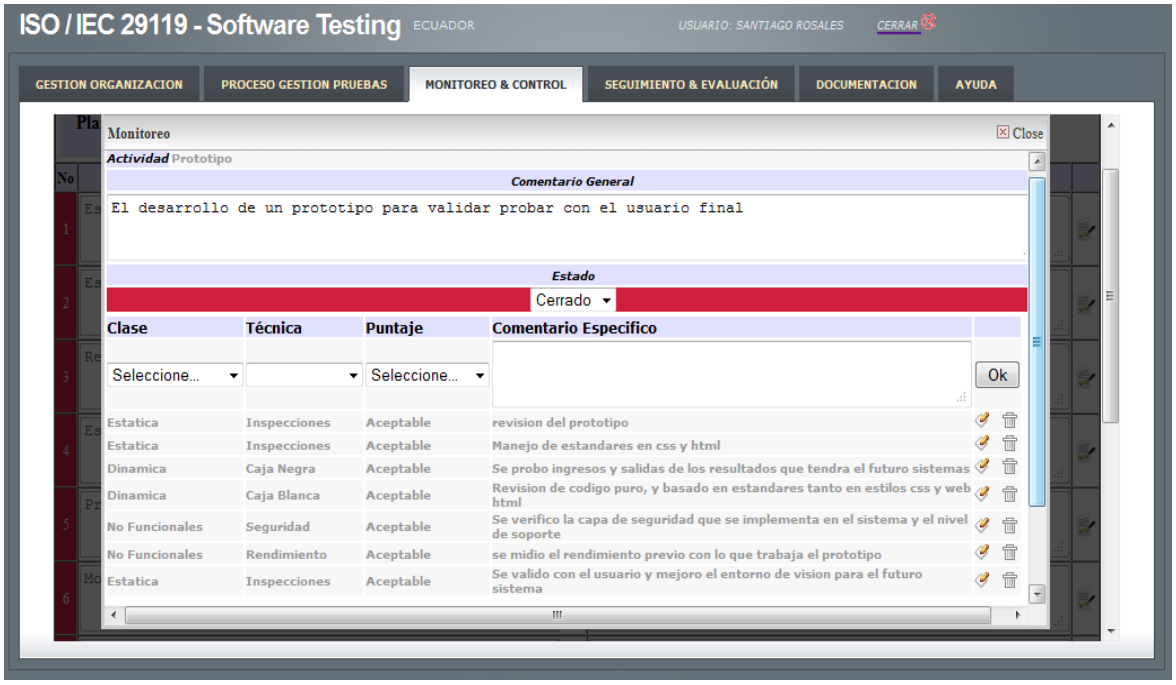
			menús, prototipos; se probó y validó cada uno de ellos
Estática	Inspecciones	Aceptable	Estilos CSS para el diseño evaluación y validación



5 Prototipo

Clase	***Técnica***	***Puntaje***	***Comentario Especifico***
Estática	Inspecciones	Aceptable	Revisión del prototipo
Estática	Inspecciones	Aceptable	Manejo de estándares en css y html
Dinámica	Caja Negra	Aceptable	Se probó ingresos y salidas de los resultados que tendrá el futuro sistemas
Dinámica	Caja Negra	Aceptable	Revisión de código puro, y basado en estándares

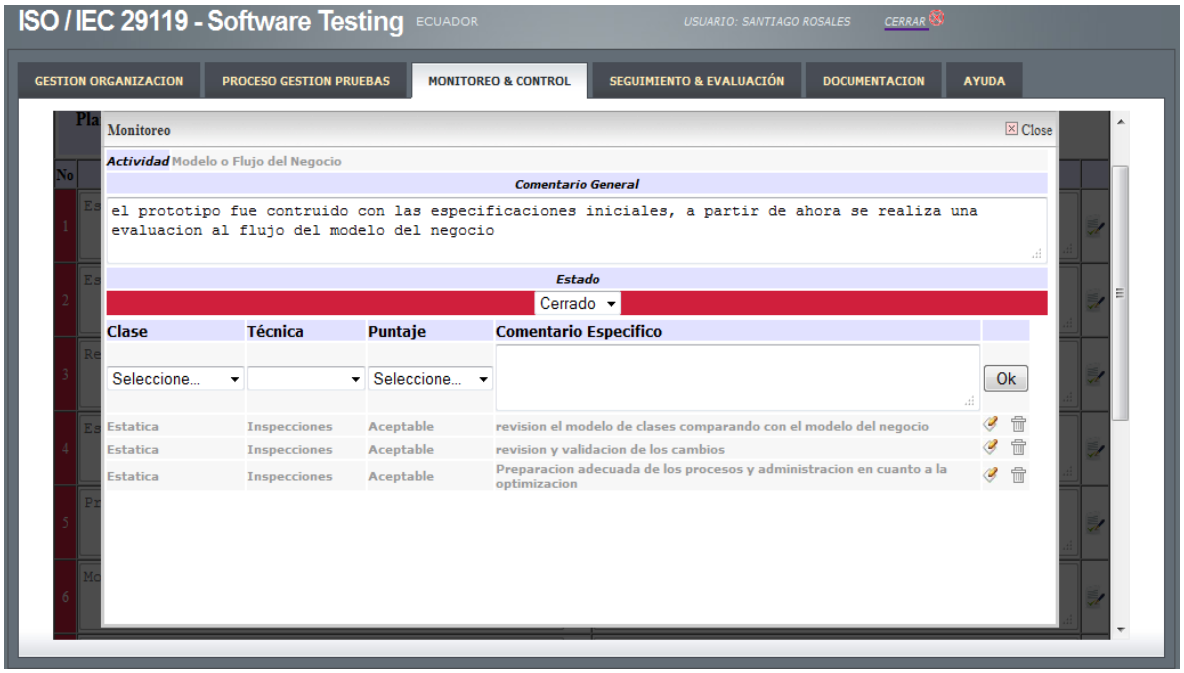
			tanto en estilos css y web html
No Funcional	Seguridad	Aceptable	Se verificó la capa de seguridad que se implementa en el sistema y el nivel de soporte
No Funcional	Rendimiento	Aceptable	Se midió el rendimiento previo con lo que trabaja el prototipo
Estática	Inspecciones	Aceptable	Se validó con el usuario y se mejoró el entorno de visión para el futuro sistema



6 Modelo o Flujo del Negocio

Clase	*Técnica*	*Puntaje*	*Comentario Especifico*

Estática	Inspecciones	Aceptable	Revisión el modelo de clases comparando con el modelo del negocio
Estática	Inspecciones	Aceptable	Revisión y validación de los cambios
Estática	Inspecciones	Aceptable	Preparación adecuada de los procesos y administración en cuanto a la optimización



7 Modelo o Flujo de Datos

Clase	*Técnica*	*Puntaje*	*Comentario Especifico*
Estática	Inspecciones	Aceptable	Revisión de los modelos negocio y datos
Estática	Inspecciones	Aceptable	Revisión de la estructura que soportará el modelo de negocio
Estática	Inspecciones	Aceptable	Validación de la

			estructura
Estática	Inspecciones	Aceptable	Revisión y aprobación del modelo final

8 Construcción del software

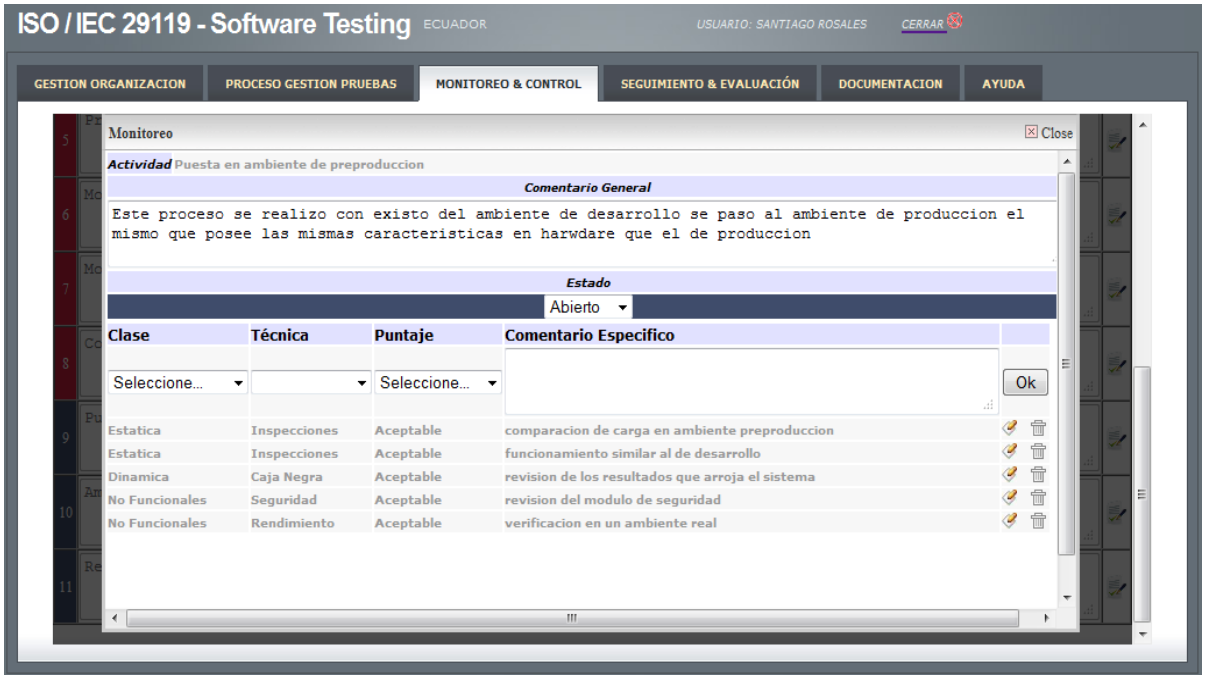
Clase	***Técnica***	***Puntaje***	***Comentario Especifico***
Estática	Inspecciones	Aceptable	Revisión de un módulo de ingreso
Dinámica	Caja Negra	Aceptable	validación de datos que acepta el sistema
Estática	Inspecciones	Aceptable	revisión y control de errores
Dinámica	Caja Negra	Aceptable	Revisión de código que tenga los estándares y no sea repetitivo en las clases y métodos
Dinámica	Caja Negra	Aceptable	Validación con el modelo

			de datos y negocio
No Funcional	Seguridad	Aceptable	Validación del método de seguridad
No Funcional	Rendimiento	Aceptable	Puesta en marcha con robots de carga y transaccional

9 Puesta en ambiente de preproducción

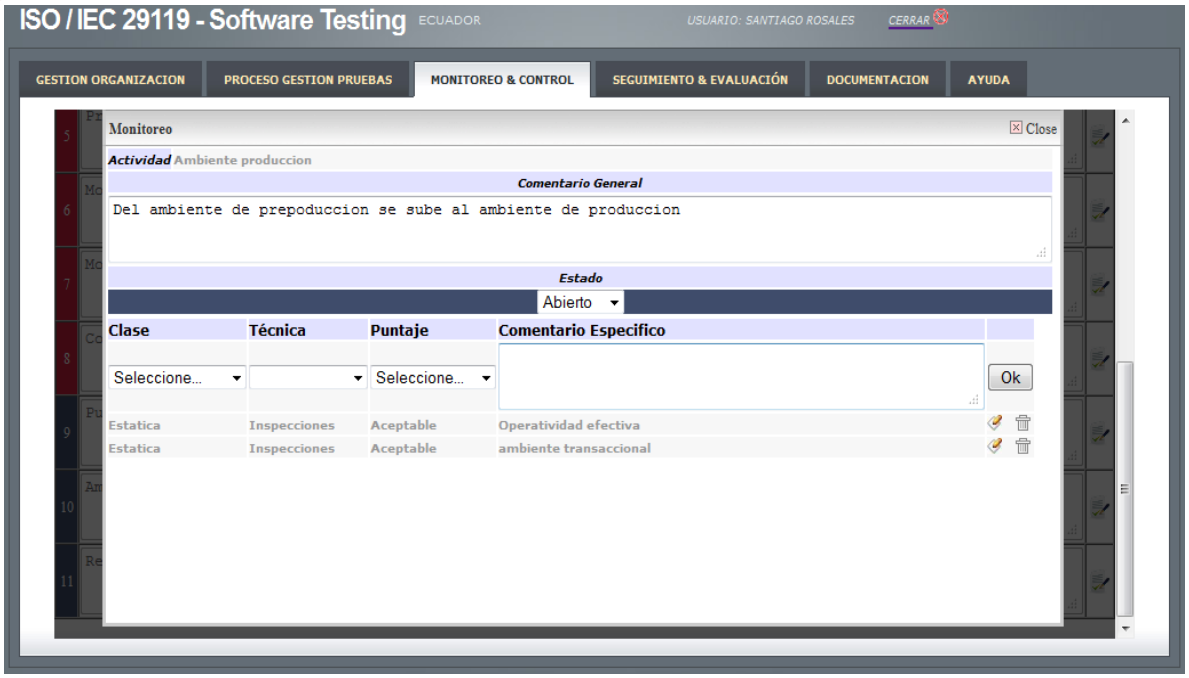
Clase	***Técnica***	***Puntaje***	***Comentario Especifico***
Estática	Inspecciones	Aceptable	Comparación de carga en ambiente preproducción
Estática	Inspecciones	Aceptable	Funcionamiento similar al de desarrollo
Dinámica	Caja Negra	Aceptable	Revisión de los resultados que arroja el sistema
No Funcionales	Seguridad	Aceptable	Revisión del módulo de seguridad

No Funcionales	Rendimiento	Aceptable	Verificación en un ambiente real la funcionalidad de todo el sistema



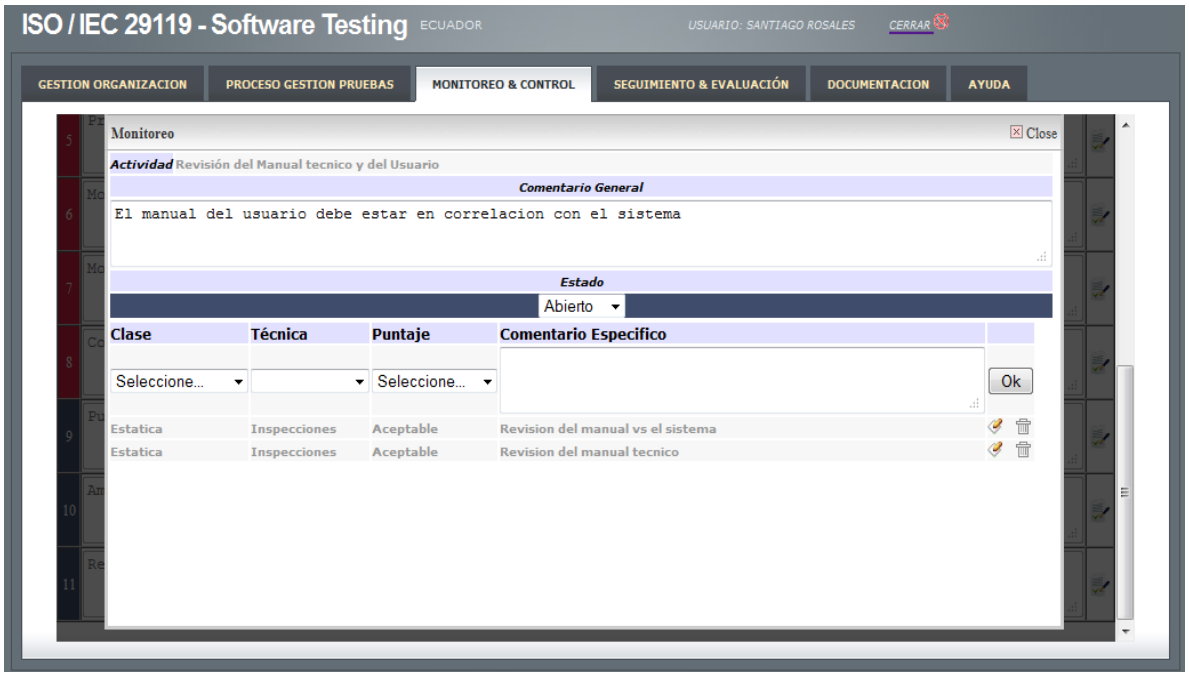
10 Ambiente producción

Clase	*Técnica*	*Puntaje*	*Comentario Especifico*
Estática	Inspecciones	Aceptable	Operatividad efectiva
Estática	Inspecciones	Aceptable	Ambiente transaccional



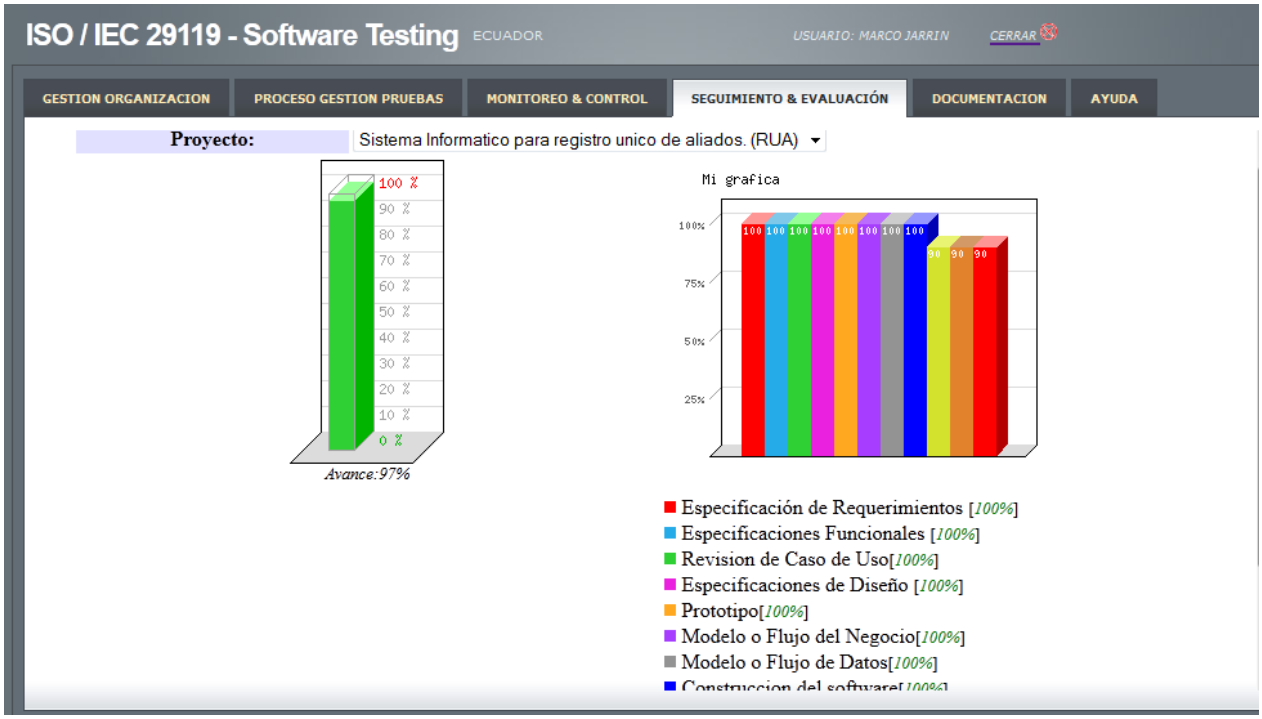
11 Revisión del Manual

Clase	***Técnica***	***Puntaje***	***Comentario Especifico***
Estática	Inspecciones	Aceptable	Revisión del manual vs el sistema
Estática	Inspecciones	Aceptable	Revisión del manual técnico



Realizado en proceso monitoreo de las actividades con las técnicas que se aplico en cada una de ellas, posterior a esto el usuario responsable de revisión y aprobación de cada actividad cambiando el estado.

Luego de todo el proceso de aprobación por parte del usuario responsable. En consecuencia el proceso de seguimiento y evaluación se puede tener una estadística general del proyecto para la gerencia.



La penalización es de un 10% sobre la actividad cuando se encuentra en estado abierto.

Resultado obtenido del caso práctico:

Una vez implementado el sistema y definido los roles de los usuarios que intervienen en el sistema, podemos apreciar que tenemos un proceso de gestión en el cual se identifica cada actividad que hacemos como un método de control y evaluación de pruebas para obtener un mejor producto de software, en definitiva estamos mejorando en gran medida la calidad con base a una norma internacional que aplica un estándar ISO que ayuda en gran medida a la estandarización de los procesos.

Enfoque de Pruebas	*Norma ISO 29119*	*Resultado*
Sistema Informático para registro único de aliados. (RUA) Desarrollo de un sistema fiable para la implementación en el Ministerio de Coordinación y Desarrollo Social. Con base en la unificación de la información de los aliados en los diferentes ministerios coordinados. Desarrollo e implementación de un software que permita tener una base única del personal coordinado por parte del súper ministerio.	Definiciones y Vocabulario.	Identificación del proyecto, definimos un para qué, con el fin que perseguimos.
Especificación de Requerimientos Especificaciones Funcionales Revisión de Caso de Uso Especificaciones de Diseño Prototipo Modelo o Flujo del Negocio Modelo o Flujo de Datos Construcción del software Puesta en ambiente de preproducción Ambiente producción Revisión del Manual	Proceso de Pruebas.	Definimos el proceso de gestión con un nivel de pruebas adecuado para mi entorno y proyectando donde mejorar.
Registro en el sistema	Documentación de Pruebas.	La información que alimentamos al sistemas, nos permite tener una base de

		conocimiento
Tutoriales Caja Negra Caja Blanca Seguridad Rendimiento	Técnicas de Pruebas	Las técnicas que empleamos en el proceso de pruebas. Son las más comunes y generalizadas.

Con el resultado obtenido, podemos identificar que mediante un proceso automatizado pudimos implementar la norma ISO 29119 que tiene una proyección global para el proceso de pruebas.

El software es una base de conocimiento que permitirá a futuro identificar las áreas más críticas y poder mejorar el proceso para obtener un mejor producto de software.

Con base en el resultado que se obtuvo se puede determinar:

	Gestiona		Monitorea		Evalúa		Base Norma ISO		Permite toma decisiones	
Herramientas	**Si**	**No**	**Si**	**No**	**Si**	**No**	**Si**	**No**	**SI**	**No**
Asistente de QA Pro 2011	X		x			x		x		x
Oracle Enterprise Manager	X		x			x		x		x
Pruebas en cualquier lugar	X		x			x		x		x
AppLabs	X		x			x		x		x
SmarteSoft	X		x			x		x		x
Sonar	X		x			x		x		x
JMETER	X		x			x		x		x
TestLink	x		x			x		x		x
SCGP-ISO 29119	x		x		x		x		x	

Con la herramienta se tiene una gestión mucho más amplia que las convencionales permitiendo mejorar de manera eficiente y ordenada los procesos, la misma que se basa en la norma ISO para pruebas mediante la cual podemos tomar decisiones de corrección tempranas, con el único fin de tener un mejor producto de software, que a la final no sea una prueba ligera y poco eficiente.

CASO DE EVALUACION DE RESULTADOS

Análisis de Instrumentos

Para el presente caso de estudio se realizo una matriz en Excel tomando como base algunos criterios importantes como son: El nivel de expertis del programador, el lenguaje de programación, el entorno de desarrollo de interfaz, el número de personas involucradas en el desarrollo. Posterior a ello se estableció parámetros para los resultados como identificación de errores, fallas y defectos, número de líneas de código LDC, el tiempo y costo.

En la empresa Omnisoft se procedió a desarrollar en paralelo un pequeño modulo de un proyecto el mismo que se llama: Sistema escolástico Spellman específicamente el módulo de parametrización.

Se realizo el caso con el equipo de desarrollo con un perfil sénior, con un IDE de desarrollo que los dos expertos manejaban, se uso Netbeans 7.2.

Una vez que se termino el proceso de desarrollo se obtuvieron los siguientes resultados.

Resultados Obtenidos

Se pudo identificar que el desarrollador 1 obtuvo un número mayor de líneas de código pero porque saltaba a doble espacio en ciertas partes de código, pero una vez revisado el código con el IDE de desarrollo el mismo que permite contabilizar de manera automática las líneas de código programadas dio como resultado que tenían igual líneas de código los dos desarrollos.

También se pudo identificar que el tiempo de desarrollo incremento a razón de que el desarrollador 2 realizaba una documentación y unas ciertas correcciones sobre su desarrollo, esto conllevo a que incremente su tiempo en el proceso de creación de software.

Una vez que los dos software estuvieron se procedió a implementar en un ambiente de pruebas con características similares en hardware, dando como resultado en primera instancia el programador 1 obtuvo un número mayor de errores, defectos y fallas.
Con este resultado de puede apreciar que en el primer caso no se implemento la norma ISO/IEC 29119 y se obtuvo un producto rápido pero con un número mayor de errores al final, el mismo que al final fue un proceso de parcheo sin documentación de los cambios que se hizo.

Por tal razón el presente investigador del proyecto de tesis propone como una alternativa bastante valedera sugiriendo como una alternativa al gran proceso del aseguramiento de la calidad en un producto de software, a través de la disminución del número de errores al final antes del paso a producción.

Proyecto	*Desarrollador*	*Perfil*	*LDC*	*LDC-CON*	*Lenguaje*	*IDE-DESARROLLO*	*Personas*	*Costo*	*días*	*Errores*	*Defectos*	*Fallas*	*Estándar ISO/IEC 29199*
Sistema Escolástico Spellman - Módulo Parametrización	Desarrollador 1	Sénior	483	483	PHP-AJAX	Netbeans7.2	1	500	5	44	15	10	NO
	Desarrollador 2	Sénior	490	483	PHP-AJAX	Netbeans7.2	1	550	7	10	4	3	SI

CAPITULO IV

4.1.- Conclusiones.

Se ha desarrollado he implementado EL DISEÑO DE UN MODELO PARA EVALUACION/PRUEBAS DEL SOFTWARE EN BASE A INGENIERIA DE PRUEBAS APLICANDO EL ESTANDAR ISO/IEC 29119, mismo que permite un mejor seguimiento de los proyectos de desarrollo de software; tiempo atrás se documentaba manualmente y posterior a ello en archivos de Excel. Esa manera de documentar conllevaba a confusiones y errores al momento de realizar o no la prueba. Con la implementación del modelo de software que estructuralmente ya contempla la norma ISO/IEC 29119 Omnisoft se permite tener una gestión más eficiente de varios proyectos al mismo tiempo.

Si bien es cierto existen herramientas que permiten guardar información de proyectos pero no se basan en estándares internacionales para el manejo de pruebas aplicados de manera al entorno nacional de nuestro país, el aporte de esta herramienta es de gran importancia para la empresa Omnisoft ya que permite tener un control más efectivo y detallado de cada uno de los casos de pruebas efectuados al software.

Cada proyecto de software realizado y gestionado a través de la herramienta permitirá una mejor toma de decisiones por parte de la gerencia o dirección de proyectos como por ejemplo identificar la fase de mayor criticidad en el ciclo de vida de desarrollo de software.

Tradicionalmente la empresa Omnisoft realizaba el desarrollo de un producto de software y a su vez las pruebas las dejaban al final días antes de la implementación y puesta en un ambiente de producción, por ende al final se detectaban errores críticos en el software y esto conllevaba a un desprestigio de la empresa y del equipo de desarrollo, con la implementación de esta herramienta de gestión de evaluación de pruebas del software en

base a ingeniería de pruebas aplicando el estándar ISO/IEC 29119, cada fase del ciclo de vida del software es gestionada, si hay una evaluación favorable puede continuar a la siguiente fase con lo cual estaremos verificando y garantizando que el producto final sea óptimo y de mejor calidad.

Así mismo, es importante destacar que para la implementación de la herramienta, se consideran aspectos relevantes en el área de prueba de software, así como algunas métricas para software, todo esto con el propósito de enriquecer los criterios de evaluación y sacar un producto de calidad.

La realización de este trabajo, los resultados obtenidos a partir de él y muchas otras experiencias, permiten darnos cuenta que la gestión de las pruebas de software es una actividad que indiscutiblemente debe llevarse a cabo en todo proceso de desarrollo en empresas tanto públicas como privadas.

Si bien es cierto este macro proceso de gestión de pruebas conlleva a un incremento en el costo y tiempo pero a la larga es mejor corregir a tiempo los errores del software que llegar al final y exista un fracaso total del proyecto.

4.2.- Recomendaciones.

- Instruir al equipo que llevará este proceso de gestión y control de pruebas en base a la norma ISO 29119.

- Capacitar sobre la norma y el manejo adecuado del sistema para mantener una estandarización en el proceso.

- Se recomienda que la herramienta a futuro sirva como base para la realización de métricas y la detección de puntos clave en el macro proceso de creación de software.

- En base a este proceso de gestión se sugiere iniciar un mecanismo para la toma de decisiones en el encaminamiento del proyecto.

- Dar a conocer la herramienta por medios como la web, permitiendo al interesado pueda descargarla de forma gratuita y sirva para toda la comunidad que crea el software a baja, media y alta escala.

4.3.- Glosario.

- **Análisis de riesgos:** proceso de evaluar riesgos identificados para estimar su impacto y probabilidad de ocurrencia.

- **Análisis estático:** análisis de artefactos software, por ejemplo: requisitos o código, llevado a cabo sin la ejecución de dichos artefactos.

- **Aseguramiento de la calidad (QA)**: conjunto de actividades diseñadas para asegurar que el proceso de desarrollo y/o mantenimiento es adecuado para que el sistema cumpla sus objetivos.

- **Caso de prueba:** conjunto de valores de entrada, precondiciones de ejecución, resultados esperados y post condiciones de ejecución desarrolladas para un objetivo o condición de prueba particular, tal como probar un determinado camino de un programa.

- **Control de calidad (QC)**: conjunto de actividades diseñadas para evaluar el producto de trabajo ya desarrollado.

- **Driver:** componente software o herramienta de prueba que reemplaza un componente que se encarga del control y/o de la llamada a un componente o sistema.

- **Informe de incidencias de pruebas**: documento que recoge cualquier evento ocurrido durante el proceso de pruebas que requiera de investigación y resolución.

- **Informe de resumen de pruebas:** documento que resume las actividades y resultados de las pruebas. También contiene la evaluación de los elementos de prueba correspondientes.

- **Peer reviews**(Revisión entre pares): las revisiones entre pares es un método de verificación importante y eficaz implementado vía inspecciones, revisiones técnicas, *walkthrough*u otros métodos de revisión entre compañeros.

- **Plan de pruebas**: documento que describe el alcance, el enfoque a seguir, los recursos a asignar y la planificación de las actividades de pruebas.

- **Prototipo:** borrador de un producto potencial o de una parte del mismo, una simulación de los requisitos.

- **Pruebas**: actividad en la cual un sistema, o uno de sus componentes, se ejecutan en circunstancias previamente especificadas, se observan los resultados, se registran y se realiza una evaluación de los mismos.

- **Pruebas dinámicas:** pruebas que implican la ejecución del software de un componente o sistema.

- **Pruebas estáticas:** pruebas de un componente o sistema a nivel de especificación o implementación sin la ejecución de dicho software, por ejemplo: revisiones o análisis estático.

- **Registro de pruebas**: registro cronológico de los detalles más relevantes relacionados con la ejecución de las pruebas.

- **Revisión:** evaluación del estado de un producto o proyecto para encontrar discrepancias con los resultados planificados y recomendar mejoras. Por ejemplo: revisión informal, revisiones técnicas, inspecciones, entre otros.

- **Riesgos**: factor que puede resultar en consecuencias negativas en el futuro. Normalmente se expresa en términos de impacto y probabilidad.

- **Sistema de pruebas:** Las pruebas forman parte de lo que se denomina sistema de pruebas. Los cuatro componentes principales de un sistema de pruebas son equipo, recursos, procesos y entorno de pruebas.

- **Stub:** implementación esquemática con un propósito especial de un componente software, usado para desarrollar o probar un componente que llama o depende de él. Reemplaza al componente llamado.

- **Test-driver*:*** son elementos de prueba que se utilizan para facilitar la comprobación de la integración de los componentes de un sistema en cada momento.

- **Tipo de prueba:** grupo de actividades de prueba dirigidas a probar un componente o sistema centrado en un objetivo de prueba específico.

- **Validación**: proceso de comprobar que un sistema cumple las necesidades y expectativas del cliente.

- **Verificación**: proceso de comprobar que un sistema cumple su especificación.

- **Walkthrough:** presentación paso a paso por el autor de un documento para reunir información y para establecer un entendimiento común de su contenido.

4.4.- Acrónimos.

- **AVAL** (*AcquisitionValidation*): Validación de la Adquisición

- **AVER** (*AcquisitionVerification*): Verificación de la Adquisición

- **CMMI®** (*CapabilityMaturityModelIntegration*): Modelo de Capacidad y Madurez Integrado.

- **CMMI-ACQ®** (*CapabilityMaturityModelIntegrationforAcquisition):* Modelo de Capacidad y Madurez Integrado para Adquisiciones.

- **CMMI-DEV®** (*CapabilityMaturityModelIntegrationforDevelopment):* Modelo de Capacidad y Madurez Integrado para Desarrollo.

- **CMMI-SVC®** (*CapabilityMaturityModelIntegrationforService):* Modelo de Capacidad y Madurez Integrado para Servicios.

- **IEEE** (*Institute of Electrical and ElectronicsEngineers*): Instituto de Ingeniería Eléctrica y Electrónica.

- **ISO** (*International OrganizationforStandarization*): Organización Internacional para la Estandarización.

- **QA** (*QualityAssurance*): Aseguramiento de calidad

- **QC** (*Quality Control*): Control de calidad

- **RFP** (*Requestforproposal*): Solicitud de propuesta

- **SVVP** (*Software Verification and Validation Plan*)**:** Plan de Verificación y Validación.

4.5. –Bibliografía.

- Beizer, B., "*Software Testing Techniques",* SegundaEdición, 2005
- Black, R., *"Managing the Testing Process"*, SegundaEdición, 2004
- Boehm, B., "*Software Engineering Economics*", 1981
- Brian Hambling, Peter Morgan, Angelina Samaroo, Geoff Thompson y Peter Williams, *Software Testing - An ISEB Foundation*, 2007
- CMMI® Product Team, *CMMI® For Acquisition, Version 1.2. Improving processes for acquiring better products and services*, 2007
- Dijkstra, Edsger, "Notes on Structured Programming", 1972
- IEEE Computer Society, *IEEE Std. 829-1998 Software test documentation*, IEEE, 1998.
- IEEE Standard for Software Verification and Validation, IEEE 1012, 2004
- Mary Beth Chrissis, Mike Konrad y Sandy Shrum, *CMMI® Second Edition. Guidelines for Process Integration and Product Improvement*, 2007
- Rex Black, *Pragmatic Software Testing: Becoming an effective and efficient test professional*, 2007

- Roger S. Pressman, *Software Engineering. A practitioner's Approach*, Quinta Edición, 2001
- Ron Burback, *Software Engineering Methodology*, 1998
- Sommerville, Ian, *Software Engineering 07 Edition*, 2005 ISO/IEC 29119 *Software Testing Standard*: http://www.softwaretestingstandard.org/
- Piattini M.G y García F.O. (2003). *Calidad en el desarrollo y mantenimiento del software*, Editorial RA-MA.
- Minguet J.M y Hernández J.F. (2003). *La calidad del software y su medida*, Universitaria RamónAreces.
- Pressman, Roger. Ingeniería del software: Un enfoque práctico. 5ta. ed. McGraw Hill, Madrid, 2002.
- Larman, Craig. UML y Patrones. Introducción al análisis y diseño orientado a objetos. Prentice Hall, México, 2006.
- Fowler, Martin. UML gota a gota. Addison Wesley Longman de México, México, 1999.
- Sintes, Anthony. Aprendiendo programación orientada a objetos en 21 lecciones avanzadas. Pearson Educación, México, 2002.
- Jacobson, I., Booch, G. y Rumbaugh, J. El proceso unificado de desarrollo de software. Pearson Educación, Madrid, 2000.
- Rosenberg, Doug y Scott, Kendall. Use Case Driven Obejct Modeling with UML: A practical approach. Addison Wesley, Boston, 2006.
- Rosenberg, Doug y Scott, Kendall. Appying Use Case Driven Object Modeling with UML: An annotated e-commerce Example. Addison Wesley, Boston, 2001.

- Booch, Grady, Rumbaugh, J. y Jacobson, I. The Unified Modeling Language User Guide. Addison Wesley, 1998.

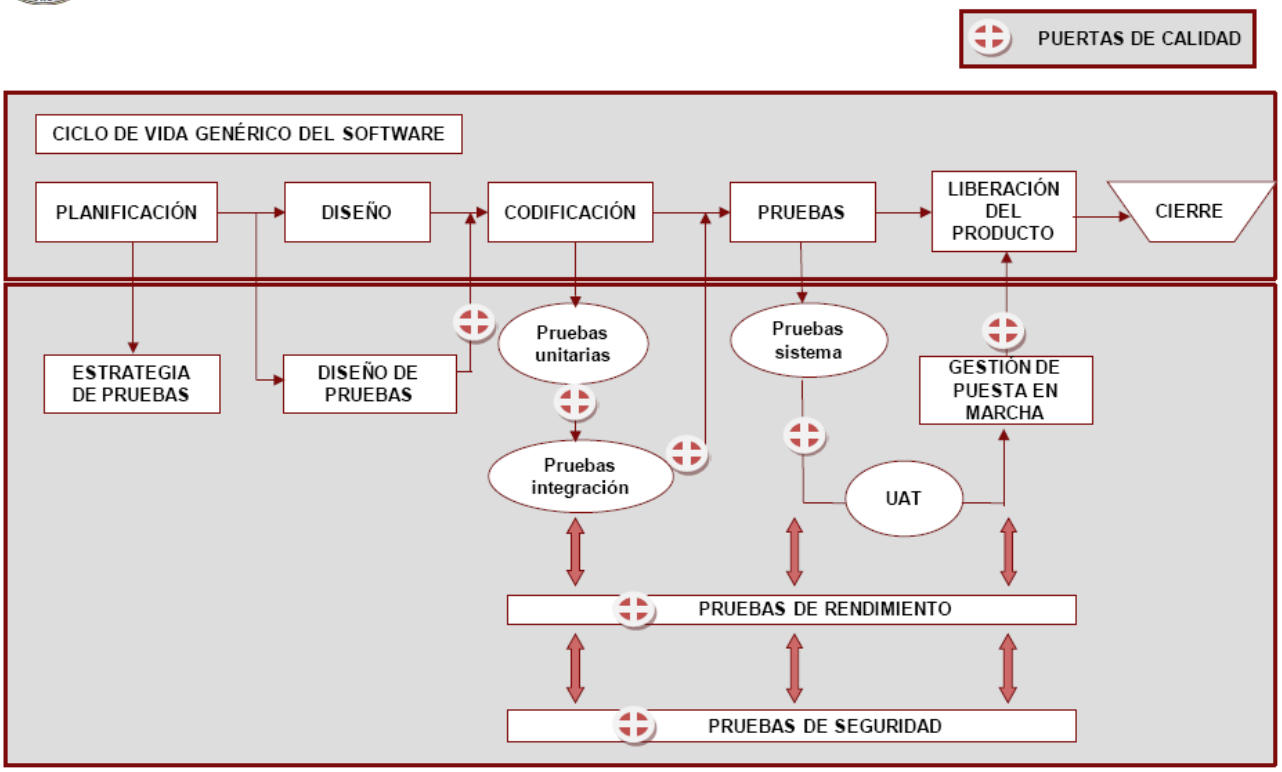
4.6. – Bibliografía Web.

- http://www.slideshare.net/marcosxm/metodologa-de-desarrollo-de-software-rad(7.5.2011)
- http://www.monografias.com/trabajos28/proyecto-uml/proyecto-uml.shtml(7.5.2011)
- http://www.cs.cinvestav.mx/Estudiantes/TesisGraduados/2005/tesisJoseJaimeL.pdf (7.5.2011)
- http://img.redusers.com/imagenes/libros/lpcu097/capitulogratis.pdf(7.5.2003)
- www.calidaddelsoftware.com/(12.5.2011)
- http://es.wikipedia.org/wiki/Software(12.5.2011)
- http://www.alegsa.com.ar/Dic/software.php(21.6.2011)
- http://biblioweb.sindominio.net/pensamiento/softlibre/softlibre007.html(21.6.2011)
- http://www.hanantek.com/win-win(15.8.2011)
- http://upsg01.foroactivo.com/t112-etapas-de-desarrollo-de-software (15.8.2011)
- www.ise.gmu.edu Department of Information and Software Engineering(17.8.2011)
- www.calidaddelsoftware.com, Calidad del Software(16.9.2011)

- http://perso.club-internet.fr/brouardf/SGBDRmerise.htm (16.9.2011)
- http://www.map.es/csi/metrica3/ (17.9.2011)
- http://www.comp.glam.ac.uk/pages/staff/tdhutchings/chapter4.html (25.10.2011)
- http://portal.newman.wa.edu.au/technology/12infsys/html/dfdnotes.doc (25.10.2011)
- http://www.yourdon.com/books/coolbooks/notes/wardmellor.html (25.10.2011)
- http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?Yourdon%2FDemarco(25.10.2011)
- http://gantthead.com/Gantthead/process/processMain/1,1289,2-12009-2,00.html(25.10.2011)
- http://java.sun.com/(25.10.2011)
- http://www.uml.org/(25.10.2011)
- http://www.rational.com/products/rup/index.jsp(25.10.2011)
- http://www.open.org.au/(25.10.2011)

ANEXO A.

Ciclo de vida del software


PUERTAS DE CALIDAD
CICLO DE VIDA GENÉRICO DEL SOFTWARE
PLANIFICACIÓN
DISEÑO
CODIFICACIÓN
PRUEBAS
LIBERACIÓN DEL PRODUCTO
CIERRE
ESTRATEGIA DE PRUEBAS
DISEÑO DE PRUEBAS
Pruebas unitarias
Pruebas integración
Pruebas sistema
UAT
GESTIÓN DE PUESTA EN MARCHA
PRUEBAS DE RENDIMIENTO
PRUEBAS DE SEGURIDAD


Calidad durante el ciclo de vida
proceso
producto
Efecto del uso
Diferentes usos
influye
influye
influye
calidad de proceso
Calidad interna
Calidad externa
Calidad de uso
Depende de
Depende de
Depende de
proveedor
usuario

ANEXO B

Elementos Notacionales en UML.

El Unified Modeling Modeling Language (UML), en un lenguaje visual de propósito general que se utiliza para especificar, visualizar, construir y documentar cualquier sistema.

UML pretende unificar las técnicas de modelado, con el propósito de capturar la información de la estructura estática del sistema, así como de su comportamiento. Para ello, el lenguaje incluye una serie de elementos que incluyen conceptos semánticos, notación y directrices, los cuales se clasifican en las siguientes áreas:

a) Estructura Estática.

Se consideran los conceptos claves de la aplicación, sus propiedades y las relaciones entre objetos.

b) Comportamiento Dinámico.

Trata con la representación del o los objetos y su interacción con el mundo, así como con los patrones de comunicación entre objetos necesarios para implementar un comportamiento específico.

c) Implementación.

Se concentra en el manejo de constructores que representan las implementaciones de ciertos elementos.

d) Organización de Modelos.

Cuando se manejan sistemas muy grandes, conviene dividir la información en un conjunto de piezas coherentes que realicen el trabajo en diferentes partes de manera concurrente.

e) Mecanismos de Extensión.

Convenciones que tienen el propósito de permitir la inclusión de nuevos elementos en forma ordenada sin necesidad de realizar cambios significativos en el lenguaje.

Los elementos de UML utilizados durante la descripción del diseño de la herramienta que se describe en este trabajo, incluyen los Diagramas de Clases y los Diagramas de Actividades, los cuales corresponden a las áreas de Estructura Estática y Comportamiento Dinámico respectivamente. Estos se abordan con mayor detalle a continuación.

Diagramas de Clases.

Un diagrama de estructura estática muestra el conjunto de clases importantes que conforman parte de un sistema, junto con las relaciones existentes entre ellas. Ejemplifica de una manera estática la estructura de información del sistema y la visibilidad que tiene cada una de las clases, dada por sus relaciones con las demás en el modelo.

A continuación se presenta una tabla que concentra a los elementos gráficos que se utilizaron en los diagramas de clases que contiene este documento.

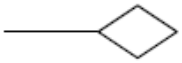
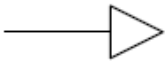
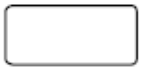
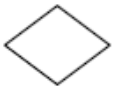
Elemento	Descripción	Representación Gráfica
Clase	Representada por un rectángulo con tres divisiones internas, que alojan el nombre de la clase, sus atributos y sus métodos, respectivamente. Son los elementos fundamentales del diagrama.	
Asociación	Permiten tener una mejor información sobre las relaciones entre los objetos de un sistema. Se representa como una línea etiquetada con la palabra que describe la relación existente.	
Agregación	Es una relación que permite describir a un "todo" y sus partes. Se representa con línea con un rombo en uno de sus extremos, el rombo debe ir situado del lado del objeto que representa el "todo".	
Generalización	Es una relación entre una descripción más general y una más especifica variedad de cosas. Aspectos como qué o quién construye a un objeto o quién lo extiende. Este elemento se utiliza para representar la herencia. Se representa con línea con un triángulo en uno de sus extremos, el triángulo debe ir situado del lado del objeto que representa la vista general.	

Diagrama de Actividades.

Un diagrama de actividades es un caso especial de un diagrama de estados en el cual casi todos los estados, son estados de acción (identifican qué acción se ejecuta al estar en él). Puede dar detalle a un caso de uso, un objeto o un mensaje en un objeto. Sirven para representar transiciones internas, sin hacer mucho énfasis en transiciones o eventos externos.

Generalmente modelan los pasos de un algoritmo.

A continuación, se presenta una tabla que concentra a los elementos gráficos que se utilizan en los diagramas de clases, así como una descripción de su significado.

Elemento	Descripción	Representación Gráfica
Estado de acción	Representa un estado con acción interna, con por lo menos una transición que identifica la culminación de la acción (por medio de un evento implícito).	
Transiciones	Las flechas entre estados representan transiciones con evento implícito. Pueden tener una condición en el caso de decisiones.	
Decisiones	Se representa mediante una transición múltiple que sale de un estado, donde cada camino tiene una etiqueta distinta.	
Fin del proceso	Establecen el punto en el que concluyen cada una de las actividades presentadas en el diagrama.	

ANEXO C

MANUAL Y PROCEDIMIENTOS DE UN SISTEMA DE TESTING DE PRUEBAS DE SOFTWARE

En base a la norma ISO/IEC 29119

José Páez Escobar

2011-2012

PRÓLOGO

Como continuación de la serie de textos que han visto la luz en estos últimos años, fruto de la colaboración entre la empresa ***Omnisof***, se presenta hoy esta nueva publicación con el fin de facilitar a todas las empresas y entidades de nuestra región, en especial las pequeñas y medianas, la implantación de un sistema de evaluación y pruebas de acuerdo con la norma internacional **ISO/IEC 29119**, en su versión del año 2007, que, junto con sus precedentes versiones, tan amplia difusión y consenso ha alcanzado a nivel mundial.

Siguiendo la idea de la anterior publicación: "DISEÑO DE UN MODELO PARA EVALUACION/PRUEBAS DEL SOFTWARE EN BASE A INGENIERIA DE PRUEBAS APLICANDO EL ESTANDAR ISO/IEC 29119 EN LA EMPRESA OMNISOFT DE LA CIUDAD DE QUITO" y como complemento de ella, el presente texto desarrolla los distintos apartados de la norma, aclarando la posible interpretación de los mismos de acuerdo con lo indicado en la norma complementaria "ISO 9000:2000 Fundamentos y vocabulario" y con las manifestaciones explícitas de expertos

pertenecientes a organizaciones consultoras y certificadoras, reiteradamente expresadas en las revistas profesionales.

A continuación de los párrafos referentes a la explicación de cada apartado, se proponen textos alternativos para la confección de un manual de calidad, se detalla su posible estructura y se presentan los distintos procedimientos necesarios para el desarrollo de un sistema de gestión de calidad.

De esta forma se pretende que cualquier empresa pueda llegar a comprender el espíritu de la norma y afrontar la confección de la documentación de su sistema de calidad.

INTRODUCCIÓN

¿Qué es ISO?

ISO (International Standard Organization) u Organización Internacional de Normalización, es un organismo que se dedica a publicar normas a escala internacional y que, partiendo de una norma ya existente de British Standard:

BS-5720, ha venido confeccionando la serie de normas ISO 9000, referidas a los Sistemas de la Calidad, desde hace varios años. La primera versión es de 1987 y sufrió una profunda revisión en 1994, por lo que esta nueva redacción del año 2000 supone la tercera modificación de su texto.

¿Qué es una norma?

Una norma es un documento que describe un producto o una actividad con el fin de que las cosas sean similares. El cumplimiento de una norma es voluntario pero conveniente, ya que de esta forma se consiguen objetos o actividades intercambiables, conectables o asimilables. La norma sirve para describir los parámetros básicos de aquello que normaliza, por lo que puede darse el caso de que, cumpliendo los requisitos mínimos definidos por la norma, dos cosas pueden tener diferencias importantes o estén adaptadas a las circunstancias particulares de cada una de ellas.

ISO/IEC 29119 Software Testing.

En mayo de 2007 cuando ISO creó un grupo de trabajo (WG26) para desarrollar un nuevo "Estándar de Pruebas Software". El objetivo principal de esta norma, actualmente en desarrollo, es que sirva de referente internacional en el ámbito de las pruebas software y que permita tanto eliminar las inconsistencias existentes entre las actuales normas, así como cubrir aquellas áreas del "testing" que simplemente no habían sido tratadas hasta ahora en el resto de normas publicadas. Esta futura norma se conoce como "ISO/IEC 29119 Software Testing", cuya publicación en versión FIS (Final International Standard) está prevista para mediados del año 2012.

La ISO/IEC 29119 cubrirá el ciclo de vida completo, a través del análisis, diseño, implementación y mantenimiento de las pruebas software. Y se basa en las principales normas que en la actualidad son referentes en pruebas del software, modelos de procesos y ciclos de vida, principalmente:

- ✓ BSI (British Standards Institution): BS 7925-1, Software Testing: Part 1-Vocabulary y BS 7925-2, Software Testing: Part 2-Software Component Testing.

- ✓ IEEE: IEEE Std. 829, Software Test Documentation y IEEE Std 1008, Software Unit Testing, IEEE Std 1012-1998 Software Verification and Validation y IEEE Std 1028-1997 Software Reviews.
- ✓ ISO/IEC: ISO/IEC 12207, Software Life Cycle Processes, ISO/IEC 15289, System and Software Life Cycle Process Information Products y ISO/IEC TR 19759, Guide to the Software Engineering Body of Knowledge.

Alcance del Proyecto.

El presente texto trata de facilitar la evaluación y pruebas del software en base a ingeniería de pruebas según la norma. Para ello se irá contemplando, apartado por apartado, la totalidad de la norma ISO/IEC 29119, señalando en cada caso la posible redacción del manual y presentando una propuesta de los procedimientos, registros y demás documentación necesaria.

Con ello se pretende que las organizaciones, dispongan o no de un sistema documental ISO, puedan adaptar fácilmente su manual y sus procedimientos, añadiendo aquellos que respondan a prescripciones de la nueva norma, no contenidas en la anterior, o prepararlo desde sus inicios, aprovechando los modelos de documentación aquí propuestos.

Requisitos generales

El sujeto de la norma, o sea quien debe de aplicarla se define con “la organización o empresa” estableciendo para ella la obligación de redactar sobre documentos, implantar y mantener vigente un modelo para evaluación/pruebas del software en base a ingeniería de pruebas aplicando el estándar ISO/IEC 29119 en la empresa Omnisoft de la ciudad de

Quito. Dicho sistema debe estar sujeto a mejora continua con el fin de incrementar la eficacia de la organización en la tarea de alcanzar los objetivos que hayan sido señalados.

La modelo señala como característica del sistema de testing de calidad, un enfoque basado en los procesos, de forma que si se consigue mejorar todos aquellos que componen las actividades de la organización se conseguirá como consecuencia la mejora del producto por ellos elaborado o la del servicio a que puedan dar lugar. En este sentido la norma unifica el concepto que define el resultado de la organización y lo denomina "producto", incluyendo como es lógico, tanto los productos fabricados como los servicios prestados, sean o no canjeables por dinero.

El Modelo describe un sistema de calidad aplicable genéricamente a todas las organizaciones aplicando pruebas de software, sin importar su tipo, su tamaño o su personalidad jurídica, por lo que puede ser implantada en todo tipo de empresas, tanto industriales como de servicios, en entidades sin ánimo de lucro y en cualquier modelo de organización pública o privada.

FUNCIONALIDAD

A continuación se detalla el modo de funcionamiento de los distintos módulos de la aplicación en su versión WEB.

El siguiente manual le irá guiando a través de las distintas opciones de la aplicación, siguiendo un orden similar al que puede encontrarse en su modo de trabajo habitual.

ACCESO AL SISTEMA

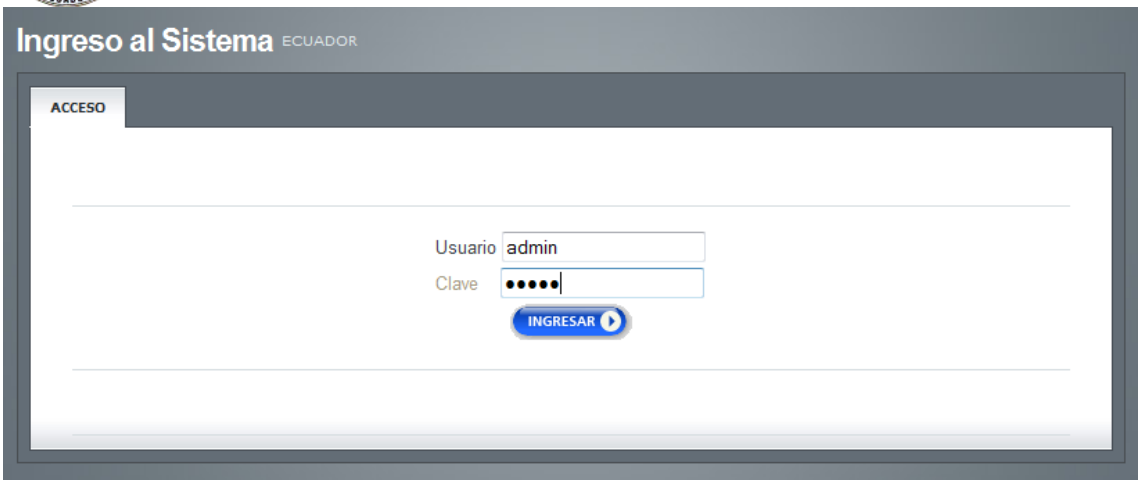
Para ingresar a la aplicación web se debe ir a la siguiente dirección web:
http://127.0.0.1/ISO/IE29119-Software Testing Ecuador

Los navegadores en los que trabaja la aplicación Web son los siguientes:

1. Mozilla Firefox 3.4.5 en adelante
2. Safari 4.0.3
3. Google Chrome 3.0.195.27
4. Opera 10
5. Internet Explorer 9

Sin embargo, la aplicación Web ha sido optimizada para funcionar y aprovechar las características de **Mozilla Firefox,** por lo tanto se recomienda acceder a la aplicación Web a través de dicho navegador.

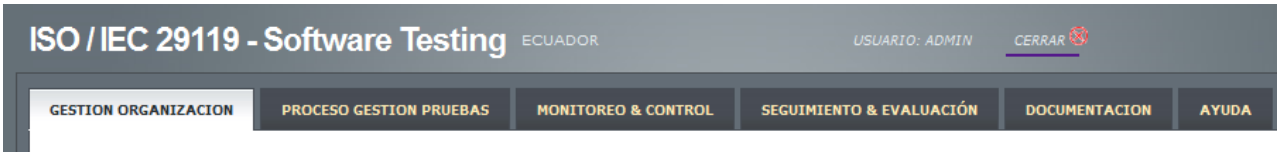
La entrada se realizará mediante una página de identificación mediante la introducción de código de usuario y clave. Cada usuario tendrá asociado un nivel con unos privilegios asociados, por ello, dependiendo del tipo de usuario que se trate, se habilitarán unas funciones u otras del programa. Una vez introducido un usuario y contraseña correcto, se accederá a la aplicación y se cargarán los módulos específicos definidos para el nivel de dicho usuario.



Esta información le habrá sido proporcionada por el Administrador del sistema una vez finalizado el proceso de registro. Se creará un par usuario/contraseña con permisos de administrador o usuario. A partir de cual se podrán crear el resto de usuarios y/o consultores.

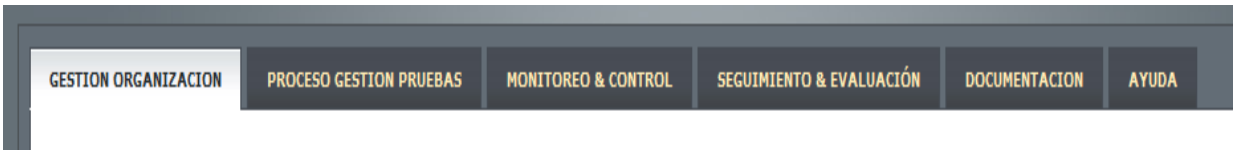
ASPECTO GENERAL DE LA APLICACIÓN.

Una vez que se ha validado correctamente en el sistema, se mostrará la página principal de la web. En la página principal de la Web, se puede observar un mensaje de bienvenida en el que se refleja el nivel del usuario que ha accedido al sistema (administrador, usuario o consultor). En la parte superior de la página se han dispuesto los enlaces con todas las funcionalidades ofrecidas por el Sistema.



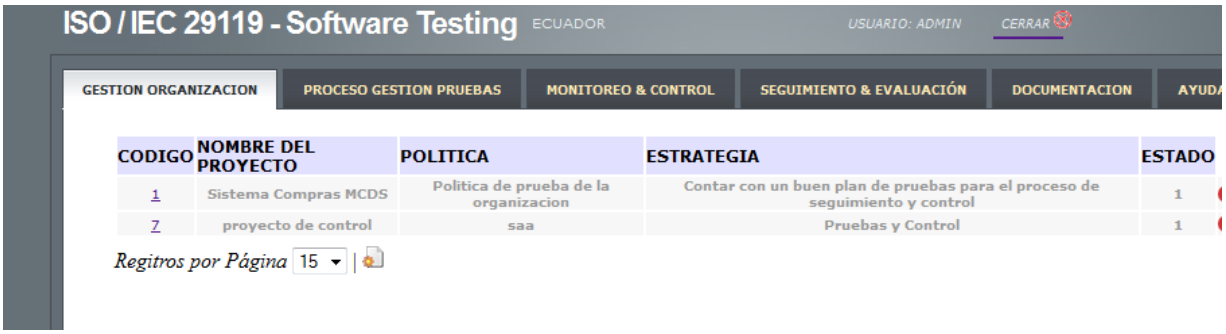
BARRA DE NAVEGACION.

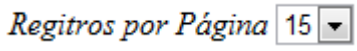
La barra de navegación de la aplicación permite una navegación a través de todo el sitio de donde se puede ingresar a las diferentes secciones de la aplicación y a la configuración de la misma.



La barra de navegación está compuesta por las siguientes secciones:

GESTION DE ORGANIZACION: En esta página se introducirá el nombre del proyecto, Política y Estrategia, además, cuenta para la creación y edición de su contenido.

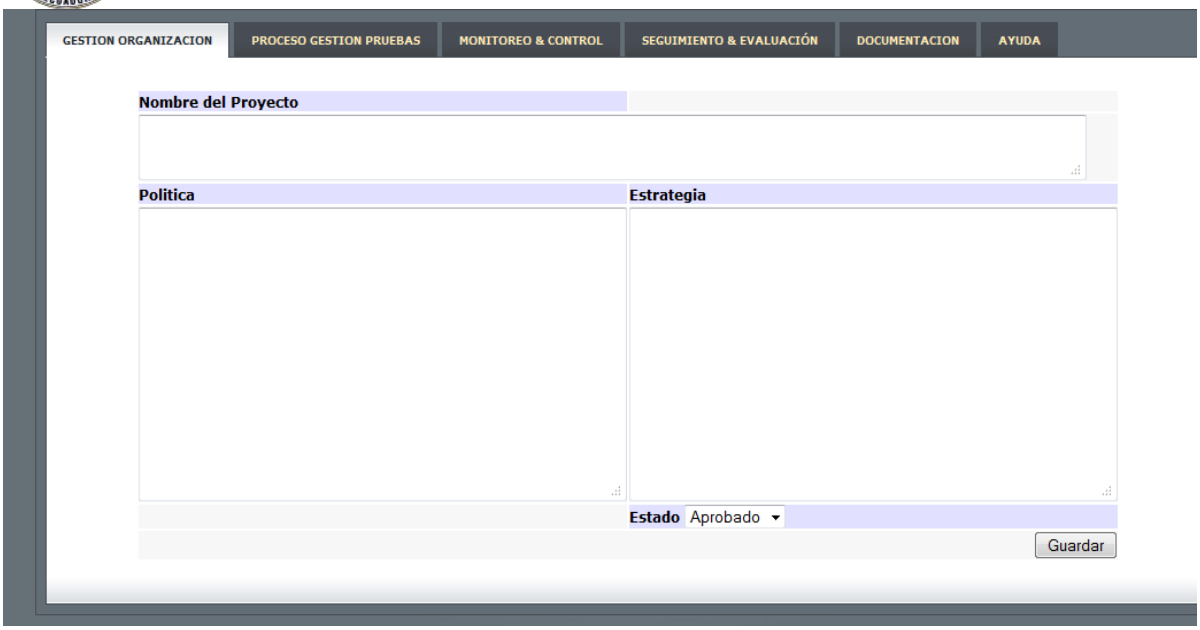




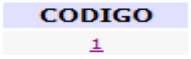
Esta opción nos permite visualizar el número de registros por página.

Nuevo. 

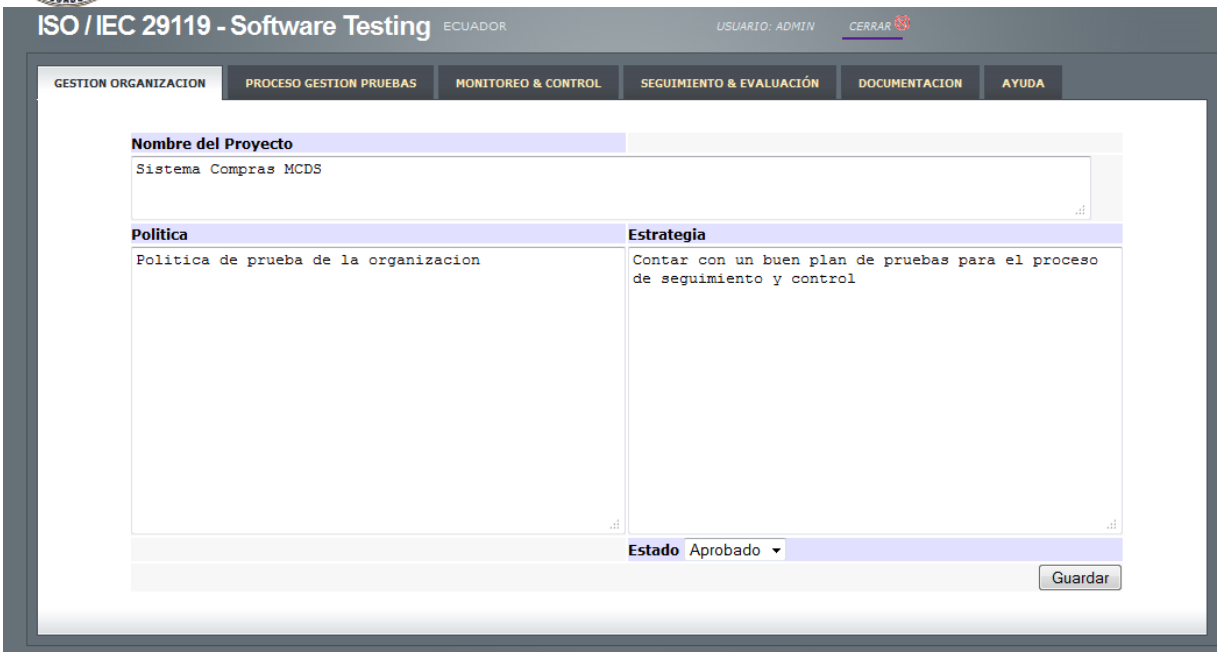
Permite crear un nuevo Proyecto en el sistema, el cual almacenará todos sus datos.



IMPORTANTE: Para que los cambios realizados sobre cualquier aspecto de la configuración tengan efecto, es necesario pulsar el botón **"Guardar"** situado en la parte inferior derecha de la página.

Modificar Proyecto. 

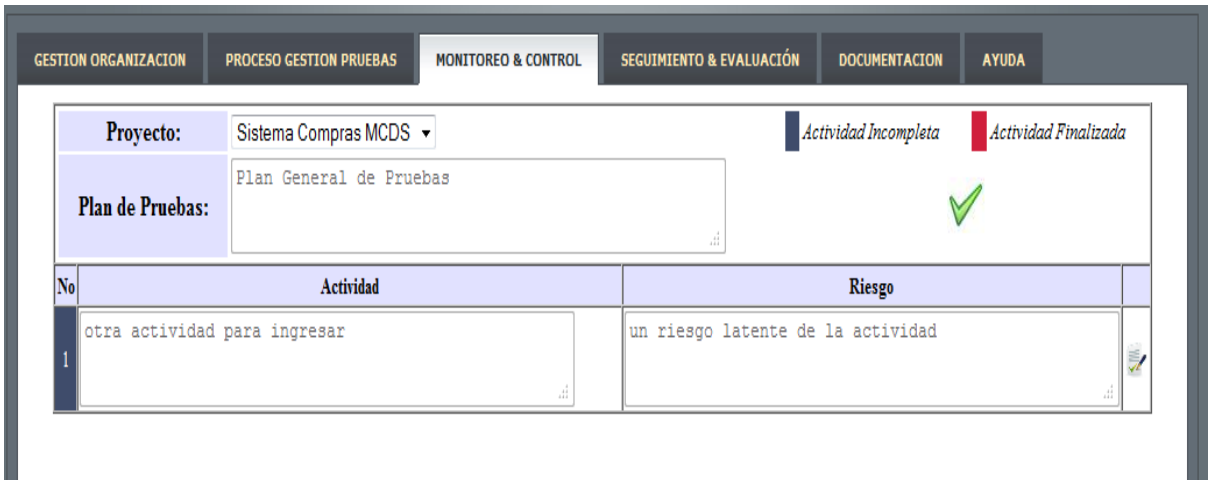
Permite modificar los datos del cliente en caso de que se los requiera.



IMPORTANTE: Para que los cambios realizados sobre cualquier aspecto de la configuración tengan efecto, es necesario pulsar el botón **"Guardar"** situado en la parte inferior derecha de la página.

PROCESO GESTION DE PRUEBAS: Esta página contiene todos los proyectos creados, su plan de pruebas, al igual que su actividad y riesgo.

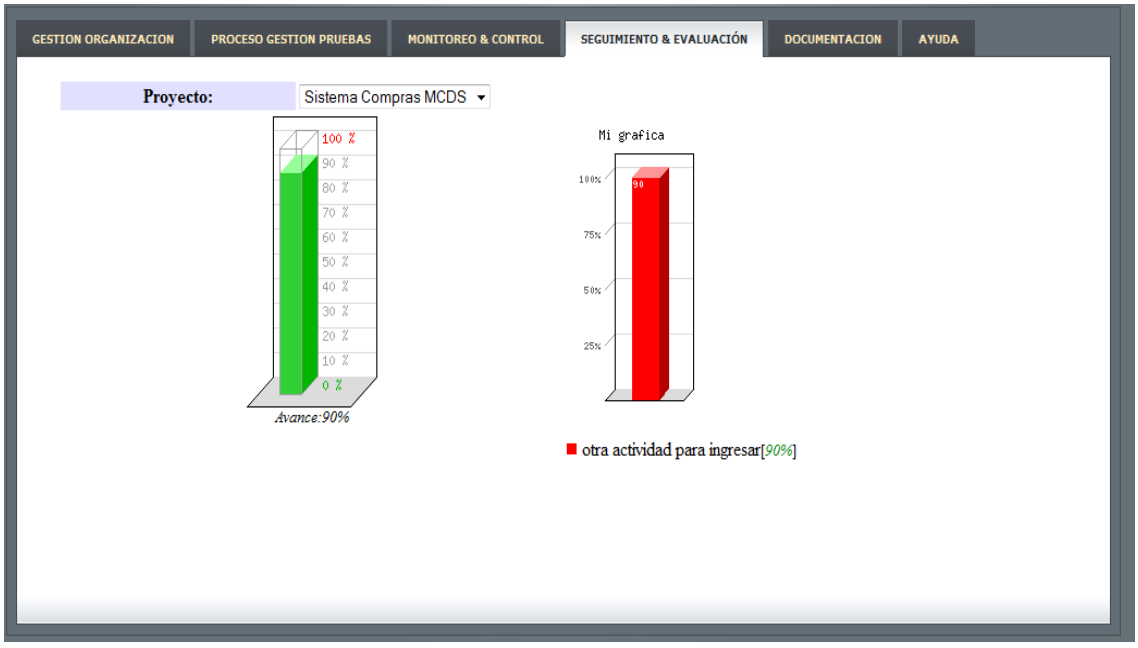
MONITOREO Y CONTROL: Como su nombre lo indica permite tener un monitoreo constante y control, al mismo tiempo permite visualizar la actividad en su estado.



. **MONITOREO:** Al dar un Clic en la pestaña de monitoreo se abre una ventana donde el usuario del sistema podrá ingresar la técnica, clase y puntaje en que se encuentra el proyecto.

IMPORTANTE: Para que los cambios realizados sobre cualquier aspecto de la configuración tengan efecto, es necesario pulsar el botón **"OK"** situado en la parte inferior derecha de la página y para modificar  o Eliminar  se deberá dar clic en los iconos indicados.

SEGUIMIENTO Y EVALUACION: Permite tener una visión más detallada del avance del proyecto en un porcentaje indicado.



DOCUMENTACION: Aquí están todos los documentos relacionados a la ISO/IEC 29119 en caso de que se lo requiera.

AYUDA: Aquí podrá encontrar el manual del funcionamiento del sistema.

MANUAL Y PROCEDIMIENTOS DE UN SISTEMA DE TESTING DE PRUEBAS DE SOFTWARE

En base a la norma ISO/IEC 29119

José Páez Escobar

2011-2012

INDICE PAG

PRÓLOGO

Como continuación de la serie de textos que han visto la luz en estos últimosaños, fruto de la colaboración entre la empresa***Omnisof***, se presenta hoy esta nueva publicación conel fin de facilitar a todas las empresas y entidades de nuestra región, en especiallas pequeñas y medianas, la implantación de un sistema de evaluación y pruebas deacuerdo con la norma internacional **ISO/IEC 29119**, en su versión del año 2007,que, junto con sus precedentes versiones, tan amplia difusión y consenso haalcanzado a nivel mundial.

Siguiendo la idea de la anterior publicación: "DISEÑO DE UN MODELO PARA EVALUACION/PRUEBAS DEL SOFTWARE EN BASE A INGENIERIA DE PRUEBAS APLICANDO EL ESTANDAR ISO/IEC 29119 EN LA EMPRESA OMNISOFT DE LA CIUDAD DE QUITO" y como complemento de ella, el presente textodesarrolla los distintos apartados de la norma, aclarando la posible interpretaciónde los mismos de acuerdo con lo indicado en la norma complementaria"ISO 9000:2000 Fundamentos y vocabulario" y con las manifestacionesexplícitas de expertos pertenecientes a organizaciones consultoras y certificadoras,reiteradamente expresadas en las revistas profesionales.

A continuación de los párrafos referentes a la explicación de cada apartado,se proponen textos alternativos para la confección de un Manual de calidad,se detalla su posible estructura y se presentan los distintos Procedimientosnecesarios para el desarrollo de un sistema de gestión de calidad.

De esta forma se pretende que cualquier empresa pueda llegar a comprenderel espíritu de la norma y afrontar la confección de la documentación de susistema de calidad.

INTRODUCCIÓN

¿Qué es ISO?

ISO (International Standard Organization) u Organización Internacional deNormalización, es un organismo que se dedica a publicar normas a escala internacionaly que, partiendo de una norma ya existente de British Standard:

BS-5720, ha venido confeccionando la serie de normas ISO 9000, referidasa los Sistemas de la Calidad, desde hace varios años. La primera versión esde 1987 y sufrió una profunda revisión en 1994, por lo que esta nueva redacciónde año 2000 supone la tercera modificación de su texto.

¿Qué es una norma?

Una norma es un documento que describe un producto o una actividad conel fin de que las cosas sean similares. El cumplimiento de una norma es voluntariopero conveniente, ya que de esta forma se consiguen objetos o actividadesintercambiables, conectables o asimilables. La norma sirve para describirlos parámetros básicos de aquello que normaliza, por lo que puede darseel caso de que, cumpliendo los requisitos mínimos definidos por la norma, doscosas pueden tener diferencias importantes o estén adaptadas a las circunstanciasparticulares de cada una de ellas.

ISO/IEC 29119 Software Testing.

En mayo de 2007 cuando ISO creó un grupo de trabajo (WG26) para desarrollar un nuevo "Estándar de Pruebas Software". El objetivo principal de esta norma, actualmente en desarrollo, es que sirva de referente internacional en el ámbito de las pruebas software y que permita tanto eliminar las inconsistencias existentes entre las actuales normas, así como cubrir aquellas áreas del "testing" que simplemente no habían sido tratadas hasta ahora en el resto de normas publicadas. Esta futura norma se conoce como "ISO/IEC 29119

Software Testing", cuya publicación en versión FIS (Final International Standard) está prevista para mediados del año 2012.

La ISO/IEC 29119 cubrirá el ciclo de vida completo, a través del análisis, diseño, implementación y mantenimiento de las pruebas software. Y se basa en las principales normas que en la actualidad son referentes en pruebas del software, modelos de procesos y ciclos de vida, principalmente:

- ✓ BSI (British Standards Institution): BS 7925-1, Software Testing: Part 1-Vocabulary y BS 7925-2, Software Testing: Part 2-Software Component Testing.
- ✓ IEEE: IEEE Std. 829, Software Test Documentation y IEEE Std 1008, Software Unit Testing, IEEE Std 1012-1998 Software Verification and Validation y IEEE Std 1028-1997 Software Reviews.
- ✓ ISO/IEC: ISO/IEC 12207, Software Life Cycle Processes, ISO/IEC 15289, System and Software Life Cycle Process Information Products y ISO/IEC TR 19759, Guide to the Software Engineering Body of Knowledge.

Consta de cuatro partes:

5. Conceptos y Vocabulario
6. Proceso de Pruebas
7. Documentación de Pruebas
8. Técnicas de Prueba

De acuerdo con el plan de trabajo aprobado por ISO en una primera fase se elaboran las partes 2 y 3, y posteriormente las 1 y 4.

Alcance del Proyecto.

El presente texto trata de facilitar la evaluación y pruebas del software en base a ingeniería de pruebas según la norma. Para ello se irá contemplando, apartado por apartado,la totalidad de la norma ISO/IEC 29119, señalando en cada caso la posibleredacción del Manual y presentando una propuesta de los procedimientos,registros y demás documentación necesaria.

Con ello se pretende que las organizaciones, dispongan o no de un sistemadocumental ISO, puedan adaptar fácilmente su Manual y sus procedimientos,añadiendo aquéllos que respondan a prescripciones de la nueva norma, nocontenidas en la anterior, o prepararlo desde sus inicios, aprovechando losmodelos de documentación aquí propuestos.

Requisitos generales

El sujeto de la norma, o sea quien debe de aplicarla se define con "la organización o empresa"estableciendo para ella la obligación de redactar sobre documentos, implantary mantener vigente un modelo para evaluación/pruebas del software en base a ingeniería de pruebas aplicando el estándar ISO/IEC 29119 en la empresa Omnisoft de la ciudad de quito. Dicho sistema debe estar sujeto a mejora continua al objeto de incrementarla eficacia de la organización en la tarea de alcanzar los objetivos que hayansido señalados.

La modelo señala como característica del sistema de testing de calidad, un enfoque basadoen los procesos, de forma que si se consigue mejorar todos aquéllos que componenlas actividades de la organización se conseguirá como consecuencia la mejoradel producto por ellos elaborado o la del servicio a que puedan dar lugar. Eneste sentido la norma unifica el concepto que define el resultado de la organizacióny lo denomina

“producto”, incluyendo como es lógico, tanto los productosfabricados como los servicios prestados, sean o no canjeables por dinero.

El Modelo describe un sistema de calidad aplicable genéricamente a todas las organizaciones aplicando pruebas de software,sin importar su tipo, su tamaño o su personalidad jurídica, por loque puede ser implantada en todo tipo de empresas, tanto industriales como deservicios, en entidades sin ánimo de lucro y en cualquier modelo de organizaciónpública o privada.

FUNCIONALIDAD

A continuación se detalla el modo de funcionamiento de los distintos módulos de la aplicación, tanto en su versión WEB.

El siguiente manual le irá guiando a través de las distintas opciones de la aplicación, siguiendo un orden similar al que puede encontrarse en su modo de trabajo habitual.

ACCESO AL SISTEMA

Para ingresar a la aplicación web se debe ir a la siguiente dirección web:
http://127.0.0.1/ISO/IE29119-Software Testing Ecuador

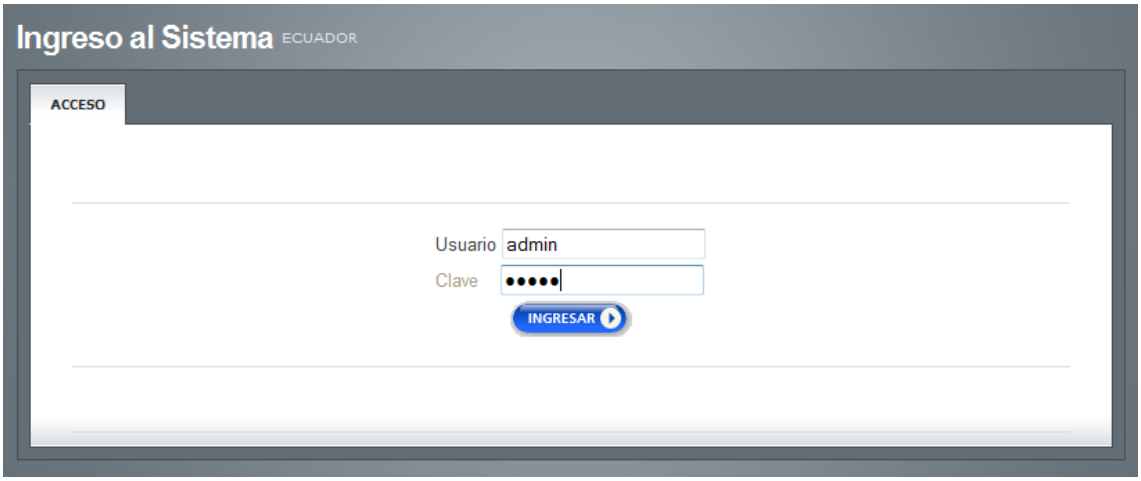
Los navegadores en los que trabaja la aplicación Web son los siguientes:

6. Mozilla Firefox 3.4.5 en adelante
7. Safari 4.0.3

8. Google Chrome 3.0.195.27
9. Opera 10
10. Internet Explorer 9

Sin embargo la aplicación Web ha sido optimizada para funcionar y aprovechar las características de **Mozilla Firefox,** por lo tanto se recomienda acceder a la aplicación Web a través de dicho navegador.

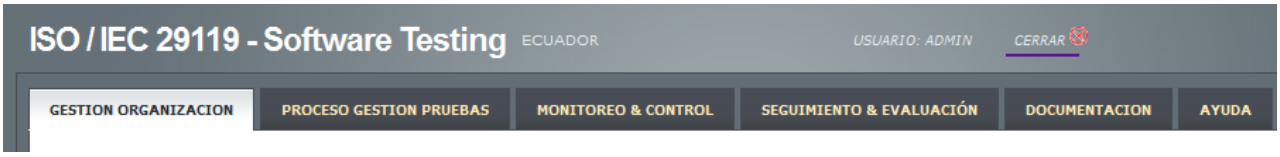
La entrada se realizará mediante una página de identificación mediante la introducción de Código de Usuario y Clave. Cada usuario tendrá asociado un nivel con unos privilegios asociados, por ello, dependiendo del tipo de usuario que se trate, se habilitarán unas funciones u otras del programa. Una vez introducido un usuario y contraseña correcto, se accederá a la aplicación y se cargarán los módulos específicos definidos para el nivel de dicho usuario.



Esta información le habrá sido proporcionada por Administrador del sistema una vez finalizado el proceso deregistro. Se le creará un par usuario/contraseña con permisos de administrador o usuario. A partir decual se podrán crear el resto de usuarios y/o consultores.

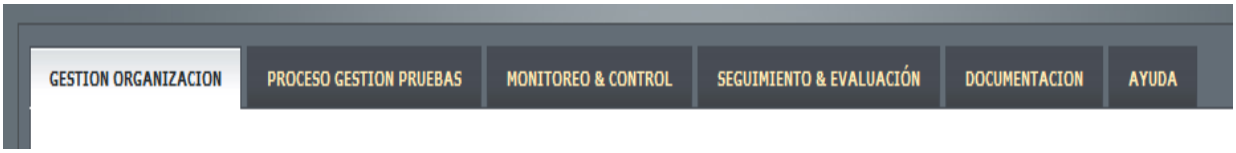
ASPECTO GENERAL DE LA APLICACIÓN.

Una vez que se ha validado correctamente en el sistema, se mostrará la página principal de la web. En la página principal de la Web, se puede observar un mensaje de bienvenida en el que se refleja el nivel del usuario que ha accedido al sistema (administrador, usuario o consultor). En la parte superior de la página se han dispuesto los enlaces con todas las funcionalidades ofrecidas por el Sistema.



BARRA DE NAVEGACION.

La barra de navegación de la aplicación permite una navegación a través de todo el sitio de donde se puede ingresar a las diferentes secciones de la aplicación y a la configuración de la misma.



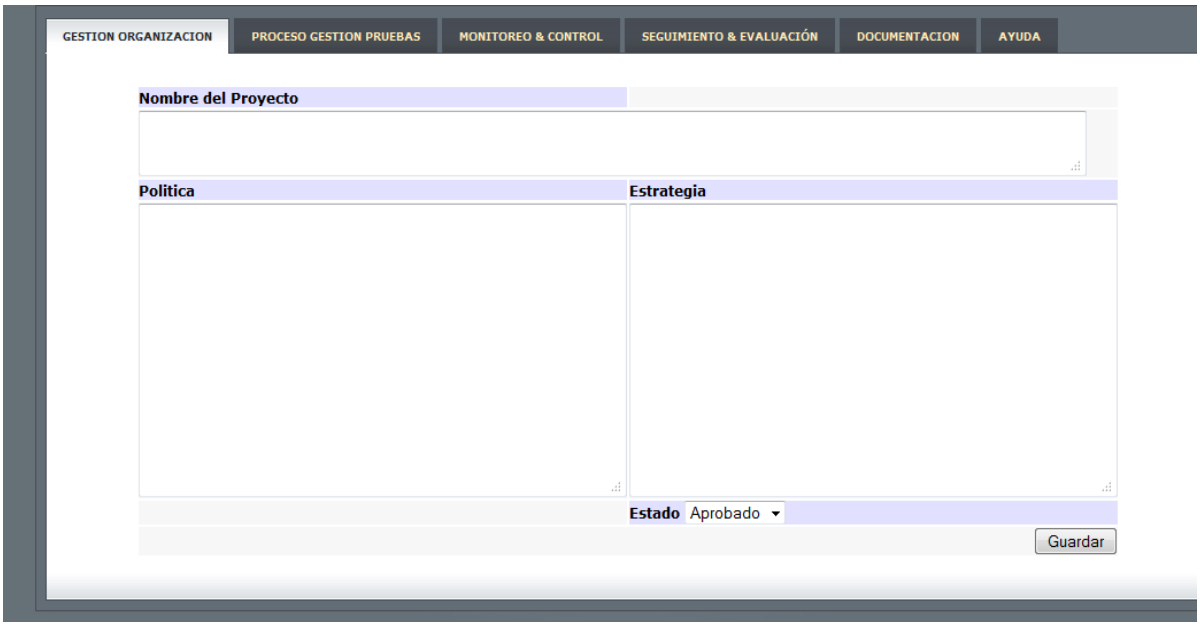
La barra de navegación está compuesta por las siguientes secciones:

GESTION DE ORGANIZACION: Es la página está el nombre del proyecto, Política y Estrategia y además cuenta para la creación y edición de su contenido.

Regitros por Página 15 Esta opción nos permite visualizar el número de registros por página.

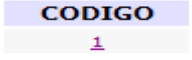
Nuevo.

Permite crear un nuevo Proyecto en el sistema, el cual almacenara todos sus datos.

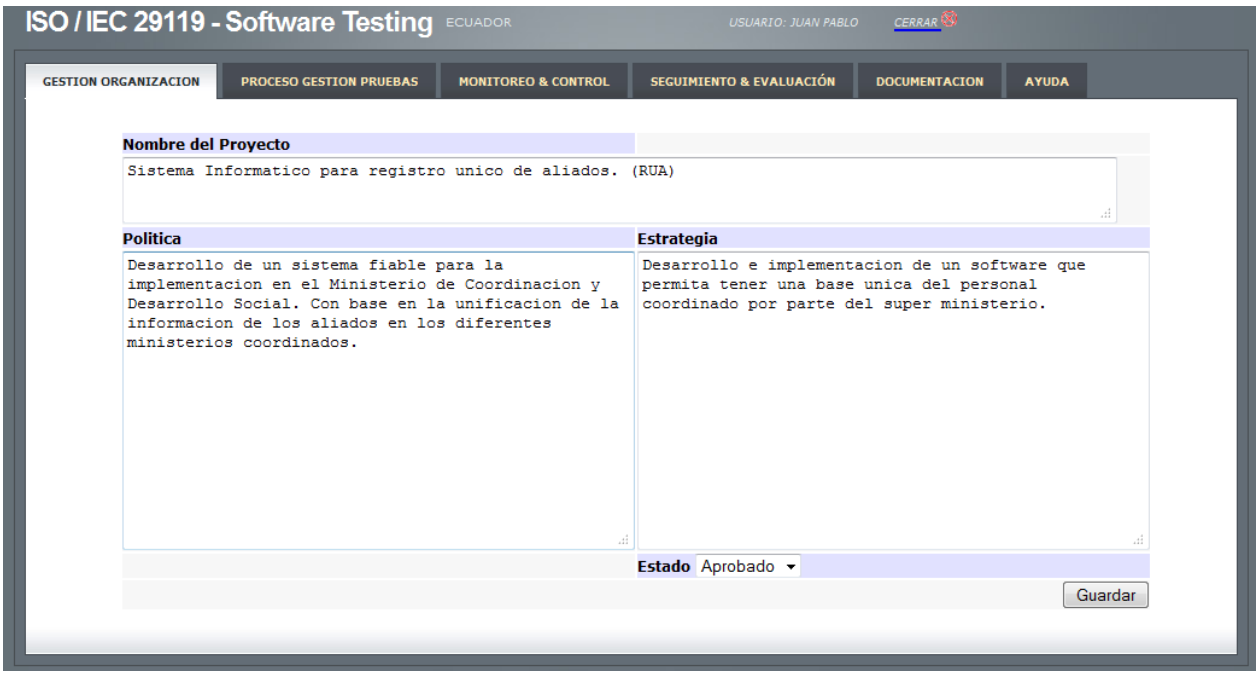


IMPORTANTE: Para que los cambios realizados sobre cualquier aspecto de la configuración tengan efecto, es necesario pulsar el botón **"Guardar"** situado en la parte inferior derecha de la página.

ModificarProyecto.

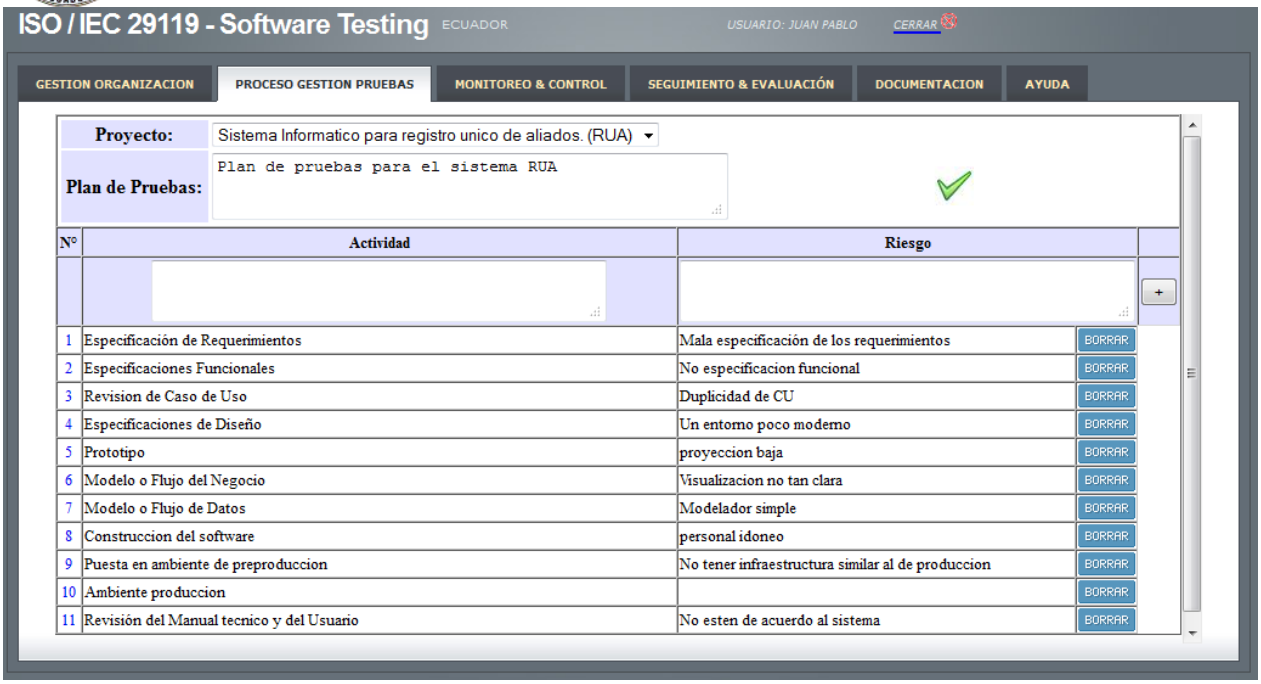


Permite modificar los datos del cliente en caso de que se los requiera.

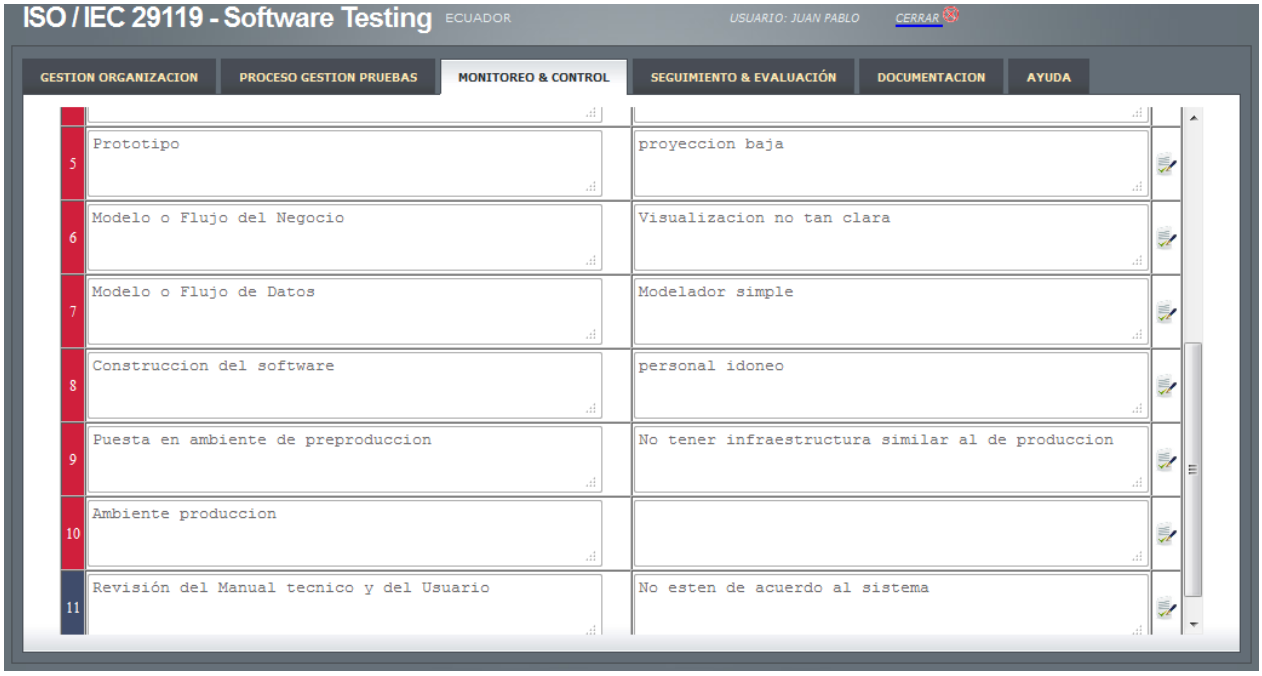


IMPORTANTE: Para que los cambios realizados sobre cualquier aspecto de la configuración tengan efecto, es necesario pulsar el botón **"Guardar"** situado en la parte inferior derecha de la página.

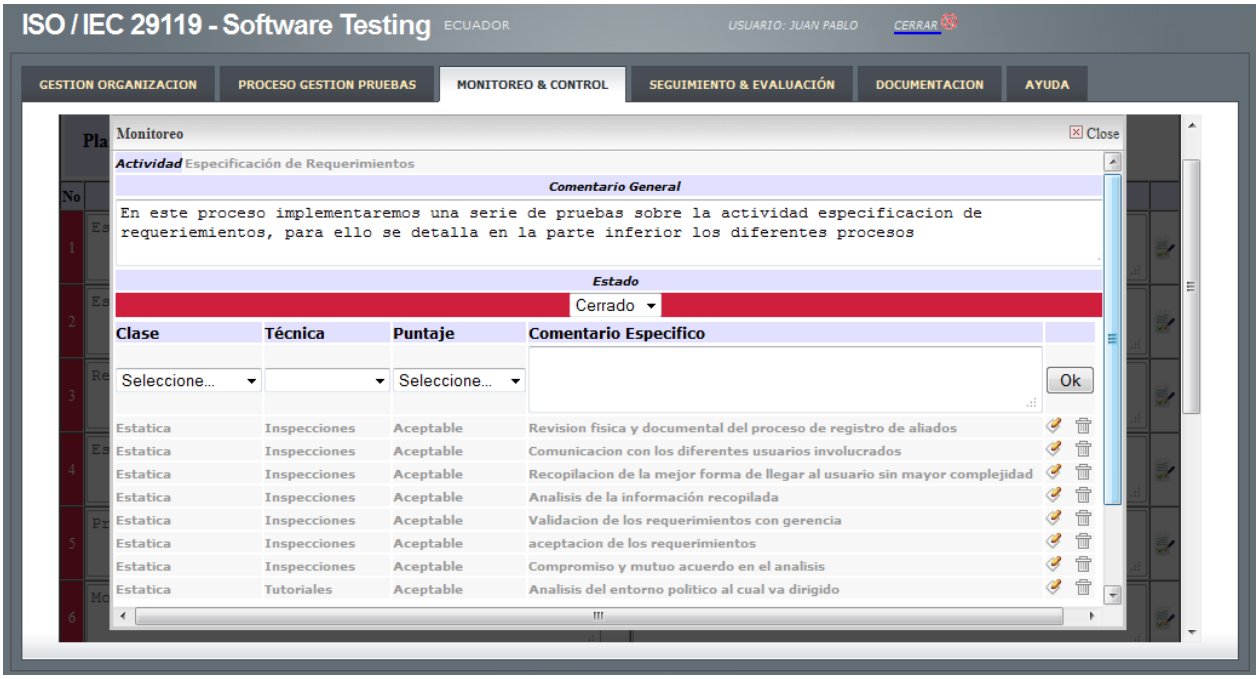
PROCESO GESTION DE PRUEBAS: En esta página contiene todos los proyectos creados, su plan de pruebas, al igual que su actividad y riesgo.



MONITOREOY CONTROL: Como su nombre lo indica permite tener un monitoreo constante y control, al mismo tiempo permite visualizar la actividad en su estado.

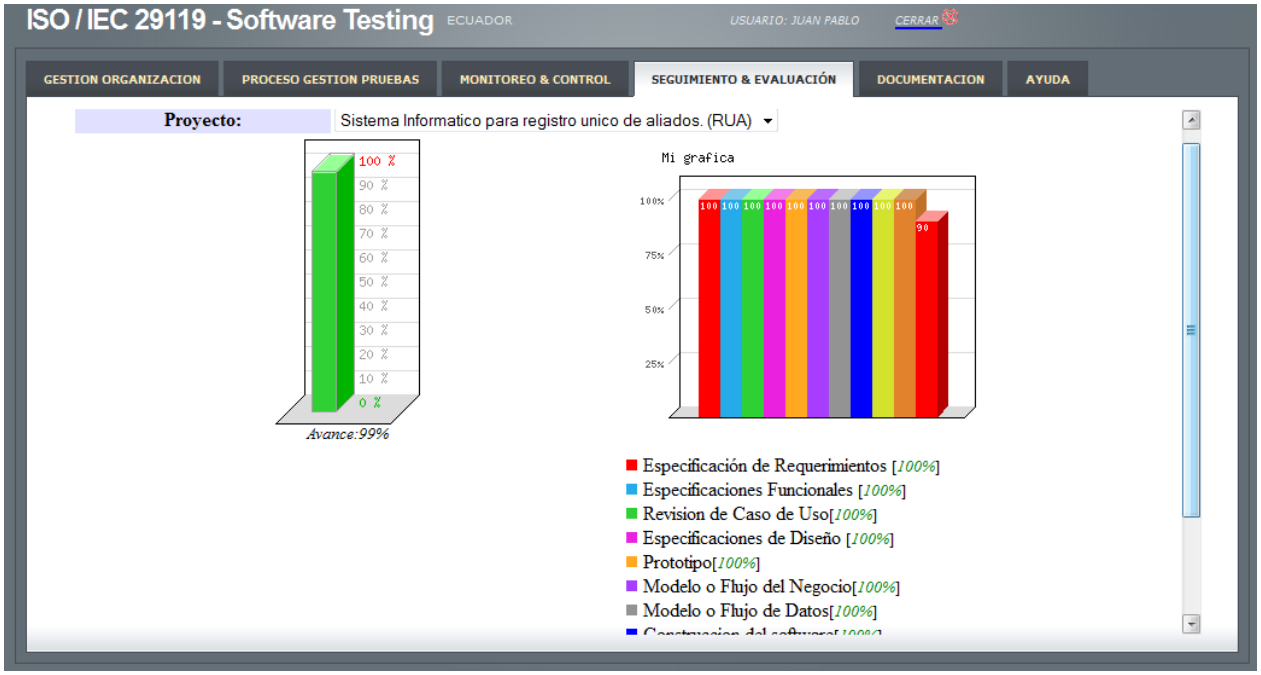


. **MONITOREO:** Al dar un Clic en la pestaña de monitoreo se abre una ventana donde el usuario del sistema podrá ingresar la técnica, clase y puntaje en que se encuentra el proyecto.

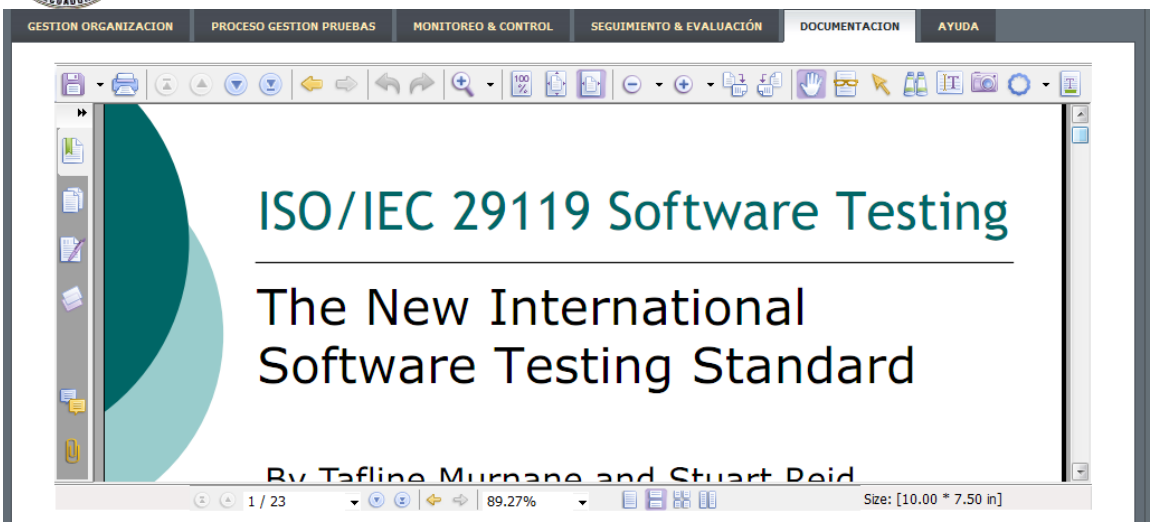


IMPORTANTE: Para que los cambios realizados sobre cualquier aspecto de la configuración tengan efecto, es necesario pulsar el botón **"OK"** situado en la parte inferior derecha de la página y para modificar  o Eliminar  se deberá dar clic en los iconos indicados.

SEGUIMIENTO Y EVALUACION: En esta parte nos permite tener una visión más detallada del avance del proyecto en un porcentaje indicado.



DOCUMENTACION: Aquí están todos los documentos relacionados a la ISO/IEC 29119 en caso de que se lo requiera.



AYUDA: Aquí podrá encontrar el manual del funcionamiento del sistema en caso de que se lorequiera**.**