METODOLOGÍAS DE TESTING DE SOFTWARE Y SU APLICACIÓN EN EL CENTRO DE INFORMÁTICA DE LA UNIVERSIDAD EAFIT

ALEJANDRO CÁLAD ÁLVAREZ
JUAN DAVID RUÍZ CALLE

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE INGENIERIA DE SISTEMAS
AREA DE CALIDAD DE SOFTWARE
MEDELLIN
2009

METODOLOGÍAS DE TESTING DE SOFTWARE Y SU APLICACIÓN EN EL CENTRO DE INFORMÁTICA DE LA UNIVERSIDAD EAFIT

ALEJANDRO CÁLAD ÁLVAREZ
JUAN DAVID RUÍZ CALLE

Proyecto de grado para optar al título de
Ingeniero de Sistemas

Asesor: Carlos Hernando Montoya
Ingeniero de Sistemas

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS
AREA DE CALIDAD DE SOFTWARE
MEDELLÍN
2009

Nota de aceptación:

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Medellín, octubre de 2009

De Alejandro

A mi familia con todo mi amor, especialmente a mis tíos Tavo y Patricia por apoyarme durante esta etapa de mi vida, aconsejarme y darme fuerzas para seguir adelante.

De Juan David

Principalmente a Dios por darme la salud física y mental para realizar todas mis actividades diarias; a mis padres, Jaime y Luz, por apoyarme desde principio a fin y mostrarme el camino de bien, hoy soy lo que soy por ellos. Por último, a mi novia María Victoria, por tanta compresión y amor.

AGRADECIMIENTOS

Agradecemos a nuestro asesor Carlos Hernando Montoya, Jefe del Centro de Informática de la Universidad EAFIT, por sacar de su apretada agenda el tiempo necesario para orientarnos en este proyecto.

A todas las personas del Centro de Informática de la Universidad que nos colaboraron de manera desinteresada, resolviendo nuestras dudas y aportando de una u otra forma para poder sacar este proyecto adelante. A Patricia, la secretaria, por organizar esas citas tan oportunas con el “Jefe” y permitirnos conseguir la reunión tan necesitada.

CONTENIDO

Pág.

LISTA DE TABLAS

Pág.

LISTA DE ILUSTRACIONES

Pág.

LISTA DE ANEXOS

Pág.

RESUMEN

Este proyecto de grado está dividido en tres grandes secciones. La primera sección describe detalladamente los principales fundamentos teóricos sobre pruebas de software, incluyendo temas como: Calidad de Software, Estrategias de prueba de software, Niveles de prueba y Técnicas de Prueba. Con base a los fundamentos teóricos presentados se desarrollan las dos secciones restantes del proyecto, una de ellas, la conformación de la Unidad de Aseguramiento de Calidad dentro del Centro de Informática de la Universidad EAFIT y la distribución y asignación de tareas que realiza para suplir la imperiosa necesidad de realizar procesos de pruebas y mejorar la calidad de los productos desarrollados. La segunda, propone una metodología sencilla y eficaz que se acople a las necesidades actuales del Centro de Informática de la Universidad EAFIT, enfocada principalmente en la ejecución de pruebas de sistema (Pruebas de interfaz de usuario, de integridad, de carga y stress).

Palabras Claves: Pruebas de software. Aseguramiento de Calidad. Calidad de Software. Metodología de pruebas. Centro de Informática.

ABSTRACT

This grade project is divided into three sections. The first of them describe the main theories foundations about software testing, including subjects like: Software Quality, Software Testing Strategies, Testing Levels and Testing Techniques. The next two sections of the project will be based on the theories foundations presented in the previous section. One of them will be the conformation of a Unit for the Software Quality Assurance, and distribution and assignation of tasks inside the unit, fulfilling the necessity of doing tests and improving the quality of product developed. This unit will be located inside the Informatics Center of EAFIT University. The last section, is a proposal of a clear and effective methodology able to fulfill the actual needs of Informatics Center of the EAFIT University, and principally focused on the execution of system tests (User Interface Tests, Integrity Tests, Load Tests and Stress Tests).

Key Words: Software Testing. Quality Assurance. Software Quality. Testing Methodology. Informatic Center.

INTRODUCCIÓN

A todo desarrollo de software es necesario realizarle pruebas de calidad que aseguren el correcto funcionamiento de la aplicación en todos sus aspectos, de ahí, la importancia que adquiere el "testing" de software dentro del ciclo de vida del desarrollo. Entiéndase por testing de software:

Un proceso concurrente del ciclo de vida de ingeniería, usando y manteniendo el testware[1] en orden de medir y mejorar la calidad del software que está siendo probado (Craig y Jaskiel, 2006).

Esta definición no puede ser tomada como un guía universal para definir testing ya que la práctica varía de acuerdo a cada proyecto y además, algunos autores como Roger Pressman, aseguran que el testing de software es una práctica más empírica que metodológica, de ahí la dificultad dentro de los equipos de desarrollo de software a la hora de probar sus aplicaciones.

La Universidad EAFIT cuenta con un equipo sólido de desarrollo, el cual ha creado diferentes aplicaciones con el fin de dar solución a las necesidades informáticas de la Universidad. Dentro del ciclo de desarrollo, el equipo de analistas realiza todos los pasos excepto el de testing, ya que surgen problemas de tipo ético y moral, más que técnicos, que impiden a los desarrolladores ejecutar la labor de testing en una forma eficiente.

"Desde el punto de vista psicológico, el análisis y el diseño del software son tareas constructivas. Cuando comienza la prueba hay un intento sutil, pero definitivo, de 'romper' lo que ha construido el ingeniero de software" (Pressman, 2006).

[1] Testware es cualquier documento o producto creado como parte del esfuerzo de prueba (casos de prueba, planes de prueba, etc.).

Por todo lo anterior, se quiere recopilar documentación sobre metodologías apropiadas y actualizadas de testing que soporten la creación de un equipo de trabajo al interior del Centro de Informática de la Universidad EAFIT, apoyado por el área de desarrollo, para realizar todas las pruebas de software a las aplicaciones desarrolladas internamente y de esta forma asegurar la calidad del producto final.

Para este proyecto de grado se harán uso de los conceptos, técnicas y mejores prácticas que resulten del marco teórico en un caso real, el aplicativo Asignación Docente de la Universidad (ZEUS), y si la magnitud del problema lo permite, se aplicarán pruebas a otra aplicación del Centro de Informática desarrollada para el Centro de Idiomas (ACADI).

1. MARCO TEÓRICO

1.1 CALIDAD DEL SOFTWARE

1.1.1 ¿Qué es calidad de software?

Desde los inicios de la ingeniería del software se ha intentado formalizar una definición de calidad que esté enmarcada dentro del contexto del desarrollo de software. Muchos autores están de acuerdo con la importancia que tiene la calidad del software para obtener aplicaciones con un rendimiento óptimo que cumplan con todos los requisitos establecidos por el cliente.

Tomando como base una definición genérica de calidad, desarrollada en el estándar ISO 9000,"*grado en el que un conjunto de características inherentes cumple con los requisitos*", se puede empezar a concebir calidad aplicada al desarrollo de software. Algunos autores han enmarcado la definición de calidad dentro de este contexto.

Autores como William E. Lewis en su libro Software "Testing And Continuous Quality Improvement" definen calidad de software desde dos puntos de vista[2]. El primero, hace alusión al cumplimiento de requerimientos; de acuerdo a eso, obtener un producto de calidad implica tener requerimientos medibles, y que se cumplan. Con este significado la calidad de un producto pasa a ser un estado binario en donde se tiene o no calidad. El segundo, tiene en cuenta al cliente; el cual define calidad de un producto o servicio como el cumplimiento de las necesidades presentadas por el mismo. Otra forma de expresar lo mencionado anteriormente es la capacidad del producto de "ajustarse para el uso".

[2] Lewis, William E. Software Testing and Continuous Quality Improvement. Awerbach. 2000. p. 3

Una definición más concreta de calidad de software, propuesta por Roger S. Pressman[3], sugiere que "*la calidad de software es el cumplimiento de los requisitos de funcionalidad y desempeño explícitamente establecidos, de los estándares de desarrollo explícitamente documentados y de las características implícitas que se esperan de todo software desarrollado profesionalmente*".

Se puede ver que los dos autores exponen versiones similares de la definición de calidad de software y en general en el mundo la definición puede variar o extenderse dependiendo del autor, y no es el objetivo de este proyecto decidir cuál es la definición correcta, sin embargo se puede concluir que la calidad de software es una compleja combinación de factores que varían entre las diferentes aplicaciones y los clientes que las solicitan.

1.1.2 Factores que determinan la calidad del software

En el año de 1970, McCall propuso unos factores de calidad que siguen vigentes en la actualidad, lo que quiere decir que los factores que afectan la calidad del software no cambian en el tiempo. Estos factores se dividen en dos grandes grupos; aquellos que pueden ser medidos directamente, es decir aquellos defectos descubiertos en las pruebas; y factores que pueden ser medidos únicamente de manera indirecta, como por ejemplo el mantenimiento y la facilidad de uso.

McCall y su equipo de trabajo plantearon una clasificación basada en tres aspectos importantes de todo producto de software: operación, que *incluye corrección, confiabilidad, usabilidad, integridad* y *eficiencia*; transición, compuesta por *portabilidad, reutilización* y *compatibilidad*; y por último revisión de un producto, donde se encuentran factores como *facilidad de mantenimiento,*

[3] Pressman, Roger S. Ingeniería del Software, un enfoque práctico. Sexta edición. Ed. MacGraw Hill, 2006, p. 463.

flexibilidad y *facilidad de prueba*. A continuación se proporcionara una breve descripción de cada factor previamente citado:

- Corrección: está compuesto por dos aspectos; cumplimiento por parte del software de las especificaciones técnicas, y satisfacción de objetivos del cliente.
- Confiabilidad: Expectativa que un programa desempeñe la función para la cual está hecho.
- Usabilidad: Esfuerzo necesario para aprender, operar y preparar los datos de entrada de un programa e interpretar la salida.
- Integridad: Grado de control sobre el acceso al software o la manipulación de los datos por parte de personas no autorizadas.
- Eficiencia: Cantidad de recursos necesarios para que un programa realice su función.
- Portabilidad: Esfuerzo necesario para transferir el programa de un entorno de hardware o software a otro.
- Reutilización: Grado en que un programa o parte de él puede usarse en otras aplicaciones.
- Compatibilidad: Esfuerzo necesario para acoplar un sistema con otro.
- Facilidad de mantenimiento: Esfuerzo necesario para localizar y corregir errores en un programa.
- Flexibilidad: Esfuerzo necesario para modificar un programa en operación.
- Facilidad de prueba: Esfuerzo que demanda un programa con el fin de asegurar que cumpla su función

Se puede observar que muchos de los factores previamente mencionados se convierten en grupo de factores cuantitativos y por tanto, desarrollar medidas que permitan realizar una adecuada medición se convierte en una actividad frustrante debido a la naturaleza subjetiva de la misma.

1.1.3 Aseguramiento de la calidad del software

Se puede definir el aseguramiento de la calidad del software como "las actividades sistemáticas que proveen evidencia del uso apropiado de la capacidad total del software"[4]. En otras palabras, el aseguramiento de la calidad del software es una colección de actividades y funciones usadas para monitorear y controlar un proyecto de software desde su concepción hasta la salida en vivo, con el fin de que los objetivos específicos se cumplan con un nivel deseado de confianza.

El aseguramiento de la calidad es un esfuerzo planeado para asegurar que el producto (software) cumpla con los criterios y los factores de calidad mencionados en el numeral anterior. Estas actividades y funciones no solo son responsabilidad del equipo asegurador de calidad, sino que incluyen también al gerente/líder del proyecto, el personal del proyecto y los usuarios.

La definición de *aseguramiento* plantea, que sí los procesos planteados dentro del plan de calidad son seguidos de acuerdo a los lineamientos pactados, asegurarán la gestión de la calidad del producto.

Los objetivos de la calidad del software se logran siguiendo el plan de aseguramiento de calidad de software, el cual establece los métodos que el proyecto debe emplear para asegurar que los documentos y/o productos generados y revisados en cada etapa, sean de alta de calidad.

[4] Lewis, Op. Cit, p. 6

El aseguramiento de la calidad del software es una estrategia adoptada por la gestión del riesgo. Considerar la calidad de software dentro de la gestión del riesgo es importante porque en muchas ocasiones la calidad tiene un alto costo en los proyecto de software. A continuación, se mostrarán algunos ejemplos de mala calidad en materia de software:

- Fallas frecuentes en la funcionalidad del software.
- Consecuencias secundarias de fallas en el software, como problemas financieros.
- Sistemas no disponibles cuando se requiere.
- Costosas mejoras en el software
- Altos costos en la detección y corrección de errores.

Parte de los riesgos con respecto a la calidad están relacionados a los defectos del producto. Un defecto es una falla o falta de cumplimiento con un determinando requisito. Si dicho requisito está planteado de una forma incorrecta, generará una mayor posibilidad de que ocurran riesgos, teniendo como resultado la aparición de posibles defectos posteriores en la aplicación y productos poco verificables. Algunas de las técnicas y estrategias de gestión de riesgos incluyen Testing de software, revisiones técnicas, revisión de pares (peers) y verificación de conformidad.

El aseguramiento de la calidad de software posee tres componentes expuestos a continuación:

- *Testing de software*: Componente usado para verificar que los requisitos funcionales de una aplicación se cumplan. El Testing de software tiene limitaciones en cuanto al aseguramiento de la calidad se requiere, ya que una vez hecho el Testing, se dejarán descubiertos posibles defectos que

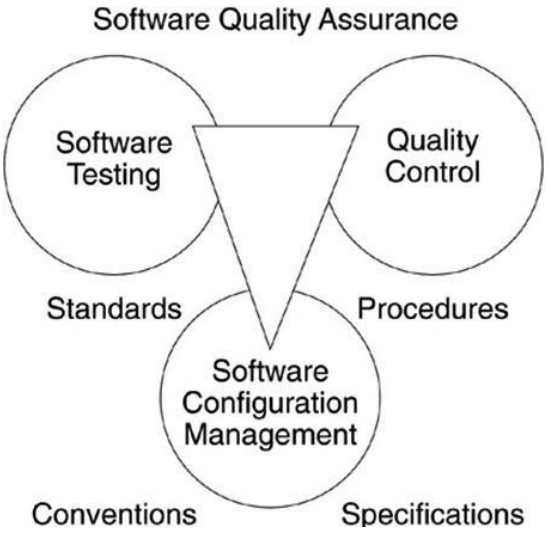
cuestionan la calidad del producto evaluado, con poco o nada de tiempo para mitigar dicha falta de calidad. En el capitulo siguiente se ampliará la información correspondiente a este componente ya que es el tema principal de este proyecto.

- *Control de calidad:* Compuesta por métodos y procesos usados para monitorear el trabajo y observar si los requisitos son cumplidos. Se enfoca en la revisión y posterior eliminación de defectos antes de la entrega del producto final. El control de calidad debe ser responsabilidad de la unidad organizacional que está encargada de generar el producto. En general el control de calidad consiste en una serie de revisiones (incluida en el plan de aseguramiento de la calidad) realizadas a un producto. Para el caso de los productos de software, el control de calidad generalmente incluye revisión de las especificaciones, inspecciones de código y documentación, y revisiones para los entregables. Usualmente las inspecciones del producto y de la documentación son llevadas a cabo en cada etapa del ciclo de vida de software, para demostrar que los ítems producidos cumplen con los criterios especificados en el plan de aseguramiento de la calidad. Estos criterios están determinados por las especificaciones de los requisitos, por la documentación del diseño, y por los planes de prueba. Dependiendo del tamaño del proyecto, el control de calidad puede estar cargo bien sea, del personal del departamento de calidad de software del proyecto, en caso de que el proyecto sea pequeño, o de una junta de control de configuración si el proyecto es grande. Dicha junta debe incluir un representante por parte del usuario, un miembro del departamento de calidad de software, y el líder del proyecto.
- *Gestión de la configuración del software:* Tiene que ver con el seguimiento y control de cambios de los elementos de software en un sistema. Controla la evolución de un sistema software por medio del manejo de versiones de los componentes de software y sus relaciones. El propósito de la gestión de la configuración del software es el de identificar todos los componentes de

software interrelacionados y controlar su evolución a través de los diferentes ciclos de vida. La gestión de la configuración del software es una disciplina que puede ser aplicada a actividades como el desarrollo de software, el control de la documentación, control de cambios y mantenimiento. Puede proveer un alto ahorro de costos en la reutilización del software porque cada componente de software y su relación con otros componentes de software ya han sido previamente definidas. Cabe mencionar también que la gestión de la configuración del software consta de actividades que aseguran que el diseño y el código estén definidos previamente, y por tanto no se puedan cambiar sin antes hacer una revisión del efecto de tal cambio. A continuación se mencionaran los elementos que hacen parte de la gestión de la configuración del software: Identificación de componentes, control de versiones, construcción de la configuración, control del cambio.

La ilustración 1 muestra que no solo es necesario tener en cuenta los tres componentes del aseguramiento de la calidad antes mencionados, sino también una serie de estándares, mejores prácticas, convenciones, y especificaciones para de cierto modo asegurar la calidad en cuanto a software se refiere.

Ilustración 1. Software Quality Assurance



Fuente: Lewis, William E. “Software Testing and Continous Quality Improvement”. 2000, Auerbach, p. 8

Cuando se habla de aseguramiento de calidad de software es de vital importancia tener un plan que garantice que todos los procesos y/o procedimientos llevados a cabo dentro del aseguramiento de la calidad se cumplan de una manera eficiente. El plan de aseguramiento de la calidad de software (SQA, por sus siglas en ingles) es un conjunto de medidas con la finalidad de asegurar ciertos niveles de calidad dentro de las fases del desarrollo de software. El plan es usado como base para comparar los niveles actuales de calidad durante el desarrollo con los niveles planeados. Dicho plan sugiere un marco de trabajo y unos lineamientos para el desarrollo de código entendible y de fácil mantenimiento. El plan de SQA determinará si los procedimientos para asegurar la calidad del software serán desarrollados dentro del equipo de desarrollo o estarán a cargo de terceros. Para desarrollar e implementar un plan de aseguramiento de calidad de software (SQA), es necesario tener en cuenta ciertos pasos mencionados a continuación:

- *Documentar el plan*: Todo plan debe estar documentado para su posterior aprobación y ejecución.
- *Obtener aprobación de la gerencia del proyecto*: La participación de la gerencia del proyecto es necesaria para poder implementar de manera exitosa un plan de SQA. La gerencia del proyecto es responsable tanto de asegurar la calidad en un proyecto de software, como de proveer los recursos necesarios para el desarrollo del mismo.
- *Obtener aprobación del equipo de desarrollo*: Los principales usuarios de un plan de SQA son los miembros del equipo de desarrollo, por lo tanto es de vital importancia su aprobación y cooperación en el plan.
- *Planear la implementación del SQA*: El proceso de planear, formular y esbozar un plan de SQA requiere de personal y acceso a recursos tecnológicos. La persona responsable de implementar el plan de SQA debe tener acceso a estos recursos y adicionalmente, necesita aprobación y soporte constante del gerente del proyecto para garantizar la disponibilidad de los mismos. El gerente del proyecto debe considerar todos los riesgos que puedan impedir que el proceso se lleva a cabo, como por ejemplo la disponibilidad de personal o equipos. Es necesario también desarrollar un cronograma que incluya revisiones y aprobaciones para el plan de SQA.
- *Ejecutar el plan de SQA*: El actual proceso de ejecutar un plan de SQA involucra puntos de auditoría que permiten monitorear dicho plan. La auditoria debe ser programada durante la fase de implementación del software, para que así el plan de SQA no se vea afectado por un mal monitoreo del proyecto de software. Las auditorias deben darse periódicamente durante el desarrollo o en las etapas especificas del proyecto.

1.2 ESTRATEGIAS DE PRUEBA DE SOFTWARE

Aclarar el concepto de estrategia dentro del contexto de pruebas de software es sumamente difícil ya que los autores usan el término “estrategia” para exponer diferentes ideas todas ellas relacionadas con proyectos de prueba de software. Para algunos[5] la estrategia son las técnicas y metodologías que deben aplicarse al momento de realizar las pruebas; otros como William Perry y Roger Pressman, recurren a la palabra estrategia dentro de dos enfoques. El primero se refiere a estrategia como el plan general que guía y permite el seguimiento del proyecto de pruebas, e integra los métodos de diseño de casos de pruebas en una serie bien planeada de pasos; y el segundo, estrategia como las técnicas y metodologías que deben aplicarse en cada uno de los tipos de pruebas que se consideran dentro de las etapas del plan general. Autores más especializados en el tema como Rick D. Craig y Stefan P. Jaskiel, consideran el concepto de estrategia mencionada por Pressman como una planeación maestra de pruebas a lo que denominan “Master Test Planning”.

A pesar de las diferencias conceptuales con las que se refieren al termino estrategia, los autores coinciden que antes de empezar a realizar cualquier actividad relacionada con pruebas de software, debe existir un plan de pruebas (De ahora en adelante plan) que soporte y considere todas las decisiones administrativas que se toman a partir de las condiciones que rodean el proyecto, como: riesgos, recursos (tanto humanos como materiales) y tiempo, para lograr el buen desarrollo del mismo. Además de lo anterior, el plan debe tener objetivos medibles y alcanzables que conlleven al desarrollo de software de calidad.

[5] DeMillo, Richard A.; McCracken, Michael, W.; Martin, R.J. y Passafiume, John F. Software Testing and Evaluation. The Benjamin/Cummings Publishing Company.

El nivel de detalle a alcanzar por el plan debe permitir al desarrollador basar su trabajo diario de acuerdo a lo estipulado en el mismo y sobre todo, servir de guía para que el gerente realice control sobre el proyecto.

1.2.1 Estrategias de pruebas: enfoque de R. Pressman

Pressman, como se mencionó anteriormente, describe la estrategia general de software "como un aspecto que reúne todos los métodos de diseño de caso de prueba en una serie bien planteada de pasos; proporcionando un mapa que describe los pasos que se darán como parte de la prueba e indica cuándo se planean y cuándo se dan estos pasos, además de cuánto esfuerzo, tiempo y recursos consumirán. Por lo tanto, cualquier estrategia de prueba debe incorporar la planeación de pruebas, el diseño de casos de pruebas, la ejecución de pruebas y la evaluación de los datos resultantes." Al mismo tiempo el autor menciona cuatro estrategias por cada una de las etapas en las que divide las pruebas de software, asegurando también, que por cada proyecto de desarrollo de software debe existir una estrategia de pruebas diferente manteniendo una estructura básica general, ya que cada proyecto tiene tanto necesidades particulares como generales (Estrategia). De esta forma se asegura que la gestión administrativa del proyecto pueda ser llevada a cabo con facilidad.

En la actualidad existen tantas estrategias de pruebas de software como enfoques de desarrollo, pero como se menciono anteriormente, todas ellas mantienen en común unas características generales. A continuación se nombran las características que se encuentran frecuentemente en las plantillas de pruebas.

- Para realizar *testing* efectivo el equipo desarrollador de software debe realizar revisiones técnicas formales (RTF) con el fin de eliminar errores antes del inicio de la etapa de pruebas.

- El *testing* empieza en un nivel bajo (de componentes) y trabaja “hacia afuera” con el fin de integrar todo el sistema. De lo menos hacia lo más. Toda estrategia de pruebas de software debe incluir pruebas de bajo nivel, necesarias para verificar que pequeños segmentos de código estén correctamente implementados (unidades lógicas), como también debe incluir pruebas de alto nivel que validen la funcionalidad general del sistema.
- Se deben aplicar diferentes técnicas de prueba en diferentes puntos del *testing*.
- Las pruebas son dirigidas por el grupo desarrollador del software, y en caso de que sea un gran proyecto, dichas pruebas son desarrolladas por un grupo de pruebas independiente (GPI).
- El *testing* y la depuración son actividades diferentes, pero la depuración debe ocupar un lugar dentro de cualquier estrategia de prueba de software.

El punto de vista de R. Pressman respecto a cómo deben realizarse las pruebas es bastante interesante en la medida que sugiere como el mejor método, el realizar pruebas constantemente desde el momento de codificación del software hasta las pruebas del sistema, repitiendo el proceso por cada cambio realizado al código a partir de un error localizado por el equipo en pruebas anteriores.

Si se considera el proceso de prueba de software desde un punto de vista procedimental, las pruebas consisten en una serie de pasos que se implementan de manera secuencial. Estos pasos empiezan desde la prueba de unidad, asegurando que funciona de manera apropiada como unidad; continúan con prueba de integración, que se realizan cuando se integran los paquetes y que considera problemas como doble verificación y construcción del programa; prueba de validación, primera prueba de alto nivel y se realiza cuando se ha integrado y probado el programa, en ella se debe evaluar los criterios de validación establecidos en la etapa de análisis de requisitos incluyendo requisitos

funcionales, de comportamiento y de desempeño; y por último se realiza la prueba de sistema, la cual queda por fuera de todos los límites del ingeniero de software, ya que cuando el producto está listo debe combinarse con otros elementos del sistema como hardware, personas, bases de datos, etc. En esta prueba se verifica que todos los elementos encajen apropiadamente y que se logre la función y el desempeño generales del sistema.

Las cuatro etapas mencionadas anteriormente son descritas para lo que el Autor denomina “software convencional”, manifestando que se pueden realizar estas mismas pruebas en arquitecturas orientadas a objetos considerando como unidad a las clases.

Muchas organizaciones desarrolladoras de software pueden aplicar diferentes estrategias para probar sus productos. En un extremo, el equipo desarrollador podría esperar a que el sistema este terminado para realizar las pruebas y esperar encontrar los errores que no se encontraron o no se preocuparon por encontrar durante todo el proceso de desarrollo. Este enfoque puede resultar muy atractivo para el equipo de pruebas, pero según la experiencia de Pressman no funciona. El resultado de realizar pruebas basadas en este enfoque es un software lleno de errores y que seguramente generará en el cliente y el usuario final una experiencia molesta. En el otro extremo, el ingeniero de software puede optar por el enfoque, mencionado anteriormente, realizando pruebas diariamente sin importar que parte del programa se haya desarrollado. El enfoque, aunque asegurará una aplicación de calidad, no es muy atractivo por el tiempo y recursos que consume. Resultado que puede obtenerse siguiendo lo descrito en la estrategia general de manera puntual y aplicando una estrategia (método) de prueba adecuada para cada una de las etapas sugeridas.

Para Tom Gilb, citado por Pressman en su libro, las cuatro etapas de toda estrategia de pruebas deben tener unas directrices o que llevan a que la estrategia de prueba de software tenga éxito. Dichas directrices fueron publicadas en el articulo “What we fail to do in our current testing culture”, y continuación serán expuestas:

- *Especificar los requerimientos del software de manera cuantificable, previa a la ejecución de las pruebas.* Aunque el objetivo primordial de la prueba es encontrar errores, una buena estrategia de prueba también evalúa otras características de la calidad, como las opciones de llevarla a otra plataforma, además de la facilidad de mantenimiento y uso. Esto debe especificarse de manera tal que permita medirlo para que los resultados de la prueba no resulten ambiguos.
- *Establecer explícitamente los objetivos de la prueba.* Los objetivos específicos de la prueba se deben establecer en términos cuantificables. Por ejemplo, dentro del plan de prueba deben establecerse la efectividad y la cobertura de la prueba, el tiempo medio de falla, el costo de encontrar y corregir defectos, la densidad o la frecuencia de las fallas restantes, y las horas de trabajo por prueba de regresión.
- *Entender las necesidades del usuario del software, y crear un perfil por cada categoría de usuarios.* Los casos de uso que describan el escenario de interacción para cada clase de usuario reducen el esfuerzo general de prueba, ya que concentran la prueba en la utilización real del producto.
- *Desarrollar un plan de pruebas enfocado en “ciclos rápidos de pruebas”.* Se recomienda que un equipo de ingeniería de software “aprenda a probar en ciclos rápidos (2% del esfuerzo del proyecto) los incrementos en el mejoramiento de la funcionalidad, la calidad, o ambas, de manera que sean útiles para el cliente y se puedan probar en el campo”. La retroalimentación

que generan estas pruebas de ciclo rápido se utilizan para controlar los grados de calidad y las correspondientes estrategias de prueba.

- *Construir software "robusto" diseñado para probarse a sí mismo.* El software debe diseñarse de manera tal que use técnicas anti error. Es decir, el software debe tener la capacidad de diagnosticar ciertas clases de errores. Además, el diseño debe incluir pruebas autorizadas y de regresión.
- *Utilizar las revisiones técnicas formales como filtro antes de comenzar las pruebas.* Las revisiones técnicas formales posibilitan descubrir inconsistencias, omisiones y errores evidentes en el enfoque de la prueba. Esto ahorra tiempo y también mejora la calidad del producto.
- *Desarrollar un enfoque de mejora continua para el proceso de prueba.* Es necesario medir la estrategia de prueba. Las medidas reunidas durante la prueba deben usarse como parte de un enfoque estadístico de control del proceso para la prueba de software.

Siguiendo el orden de ejecución de un proyecto y con todas las estrategias propuestas para concluir de forma eficaz un proyecto de pruebas de software, Pressman toca un punto muy importante relacionado con esto último. Comúnmente se da por sentado que las pruebas tienen una duración o terminación conocida pero es común que surjan muchas dificultades a la hora de conocer realmente cuando se deben dar por terminadas las pruebas. Existen varias aproximaciones que tratan de resolver dicha problemática. Una de ellas sugiere que en realidad las pruebas nunca terminan, sino más bien se transfiere la responsabilidad de hacerlas, ya que una vez terminado el producto el encargado de seguir haciendo las pruebas es el usuario. Otra de las aproximaciones propuestas, pero muy poco aceptada, dice que las pruebas acaban cuando se agota el tiempo y/o el dinero. En general, uno de los criterios más utilizados para determinar cuándo acabar las pruebas es la experiencia adquirida en proyectos

anteriores. Son también de gran ayuda modelos estadísticos que recopilan ciertas métricas de software.

1.2.2 Estrategias de pruebas: Enfoque de Rick Craig y Stefan Jaskiel

Craig y Jaskiel, autores del libro “Systematic Software Testing”, aseguran que una de la claves para alcanzar el éxito en las pruebas de software es la planeación de pruebas de software. Es común que en la mayoría de los proyectos informáticos no se dé la debida importancia a la planeación debido a la falta de tiempo, recursos, entrenamiento o vías culturales generando productos que en gran medida no cumplen las expectativas de los clientes. Las pruebas de software no son ajenas a error y de ahí la necesidad de crear un plan que determine los lineamientos para un buen desarrollo de pruebas.

1.2.2.1 Niveles de planeación de pruebas. Según el estándar 829-1998 propuesto por la IEEE para la documentación de pruebas de software existen 4 niveles: pruebas de unidad, pruebas de integración, pruebas de sistema, y pruebas de aceptación (o validación). Para cada uno de estos niveles debe existir un plan correspondiente, definido de acuerdo al nivel de complejidad y la duración del proyecto. Se debe considerar entonces un *plan maestro de pruebas* (MTP por sus siglas en ingles) que orqueste las pruebas en todos los niveles.

Adicionalmente al MTP, es común que sea necesario crear planes de pruebas detallados. En un proyecto largo y complejo, a menudo vale la pena crear un plan de pruebas de aceptación, sistema, integración, de unidad, y muchos otros planes, dependiendo del alcance del proyecto en cuestión. Proyectos pequeños, es decir, proyectos con alcances cortos, menos número de participantes, y organizaciones de menor tamaño, puede que solo necesiten un solo plan para abarcar todos los niveles de las pruebas. Decidir el número y alcance de los planes de prueba debe ser la primera estrategia de decisión desarrollada dentro del plan de pruebas.

Mientras la complejidad del proceso de pruebas se incrementa, se incrementa exponencialmente la necesidad de tener un buen MTP.

El ingeniero encargado del plan de pruebas debe ver el MTP como su máximo canal de comunicación con todos los participantes del proyecto. El proceso de prueba tiene como objetivo generar un documento que permita a todas las partes involucradas en las pruebas a decidir proactivamente cuales son los aspectos que se presentan en cualquier proyecto de testing como: la utilización de los recursos, responsabilidades, roles, riesgos y prioridades. Al comparar este enfoque con el enfoque de Pressman, se puede observar que ambos usan el MTP o la estrategia, según sea el caso, como guía para los miembros del equipo y posibilitarles herramientas que faciliten terminar el proyecto de prueba.
Dos aspectos importantes a tener en cuenta a la hora de diseñar un plan de pruebas son:

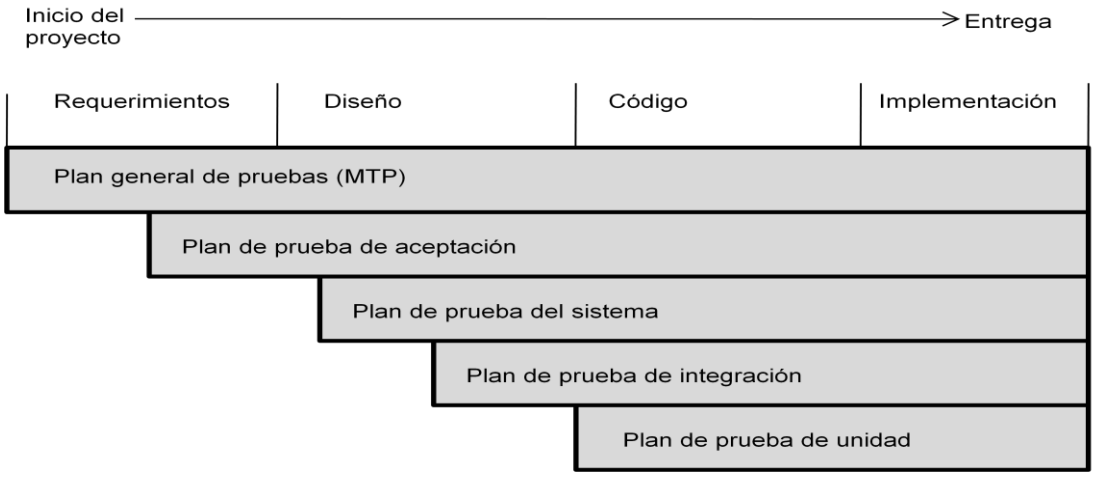
- La planeación de pruebas debe ser un complemento del planeación general del proyecto, ya que lo que se defina en el plan de pruebas debe estar reflejado en el plan general del proyecto.
- La planeación de pruebas debe ir separada del diseño de pruebas.

1.2.2.2 Tiempos de actividad. La planeación de prueba debe empezar tan pronto como sea posible. Generalmente es deseable empezar el MTP casi en el mismo tiempo que se desarrolla el plan del proyecto y la especificación de requisitos.

Si la planeación se empieza tempranamente, puede y debe afectar significativamente el impacto del contenido del plan del proyecto. El plan de pruebas de aceptación puede empezar tan pronto como el proceso de definición de requisitos ha empezado. Igualmente las pruebas de sistema, integración y unidad deben empezar tan pronto como sea posible.

En la siguiente figura se muestran los tiempos aconsejados para iniciar los planes de cada etapa.

Ilustración 2. Tiempos de planeación de pruebas



1.2.2.3 Componentes del plan. A pesar que la IEEE propone un formato que contiene varios aspectos que deben tenerse en cuenta al momento del desarrollo del plan de pruebas no es una exigencia considerarlos todos siempre que se realice un plan de prueba, es decir, la propuesta es flexible y puede ser modificada agregando o eliminando secciones dentro del mismo. A continuación se mencionaran cada uno de los componentes propuestos por la IEEE, incluyendo además algunas secciones propuestas por Craig y Jaskiel (identificadas con un asterisco) que pueden ser usadas al momento de crear cualquier clase de plan, ya sea el plan maestro, aceptación, sistema, integración, unidad o cómo haya nombrado los niveles del plan prueba dentro del proyecto.

- Identificador del plan. El objetivo del identificador del plan es brindar la posibilidad de seguimiento de las versiones del plan de prueba. Es por eso que a cada una de las versiones debe asignárseles un número de identificación. Cuando la organización que desarrolle el proyecto tenga algún estándar para

control de documentación, debe implementarlo para las versiones del plan. El identificador del plan es un número único generado por la organización que gestiona el proyecto y este se usa, como se menciono antes para identificar la versión del plan, el nivel, y la versión de software a la que pertenece dicho plan.

No puede olvidarse que el plan de pruebas es como cualquier otra documentación de desarrollo de software, es dinámico y por lo tanto debe mantenerse actualizado.

- Tabla de contenidos*. Debe incluir todos los temas que se trabajen en el plan, así como también referencias, glosarios, y apéndices. Si es posible, es aconsejable desarrollar la tabla de contenidos hasta dos o más niveles de profundidad para que el lector pueda ver en detalle el contenido del plan.
- Referencias*. Según el estándar 829-1998 de la IEEE para la documentación de pruebas, las referencias están incluidas dentro de la *Introducción* pero Craig y Jaskiel consideran que debe ser dividido cada ítem en una sección para hacer énfasis en su importancia, los ítems son: Autorización del proyecto, plan del proyecto, plan de aseguramiento de calidad, plan del manejo de la configuración, políticas relevantes y estándares relevantes. La IEEE también sugiere que para planes multinivel, cada nivel inferior debe referenciar al siguiente nivel superior.
- Glosario*. El glosario es usado para definir algunas de las palabras y términos utilizados en el documento del plan. Es conveniente que cuando se realice el glosario se tenga en cuenta el público que recibirá el documento, recuerde que pueden existir diferentes perfiles de personas dentro de proyecto y que cada uno tiene conocimientos técnicos diferentes.
- Introducción. Según la plantilla sugerida por la IEEE la introducción está compuesta por dos componentes principales: una descripción del alcance del

proyecto, que incluye las características, historia, entre otros.; y una introducción que describa el alcance del plan.

- Ítems a probar. La sección describe de una manera secuencial, que elementos se van a “testear” de acuerdo al alcance del proyecto y debe realizarse con la colaboración del gerente y el equipo de desarrollo. Este componente del plan puede variar entre niveles; para niveles superiores los ítems se deben organizar por versiones o aplicaciones, mientras que para los niveles inferiores dichos ítems se pueden organizar en módulos, unidades o programas. Los ítems o elementos que vayan a ser excluidos de las pruebas deben ser especificados en esta sección.
- Riesgos*. Craig y Jaskiel sugieren que el propósito de incluir los riesgos de software dentro de un plan de software es el de determinar el enfoque que deben tener las pruebas. Explican qué determinar los riesgos de software ayuda a los ingenieros a establecer una prioridad entre las pruebas y a concentrarse en aquellas áreas donde puede presentarse mayor riesgo a fallas o que tengan alto impacto para el cliente en caso de fallas. Algunos de los riesgos más comunes pueden ser: Interfaces con otros sistemas, alta complejidad del software, módulos que históricamente han presentado fallas o con muchos cambios, aspectos relacionados con la seguridad, desempeño, fiabilidad.

  Para determinar las áreas que sean propensas a falla dentro del proyecto en el que está trabajando es aconsejable realizar una lluvia de ideas, preguntando a los miembros del equipo “¿Qué los preocupa?”, sin usar la palabra riesgo dentro de la pregunta, ya que por experiencia de los Craig y Jaskie dicha palabra puede intimidar a los participantes.
- Características a probar. A diferencia de la sección “ítems a probar”, donde se tienen en cuenta los ítems que se deben probar desde el punto de vista del equipo desarrollador, esta sección del plan contiene una lista de los elementos que serán probados desde la vista del usuario o el cliente.

Por cada característica mencionada dentro de la lista debe haber una columna por cada uno de los siguientes aspectos: tipo de requisito al que pertenece el ítem, frecuencia de ocurrencia, impacto y prioridad. Para definir la escala por cada característica de la lista, pueden usar las concepciones definidas, para el proyecto en general, respecto a los riesgos.

- Características no probadas. En esta sección del plan debe enunciar todas las características o elementos que no serán tenidos en cuenta dentro de las pruebas, explicando el motivo para descartarlas dentro del plan. Existen varias razones por las cuales no se prueba una característica. Puede que la característica no haya cambiado, no esté disponible para el uso, etc. Pero por la razón que sea la característica debe aparecer en esta lista, generalmente la razón principal es un riesgo bajo dentro del proyecto.
- Estrategia*. La estrategia es considerada por Jaskiel y Craig, el corazón del plan de pruebas. Esta sección contiene una descripción detallada de cómo serán desarrolladas las pruebas y también una explicación de los temas que tengan un alto impacto tanto para las pruebas, como para el proyecto en general. La estrategia incluye una serie de secciones o componentes explicados en detalles en el siguiente numeral.
- Criterio de aceptación/rechazo de los ítems. Esta sección describe los criterios de aceptación y rechazo sobre los ítems expuestos en la sección “Ítems a probar”. Generalmente dichos criterios se aplican a los casos de pruebas y están expresados por: número, tipo, severidad y ubicación del error, usabilidad, fiabilidad y/o estabilidad. Dependiendo del nivel, y del proyecto, los criterios de aceptación/rechazo pueden variar. Algunos ejemplos de criterios incluyen: porcentaje de casos de pruebas aprobados, cantidad, severidad y distribución de defectos, alcance de los casos de pruebas.
- Criterios de suspensión y requisitos de reanudación. En esta sección se identifican las condiciones que obligan una suspensión temporal de las pruebas y los criterios de reanudación de las pruebas. Los criterios de

suspensión pueden incluir: tareas incompletas en la ruta critica del proyecto, muchos errores encontrados, errores críticos. Ambientes de pruebas incompletos, faltas de recursos.

- Entregables de pruebas. En esta sección se define una lista de entregables que incluye documentos, herramientas y otros componentes a desarrollarse y mantenerse bajo los esfuerzos de las pruebas. Ejemplos de entregables de pruebas pueden ser: plan de pruebas, especificaciones de diseño, casos de pruebas, procedimientos, logs, reporte de incidentes, resumen, entre otras. Es importante aclarar que no es válido incluir dentro de la lista de entregables el software que está siendo puesto a prueba.

  Los elementos que hayan servido de soporte para las pruebas deben ser identificados en el plan general del proyecto como entregables y deben tener una asignación adecuada de recursos en el sistema de seguimiento del proyecto. Esto garantizará que el proceso de pruebas tenga una visibilidad importante dentro del proceso de seguimiento del proyecto y que las tareas resultantes de las pruebas para generar los entregables sean iniciadas a tiempo. Las dependencias entre los entregables de las pruebas y las entregables del producto de software deben ser plasmadas en un diagrama de Gantt (se exponen en la sección “Cronograma”).

- Tareas. Definida por la IEEE para identificar el conjunto de tareas necesarias para llevar a cabo las pruebas. También se listan todas las dependencias entre tareas y cualquier habilidad especial requerida.

- Necesidad del entorno. Las necesidades del entorno incluyen, hardware, software, datos, lugares apropiados, accesos, y otros requisitos que permitan el normal desarrollo de las pruebas. Se debe entonces crear un entorno lo más cercano a la realidad del sistema que va a ser sometido a pruebas. En caso de que el sistema este diseñado para ser ejecutado con múltiples configuraciones (bien sea de hardware o de software), se debe decidir si bien se replican todos los escenarios los mas riesgosos o las más comunes. Una vez se hayan

determinado los escenarios se deben listar, tantos las configuraciones de hardware como las de software. Adicional a esto es necesario identificar de donde vendrán los datos para generar una base de datos referentes a las pruebas. Dicha base de datos puede contener: datos de producción, de compra y los suministrados por el usuario.

- Responsabilidades*. Craig y Jeskiel formularon una matriz que muestra como está distribuida la responsabilidades de las diferentes actividades que requieren las pruebas frente a los diferentes miembros del equipo encargado de realizar el Testing.

  La matriz contiene responsables, y tareas o actividades a realizar. Los responsables se pueden categorizar en: Director de desarrollo, director de pruebas, ejecutores de pruebas, coordinador del entorno de pruebas y director de librerías. Y algunos ejemplos de tareas a realizar pueden incluir: coordinar el desarrollo del MTP (plan general de pruebas), desarrollar el plan para la prueba de sistema, de unidad e integración, mantener el entorno de pruebas, entre otros.

- Necesidades de personal y entrenamiento. Dependiendo del alcance, los objetivos y otros factores influyentes en el proyecto, se determina la cantidad de personas y las cualidades necesarias que deben tener para llevar a cabo el Testing. Algunos ejemplos de capacitaciones pueden incluir: usos de cierta herramienta en particular, metodologías de pruebas, interfaces, entre otras. Las capacitaciones pueden variar de acuerdo al proyecto.

- Cronograma. El cronograma de pruebas debe ser generado a partir de las fases existentes en el plan del proyecto como las fechas de entrega de varios documentos y módulos, la disponibilidad de recursos, interfaces, etc. Después añadir las fases de las pruebas. Estas fases serán diferentes para cada nivel de planificación existente dependiendo del nivel del plan de pruebas creado. En un MTP, las fases se construirán alrededor de los eventos importantes

como las revisiones de requerimientos y diseño, entrega de código, terminación del manual de usuario y disponibilidad de interfaces.

Inicialmente es de mucha ayuda construir un cronograma genérico sin fechas; esto es, identificar el tiempo requerido para las tareas, las dependencias, etc. Todo esto sin especificar tiempos de inicio o entrega. Normalmente este cronograma debe ser una grafica de un diagrama de Gantt en orden de mostrar las dependencias entre tareas.

- Riesgos de planeación*. Jaskiel y Craig argumentan que cualquier actividad que suponga en un obstáculo para el cronograma de las pruebas se considera un riesgo de planeación. Dentro de los riesgos de planeación más comunes se encuentran: fechas de entregas imposibles de cumplir, disponibilidad del personal, presupuesto, insuficiencia de inventario de herramientas, alcance de las pruebas mal propuesto, entre otras. Por otra parte esta sección debe mencionar las contingencias que se deben tener en cuenta para mitigar los riesgos antes mencionados. Algunas de dichas contingencias pueden ser: reducción del alcance de la aplicación, retrasar la implementación, añadir recursos adicionales, reducir procesos de calidad.
- Aprobaciones. Las aprobaciones deben ser firmadas o aceptadas por miembros del equipo de pruebas que estén en capacidad de determinar si el software está listo para avanzar hacia el siguiente paso. Para cada uno de los niveles de pruebas, existirán aprobaciones que estarán a cargo de los directores de dichos niveles, por ejemplo, para el plan de prueba de unidad la persona que autoriza debe ser el gerente de desarrollo.

  Las personas que firman deben incluir la fecha en el que realizan la aprobación para que exista un registro dentro del proyecto.

1.2.2.4 Componentes de la estrategia

- Selección de metodologías. Para definir la metodología a utilizar en una estrategia de pruebas es necesario tener en cuenta una serie consideraciones básicas como: el número de miembros del equipo de pruebas designados para la planeación el diseño y las pruebas, el inicio de las pruebas, la utilización de pruebas piloto, las técnicas y niveles de pruebas (Unidad, Integración, Aceptación o de Sistema) a utilizar, entre otras. Por ejemplo, para la selección de los niveles de pruebas que se van a aplicar en un determinado proyecto se deben tener en cuenta factores que influyen en la decisión como lo son las políticas, los riesgos, la complejidad del sistema, los recursos y el tiempo disponible entre otros.
- Recursos. Se necesita tener una estrategia solida frente al manejo de los recursos, sobre todo humanos, ya que estos son escasos y son la clave del éxito en cualquier tipo de proyectos incluyendo proyectos de software.

  En casos donde no se tenga un equipo de pruebas o se tenga uno pero con poco personal o dedicados a otras actividades diferentes a las de pruebas, se deberá acudir a la contratación de personal adicional lo cual retrasaría todo el proceso debido a la curva de aprendizaje. También incrementaría costos. Por otro lado, se puede conformar un equipo de pruebas desde el inicio del proyecto, pero aumentaría costos al tener sin hacer nada a consultores que por lo general devengan un buen salario. Por lo cual se hace necesario encontrar un equilibrio entre los dos puntos.
- Determinación del alcance. Actualmente existen diferentes medidas para el alcance de las pruebas, entre las cuales se encuentran las coberturas de requisitos, diseño e interfaces. Pero una de las más conocidas y usadas, la cobertura de código, se encarga de medir el porcentaje de condiciones, ramificaciones o rutas de código ejecutadas por un grupo especifico de casos

de pruebas. Para determinar que partes del código fuente han sido ejecutadas es necesaria la ayuda de herramientas especializadas que faciliten dicha labor.

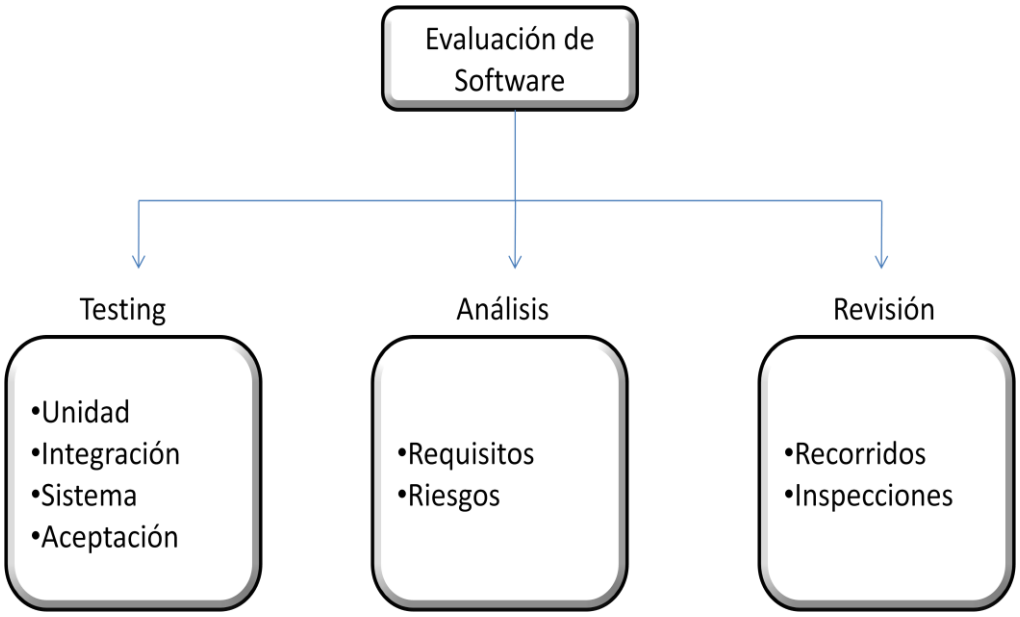
Otra de las medidas utilizadas para el reconocimiento del alcance en las pruebas, el cubrimiento de los requisitos, mide el porcentaje de los requisitos de negocio cubiertos por los casos de prueba. De forma similar se mide el cubrimiento de la interface al considerar las interfaces que están siendo intervenidas por las pruebas. Y en lo concerniente a la medición del alcance que puedan tener las pruebas en el diseño, se determina que partes del mismo son cubiertas por las pruebas.

- Recorridos e inspecciones. Las revisiones, junto con el Testing y el análisis, hacen parte del proceso de evaluación del software (ver Ilustración 2). En las revisiones se encuentran técnicas de verificación, como revisiones formales de código, requisitos y diseño, que no hacen parte de actividades de Testing como tal, pero afectan de manera significativa la estrategia, por lo que se deben considerar en el plan general de pruebas (MTP).

  Dos de los tipos de revisiones más comunes se encuentran los recorridos y las inspecciones. Un recorrido, hablando en términos de pruebas de software, se conoce como la revisión de pares a un producto de software en específico, que consiste en recorrer de manera secuencial el software para determinar la calidad que posee el producto y descubrir posibles defectos. Y en el caso de las inspecciones, tienen la misma razón de ser de los recorridos, pero son pruebas mucho más rigurosas y emplean procesos de control estadístico con el fin de medir la efectividad de la inspección e identificar oportunidades de mejoramiento en el proceso de desarrollo de software. La IEEE define la inspección como “la técnica de evaluación formal de software, en la cual los requisitos, el diseño o el código son examinados en detalle por una persona o un grupo, diferente al autor, con el objetivo de detectar fallas, violaciones de estándares de desarrollo y otros problemas”. Según Craig y Jaskiel, la inspección se rige por las siguientes observaciones a continuación

mencionadas: Verificar que los elementos del software satisfagan las especificaciones, verificar que los elementos del software se ajustan a los estándares aplicables, identificar las desviaciones de los estándares y especificaciones anteriormente mencionados, recolectar datos provenientes de la ingeniería de software (como por ejemplo datos de defectos o esfuerzos), y por ultimo no examinar aspectos alternativos o estilísticos.

Ilustración 3. Proceso evaluación de software



Dentro de las diferencias entre recorrido e inspección, planteadas por Craig y Jaskiel, se puede citar la informalidad del recorrido frente a la formalidad de la inspección, resaltando los propósito de cada una de ellas, siendo el de la primera muy subjetivo dependiendo de la persona encargada, mientras que en la segunda se utilizan técnicas de medición para la mejora de determinados procesos.

Debido a que tanto el proceso de recorrido como el de inspección requieren tanto de esfuerzo humano como de muchos recursos para llevarse a cabo, es que existe la necesidad de priorizar la inspección y recorrido de ciertos elementos, ya que resultaría casi imposible evaluar todos los componentes de un producto de software.

Por otro lado, se debe tener en cuenta el tipo de personal que se necesitará para realizar bien sea la inspección o los recorridos. Para realizar dichas actividades, es ideal tener personal de prueba que esté involucrado con las pruebas del nivel de sistema y tengan habilidades en la revisión de código.

- Gestión de la configuración. En esta sección de la estrategia incluida en el plan general de pruebas (MTP), es necesario considerar el manejo que se le dará a la gestión de la configuración dentro del marco de las pruebas. Dicha gestión de la configuración incluirá la gestión de cambios, de vital importancia para el proyecto en general, llevando a cabo un control de versiones del software y sus respectivas documentaciones que vayan siendo “testeadas”, así como también para un generar un proceso que permita tomar decisiones útiles para la priorización de errores. De los niveles de pruebas expuestos en el numeral anterior, Craig y Jaskiel sugieren utilizar el nivel de aceptación para validar el proceso de la gestión de la configuración.
- Recolección y validación de métricas. Es necesario discutir e incluir las métricas a utilizar en el proyecto dentro de la estrategia del plan general de pruebas, debido a que todo esfuerzo en cuanto a pruebas de software se refiere, debe medirse en términos de efectividad, calidad, cumplimiento al cronograma, entre otros.
- Herramientas y automatización. Otra de las consideraciones a tener en cuenta en la estrategia es el uso de herramientas de automatización útiles para el buen desarrollo del proyecto. Es necesario entonces generar un buen plan sobre el empleo de dichas herramientas, que incluya las respectivas capacitaciones de estas hacia los miembros del equipo.

- Cambios al plan de pruebas. En esta sección se debe abordar el manejo de cambios en el plan general de pruebas, así como también en los planes de los niveles existentes, debido a los constantes cambios que se presentan a lo largo del proyecto. Por eso es necesario que el encargado de coordinar la estrategia de prueba tenga en cuenta las siguientes consideraciones: determinar qué cambios deberán ir al proceso de aprobación nuevamente (por más pequeño que sean), precisar la frecuencia de actualización del plan de pruebas, determinar si el plan debe pasar por el proceso de gestión de configuración, como publicar el plan, entre otras consideraciones.
- Reuniones y comunicaciones. Resulta conveniente incluir dentro del plan general de pruebas (MTP) los reportes, comunicaciones y reuniones que se llevaran a cabo durante la duración del proyecto. Debe existir entonces unos métodos preestablecidos de comunicación como reuniones y reportes acerca del estatus de las pruebas, mesa de de control de cambios, presentación a usuarios y/o gerentes, entre otros métodos.
- Otras consideraciones sobre la estrategia. Durante la ejecución del proyecto pueden ocurrir una serie de consideraciones acerca de la estrategia no incluidas en el plan general de pruebas. A continuación mencionaran dichas consideraciones: múltiples entornos de producción, seguridad multinivel, beta Testing, mantenimiento y configuración de entornos de prueba, uso del soporte contractual, calidad de software desconocidas, entre otras más.

1.3 NIVELES DE PRUEBA

En esta sección se describen detalladamente los aspectos más importantes de cada uno de los cuatro niveles principales de las pruebas de software mencionados en la sección anterior.

Todas las notas presentadas a continuación serán basadas en lo propuesto por el estándar de la IEEE para las pruebas de software e interpretado por autores que tienen bastante recorrido y experiencia en el tema, como son Roger Pressman, Rick D. Craig y Stefan P. Jaskiel.

1.3.1 Pruebas de unidad

De acuerdo a Rick y Jaskiel, las pruebas de unidad juegan un papel muy importante dentro de las pruebas de software y a pesar de esto es uno de los niveles que menos atención se le presta y por ello es que dichas pruebas se realizan de manera deficiente.

1.3.1.1 Obstáculos comunes en las pruebas de unidad. Los problemas dentro de las pruebas de unidad se generan principalmente debido a la educación que reciben los desarrolladores de software que se realizan en proyectos de desarrollo. A estas personas se les ha enseñado que su productividad se mide por la cantidad de código entregado y aunque el gerente puede asegurar la importancia que tienen las pruebas de unidad dentro del proyecto, constantemente presiona al equipo desarrollador a entregar código lo más pronto posible cuando se aproxima la fecha de entrega del producto final. El otro problema es generado por la actitud del desarrollador. Generalmente los desarrolladores son demasiado optimistas y aseguran que el código desarrollado va a funcionar correctamente, dejando las pruebas de unidad para algo que se realiza solo si el tiempo lo permite.

Una extensión de este problema es una situación normal dentro de las compañías desarrolladoras de software, que usualmente no entrenan a su personal para realizar pruebas de unidad. Muchos gerentes e incluso algunos desarrolladores creen que por tener conocimientos para escribir un código de software, tienen también conocimiento para realizar pruebas.

Finalmente, las herramientas y procedimientos adecuados deben ponerse a disposición de los desarrolladores para que puedan realizar las pruebas de software. Herramientas como debuggers, sistemas de gestión de librerías, sistemas de rastreo de errores, entre otros. Dependiendo de la organización y el proyecto pueden considerarse diferentes herramientas.

Los tres párrafos anteriores describen algunas de las situaciones más comunes dentro de una organización. Considerar estos aspectos ayudará ampliamente a mejorar la calidad del software.

1.3.1.2 Educación. Para empezar a implementar efectivamente pruebas de unidad es necesario educar a los desarrolladores en cómo realizar pruebas. El entrenamiento debe incluir preparación tanto en metodologías de pruebas como entrenamiento específico en técnicas y herramientas de pruebas de unidad. Los gerentes de desarrollo también se deben ver beneficiados por la participación en el entrenamiento. El entrenamiento no solo provee a los gerentes conocimiento para entender la disciplina de prueba sino también presentará un apoyo que permite el aumento de confianza en el personal.

De la adquisición de conocimientos por parte del equipo de desarrollo depende el éxito de las pruebas de unidad y en general de todos los niveles de prueba, ya que realizar unas buenas pruebas de unidad facilita en gran medida el desarrollo de las pruebas de alto nivel, debido a que se encuentran defectos en etapas más tempranas del ciclo de desarrollo.

1.3.1.3 Estándares y Requerimientos. Una forma efectiva para desarrollar un estándar es proporcionar ejemplos únicos por cada uno de los documentos requeridos. Como mínimo deben hacerse ejemplos de los siguientes documentos: Plan de prueba de unidad, diseño de prueba, casos de prueba, procedimiento de prueba, reporte de error y reporte resumen de las pruebas. Se considera

conveniente proporcionar ejemplos tanto de proyectos pequeños como proyectos grandes ya que la documentación requerida para proyectos grandes generalmente es mayor.

Comúnmente la documentación requiere, además de los ejemplos, una serie de diagramas de flujos que describan como deben lograrse tareas como reportar un error, revisar el código, donde y como guardar casos de pruebas, cuándo y cómo implementar una inspección o un recorrido y como usar un registro de prueba.

Dentro del proyecto es recomendable seleccionar una persona (o un grupo) que realice un papel de protector para todos los miembros del grupo. Es decir, a esta persona recurren los otros miembros del equipo cuando necesitan ayuda.

Una práctica recomendada es la asignación de la creación y documentación de los procesos a los mismos desarrolladores, con esto se logra que no vean el proceso como una orden que se realiza por burocracia y que no tiene implicación alguna dentro del proyecto.

1.3.1.4 Gestión de la configuración. Lograr el objetivo de las pruebas de unidad requiere que se fijen procedimientos de control de código fuente. Dichos procedimientos pueden realizarse desde dos perspectivas; permitir que el desarrollador realice control de su propio código durante las pruebas de unidad o poner el código bajo la gestión de configuración. Claro está, que los dos enfoques tienen sus problemas. En el primero al desarrollador no le gustará realizar pruebas de unidad formales para identificar defectos en su propio código ya que no le gustará admitir que su trabajo no es perfecto. El segundo enfoque puede presentar problemas en la medida que se establezcan controles que son extremadamente rígidos, especialmente en el proceso de desarrollo. Lo adecuado es entonces, realizar un control al código fuente en el nivel de equipo de desarrollo o incluso a nivel de desarrollador (si existen lineamientos que permitan realizarlo),

pero no si existen dichos procedimientos, será imposible diferenciar entre depuración o pruebas de unidad.

1.3.1.5 Métricas. Otro beneficio de las pruebas de unidad formales, es que permite la recolección de métricas de unidad-nivel, las cuales ayudan a identificar tempranamente áreas problemáticas en el proceso de desarrollo.

Recolectar métricas durante el proceso de prueba de unidad también facilita la estimación de tiempo y recursos para trabajos futuros y además, ayuda a identificar áreas susceptibles al mejoramiento de procesos. Cabe anotar que al implementar cualquier forma de medición debe entenderse la cultura organizacional ya que puede causar que los miembros del grupo de desarrollo se sientan amenazados, pensando se está midiendo su capacidad personal y no como sucede en realidad, donde se está midiendo son los productos y procesos.

1.3.1.6 Reutilización de los elementos de prueba. El momento por el que está atravesando la sociedad requiere que se optimicen procesos y se realicen lo más pronto posible dentro de unos parámetros de calidad. Esto se puede lograr en los procesos de prueba reutilizando la documentación creada en la fase de prueba de unidad como casos de prueba, plan de prueba, etc. Es necesario aclarar que para lograr el objetivo de la reutilización todos los productos deben estar correctamente documentados.

1.3.1.7 Revisiones, Recorridos e Inspecciones. No existe claridad sobre que debe ser inspeccionado, si el código, los casos de pruebas o los planes de pruebas y tampoco sobre cuándo deben realizarse las inspecciones, si deben hacerse antes de las pruebas de unidad, después o ambos, antes y después. Lo único que está claro es que el uso de código de inspección complementa sistemáticamente las pruebas de unidad y ayuda a definir cuando ejecutarse las

pruebas de unidad. Las inspecciones son también muy efectivas encontrando defectos que son difíciles de detectar durante las pruebas de unidad y viceversa.

1.3.2 Pruebas de integración

Las pruebas de integración se realizan para asegurar que varios componentes del sistema interactúen y se comuniquen correctamente. Las pruebas pueden realizarse en varios niveles. En el nivel más bajo, generalmente se realizan por el equipo de desarrollo para asegurar que las unidades juntas funcionen correctamente. En niveles más altos, las pruebas pueden ser hechas por el equipo de pruebas o de desarrollo, y examina como las piezas del sistema funcionan conjuntamente.

1.3.2.1 Análisis de audiencia. El plan de pruebas de integración es generalmente usado por los desarrolladores como un “mapa” para ver como las partes individuales del sistema encajan cuando se ponen todas juntas. El grupo de pruebas también usa el plan para asegurarse que no están realizando las mismas tareas que el grupo de desarrollo u otros grupos de pruebas.

Las pruebas de integración han tomado una importancia significativa, especialmente en proyectos grandes, complejos, y en particular en nuevos desarrollos. Algunos errores de programación solo pueden ser descubiertos cuando las partes individuales están integradas.

1.3.2.2 Tiempos de actividad. Generalmente los miembros del equipo quieren empezar a realizar el plan de pruebas de integración lo más pronto posible pero es aconsejable empezarlo cuando el diseño de la aplicación está estabilizado.

Los desarrolladores juegan un papel importante en la creación del plan de integración porque eventualmente ellos van a participar en las pruebas, ya que se

considera que son ellos las personas más aptas para realizarlas. Esto sucede porque el equipo de desarrollo no sabrá si han creado un buen sistema sin hacer al menos una prueba de integración.

Es deseable empezar a realizar las pruebas de integración por los componentes más riesgosos o aquellos que hacen parte del camino crítico, de esta forma se descubren los errores graves lo antes posible y no en la parte final del ciclo, algo que podría resultar muy costoso.

Los sistemas son construidos por la combinación iterativa de grandes piezas de código, por lo tanto pueden existir varios niveles de integración. Los primeros niveles generalmente se consideran una extensión de las pruebas de unidad mientras que los niveles más altos integran módulos completos y son considerados pruebas de construcción. Lógicamente estas pruebas las realizan el grupo de desarrolladores.

1.3.2.3 Fuentes de información. Las pruebas de integración se realizan con base en las especificaciones de diseño, que son las principales fuentes de informacion. En niveles más altos las pruebas se basan en diseños de arquitectura para que coincida con las especificaciones de diseño definidas en el proyecto.

1.3.2.4 Consideraciones del plan de prueba de integración. Para definir el tipo de pruebas de integración es necesario considerar los siguientes hechos:

- ¿Qué módulos deben ser probados como un grupo?
- ¿Cuáles son las principales sub-integraciones?
- ¿Cuáles son las características críticas?
- ¿Cuántas pruebas son apropiadas?
- ¿Existe alguna implementación basada en los objetivos de las pruebas?

- ¿Cómo puede un problema aislarse?

1.3.2.5 Gestión de la configuración. En las primeras etapas de las pruebas de integración, la gestión se realiza de forma informal entre los desarrolladores y los líderes del proyecto. Cuando la integración se torna más compleja, la gestión debe convertirse más formal ya que los cambios en el sistema pueden afectar potencialmente el código y las pruebas de otros miembros del equipo.

1.3.3 Pruebas de sistema

Cuando la mayoría de las personas piensan en pruebas, lo primero que se les viene a la mente son las pruebas de sistema. En pruebas de sistema es donde la mayoría de los recursos del proyecto van a parar, ya que pueden realizarse por largos períodos de tiempo y tienden a ser tan exhaustas como lo permita el nivel de riesgo tolerado y los recursos asignados a esta labor. Adicional a las pruebas funcionales, es normal que se realicen durante las pruebas del sistema pruebas de carga, desempeño, confiabilidad y otro tipo de pruebas si no existe un equipo encargado de realizar dichas actividades.

1.3.3.1 Análisis de audiencia. El plan de pruebas del sistema debe ser escrito por el director del equipo de pruebas (si está definido). Él es el responsable de coordinar con los desarrolladores y usuarios para obtener de ellos su punto de vista sobre las áreas que los afectan. En caso que no haya equipo independiente de pruebas, los encargados de realizar el plan son los miembros del equipo desarrollador o en algunos casos los usuarios.

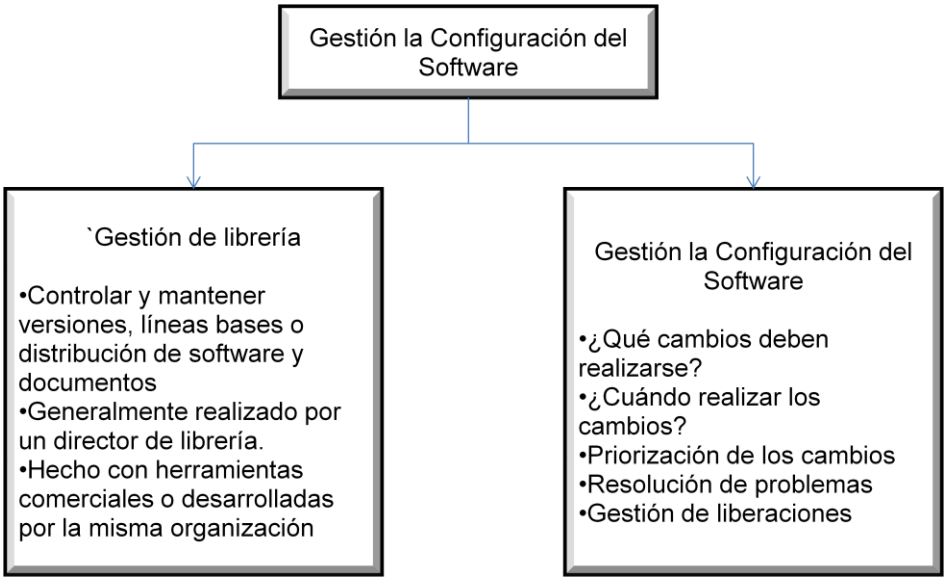
1.3.3.2 Fuentes de información. El plan de prueba del sistema debe estar en concordancia con el MTP y busca ampliar esas partes del plan que apuntan a las pruebas de sistemas. El plan de pruebas del sistema se construye basado en las especificaciones de los requisitos, la documentación de diseño, y la

documentación de los usuarios, en caso que exista. Otros documentos como las especificaciones funcionales, materiales de entrenamiento y demás, son muy útiles.

1.3.3.3 Gestión de la configuración de software. La gestión de la configuración de software es crítica para las pruebas de software. Es por eso que si se realiza una gestión pobre, el esfuerzo de prueba y en general el proyecto entero puede fallar. Craig y Jaskiel consideran que la gestión de la configuración de software es un proceso que está compuesto por dos funciones diferentes pero relacionadas, gestión de librerías y junta de control de cambios (CCB por sus siglas en Ingles).

- Gestión de librerías. Controla y mantiene las versiones, las líneas bases, las "construcciones" y las liberaciones de software y documentos al público. Usualmente el trabajo lo realiza un líder de configuración ayudado por herramientas especializadas.
- Junta de control de cambios (CCB), por su nombre en Inglés, Control Configuration Board, o Comité de Control de Cambios. Esta junta define que cambios deben realizarse, cuando se realizan, priorizan su ejecución y además son los encargados de la solución de incidentes, defectos, mejoramientos, y gestionar las liberaciones.

Ilustración 4: Funciones de la gestión de la configuración



1.3.3.4 Criterios de entrada/salida. Considerar los criterios de entrada y salida de una etapa del proceso de prueba a otra dentro del MTP es muy importante. Si las pruebas del sistema empiezan antes de que se hayan terminado las pruebas de integración, muchos de los errores de programación que pueden ser encontrados durante la etapa de desarrollo seguramente serán descubiertos por los miembros del equipo de prueba durante las pruebas del sistema y por realizar esto en una etapa superior, los costos de encontrar el error y corregirlo aumentan considerablemente. Sumado a lo anterior, el producto resultante de las correcciones debe volver a entrar al proceso de prueba.

Si el proceso de prueba se desarrolla en el orden normal, empezando cada etapa después de la finalización de la anterior, el problema descrito anteriormente podría solucionarse durante la etapa de desarrollo sin seguir un procedimiento formal, sin llegar a involucrar a más personas que al desarrollador. Depurar errores encontrados por el equipo de pruebas en la fase de pruebas del sistema, es

comúnmente más difícil que los errores de programación encontrados durante las fases de integración o de unidad.

Entonces para solucionar la incertidumbre de cuando terminan las pruebas de la etapa de desarrollo (Unidad e integración) y cuando empezar con las pruebas del sistema, es necesario definir y comunicar criterios de salida de las pruebas de desarrollo y criterios de entrada a las pruebas del sistema. Comúnmente los criterios de entrada a una etapa incluyen todos o algunos de los criterios de salida de la etapa anterior.

1.3.3.5 Prueba de humo. La prueba de humo se realiza dentro de fase de desarrollo (Unidad e integración) y es un grupo de casos de prueba que establece que el sistema es estable y la mayoría de sus funcionalidades están presentes y funcionan bajo condiciones normales. El propósito de la prueba de humo no es encontrar errores de programación sino la construcción de productos con mayor calidad. Además, proporciona a los desarrolladores una meta y les hace saber cuándo alcanzarán un grado de estabilidad.

La prueba, está diseñada como un mecanismo para marcar el ritmo en proyectos en los cuales el tiempo es crítico, lo que permite que el equipo de software evalué su proyecto con frecuencia. En esencia la prueba de humo abarca las siguientes actividades:

- Los componentes de software traducidos a código se integran en una “construcción”, la cual incluye todos los archivos de datos, las librerías, los módulos reutilizables y los componentes de ingeniería que se requieren para implementar una o más funciones del producto.
- Se diseña una serie de pruebas para exponer errores que impedirán que la construcción realice apropiadamente su función. El objetivo es descubrir

errores “paralizantes” que tengan la mayor probabilidad de retrasar el proyecto de software.

- La construcción se integra con otras construcciones, y diariamente se aplica una prueba de humo a todo el producto en la forma actual. El enfoque de la integración que se realice puede ser descendente o ascendente.

La prueba de humo debe ser creada por el grupo de pruebas y contar al menos con el consentimiento del grupo de desarrollo. De otra forma, los desarrolladores sentirán que los miembros del equipo de pruebas están imponiendo sobre ellos una forma de trabajar.

Al ser una metodología que implica ejercitar todo el sistema diariamente, no es necesario que sea exhaustiva, pero debe tener la capacidad de exponer los problemas importantes de forma rápida. La prueba está diseñada con el fin de marcar el ritmo a los proyectos, ya que permite que el equipo pueda evaluar su progreso con frecuencia.

El truco para establecer una buena prueba de humo es crear un grupo de pruebas de alto alcance en vez de profundidad. Recuerde que el propósito de estas pruebas es demostrar estabilidad del sistema, no encontrar cada error de programación dentro del mismo.

1.3.4 Pruebas de aceptación

Las pruebas de aceptación están basadas principalmente en los requisitos de usuario, y su razón de ser es demostrar que dichos requisitos se hayan cumplido. Es válido señalar que dichas pruebas deben ser las primeras en tener en cuenta dentro de la planeación debido al enfoque que poseen. Debido a que las pruebas de aceptación son utilizadas para modelar el comportamiento del sistema cuando

este ya esté finalizado. Es pertinente que estas pruebas las realicen los usuarios, o alguien que los represente, dentro de un entorno lo más real posible.

1.3.4.1 Análisis de audiencia. Como se menciona anteriormente, las pruebas de aceptación están enfocadas sobre los requisitos de usuarios, son ellos los que hacen parte activa de la audiencia de dichas pruebas. También debe ser incluido en este grupo el encargado de manejar las pruebas del sistema ya que su criterio de salida se debe considerar para el criterio de entrada de las pruebas de aceptación. Debido al lenguaje no técnico de las pruebas, la audiencia puede incluir desde personal técnico avanzado hasta usuarios dedicados al negocio como tal.

1.3.4.2 Tiempos de actividad. La planeación de la prueba de aceptación deberá comenzar a principios del ciclo de vida, cuando los requisitos de alto nivel sean levantados, debido a la importancia que tiene el criterio generado por dichas pruebas para la aceptación del proyecto en general.

La duración de las pruebas de aceptación, dependiendo de la magnitud del proyecto, pueden llegar a ser de unos pocos días o semanas. Pero se presentan casos en donde dicha duración puede durar mucho más tiempo, por lo que se debe hablar ya no de pruebas de aceptación sino de pruebas de sistemas, según lo expuesto por Craig y Jaskiel. Una de las diferencias más notorias entre las pruebas de aceptación de sistema está en quien lleva a cabo dichas pruebas. Entonces existe la necesidad de considerar la posibilidad de combinar las dos pruebas, dependiendo de si se dan dos situaciones en particular: cuando los usuarios tienen un rol activo en las pruebas de sistema, y cuando los usuarios no tienen rol diferenciado en ninguna de las dos pruebas.

1.3.4.3 Fuentes de información. Para generar correctamente el plan de pruebas de aceptación, se deben tener en cuenta documentos clave como el plan del

proyecto, el plan general de pruebas, la documentación de los requisitos, y si está disponible el manual de usuario de la aplicación. Para generar los casos de pruebas en este nivel, es ideal utilizar las especificaciones de los requisitos, pero muchos proyecto no poseen una documentación de requisitos como tal, por lo que es necesario emplear otros métodos o documentos que permitan realizar dichos casos de pruebas.

1.3.4.4 Responsabilidades del usuario. Los usuarios, o en su defecto sus delegados, juegan un papel determinantes dentro del procesos de las pruebas de aceptación. Algunas de las responsabilidades adquiridas por estos personajes incluyen: definir los requisitos; identificar los riesgos del negocio; crear, actualizar y/o revisar el plan de pruebas de aceptación; definir un escenario realista, proveer datos verídicos de las pruebas; ejecutar las pruebas; revisar la documentación; revisar los resultados y tener criterios de aceptación.

Se deben consideran entonces los siguientes elementos dentro de las responsabilidades del usuario:

- Pruebas de usabilidad. Una de las pruebas más difíciles de lograr es la prueba de usabilidad, debido a los diferentes puntos de vista que existen sobre la usabilidad. Parte de la dificultad radica en describir ciertos requisitos, por lo que se presentan en muchas ocasiones requisitos mal planteados, dando como resultado pruebas mal planeadas y ejecutadas. Otro de los obstáculos que pueden presentan las pruebas de usabilidad es la dificultad de aplicar las pruebas de aceptación en las fases del ciclo de vida del proyecto.
- Laboratorios de usabilidad. Los laboratorios de usabilidad son un tipo especial de entorno creado para evaluar la usabilidad del prototipo de un producto específico. El objetivo de dichos laboratorios es descubrir problemas de usabilidad antes que el sistema sea desarrollado, ahorrándose el costo de construir un sistema defectuoso.

Los actores principales en una prueba de usabilidad son: el usuario, el observador y un mecanismo de grabación. En el laboratorio de usabilidad el usuario se sienta en un cuarto de evaluación donde él o ella utilizan el prototipo. Las videocámaras graban las actividades del usuario, su expresión facial, las posiciones de su cuerpo y otros factores que pueden indicar la usabilidad del software que está a prueba. Al otro lado se encuentra un cuarto de observación donde hay un espejo de una sola vía (como los que utilizan en los cuartos de interrogatorios) y los observadores dispuestos a observar al usuario usar el sistema. En ciertas ocasiones los observadores le piden explicaciones al usuario de cómo resolvió una situación adversa, pero en general la comunicación entre el usuario y los observadores es mínima. Dentro del equipo de observadores puede haber miembros de laboratorio, desarrolladores y/o testers. Los resultados encontrados en la sesión realizada en el laboratorio de usabilidad serán usadas para identificar y corregir deficiencias de usabilidad.

- Pruebas alfa y beta. Las pruebas *alfa* pertenecen al nivel de aceptación y se llevan a cabo en la fase de desarrollo. Algunos proyectos involucran a usuarios en este tipo de pruebas, para hacerlas más realistas. Mientras que las pruebas *beta*, pertenecen al mismo nivel de pruebas (aceptación), pero enfocadas totalmente a la interacción con el usuario.
- Trazabilidad de requisitos. La trazabilidad de requisitos es un proceso importante en el nivel de aceptación, ya que garantiza que una o más pruebas están direccionadas a cada requisito correspondiente.

1.3.4.5 Gestión de la configuración. Durante las pruebas de aceptación, la gestión de la configuración de un software bajo prueba debe ser tan formal como lo es en un ambiente de producción. Debido a que el plan y las pruebas de aceptación son realizadas un poco antes de la fecha de implementación, resulta problemático encontrar muchos defectos o defectos de gran impacto, ya que hay

muy poco tiempo para efectuar todos los cambios necesarios en caso de encontrarlos. Lo ideal en estos casos, es que las pruebas de aceptación sean lo suficientemente extensas como para encontrar los errores de mayor impacto en los primeros niveles de las pruebas y no al final donde, por los problemas antes mencionados, generarían retrasos en el proyecto o un producto de baja calidad al usuario final.

1.3.4.6 Criterios de salida. Las categorías de medidas para cada uno de los niveles existentes en las pruebas son iguales, pero difieren en los valores y datos presentados en cada una de ellas. El criterio de salida para las pruebas de aceptación es de especial cuidado, ya que también son la salida para el sistema completo.

1.3.4.7 Estrategias de lanzamiento. Cuando las pruebas de aceptación han concluido satisfactoriamente, el sistema es entregado a los usuarios. Una de las estrategias para realizar dicho lanzamiento es la de entregar e instalar todo el sistema a cada uno de los clientes en simultaneo, pero dicha estrategia trae consigo el riesgo de encontrar problemas en la funcionalidad o la instalación que afectan al sistema en general. Por lo que Craig y Jaskiel sugieren unas estrategias para mitigar este y otros riesgos, a continuación expuestas.

- Pruebas pilotos. A diferencia de las pruebas beta (pruebas con usuarios en ambiente reales), las pruebas piloto consisten en ejecutar el sistema en un ambiente de producción, pero con uno o pocos usuarios.
- Implementación gradual. Otra de las alternativas planteadas, la implementación gradual, sugiere entregar el sistema a unos pocos usuarios a la vez. La desventaja de esta técnica radica en el tiempo disponible para entregarle a cada usuario el producto terminado

- Implementación por fases. La implementación por fases se da cuando se entrega el sistema a los usuarios incrementalmente. La complicación que posee esta técnica se halla en que la integración de todo el sistema puede llegar a ser complicada y habrá más pruebas de regresión a todo el sistema
- Implementación paralela. Esta técnica sugiere que se ejecuten al mismo tiempo (en ambiente de producción), tanto la aplicación existente como el nuevo sistema. Esta implementación requerirá recursos adicionales de software, hardware y personal.

1.3.4.8 Ambientes de Prueba. En las pruebas de aceptación es muy importante construir un ambiente tan realista como sea posible. Cualquier diferencia entre el ambiente de prueba y la realidad puede ser un riesgo. En las pruebas de alto nivel (Sistema y aceptación) el equipo de pruebas tiene que comparar el ambiente de pruebas con la realidad, buscar diferencias y tratar de eliminarlas por completo. Si esto no es posible, los miembros del equipo de prueba tienen que buscar la forma de manejar el riesgo de una situación irreal.

1.4 TÉCNICAS DE PRUEBAS

En lo que concierne a las técnicas de pruebas existen dos grandes grupos o categorías, pruebas de caja negra (o pruebas de comportamiento) y caja blanca. Las técnicas enmarcadas dentro de la categoría de caja negra, son pruebas basadas en requisitos, y como su nombre lo indica, son tratadas como una caja negra ya que se desconoce el trabajo interno del proceso que se está probando. Su funcionamiento se basa en darle un estimulo al sistema (entrada) y esperar que el resultado (salida) de ese estimulo sea aceptado de acuerdo con ciertos criterios, dependiendo de la prueba, de lo contrario la prueba será rechazada. A diferencia de las pruebas de caja negras, en las de caja blanca (también llamadas pruebas estructurales) se tienen tanto entradas como salidas, pero en este caso si

se tiene en cuenta el desarrollo del proceso interno para determinar la aceptación de la prueba.

También existen otros tipos de prueba como las pruebas de desempeño, en donde se encuentran clasificadas las pruebas de carga y estrés que posteriormente serán analizadas.

1.4.1 Pruebas de caja negra

A continuación se expondrán las técnicas de pruebas más utilizadas dentro de la categoría de caja negra. Estas incluyen pruebas de partición equivalente, análisis del valor limite, tablas de decisión, diagramas de causa-efecto y arreglos ortogonales.

1.4.1.1 Partición equivalente. La técnica de partición equivalente, muy común entre los testers y programadores, consiste básicamente en identificar y categorizar entradas que sean tratadas de manera similar por un sistema, y que produzcan el mismo resultado. Siendo más formales, la partición equivalente consta de clases de equivalencia que representan un conjunto de estados validos o inválidos para condiciones de entrada determinadas. En la tabla 1 se pueden ver los estados (valido o inválido) de las clases de equivalencia necesarias para las cuatro condiciones de entradas existentes. Un ejemplo claro del uso de la partición equivalente, propuesto por Craig y Jaskiel, es el de asignación de sillas en un avión, en donde se categorizan por clase ejecutiva y clase económica, dentro de las cuales pueden surgir otras categorías como la posición del asiento en el avión (ventana o pasillo). Se debe entonces categorizar o particionar los casos de prueba de manera correcta para evitar pruebas redundantes, en el caso que se consideren dos categorías diferentes siendo estas iguales; o pruebas faltantes, en el caso que se consideren dos categorías iguales siendo estas diferentes.

Tabla 1. Particiones equivalentes

Condición de entrada	Clases de equivalencia	
	Valida	**Invalida**
Especifica un rango	1	2
Requiere un valor especifico	1	2
Especifica un miembro de un conjunto	1	1
Es booleana	1	1

1.4.1.2 Análisis del valor limite. La prueba del análisis del valor límite consiste en tomar los valores límite (tanto superior como inferior) de una partición equivalente y analizar su comportamiento. Se prueban los valores limites ya que en muchos casos estos son dados a fallas dentro del sistema. También es recomendable analizar valores por encima del límite superior y por debajo del límite inferior.

1.4.1.3 Tablas de decisión. Este método utiliza una tabla que lista todas las posibles condiciones (entradas) y todas las posibles acciones (salidas). En toda tabla de decisión existen, además de condiciones y acciones, “reglas” para cada posible combinación de condiciones, en donde cada una de estas se identifica con un “si” o “no” dependiendo de si cumple la regla o no (o en su defecto 1 y 0), y para el caso de las acciones se pueden representar con una “x”. Dichas tablas no son usadas para documentar todo un sistema ya que requieren de trabajo intensivo para crearlas, pero son particularmente útiles para describir los componentes críticos de un sistema los cuales pueden ser definidos por un conjunto de reglas. Por ejemplo: pago de nomina, reglas de aseguramiento, calendario de amortizaciones, etc.

La tabla se lee de la siguiente manera: Leer la primera condición y seguir la fila a la derecha de la condición hasta que la regla satisfaga dicha condición. Después mirar cada regla que haya obtenido un resultado positivo sobre la condición, y seguir la columna hacia abajo hasta que la acción correspondiente tenga un “si”. Por último, cuando la última condición este satisfecha, aplicar las acciones indicadas por columna de la regla en la que se termino de leer. En la tabla 2 se expone un ejemplo sencillo que clarifica la técnica de tablas de decisión, por medio de la resolución de problemas de impresoras donde se ve claramente que las condiciones representan las entradas, las acciones a las salidas y las reglas a los casos de prueba.

Tabla 2. Tabla de decisión

		Reglas							
Condiciones	La impresora no imprime	Si	Si	Si	Si	No	No	No	No
	Se enciende una luz roja	Si	Si	No	No	Si	Si	No	No
	La impresora no es reconocida por el PC	Si	No	Si	No	Si	No	Si	No
Acciones	Verificar el cable de energía			X					
	Verificar el cable de conexión hacia el PC	X		X					
	Asegurar que el driver de la impresora este instalado	X		X		X		X	
	Verificar/Reemplazar la tinta	X	X			X	X		
	Verificar si hay papeles atascados		X		X				

Fuente: http://en.wikipedia.org/wiki/Decision_table.

1.4.1.4 Diagramas de causa-efecto. También llamados grafos de causa-efecto, los diagramas de causa-efecto son técnicas que proporcionan una solida representación de las condiciones lógicas y sus correspondientes acciones. El diagrama contiene un maquina de estado finito, cuya funcionalidad y salida es dependiente de las entradas actuales más las entradas anteriores. El resultado de las entradas anteriores se llama “estados”, y los comandos que cambios de un estado a otro son llamados “transiciones”.

1.4.1.5 Arreglos ortogonales. La prueba de arreglos ortogonales es una metodología utilizada para determinar qué casos de pruebas escribir y ejecutar cuando se tiene gran cantidad de variables a probar y la limitante de recursos hace imposible realizar pruebas a todas estas. Craig y Jaskiel sugieren el siguiente ejemplo para explicar el método de arreglos ortogonales en las pruebas de software: se tiene un sitio web que está alojado en varios servidores (IIS, Apache, Tomcat) con distintos sistemas operativos (WinS2000, WinS2003, Linux) y está siendo visto desde diferentes navegadores (Opera, Internet Explorer, Firefox) que poseen plug-ins distintos (Quick Player, Real Player, Media Player). La tabla 3 muestra el arreglo ortogonal del ejemplo anterior, donde se ve claramente que todos los pares de combinaciones han sido probados.

Tabla 3. Tabla ortogonal

Casos de Prueba	Explorador	Plug-in	Servidor	O.S.
1	Opera	QuickPlayer	IIS	WinS2003
2	Opera	RealPlayer	Apache	WinS2000
3	Opera	MediaPlayer	Tomcat	Linux
4	IE	QuickPlayer	Apache	Linux
5	IE	RealPlayer	Tomcat	WinS2003
6	IE	MediaPlayer	IIS	WinS2000
7	Firefox	QuickPlayer	Tomcat	WinS2000
8	Firefox	RealPlayer	IIS	Linux
9	Firefox	MediaPlayer	Apache	Win2003

Fuente: Craig, Rick. Jaskiel, Stefan. “Systematic Software Testing”. p. 172.

1.4.2 Pruebas de caja blanca

Las pruebas de caja blanca, también llamadas pruebas de caja de cristal, permiten examinar la estructura interna de uno o varios componentes que están siendo puestos a pruebas con el fin de crear casos de prueba basados en la implementación de dichos componentes. Estos casos de pruebas generados por los métodos de caja blanca tienen la finalidad de:

- Garantizar que todos los caminos independientes dentro de un modulo se han ejercitado al menos una vez.
- Ejercitar todas las decisiones lógicas bien sean falsas y/o verdaderas.
- Ejecutar todos los bucles en sus respectivos límites y dentro de sus límites operacionales.
- Ejercitar las estructuras internas de datos con el fin de asegurar su integridad.

1.4.2.1 Pruebas de ruta básica. La prueba de ruta básica, propuesta por Tom McCabe, permite al analista de pruebas "derivar una medida de complejidad lógica a partir de un diseño procedimental, y usar esta medida como guía para definir un conjunto base de rutas de ejecución"[6]. Los casos de prueba resultantes del conjunto base de rutas, ejecutarán todas las sentencias del programa al menos una vez durante las pruebas.

El método de ruta básica puede ser aplicado tanto a un diseño procedimental como a código fuente, y presenta una serie de pasos mencionados a continuación.

- Usando el diseño o el código como base, dibujar el correspondiente diagrama de flujo.
- Determinar la complejidad ciclomática del diagrama de flujo resultante.
- Determinar un conjunto base de caminos linealmente independientes.
- Preparar los casos de pruebas que obliguen la ejecución de cada camino perteneciente al conjunto base.

1.4.3 Pruebas de interfaz de usuario

Las pruebas de interfaz de usuario, como su nombre lo indica, se enfocan en la realización de pruebas hechas a la interfaz con la cual el usuario final interactúa. Las pruebas a la interfaz de usuario se realizan con el fin de ejecutar la aplicación en cuestión, teniendo en cuenta aspectos específicos de la interacción del usuario, expresados en la sintaxis y semántica de la interfaz. Estas pruebas brindarán una evaluación final de la usabilidad de la aplicación puesta a prueba.

La estrategia general de las pruebas de interfaz, propuesta por Roger Pressman, consta de: descubrir errores relacionados con mecanismos de interfaz, (como por

[6] Definición de Ruta básica propuesta por Tom McCabe y Pressman, Roger. "Software Engineering, A Practitioner's Approach

ejemplo los errores que se presentan en la ejecución de un link) y revelar errores en la implementación de la semántica de navegación, funcionalidad o presentación de contenido. Pressman sugiere unos objetivos para cumplir dicha estrategia[7]:

- Las características de la interfaz deben ser probadas para asegurar que las reglas de diseño, estéticas y relacionadas con contenido visual estén disponibles y libre de errores para el usuario.
- Probar de manera individual los mecanismos de la interfaz.
- Cada mecanismo de la interfaz debe probarse dentro del contexto de un caso de uso, para una categoría de usuario en específico.
- Toda la interfaz debe ser probada dentro del contexto de un caso de uso, con el fin de descubrir errores en la semántica de la interfaz.
- La interfaz debe ser probada dentro de una variedad de ambientes (navegadores, sistemas operativos, etc.) para asegurar la compatibilidad.

La interacción de los usuarios con el sistema es posible mediante los mecanismos que posee la interfaz de usuario. Dichos mecanismos se mencionan a continuación con su respectiva forma de probarlos:

- **Links**: Cada link de navegación de ser probado con el fin de asegurar que la función o contenido apropiado sea presentado al usuario. El analista de pruebas debe entonces generar una lista de links asociados al diseño de la interfaz de usuario, y luego ejecutarlas individualmente.

- **Formularios**: Según Pressman, las pruebas en un nivel general deben asegurar lo siguiente: las etiquetas deben identificar correctamente los campos

[7] Pressman, Roger. “Software Engineering, A practitioner’s Approach”- p. 573.

dentro del formulario correspondiente, y los campos obligatorios deben estar resaltados para una visualización correcta por parte del usuario; el servidor debe recibir toda la información contenida en el formulario, y garantizar que no se pierda ninguna información durante la transmisión de datos entre el cliente y el servidor; las características por defecto apropiadas, deben ser usadas cuando el usuario no selecciona una opción de un menú determinado; las funciones de los navegadores no deben corromper los datos ingresados en el formulario; y por último los procedimientos que realizan chequeo de errores sobre datos ingresados funcionen de manera correcta y generen mensajes de error pertinentes.

- De manera más especifica las pruebas de interfaz a formularios deben asegurar lo siguiente: los campos del formulario deben contener el tipo de dato apropiado con su respectiva amplitud; los formularios deben establecer una restricción apropiada que impida al usuario ingresar cadenas de caracteres más largas de lo permitido, o en su defecto tipos de datos donde no correspondan; todas las opciones en un menú desplegable deben tener un orden pertinente para el usuario; la función de “auto-relleno” de los navegadores no debe generar errores en el ingreso de los datos; y la tecla “tab” debe mover el cursor de manera secuencial entre los campos del formulario.

- **“Scripting del lado del cliente”**: Las pruebas de caja negra son llevadas a cabo para descubrir errores en procesamiento. Muchas veces, dichas pruebas se realizan junto con las pruebas de formularios, porque las entradas de los scripts son usualmente derivadas de datos provenientes como parte del procesamiento de los formularios. Una prueba de compatibilidad deberá asegurar que el lenguaje del script que se haya escogido trabaje en el ambiente de configuración de la aplicación.

- **HTML dinámico**: Cada página web que tenga contenido HTML dinámico debe ser ejecutada para asegurar que dicho contenido es presentado correctamente.

Adicional a esto, se debe ejecutar una prueba que asegure la compatibilidad del contenido HTML dinámico con el ambiente de configuración que soporta la aplicación.

- **Ventanas emergentes**: Se deben realizar una serie de pruebas sobre las ventanas emergentes para asegurar lo siguiente: la ventana emergente debe estar dimensionada y posicionada apropiadamente; la ventana emergente no debe ocultar la ventana original de la aplicación; el diseño visual de las ventanas emergentes debe ser consistente con el diseño visual de toda la interfaz; y por último, las barras de desplazamiento y otros mecanismos de control utilizando para el funcionamiento de las ventanas emergentes, deben estar ubicadas y funcionado correctamente. Cabe anotar que las ventanas emergentes deben usarse de manera prudente, o preferiblemente no utilizarlas ya que para la mayoría de los usuarios es molesto y engorroso el uso de estas.
- **Streaming**: Las pruebas hechas a contenidos streaming deben demostrar que el contenido que se está transmitiendo sea presentado correctamente, y además, pueda ser pausado y reiniciado sin ningún problema.
- **Cookies**: Para realizar pruebas sobre cookies es necesario realizarlas tanto del lado del servidor así como del cliente. Las pruebas en el lado del servidor deben garantizar que la cookie contenga la información correcta y sea transmitida al cliente cuando este requiere una función o información específica. Además la persistencia del cookie debe ser probada para asegurar que la fecha de terminación de esta sea la apropiada. En tanto, las pruebas del lado del cliente, deben determinar si la aplicación vincula correctamente las cookies a una petición en específico.

1.4.3.1 Pruebas de usabilidad. Según Pressman, las pruebas y revisiones de usabilidad están diseñadas con el objetivo de determinar sí la interfaz de la aplicación permite una fácil utilización de la misma por parte de los usuarios finales. Sugiere Pressman entonces, que dichas pruebas sean diseñadas por el

equipo de pruebas, pero ejecutadas por usuarios (o en su defecto representantes de estos). Para esto es ideal seguir los siguientes pasos:

- Definir un conjunto de categorías para las pruebas de usabilidad e identificar metas para cada una de ellas.
- Diseñar pruebas que permitan evaluar los objetivos previamente definidos para cada categoría.
- Seleccionar las personas que harán parte de las pruebas.
- Capacitar a los participantes con la aplicación a la cual se le va a realizar la prueba.
- Desarrollar un mecanismo para valorar la usabilidad de la aplicación.

Para la consecución del primer paso en las pruebas se deben identificar las categorías o aspectos de usabilidad, seguido de la meta u objetivo que se desee cumplir. A continuación se mencionaran los aspectos más importantes con su correspondiente objetivo:

- Interactividad: Determinar si los mecanismos de interacción, (como por ejemplo los menús, botones, etc.) son fáciles de usar y entender.
- Disposición: Asegurar que los mecanismos de navegación, el contenido y las funciones estén dispuestas de manera que los usuarios pueden encontrarlos de forma fácil y rápida.
- Facilidad de lectura: Garantiza que lo publicado en la aplicación, tanto escrito como grafico, sea de fácil entendimiento.
- Estética: Asegurar que la interfaz tenga un diseño (colores, figuras, etc.) agradable a la vista de la usuarios.

- Características de presentación: Garantizar que la aplicación utilice de manera óptima el tamaño y la resolución de la pantalla.
- Sensibilidad de tiempo: Determinar si es necesario utilizar funciones o características asociadas con la utilización de tiempos.
- Personalización: Determinar si algunos componentes de la aplicación son personalizables de acuerdo con los gustos y necesidades del los usuarios.
- Accesibilidad: Determinar si la aplicación tendrá opciones de accesos para usuarios con algún tipo de discapacidad.

1.4.4 Pruebas de desempeño

Debido a que la mayoría de las aplicaciones desarrolladas por el Centro de Informática de la Universidad EAFIT son aplicaciones Web, es necesario realizar pruebas de desempeño. Las pruebas de desempeño son usadas para descubrir problemas que pueden resultar debido a factores como: falta de recursos en el servidor, ancho de banda inapropiados, inadecuada capacidad de las bases de datos, debilidades del sistema operativo, funcionalidades de la aplicación mal diseñadas, entre muchos otros factores que pueden llevar a disminuir el desempeño del servidor. El propósito de las pruebas de desempeño pueden definirse en dos: Entender como el sistema responde a la carga (por ejemplo, número de usuarios, número de transacciones, o volumen promedio de datos), y recoger métricas que permitan modificaciones de diseño para mejorar el desempeño de la aplicación.

Todas las pruebas de desempeño, nombradas anteriormente, son diseñadas para simular situaciones del mundo real. Como el número usuarios conectados simultáneamente a la aplicación crece, o el número de transacciones se incrementa, o la cantidad de información, descargada o subida, aumenta, las pruebas de desempeño deben ser capaces de contestar las siguientes preguntas:

- ¿El tiempo de respuesta de servidor cae hasta niveles notables e inaceptables?
- ¿En qué punto, en términos de usuarios, transacciones, o carga de datos, el desempeño del servidor se convierte en inaceptable?
- ¿Cuáles son los componentes que son responsables del bajo desempeño?
- ¿Cuál es el tiempo de respuesta promedio para los usuarios bajo diferentes condiciones de carga?
- ¿Tiene algún impacto el bajo desempeño del servidor en la seguridad del sistema?
- ¿La exactitud y confianza de la aplicación se ve afectada con el aumento de carga?
- ¿Qué pasa cuando las cargas realizadas son más grandes que la capacidad máxima del servidor?

Para que los desarrolladores puedan contestar estas preguntas, se realizan dos pruebas diferentes de desempeño: Pruebas de Carga y Pruebas de Estrés.

1.4.4.1 Pruebas de Carga. La intención de las pruebas de carga es determinar cómo la aplicación y su servidor responde a varias condiciones de carga. La prueba se realiza teniendo en cuenta tres variables:

N, el número de usuarios conectados
T, el número de transacciones en línea por usuario por unidad de tiempo
D, la carga de datos procesada por el servidor por transacción

En cada caso, estas variables son definidas dentro de límites normales de operación. Mientras cada condición es ejecutada se realizan una o más de las

siguientes mediciones: Promedio de respuesta del usuario, promedio de descarga para una unidad de datos estandarizada o tiempo promedio de proceso de una transacción. El equipo de pruebas analiza estas mediciones para determinar si un descenso en el desempeño puede ser rastreado con una combinación determinada de N, T o D.

Las pruebas de carga también pueden ser usadas para valorar las velocidades de conexión recomendadas para los usuarios de la aplicación. Rendimiento promedio, P, es calculado de la siguiente forma:

$$P = N \times T \times D$$

Para ejemplificar esa medida Pressman menciona un ejemplo de una página de noticias de deportes. En determinado momento hay conectados 20000 usuarios que realizan un requerimiento cada dos minutos en promedio. Cada transacción requiere que la aplicación descargue una nueva noticia que tiene en promedio 3Kb de peso. Entonces aplicando la formula:

$$P = [20000 \times 0.5 \times 3Kb]/60 = 500 \text{ Kbytes/seg}$$
$$P = 4 \text{ megabits por segundo}$$

La conexión del servidor debe entonces soportar esta tasa de transmisión y debe ser probado para asegurase que sea así.

1.4.4.2 Pruebas de estrés. Las pruebas de estrés se derivan de las pruebas de carga, pero a diferencia de estas, las pruebas de estrés están forzadas a llegar y sobrepasar los límites operacionales de la prueba. Para Pressman este tipo de pruebas intentan responder a cada una de las siguientes preguntas:

- ¿El sistema va disminuyendo su capacidad de manera paulatina o por el contrario apaga el servidor una vez llega a su capacidad limite?
- ¿El servidor genera mensajes alertando que no está disponible, y están los usuarios enterados de que el servidor esta caído?
- ¿El servidor “encola” las peticiones por recursos y vacía la cola una vez que la demanda de capacidad haya disminuido?
- ¿Las transacciones se pierden cuando la capacidad del servidor es excedida?
- ¿La integridad de los datos se ve afectada por la capacidad límite del servidor?
- ¿Qué valores de N, T y D (definidos previamente en “pruebas de carga”) obligan al servidor a fallar? ¿Cómo se manifiesta dicha falla? ¿Existen notificaciones automáticas dirigidas al equipo de soporte encargado de dicho servidor?
- ¿Si el sistema falla, cuánto tiempo se demorara en restablecer el servicio?
- ¿Existen ciertas funcionalidades de la aplicación (como por ejemplo procesamiento intensivo o streaming de datos) que se deshabilitan cuando el sistema llega a un 80% o 90% de su capacidad?

2. UNIDAD DE ASEGURAMIENTO DE CALIDAD DEL CENTRO DE INFORMÁTICA

El centro de informática de la Universidad EAFIT se ha caracterizado por su iniciativa y creatividad a la hora de desarrollar software, permitiendo a la institución diferenciarse entre las demás universidades por los servicios en línea que presta a la comunidad universitaria y además, por general valor adicional comercializando los productos desarrollados.

Hasta este momento todos los desarrollos llevados a cabo dentro del Centro de Informática han pasado por procesos de pruebas entregados a terceros, incrementando el costo de producción y generando el riesgo de que información confidencial o el mismo desarrollo llegue a manos de otras personas por manejo inescrupuloso de la información por parte de la empresa que está prestando el servicio. O incluso, puede presentarse el caso donde las pruebas a la aplicación sean realizadas por los mismos desarrolladores, los cuales no ejecutan una metodología estandarizada que permita llegar a encontrar la mayoría los errores de programación presentes en el desarrollo.

A continuación se presenta una aproximación para la conformación de una Unidad de prueba considerando su ubicación dentro del organigrama del Centro de informática, describiendo las funciones que debe desarrollar, y definiendo los roles y características de cada uno de los miembros.

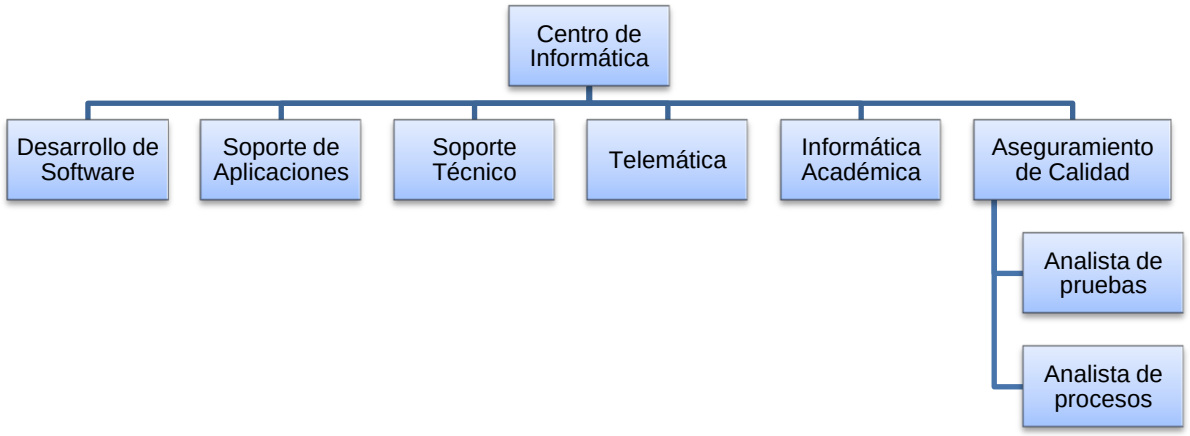
2.1 UBICACIÓN DENTRO DE LA ORGANIZACIÓN

Existen varias formas de organizar una unidad de prueba, pero realmente no existe un consenso sobre que es adecuado o que es erroneo, de hecho, expertos aseguran que un tipo de organización puede funcionar para unas compañías y fallar para otras. ¿Cómo debe estar estructurada entonces el área de

Aseguramiento de Calidad del Centro de informática de la Universidad EAFIT? Esta respuesta depende principalmente de las políticas administrativas, cultura corporativa, habilidades y conocimiento de los participantes y del riesgo que implica realizar una modificación en la estructura del Centro de informática.

De acuerdo con las reuniones realizadas con Jefe del Centro de informática se llegó a la conclusión que la mejor opción de organización para el área de Aseguramiento de Calidad es crear un grupo independiente de pruebas compuesto por dos analistas, uno de procesos y otro de pruebas, y un coordinador dependiente del Jefe del Centro de Informática. En el siguiente organigrama puede verse la propuesta de organización del área de Aseguramiento de Calidad.

Ilustración 5. Organigrama del Centro de Informática



Fuente: Jefe del centro de Informática de la Universidad EAFIT

2.2 FUNCIONES

En la unidad de Aseguramiento de Calidad del Centro de informática, existirán dos funciones principales (como se muestra en la Ilustración 5), procesos y pruebas. El foco central de este proyecto está orientado a la descripción e implementación de los procesos y funciones llevados a cabo por la sección de pruebas. Es válido entonces aclarar que la creación de la sección de procesos no será propuesta en este proyecto.

A continuación se expondrán las funciones principales, con sus respectivos encargados, que debe desarrollar la sección de pruebas dentro de la Unidad de Aseguramiento de la calidad:

- Desarrollar la etapa de pruebas del ciclo de desarrollo de software según la metodología propuesta en el capitulo siguiente, tanto en el software que este en desarrollo, como también a las aplicaciones que se encuentran codificadas totalmente. Los encargados de realizar esta labor en primer lugar, deben ser los analistas de prueba orquestados por el Coordinador de la Unidad de Aseguramiento de Calidad con la ayuda indispensable de la unidad de Desarrollo de Software (que suministra tanto la aplicación como su documentación) y de la unidad de Soporte Técnico (que se encarga de la configuración del ambiente de pruebas).
- Determinar, de acuerdo a los resultados y conclusiones arrojadas por las pruebas, los aplicativos que saldrán a producción para el uso de toda la comunidad Eafitense. Esta función debe estar a cargo del Coordinador de la Unidad de Aseguramiento de la Calidad junto con el Coordinador de la Unidad de Desarrollo y el jefe del Centro de Informática.
- Revisar que el esquema de pruebas propuesto sea el adecuado para el Centro de Informática y realizar continuas revisiones del mismo con el fin de mejorarlo.

2.3 MIEMBROS DE LA UNIDAD

El grupo que integra la unidad de Aseguramiento de Calidad está conformado por: el Coordinador y los analistas. Para la sección de pruebas, inicialmente, es ideal tener (sugerido por el jefe del Centro de Informática) un Coordinador que estará a cargo de dos analistas. Para cada uno de estos roles, el interesado en pertenecer a la unidad debe cumplir con ciertos perfiles dependiendo del cargo, a continuación se exponen cada uno de ellos:

2.3.1 Perfil del analista

Encontrar y luego contratar a la persona ideal para un trabajo en específico son tareas bastantes arduas para cualquier organización hoy en día. Por ejemplo, el Coordinador de la unidad de Desarrollo de Software contrata normalmente jóvenes recién graduados en alguna carrera afín (como ingeniería de sistemas, ciencias de la computación o ingeniería de software) al proyecto que desarrollarán. Desafortunadamente para el futuro Coordinador de Aseguramiento de Calidad, encontrar analistas de prueba es un poco complicado por el hecho de que poco universidades ofrecen currículos enfocados en pruebas de software, por lo que el coordinador debe contratar personas que, en general saben poco de pruebas y capacitarlas, o en su defecto gente especialista en desarrollo o infraestructura, que igual deberán capacitar.

A continuación se menciona algunas de las características más importantes que necesita un analista para desempeñar de manera exitosa su trabajo:

- Ser inquisitivo
- Tener un conocimiento funcional del negocio
- Estar orientado hacia el detalle

- Ser lógico
- Ser de mente abierta
- Tener experiencia técnica, sin necesidad de tener experiencia en programación
- Tener experiencia en pruebas
- Ser flexible
- Tener disposición para el trabajo en equipo
- Trabajar bajo presión
- Tener conocimiento de ciertas herramientas
- Tener sentido común
- Entender el ciclo de vida del desarrollo de software

Cabe anotar que no es necesario que un analista tenga que cumplir todas estas habilidades para poder hacer parte de la unidad de Aseguramiento de Calidad.

2.3.2 Perfil del coordinador

Según Henry Mintzberg, importante autor de administración, un administrador (o coordinador en este caso) tiene cierta autoridad sobre una organización. Dicha autoridad genera un estatus que lleva al coordinador a asumir roles interpersonales, que le permiten tener acceso a la información, y finalmente este estará en capacidad de tomar decisiones[8]. Todos estos roles aplican de manera general para el Coordinador de Aseguramiento de Calidad y a continuación se mencionaran en detalle cada uno de estos:

[8] Craig, Rick D., y Stefan P. Jaskiel, "Systematic Software Testing". 5a edición, Artech House Publishers, 2006.

- *Rol interpersonal*: Debido a la naturaleza de las pruebas de software, el Coordinador de Aseguramiento de Calidad deberá coordinar y trabajar con muchas personas miembros de distintas unidades dentro del Centro de Informática, de la misma Universidad, y agentes externos, por lo que se hace indispensable que el coordinador sepa manejar unas buenas relaciones con todos ellos.

  Otro de los roles interpersonales que debe cumplir el coordinador, será el de líder, ya que a su cargo tendrá analistas que dependerán de este para llevar a cabo las tareas de pruebas a cabalidad.

- *Rol informacional*: El coordinador de la unidad de Aseguramiento de la Calidad recibirá toda la información correspondiente al proyecto de prueba, que podrá incluir reportes de estado, planes, estimados, y reportes de actividades diarias. Además el coordinador asumirá el rol de comunicar todo lo que pase en lo referente a las pruebas a los miembros de la unidad, la alta gerencia, la unidad de Desarrollo y la comunidad de usuarios en general.

- *Rol decisional*: El coordinador de la unidad deberá asumirá este rol, teniendo en cuenta que es responsable de todas las decisiones que se tomen con respecto a las pruebas. Una de las decisiones más importantes que debe tomar el coordinador es determinar si la aplicación que está siendo objeto de pruebas puede salir a producción.

3. METODOLOGÍA DE PRUEBA

De acuerdo a los dos principales enfoques de estrategias de pruebas presentados en el marco teórico, se puede concluir que la estrategia se apoya en una metodología desarrollada de acuerdo a las necesidades particulares del proyecto u organización para la que se crea, y que tiene como finalidad estandarizar, aumentar la trazabilidad y mejorar el resultado de las pruebas, y por tanto la calidad del software desarrollado.

Considerando las necesidades actuales del Centro de Informática de la Universidad EAFIT a continuación se plantea una metodología de prueba que permita a los miembros del área de Aseguramiento de Calidad contar con un procedimiento claro y efectivo basado en los enfoques de Pressman, Craig y Jaskiel expuestos con anterioridad. El objetivo de la Metodología de prueba es proporcionar a los analistas de pruebas, procedimientos y herramientas de fácil uso, poco densos y de gran efectividad a la hora de realizar pruebas de sistema a los desarrollos del Centro de Informática. En un futuro, es posible que la metodología soporte pruebas de otros niveles como Unidad, Integración o Aceptación. A continuación se muestra una tabla resumen con los pasos principales para llevar a cabo la metodología propuesta.

Tabla 4. Resumen metodología

3.1. PLANEACIÓN DE PRUEBAS
3.2. PREPARACIÓN DEL AMBIENTE DE PRUEBAS
3.3. EJECUCIÓN DE PRUEBAS DE INTERFAZ
3.3.1. EJECUCIÓN DE PRUEBAS DE USABILIDAD
3.4. EJECUCIÓN DE PRUEBAS DE INTEGRIDAD
3.5. EJECUCIÓN DE PRUEBAS DE DESEMPEÑO: CARGA
3.6. EJECUCIÓN DE PRUEBAS DE DESEMPEÑO: STRESS
3.7. PRESENTACIÓN DE RESULTADOS
3.8. CONCLUSIONES

3.1 PLANEACION DE PRUEBAS

1. Determinar la aplicación que va a ser sometida al proceso de pruebas. Esto debido a que todo desarrollo que se lleve a cabo en el CINF no puede ser puesto a prueba por diversas razones (falta de recursos, disponibilidad de tiempo, poca criticidad, etc.). Por lo tanto, se hace necesario que el coordinador de Desarrollo y el coordinador de Soporte a aplicaciones determinen que aplicación debe ser sometida al proceso de prueba y determinar el alcance de las pruebas a realizar, en caso que la aplicación requiera ser probada.

2. Definir qué tipos de pruebas se realizarán a la aplicación previamente seleccionada. Esta actividad debe ser llevada a cabo por el coordinador de Aseguramiento de Calidad.

3. Una vez se hayan determinado los tipos de pruebas que se van a realizar, es necesario seleccionar los ítems y características de la aplicación a probar.

4. Determinar que partes o módulos de la aplicación tienen más riesgo a fallas. Esto con el fin de dar prioridad a ciertos partes o módulos de la aplicación, que tienen más probabilidades de fallas, en el proceso de pruebas.

5. Definir claramente los criterios de aceptación.

6. Realizar un cronograma con todas las actividades consideradas dentro de todo el proceso de pruebas.

3.2 PREPARACIÓN DEL AMBIENTE DE PRUEBAS

1. Informar al Coordinador de Aseguramiento de Calidad, por parte del Coordinador de Desarrollo de Software, que el desarrollo de la aplicación que va a hacer sometida al proceso de pruebas ha concluido (Ver anexo A).

El Coordinador de Desarrollo debe adjuntar toda la información necesaria para que pueda ser implementado el ambiente de pruebas de la aplicación, incluyendo:

- Requisitos funcionales y no funcionales
- Manual de usuario (Sí aplica)
- Casos de uso
- Diagrama de datos (especificando de donde deben obtenerse los datos para poblar la base de datos)
- Especificaciones técnicas:

  -Hardware y software necesarios

  -Servidor de aplicaciones

  -Motor de bases de datos

- Usuarios y contraseñas para acceder a las bases de datos.

2. Solicitar al Área de Soporte Técnico la implementación del ambiente de pruebas (Ver anexo B) para la aplicación a probar. Esta solicitud la realiza el Coordinador de Aseguramiento de la Calidad inmediatamente después de recibir la información enviada por el Coordinador de Desarrollo de Software.

El ambiente de pruebas debe incluir:

- Asignación de un servidor con las especificaciones técnicas requeridas.
- Creación de los perfiles de usuario necesarios para tener acceso a todas las instancias del aplicativo.
- Instalación del motor de la base de datos que alojará los datos para realizar las pruebas.

3. El Coordinador de Soporte Técnico debe informar cuando esté listo el ambiente de prueba por medio de una carta (Ver anexo C) dirigida al Coordinador de Aseguramiento de Calidad.

3.3 EJECUCIÓN DE PRUEBAS DE INTERFAZ

De acuerdo al desarrollo teórico de las pruebas de interfaz presentado en el Marco Teórico de este documento, se realizará la descripción de los pasos que deben seguir los analistas al realizar pruebas sobre el interfaz de usuario.

1. Realizar una lista de pantallas de la aplicación y categorizarlas por tipo de usuario. Existen pantallas comunes a varios tipos de usuarios, por ejemplo, la pantalla de ingreso.

2. Seleccionar las pantallas que sean comunes a más usuarios, ordenarlas empezando por las que tengan en común más tipos de usuarios y terminando con aquellas propias de cada uno, esto se hace con el fin de agilizar el proceso de prueba.

Se recomienda realizar un mapa jerárquico, abarcando todas las pantallas, que refleje la navegación por la aplicación. Realizar el ordenamiento es más fácil a partir del mapa.

3. Definir de los siguientes tipos de prueba aquellas que sean convenientes a aplicar para la pantalla seleccionada de la lista anterior, para posteriormente diligenciar las plantilla (Ver anexo D). Cada analista determina el número de pruebas a aplicar por pantalla.

- *Pruebas de link.* Para probar un link es necesario dar clic en él y comprobar si está funcionando y re-direccionando al contenido apropiado.
- *Pruebas de formulario:* Realizar pruebas de formulario incluye:

  -Asegurar que las etiquetas identifiquen correctamente los campos correspondientes.

  -Comprobar que los campos obligatorios están resaltados para una visualización correcta.

  -Los campos del formulario deben contener el tipo de dato apropiado con su respectiva amplitud.

  -Los formularios deben establecer una restricción apropiada que impida al usuario ingresar cadenas de caracteres más largas de lo permitido, o en su defecto tipos de datos donde no correspondan.

  -Todas las opciones en un menú desplegable deben tener un orden pertinente para el usuario.

-La función de “auto-relleno” de los navegadores no debe generar errores en el ingreso de los datos.

-La tecla tabulador debe mover el cursor de manera secuencial entre los campos del formulario.

-Además, el analista por cada campo del formulario debe tener en cuenta las posibles condiciones de entrada, utilizando las técnicas de partición equivalente y análisis de valor límite, y de acuerdo ellas definir la cantidad y el valor de los datos a ingresar para realizar las pruebas de forma eficiente. Las posibles condiciones de entrada son:

El valor del campo está especificado dentro de un rango. Para realizar pruebas a este tipo debe ingresarse tres valores validos, un valor dentro rango y los dos valores límite, y dos valores inválidos, es decir fuera del rango.

El valor del campo requiere un valor específico. Debe ingresarse el valor valido y dos inválidos, uno mayor y el otro menor que el valor aceptado.

El valor del campo especifica un miembro de un conjunto. Debe ingresarse un valor dentro del conjunto (valor valido) y otro por fuera de él.

El valor del campo especifica una variable booleana. Debe ingresarse un valor las dos posibles opciones de la variable.

- *Pruebas scripts del lado del cliente:* Realizar estas pruebas implica comprobar que los scripts de la pantalla realicen un chequeo sobre los datos ingresados en los formularios de dicha pantalla y además, generen los mensajes de error pertinentes.
- *Pruebas del contenido HTML Dinámico:* Debe comprobarse que cada página que contenga contenido HTML dinámico lo presente de forma adecuada y pertinente.
- *Pruebas de ventanas emergentes:* Las pruebas de ventanas emergentes buscan lo siguiente: la ventana emergente debe estar dimensionada y

posicionada apropiadamente; la ventana emergente no debe ocultar la ventana original de la aplicación; el diseño visual de las ventanas emergentes debe ser consistente con el diseño visual de toda la interfaz; y por último, las barras de desplazamiento y otros mecanismos de control utilizando para el funcionamiento de las ventanas emergentes, deben estar ubicadas y funcionado correctamente.

- *Pruebas de "streaming":* Las pruebas hechas a contenidos streaming deben demostrar que el contenido que se está transmitiendo sea presentado correctamente, y además, pueda ser pausado y reiniciado sin ningún problema.

4. Realizar las pruebas a las pantallas seleccionadas con anterioridad y anotar los resultados en la plantilla de pruebas de interfaz (Ver anexo D). Una copia de dicha plantilla debe ser enviada al analista de desarrollo después de realizar las pruebas para que él depure los errores encontrados.

5. El analista de desarrollo debe informar al analista de prueba sobre la depuración de los errores encontrados en el proceso de prueba (Ver anexo E).
El analista de desarrollo debe nombrar en las anotaciones los errores que no fueron depurados, así como también las posibles pantallas que pudieron verse afectadas por el proceso de depuración.

6. Realizar las pruebas a los errores corregidos y a todas aquellas pantallas que pudieron verse afectadas en el proceso de depuración. Las pantallas que pudieron verse afectadas por la depuración son recomendadas por el analista de desarrollo que realiza las correcciones.

En caso de encontrar algún error, bien sea nuevo o que haya sido informado anteriormente al equipo de desarrollo, el analista de pruebas debe incluirlo dentro de una plantilla (Ver anexo D), incrementando el número de versión para

identificar que se ha realizado una prueba nueva a la pantalla. En número de versión se incrementa en la medida que se vayan realizando más pruebas a la misma pantalla.

7. Definir si los errores encontrados en la última revisión realizada afectan gravemente el desempeño de la aplicación. Para ello, debe convocarse al coordinador de la Unidad de Aseguramiento de la Calidad, al Coordinador de Desarrollo de Software y a los analistas, tanto de prueba como de desarrollo, a una reunión para definir el impacto que generan los errores encontrados en la última prueba. Es aconsejable cuantificar y priorizar el impacto entre uno y cinco, siendo el indicador de mínimo impacto uno, y cinco, el de mayores implicaciones y por lo tanto, son estos últimos, los que deben depurarse antes de lanzar la aplicación en vivo.

Los asistentes a la reunión, están en libertad, de acuerdo a su experiencia y a las necesidades del proyecto, de definir las acciones que se deben tomar para solucionar los problemas mencionados anteriormente. El resultado de la reunión debe ser escrito en el formato de reuniones con el que cuenta el centro de informática (Ver anexo F).

Una vez el analista de desarrollo realice la depuración, el proceso debe volver al paso 5 de esta metodología y repetir el ciclo hasta que los miembros de la reunión consideren que el software que se está probando cumple con las requerimientos a nivel de interfaz.

Ejecución de pruebas de usabilidad

Las pruebas de usabilidad son una extensión de las pruebas de interfaz, ya que están directamente relacionadas con la apariencia, distribución y usabilidad de la misma. Pero las pruebas son realizadas por un pequeño grupo de usuarios seleccionados para tal fin.

A continuación se describen los pasos, incluyendo la encuesta al usuario, que deben hacerse cuando se vayan a realizar pruebas de usabilidad a una aplicación del Centro de Informática.

1. Definir de las opciones presentadas a continuación, un conjunto de categorías para las pruebas de usabilidad, ya que las preguntas de la encuesta que se le realizará al usuario (Ver anexo J) dependerán del conjunto de categorías seleccionado.

La selección de cada categoría depende de la aplicación a probar ya que no todas las categorías son evaluadas en las diferentes aplicaciones.

- Interactividad: Determina si los menús, botones, barras de desplazamiento, ayuda, y otros mecanismos de interactividad, son fáciles de usar y entender.
- Disposición: Asegura que los mecanismos de navegación, el contenido y las funciones estén dispuestas de manera que los usuarios pueden encontrarlos de forma fácil y rápida.
- Facilidad de lectura: Garantiza que lo publicado en la aplicación, tanto escrito como gráfico, sea de fácil entendimiento.
- Estética: Asegura que la interfaz tenga un diseño (colores, figuras, etc.) agradable a las vista de usuarios.
- Características de presentación: Garantiza que la aplicación utilice de manera óptima el tamaño y la resolución de la pantalla.
- Sensibilidad de tiempo: Determina si es necesario utilizar funciones o características asociadas con la utilización de tiempos.
- Personalización: Determina si algunos componentes de la aplicación son personalizables de acuerdo con los gustos y necesidades del los usuarios.

2. Realizar la encuesta que se le dará a los usuarios. Para realizar la encuesta refiérase al formato de encuesta (Ver anexo J) y seleccione las preguntas que considere pertinentes de acuerdo a las características seleccionadas en el numeral anterior.

3. Seleccionar un grupo impar de usuarios, en lo posible mas de 7 (O el número que considere apropiado el Coordinador de Aseguramiento de la Calidad, asegúrese que el número sea impar) de la aplicación a probar para luego realizarles la encuesta.

4. Capacitar los usuarios seleccionados en la aplicación a probar.

5. Darles acceso a la aplicación por el período de tiempo que considere el Coordinador de Aseguramiento de la Calidad.

6. Distribuir la encuesta entre los usuarios seleccionados.

7. Revisar las respuestas de los usuarios y sacar un promedio por cada pregunta, así:

- Por cada pregunta debe sumar los valores equivalentes a la respuesta de esa pregunta y luego dividirlo por el número total de personas que contestaron la pregunta.
- En caso que la pregunta sea de opción Si/No, debe tomar la decisión por mayoría numérica.
- Las respuestas a preguntas abiertas como comentarios y sugerencias, deben ser resumidas por el analista de pruebas.

El Coordinador de Aseguramiento de Calidad y al Coordinador de Desarrollo de Software son los encargados de definir si las consideraciones hechas por los usuarios son viables y vale la pena implementarlas. Para registrar las decisiones de la reunión deben usar el formato para reuniones (Ver anexo F).

8. En el caso que existan cambios a realizar, enviar una copia al analista de desarrollo para que los implemente. Si se adicionan nuevos módulos a la aplicación es necesario que se le realicen las pruebas que sean convenientes.

3.4 EJECUCIÓN DE PRUEBAS DE INTEGRIDAD

A continuación se describen los pasos que deben llevarse a cabo en el proceso de pruebas cuando se realicen pruebas de integridad. Tenga en cuenta que estas pruebas se realizan a atributos de registro es decir, campos que almacenen información en la base de datos.

1. Seleccionar la pantalla a probar de acuerdo los numerales 1 y 2 de las pruebas de interfaz.

2. Determinar los campos a probar de la pantalla seleccionada, teniendo en cuenta que para el interés de estas pruebas deben ser campos que escriban en las bases de datos.

3. Probar los campos con valores de prueba, y diligenciar la plantilla correspondiente (Ver anexo G), basados en las técnicas de partición equivalente y análisis de valor limite. Por ejemplo:

- Un campo de tipo numérico, como el documento de identidad, debe ser probado con los siguientes valores: un nulo, un valor valido y dos valores no validos.

Después de ingresar los datos en el formulario, el analista debe verificar en las tablas de la base de datos que el valor ingresado sea igual al valor escrito y además, este en la tabla adecuada.
5. Una vez terminada la prueba, una copia de dicha plantilla debe ser enviada al analista de desarrollo para que depure los errores encontrados.

6. El analista de desarrollo debe informar al analista de prueba sobre la depuración de los errores encontrados en el proceso de prueba (Ver anexo E).

El analista de desarrollo debe describir en las anotaciones los errores que no fueron depurados, así como también los posibles campos que pudieron verse afectados por el proceso de depurado.

7. Realizar las pruebas a los errores corregidos y a todos aquellos campos que pudieron verse afectados en el proceso de depuración. Los campos que pudieron verse afectados por la depuración son recomendados por el analista de desarrollo que realiza la depuración.

En caso de encontrar algún error, bien sea nuevo o que se haya informado anteriormente al equipo de desarrollo, el analista de pruebas debe incluirlo dentro de una plantilla (Ver anexo F), incrementando el número de versión para identificar que se ha realizado una prueba nueva a la pantalla. En número de versión se incrementa en la medida que se vayan realizando más pruebas a la misma pantalla.

8. Definir si los errores encontrados en la última revisión realizada afectan gravemente el desempeño de la aplicación. Para ello, debe convocarse al coordinador de la Unidad de Aseguramiento de la Calidad, al Coordinador de Desarrollo de Software y a los analistas, tanto de prueba como de desarrollo, a una reunión para definir el impacto que generan los errores encontrados en la

última prueba. Es aconsejable cuantificar el impacto entre uno y cinco, siendo el indicador de mínimo impacto uno, y cinco, el de mayores implicaciones y por lo tanto, son estos últimos, los que deben depurarse antes de lanzar la aplicación en vivo.

Los asistentes a la reunión, están en libertad, de acuerdo a su experiencia y las necesidades del proyecto, de definir las acciones que se deben tomar para solucionar los problemas mencionados anteriormente. El resultado de la reunión debe ser escrito en el formato de reuniones con el que cuenta el centro de informática (Ver anexo E).

Una vez el analista de desarrollo realice la depuración, el proceso debe volver al paso 6 de esta metodología y repetir el ciclo hasta que los miembros de la reunión consideren que el software que se está probando cumple con los requerimientos de integridad.

3.5 EJECUCIÓN DE PRUEBAS DE DESEMPEÑO: CARGA

1. Determinar el número de usuarios que tienen acceso a la aplicación.

2. Seleccionar la herramienta de prueba adecuada teniendo en cuenta factores como: el sistema operativo en el que se van a realizar las pruebas, el costo y la cantidad de usuarios virtuales soportados por la herramienta.

Para esta metodología se sugiere el Software de Prueba WAPT (Web Application Testing), ya que se realizaron pruebas con la versión “Trial” de la herramienta y cumplió con muchas de las necesidades que se presentaron a la hora de construir la metodología.

3. Definir los casos de pruebas teniendo en cuenta las siguientes variables:

- Número de usuarios virtuales, teniendo en cuenta que este número este dentro del límite de usuarios que tienen acceso a la aplicación. La cantidad de usuarios virtuales por cada caso de prueba debe ser fija.
- Duración total de la prueba. Se da en minutos
- Tiempo entre clics de los usuarios virtuales.
- Creación de los perfiles a asignar para los usuarios virtuales. Se recomienda crear un perfil por cada rol que exista en la aplicación.

Debe tener en cuenta que, si la herramienta seleccionada lo permite, debe activar el “ip-spoofing”, que asigna diferentes direcciones IP a los usuarios virtuales.

Organizar los casos de prueba definidos en una tabla así (también esta adjunta al Anexo K)

Tabla 5. Casos de prueba

Número del caso de prueba	Perfil	Número de usuarios virtuales	Duración de la prueba	Tiempo entre clics
Número consecutivo para cada una de los casos de prueba	Perfil de usuario con el que se realizara la prueba	Cantidad de usuarios virtuales con los que se cargara la aplicación	Tiempo total que durara la prueba, dada en minutos	Tiempo entre clics que dura un usuario virtual

Después de realizar las configuraciones adecuadas es necesario que almacene una secuencia de recorrido por la aplicación. Esta secuencia permitirá a la herramienta simular el comportamiento de los usuarios virtuales creados.

4. Realizar pruebas para cada uno de los diferentes casos de prueba creados en el numeral anterior.

5. De todos los resultados obtenidos en las pruebas por cada caso de prueba, tener en cuenta los siguientes:

- Paginas ejecutadas.
- Paginas con error.
- Clics ejecutados.
- Clics con error.
- Kilo Bytes enviados.
- Kilo Bytes recibidos.
- Tiempo de respuesta por página.
- Porcentaje de errores HTTP.
- Porcentaje de Timeouts.
- Velocidad del usuario recibiendo.
- Velocidad del usuario mandando.
- Porcentaje total de errores.

Registrar los resultados en el formato adecuado (Ver Anexo K).

6. Presentar los resultados en una reunión donde estén presentes el Coordinador de Desarrollo de Software y el Coordinador de Aseguramiento de la Calidad, en la cual definan si los resultados obtenidos en las pruebas de carga cumplen con las expectativas de rendimiento de la aplicación. En caso contrario, debe programarse los correctivos necesarios para luego, volver a realizar las pruebas de carga. El resultado de la reunión debe ser escrito en un documento (Ver anexo E) para que quede constancia de las decisiones tomadas.

El proceso se repite hasta que los miembros de la reunión definan que el aplicativo cumpla con las expectativas de rendimiento.

3.6 EJECUCIÓN PRUEBAS DE DESEMPEÑO: ESTRÉS

Para realizar las pruebas de estrés se recomienda utilizar la misma herramienta utilizada para realizar las pruebas de carga.

1. Definir los valores de las variables iníciales, en la tabla dispuesta la plantilla de stress (Ver Anexo L) necesarias para realizar las pruebas de stress, teniendo en cuenta lo siguiente:

- Para este tipo de prueba es necesario que se configure el programa de forma que el número de usuarios virtuales crezca de manera incremental.
- La duración total de la prueba debe ser igual o mayor que la duración fijada para las pruebas de carga.
- El tiempo entre clics de los usuarios virtuales debe ser menor al tiempo fijado en las pruebas de carga.
- Utilizar los mismos perfiles de las pruebas de carga para los usuarios virtuales.

De nuevo, debe tener en cuenta que si la herramienta seleccionada lo permite debe activar el “ip-spoofing”, que asigna diferentes direcciones IP a los usuarios virtuales.
Una vez fijados los parámetros adecuados para ejecutar la prueba, utilizar la secuencia de navegación, creada para las pruebas de carga, para ejecutar la prueba de stress.

2. Realizar prueba de estrés para el caso de prueba planteado en el numeral anterior.

3. De los resultados obtenidos en la prueba, tener en cuenta los siguientes ítems por cada perfil de usuario definido:

- Periodo de tiempo de ejecución
- Número de usuarios virtuales en ese período de tiempo
- Paginas ejecutadas
- Paginas con error
- Clics ejecutados
- Clics con error
- Kilo Bytes enviados
- Kilo Bytes recibidos
- Porcentaje de errores HTTP
- Porcentaje de Timeouts
- Velocidad del usuario recibiendo
- Velocidad del usuario enviando
- Porcentaje total de errores

Registrar los resultados en la platilla designada para este numeral (Ver Anexo L).

4. Presentar los resultados en una reunión donde estén presentes el Coordinador de Desarrollo de Software y el Coordinador de Aseguramiento de la Calidad, en la cual definan si los resultados obtenidos en las pruebas de estrés cumplen con las expectativas de rendimiento de la aplicación. En caso contrario, debe programarse los correctivos necesarios para luego, volver a realizar las pruebas de carga. El resultado de la reunión debe ser escrito en un documento (Ver anexo F) para que quede constancia de las decisiones tomadas.

El proceso se repite hasta que los miembros de la reunión definan que el aplicativo cumpla con las expectativas de rendimiento.

3.7 PRESENTACIÓN DE RESULTADOS

Después de realizar todo el proceso de prueba propuesto por la metodología construida es necesario presentar un resumen de los errores encontrados. El resumen debe incluir: todos los errores encontrados en todas las pruebas, identificados por un código que los relacionen con el tipo de prueba donde fue descubierto; su estado, es decir, si fue solucionado o está pendiente; la prioridad de solución, en caso que esté pendiente de solucionar; y observaciones sobre el error (Ver anexo H).

Por último el Coordinador de Aseguramiento de Calidad aprueba el software sometido a pruebas se encuentra listo para salir a producción y lo comunica por medio de una carta al Coordinador de Desarrollo de Software (Ver anexo I).

3.8 CONCLUSIONES DE LA METODOLOGIA DE PRUEBA

En esta sección el Coordinador de Aseguramiento de Calidad describe las conclusiones del proyecto prueba, incluyendo consejos (si existen) para mejorar la metodología y describiendo los problemas que se presentaron, para que sirvan de guía para los proyectos futuros.

4. CONCLUSIONES

El entorno dinámico de las organizaciones consumidoras de software, el aumento de oferta de productos de calidad y el incremento de proveedores de servicios informáticos, han llevado a los usuarios de los productos de software a convertirse en agentes activos del proceso de desarrollo de software sacando máximo provecho de la aplicaciones, realizando críticas sobre el desempeño, y sobre todo forzando a los productores a desarrollar aplicativos más eficientes, con mayor funcionalidad y sobre todo, productos de mejor calidad.

El Centro de Informática de la Universidad EAFIT se ha convertido en un modelo de desarrollo nacional en el campo de creación de aplicaciones para la gestión de instituciones educativas, incluyendo otras universidades y colegios. Por ello fue necesario proponer un esquema de unidad de Aseguramiento de Calidad que habilite la realización de pruebas a las aplicaciones desarrolladas dentro del Centro, enmarcando las pruebas dentro de una metodología ajustada a las necesidades particulares de facilidad y eficacia que requiere el Centro de Informática. Todo este trabajo se realiza con el fin de garantizar que las futuras aplicaciones desarrolladas y probadas bajo la metodología propuesta alcancen un nivel de calidad competitivo respecto a los estándares mundiales vigentes sobre el tema, cumpliendo y sobrepasando las expectativas de los usuarios y clientes de dichas aplicaciones.

La metodología propuesta es un paso importante hacia el mejoramiento de los productos desarrollados por el Centro de Informática, sin embargo la metodología no soporta todo el proceso de desarrollo de software. Para ello es necesario que se implemente un complemento a la metodología desarrollada, que permita abarcar el proceso completo de desarrollo, incluyendo pruebas de unidad, integridad y aceptación. Además, sería ideal que la metodología se perfeccione

enmarcándola sobre estándares internacionales de mejores prácticas como ITIL y CMMI, los cuales permitirían generar un valor agregado tanto al proceso de prueba como al resultado final.

5. TRABAJOS FUTUROS

- Incluir en la metodología propuesta estándares internacionales como CMMI, ITIL y otros que contribuyan a la mejora continua de la misma.
- Definir la utilización de una herramienta solida y confiable que permita la realización de las pruebas de carga y estrés, para su posterior uso dentro de la metodología propuesta.
- Investigar y proponer una herramienta que permita la automatización el proceso de uso y manejo de las plantillas de pruebas. Esto con el fin de agilizar el proceso de ejecución de pruebas.
- Realizar un gestor de conocimiento que soporte toda la gestión documental de las pruebas y sirva como una base de conocimiento para los futuros proyecto de pruebas.

BIBLIOGRAFÍA

Calidad del software. Disponible en:
http://www.info- ab.uclm.es/asignaturas/42579/slides/tema1.pdf

CRAIG, Rick D. y STEFAN P., Jaskiel, “Systematic Software Testing”, 5a edición, Artech House Publishers. 2006

Decision Tables, Disponible en: http://en.wikipedia.org/wiki/Decision_table

DEMILLO, Richard A.; MCCRACKEN, Michael W.; MARTIN, R.J. y PASSAFIUME, John F. “Software Testing and Evaluation”. The Benjamin/Cummings Publishing Company.

Estrategias de Prueba de Software.
Disponible en: http://www.softwareprojects.org/software-test-strategy.html

GILB, Tom. “What we fail to do in our current testing culture”. Disponible en:
http://saturn.uni-mb.si/~razno/vv_iii/what%20we%20fail%20to%20do%20in%20our%20current%20testing%20culture%20by%20tom%20gilb.txt. 1995

LEWIS, William E. “PDCA/TEST, A Quality Framework for Software Testing”. Auerbach, 1998

LEWIS, William E. “Software Testing and Continous Quality Improvement”. Auerbach,. 2000

MICROSOFT. Visual Studio Team System. “Trabajar con pruebas de carga”. Disponible en: http://msdn.microsoft.com/es-es/library/ms182561(VS.80).aspx

MITZBERG, Henry. “Ten Managments Roles”. Disponible en: http://www.sayeconomy.com/ten-management-roles-by-henry-mintzberg/

Open Source Perfomance Testing Tools. Disponible en: http://www.opensourcetesting.org/performance.php

OSKARSSON, Östen y GLASS, Robert L. “An iso 9000 approach to building quality software”, Prentice Hall. 1996

Performance Testing, Load Testing, and Stress Testing, Disponible en: http://www.devbistro.com/articles/Testing/Performance-Testing-Load-Testing

Performance vs load vs stress testing, Disponible en: http://agiletesting.blogspot.com/2005/02/performance-vs-load-vs-stress-testing.html

PERRY, William. “Effective Methods for Software Testing”. Wiley-Qed. 1995

PRESSMAN, Roger S. “Ingeniería del software: Un enfoque practico”, 6a edición, Mc Graw Hill. 2006

Software Testing, Disponible en: http://en.wikipedia.org/wiki/Software_testing

SOMERVILLE, Ian. “Ingeniería de Software”. 6a edición, Addison Wesley.

Stress testing, Disponible en: http://en.wikipedia.org/wiki/Stress_testing (software) 2002

Testware, Disponible en: http://en.wikipedia.org/wiki/Testware

WALLMÜLLER, Ernest. “Software Quality Assurance, a practical approach”, Prentice Hall. 1994

Web Site Test Tools and Site Management Tools. Disponible en: http://www.softwareqatest.com/qatweb1.html

ANEXO A. INFORME DE DESARROLLO CONCLUIDO

Señor

Nombre completo

Coordinador de Aseguramiento de Calidad

ASUNTO: Informe de concluido el desarrollo de la aplicación *Nombre de la aplicación*.

Por medio de la presente se informa que se ha concluido el proceso de desarrollo de la aplicación "*Nombre de la aplicación"*. A partir de este momento pueden implementar el ambiente de prueba, adjunto la información necesaria.

A continuación se listan los posibles elementos que contiene la documentación necesaria para implementar el ambiente de pruebas:

- Requisitos funcionales y no funcionales
- Manual de usuario (Sí aplica)
- Casos de uso
- Especificaciones técnicas:
    i. Hardware y software necesarios
    ii. Servidor de aplicaciones
    iii. Motor de bases de datos
    iv. Ubicación de los datos

Anotaciones:

Cualquier anotación que considere pertinente

_____

Coordinador de Desarrollo de Software

ANEXO B. SOLICITUD DE AMBIENTE DE PRUEBAS

Señor
Nombre completo
Coordinador de Soporte Técnico

ASUNTO: Solicitud de implantación del ambiente de pruebas para la aplicación "*Nombre de la aplicación"* y población de las bases de datos.

Por medio de la presente se solicita al Área de Soporte Técnico gestionar la implementación del ambiente de pruebas para la aplicación *"Nombre de la aplicación"*.

Se adjunta la documentación y especificaciones técnicas necesarias para el correcto funcionamiento de la misma.

El ambiente de pruebas debe incluir:

- Asignación de un servidor con las especificaciones técnicas requeridas.
- Creación de los perfiles de usuario, necesarios para tener acceso a todas las instancias del aplicativo.
- Instalación del motor de la base de datos que alojará los datos para realizar las pruebas.

Anotaciones:

Informar, cuando esté listo el ambiente de prueba al Coordinador de Aseguramiento de Calidad.

_____

Coordinador de Aseguramiento de Calidad

ANEXO C. INFORME DE AMBIENTE DE PRUEBAS

Señor
Nombre completo
Coordinador de Aseguramiento de Calidad

ASUNTO: Informe de implementación del ambiente de pruebas para la aplicación "*Nombre de la aplicación"*.
Por medio de la presente se informa al Área de Aseguramiento de Calidad que el ambiente de pruebas para la aplicación "*Nombre de la aplicación"* se encuentra listo y pueden empezar a realizar las pruebas convenientes.

Anotaciones:
En caso que el Área de Soporte Técnico no pueda poblar alguna tabla o base de datos, debe informar al Equipo de Aseguramiento de Calidad para que aquellas que no pudieron ser pobladas, sean pobladas por este último equipo.

_____
Coordinador de Soporte Técnico

ANEXO D. PLANTILLA DE INTERFAZ

Nombre de la interfaz: *Nombre de la interfaz probada*

Descripción: *Describir el usuario al que pertenece la interfaz y la función de la pantalla probada.*

Número	**Rol/Usuario**	**Campo**	**Tipo**	**Prueba realizada**	**Datos de prueba**	**A**	**R**	**Comentarios**
Consecutivo del caso de prueba	*Rol y usuario con el cual se ejecuto la prueba*	*Nombre del campo probado*	*De dato aceptado por el campo*	*Nombre del tipo de prueba*	*Datos ingresados para pruebas.*	*Marcar con "X" si lo acepta*	*Marcar con "X" si lo rechaza*	

_____ _____

Analista de prueba Analista de desarrollo

Se adjunta impresión de la pantalla probada

.

ANEXO E. INFORME DE CORRECIÓN

Señor
Nombre completo
Analista de prueba

ASUNTO: Informe de corrección de errores de la plantilla de pruebas de código *“código de la plantilla depurada”.*

Por medio de la presente me permito informarle que se ha realizado la depuración de los errores presentados en la plantilla de código *“Código de la plantilla”*.

Anotaciones:
Describir en anotaciones los errores que no pudieron ser solucionados, incluyendo la causa para no depurarlos y la acción a tomar por cada uno ellos.
En caso que considere que alguna pantalla pude verse afectada por el proceso de depurado debe nombrarla en este espacio.

_____
Analista de desarrollo

ANEXO F. REUNIONES

Proyecto:

Fecha:

Hora:

Lugar:

Asistentes

Nombre	Rol / Cargo	Dependencia

Invitados

Nombre	Rol / Cargo	Dependencia

Ausentes

Nombre	Rol / Cargo	Dependencia

Asunto

Desarrollo

Compromisos

Descripción	Responsable	Fecha Límite

ANEXO G. PLANTILLA DE INTEGRIDAD

Nombre de la interfaz: *Nombre de la interfaz probada*

Descripción: *Describir el usuario al que pertenece la interfaz y la función de la pantalla probada.*

Número	**Rol/Usuario**	**Campo**	**Tipo**	**B.D. y Tabla**	**Prueba realizada**	**Datos de prueba**	**A**	**R**	**Comentarios**
Consecutivo del caso de prueba	*Rol y usuario con el cual se ejecuto la prueba*	*Nombre del campo probado*	De dato aceptado por el campo	Nombre de la B.D. y tabla en la que escribe	Nombre del tipo de prueba	Datos ingresados para pruebas.	Marcar con “X” si lo acepta	Marcar con “X” si lo rechaza	Si se presenta un error de escritura debe explicarlo acá.

_____ _____

Analista de prueba Analista de desarrollo

Se adjunta impresión de la pantalla probada.

ANEXO H. PLANTILLA FINAL

Código	**Tipo Error**	**Nombre Error**	**Estado**	**Prioridad de Solución**	**Observaciones**
Código del error compuesto por Código de la Plantica y el consecutivo del caso de prueba, por ejemplo PInterfaz0001-1	*Tipo de error*	*Nombre del error*	*Estado en el que está error, corregido o pendiente*	*Número asignado por el coordinador de Aseguramiento de calidad, siempre entre 1 y 5, siendo 1 el más bajo y 5 el más alto*	*Observaciones pertinentes sobre el error.*

_____ _____

Analista de prueba Analista de desarrollo

ANEXO I. INFORME DE CONCLUSIÓN DE PRUEBAS

Señor
Nombre completo
Coordinador de Desarrollo de Software

ASUNTO: Informe de concluido de las pruebas a la aplicación "*Nombre de la aplicación*"

Por medio de la presente se informa que se ha concluido el proceso prueba de la aplicación "*Nombre de la aplicación"*. A partir de este momento pueden sacar la aplicación a producción.

_____
Coordinador de Desarrollo de Software

ANEXO J. ENCUESTA PRUEBAS DE USABILIDAD

Por favor califique cada uno de las siguientes características, utilizando la escala de calificación que sigue:

Calificación 5: Si está totalmente de satisfecho
Calificación 4: Si está satisfecho
Calificación 3: Si está medianamente satisfecho
Calificación 2: Si está insatisfecho
Calificación 1: Si está totalmente insatisfecho

Id.	Pregunta	Respuesta						
1.	La facilidad de uso de:		**5**	**4**	**3**	**2**	**1**	**N/A**
		1.1. Menús						
		1.2. Botones						
2.	La disposición de:		**5**	**4**	**3**	**2**	**1**	**N/A**
		1.1. Menús						
		1.2. Contenido						
		1.3. Barras de desplazamiento						
		1.4. Funciones						
		1.5. Gráficos						
3.	La facilidad de lectura de:		**5**	**4**	**3**	**2**	**1**	**N/A**
		1.1. Contenido escrito						
		1.2. Contenido gráfico						
4.	El estética de:		**5**	**4**	**3**	**2**	**1**	**N/A**
		1.1. Colores						
		1.2. Figuras						
		1.3. Tablas						
		1.4. Botones						
		1.5. Letra						

Id.	Pregunta	Respuesta					
5.		5	4	3	2	1	N/A
	La distribución de la aplicación en la pantalla						
6.		5	4	3	2	1	N/A
	El tiempo de respuesta de la aplicación						

Id.	Pregunta	Respuesta
7.	Considera usted que la aplicación debe permitir personalizarse	S/N

Id.	Pregunta	Respuesta
8.	¿Qué piensa usted que falta en la aplicación?	
9.	Comentarios y sugerencias	
10.	Nombres y apellidos	
11.	Email	

ANEXO K. PLANTILLA DE CARGA

Nombre de la aplicación: *Nombre de la aplicación probada*

Nombre de la herramienta: *Nombre de la herramienta usada para realizar las pruebas de carga.*

Número del caso de uso	Perfil	Número de usuarios virtuales	Duración de la prueba	Tiempo entre clics
Número consecutivo para cada una de los casos de prueba	Perfil de usuario con el que se realizara la prueba	Cantidad de usuarios virtuales con los que se cargara la aplicación	Tiempo total que durara la prueba, dada en minutos	Tiempo entre clics que dura un usuario virtual

Número de caso prueba	**Paginas ejecutadas / Paginas con error**	**Clics ejecutados / Clics con error**	**Tiempo de respuesta**	**Kbytes enviados / Kbytes recibidos**	**Porcentaje de errores HTTP**	**Porcentaje de Timeouts**	**Velocidad del usuario recibiendo / Velocidad del usuario mandando**	**Porcentaje total de errores**
Consecutivo del caso de prueba	*Número de páginas ejecutadas y número de páginas que presentaron error*	*Número de clics ejecutados y número de clics que presentaron error*	*La pagina donde se presento el error*	*Numero de Kbytes enviados y Numero de Kbytes recibidos por perfil*	*Porcentaje de errores HTTP presentados en la prueba*	*Porcentaje de Timeouts presentados en la prueba*	*Velocidad del usuario tanto recibiendo como mandando*	*Porcentaje total de errores presentados en las pruebas.*

_____ _____

Analista de prueba Analista de desarrollo

ANEXO L. PLANTILLA DE STRESS

Nombre de la aplicación: *Nombre de la aplicación probada*

Nombre de la herramienta: *Nombre de la herramienta usada para realizar las pruebas de carga.*

Rango de usuarios virtuales	**Frecuencia de usuario virtuales**	**Duración de la prueba**	**Tiempo entre clics**
Rango de usuarios virtuales con los que se cargara la aplicación periódicamente	*Frecuencia con la que irán ingresando los usuarios virtuales a la aplicación. Esta frecuencia está dada en Usuarios/Segundos*	*Tiempo total que durara la prueba, dada en minutos*	*Tiempo entre clics que dura un usuario virtual*

Perfil de usuario	**Periodo de tiempo**	**Número de usuarios virtuales**	**Paginas ejecutadas / Paginas con error**	**Clics ejecutados / Clics con error**	**Kbytes enviados / Kbytes recibidos**	**Porcentaje de errores HTTP**	**Porcentaje de Timeouts**	**Velocidad del usuario recibiendo / Velocidad del usuario envidando**	**Porcentaje total de errores**
Perfil de usuario en donde se presento el error	*Periodo de tiempo en el que se presentaron errores*	*Número de usuarios virtuales que habían en el periodo de tiempo*	*Número de páginas ejecutadas y número de páginas que presentaron error en el periodo de tiempo*	*Número de clics ejecutados y número de clics que presentaron errores en el periodo de tiempo*	*Número de Kbytes enviados y Numero de Kbytes recibidos en el periodo de tiempo*	*Porcentaje de errores HTTP presentados en el periodo de tiempo*	*Porcentaje de Timeouts presentados en el periodo de tiempo*	*Velocidad del usuario tanto recibiendo como mandando en el periodo de tiempo*	*Porcentaje total de errores presentados en las pruebas en el periodo de tiempo*

_____ _____

Analista de prueba Analista de desarrollo

ANEXO M. RESULTADOS PRUEBAS ZEUS

A continuación se adjuntan los resultados de las pruebas realizadas a la aplicación de asignación docente, **Zeus**. Los resultados presentados hacen parte del primer ciclo de pruebas y junto con este proyecto se presentarán por primera vez al Centro de Informática de la Universidad EAFIT, el cual, debe continuar con la metodología presentada para probar la eficacia de la misma.

Nota: No se realizó todo el ciclo de la metodología porque el tiempo para completarlo no se ajusta al cronograma disponible para la realización de este trabajo de grado.