PROYECTO DE GRADO
"METODOLOGÍA PARA TESTING DE SOFTWARE BASADO EN COMPONENTES"

JUAN CAMILO FRANCO OCHOA

INGENIERÍA DE SISTEMAS

UNIVERSIDAD EAFIT
MEDELLÍN
MAYO / 2010

TABLA DE CONTENIDO

1. INTRODUCCIÓN.

Como bien se sabe, el desarrollo de software es una actividad que está sujeta a los errores humanos, por tanto es probable encontrarse en cualquier desarrollo con defectos, errores y fallas. Para tratar de evitar esas situaciones al máximo existen ya definidas metodologías de desarrollo que si bien no garantizan la eliminación total de errores si disminuyen altamente la probabilidad de falla. Pero aún así y por más que se quieran evitar este tipo de sucesos, se dificulta ya que son inesperados, es allí donde cobra importancia un adecuado proceso de pruebas de software que permita certificar desde el punto de vista de la calidad un producto de software, si bien es una tarea que necesita de diversos recursos como el esfuerzo, planeación, organización, documentación y tiempo no se debe tomar como un impedimento, sino como una inversión que se verá retribuida en la satisfacción del usuario final o cliente.
Con el propósito entonces de realizar estas pruebas de software y gestionar los errores o fallas que se pueden presentar durante el desarrollo, a través de una manera eficiente han surgido diferentes metodologías de Testing aplicadas a los diversos métodos de desarrollo, en especial a los más populares como son el método de desarrollo RUP, cascada y espiral.

Sin embargo y a pesar de que los anteriores métodos de desarrollo que han sido mencionados siguen siendo populares, hay otros métodos que han probado también ser muy eficientes, es el caso del método del desarrollo de software basado en componentes que cada vez es más usado como modelo para el desarrollo. Este modelo tiene como gran ventaja la reutilización de código y es allí precisamente donde radica su fortaleza haciendo de este método una gran alternativa de desarrollo de software que cada día tiene más acogida entre el gremio de desarrolladores ya que facilita la labor de estos, disminuyendo los tiempos y aumentando considerablemente la eficiencia durante la etapa de construcción de un software.

Si bien es un excelente modelo de desarrollo, al igual que los demás tampoco garantiza que el desarrollo final esté exento de errores, hace falta entonces una apropiada metodología de testing que se adapte a este modelo y que permita fortalecer al desarrollo de software basado en componentes cada vez más como una alternativa que además de contar con sus múltiples ventajas para el desarrollo, cuente con su propia metodología de testing, proporcionándole un valor agregado que se considera de gran importancia para un modelo de desarrollo ya que sin una adecuada

metodología adaptada al modelo que gestione los errores y las fallas, podrían presentarse muy probablemente una serie de eventos que suceden a menudo en muchas compañías cuya razón social es el desarrollo de software.

Independientemente de cuál sea la metodología para la construcción del software que estas compañías utilizan, como lo decía anteriormente, hay eventos que repetidamente se presentan a la hora de encontrar un defecto (issue) cuando no se tiene una metodología de testing adecuada, el desarrollo basado en componentes no es indiferente a estos eventos, los cuales citaré a continuación:

Sucede muy a menudo debido a las mismas características que tiene este modelo de desarrollo que el mismo desarrollador no sabe cómo replicar el issue detectado, ya que por la naturaleza del mismo modelo que implica la reutilización de código, el desarrollador no puede reproducir el issue y por tanto no lo puede solucionar, o en caso de poder reproducirlo, a menudo el desarrollador piensa que ha solucionado el issue, realiza modificaciones sobre algún componente que funcionaba pero igual el issue persiste a causa de que realmente no sabe cómo debía funcionar. O en caso de poder solucionarlo nadie se entera, al cliente nunca le llega el aplicativo solucionado debido a una comunicación deficiente con el mismo.

Otra situación recurrente, que no solo sucede en el desarrollo basado en componentes, es cuando alguien encuentra un issue y habla con distintos desarrolladores para que se solucione, ellos a su vez comienzan a solucionarlo pero entorpeciéndose la labor unos con otros y mal gastando tiempo, o simplemente el issue llega al equipo de desarrollo. El desarrollador 1 piensa que lo tiene que solucionar el desarrollador 2 y el 2 piensa que es tarea de 1. Nadie lo soluciona al final y todo debido a que no hay una asignación adecuada de los issues.

Cuando no se tiene una metodología de testing establecida, es común notar el desorden que implica no tenerla, y notar por ejemplo que no se lleva un registro de issues, el desarrollador simplemente toma nota por ejemplo en un trozo de papel cuando se le notifica del issue encontrado pero luego como sucede frecuentemente, se le olvida y lo pierde. O en muchas ocasiones muchos issues son reportados al desarrollador personalmente de manera que este no puede solucionar ninguno porque no tiene tiempo.

En otras ocasiones, las compañías si intentan tener una metodología de testing adaptada para el desarrollo de sus aplicaciones pero no hacen el trabajo completo, simplemente tienen un sistema donde se reportan los issues, pero no se hace la correcta gestión de los mismos. Es aquí donde se empiezan a presentar situaciones en donde el desarrollador no posee la suficiente información para tomar las decisiones más acertadas. Por ejemplo, siempre existen issues, unos más importantes que otros, y supongamos como casi siempre sucede que el tiempo está limitado, los desarrolladores pierden el tiempo solucionando issues triviales mientras que los issues críticos siguen sin ser solucionados, esto a causa de que los issues no están priorizados. O la persona que está pendiente que se solucionen los problemas interrumpe constantemente al desarrollador para consultar el estado de los issues, esto podría ser fácilmente controlado de tenerse implementada una metodología que gestione el listado de los issues con sus respectivos estados de resolución.

Debido a lo citado anteriormente, se ha hecho necesario desarrollar una metodología que permita asegurar en gran medida la calidad de los productos de software y obtener un mejoramiento continuo de todos los procesos relacionados con el desarrollo de software basado en componentes. Para ello, tomaré como base una técnica de prueba existente llamada “Prueba de Caja Negra”, la prueba de caja negra verifica que el ítem que se está probando, cuando se dan las entradas apropiadas, produce los resultados esperados, es decir una prueba de tipo funcional (y sin desconocer la existencia de las demás pruebas como las no funcionales, las de stress, las de seguridad, entre otras) son las que más tiempo tardan en realizarse pues acompañan casi todo el ciclo de vida del producto.

2. OBJETIVOS.

2.1 Objetivo General.

El objetivo general del proyecto es proponer una metodología para la realización de pruebas de software que haya sido desarrollado basado en componentes y que permita realizar una adecuada gestión de los issues que se presentan durante el desarrollo del producto; contribuyendo de esta manera a la disminución de los costos en producción y a la optimización del proceso de este tipo de desarrollo.

2.2 Objetivos Específicos.

- Aplicar los conocimientos adquiridos durante la carrera de Ingeniería de Sistemas, y al conocimiento que a diario se adquiere en la compañía donde actualmente laboro, ya que tanto el asesor como el autor de este proyecto laboramos en compañías dedicadas al Testing de Software, y aplicar este conocimiento en el desarrollo de una metodología para realizar pruebas de software que permita apoyar el área de pruebas de las empresas desarrolladoras de una manera gestionada; y de esta manera minimizar los graves impactos que generan la aparición de errores en sus procesos.

- Definir una secuencia de actividades para la realización de las pruebas funcionales sobre aplicaciones que han sido desarrolladas con el método de desarrollo basado en componentes enfocándonos principalmente en las pruebas sobre la reutilización de componentes, la cual supone la mayor ventaja de usar este método de desarrollo y la integración de componentes, principal punto crítico en el desarrollo con este método.

- Definir una serie de artefactos que permitan un mayor control y una mayor eficacia a la hora de realizar las pruebas funcionales sobre estas aplicaciones que han sido desarrolladas con el método de desarrollo basado en componentes.

- Especificar los roles definidos por el método de desarrollo para software basado en componentes y las funciones que tienen dentro del procedimiento propuesto.

- Analizar e identificar las principales causas generadoras de issues en el proceso de desarrollo de software basado en componentes de las empresas.

3. JUSTIFICACIÓN DEL PROYECTO.

La primera justificación para llevar a cabo este proyecto surge a partir del conocimiento previo que se tiene de la cantidad de procedimientos para realizar pruebas que existen actualmente, sin embargo también se sabe que muchos de estos procedimientos son privados y muy concretos sobre la empresa y no sobre el producto, otros son demasiado complejos para que una empresa pequeña los pueda utilizar, otros que no se encuentran bien gestionados, pues utilizan herramientas que no son las más adecuadas como archivos planos y los desarrolladores no pueden hacer seguimiento a sus productos y por último existen otros que se enfocan a diferentes métodos de desarrollo de software pero ninguno al particular que nos concierne, el desarrollo de software basado en componentes. Debido a esto y queriendo aprovechar el cargo como analista de pruebas que ocupo actualmente en una reconocida compañía de testing de software y ya que siempre he tenido interés en el tema surge la idea de realizar este proyecto además con el valor agregado de que hago parte y laboro diariamente con este tema y he conocido las debilidades, oportunidades y fortalezas del testing que existen en nuestro entorno.

La segunda justificación para ya entrar más en la parte técnica es que para asegurar el correcto funcionamiento de un sistema de información y prevenir los posibles fallos que pueda generar el sistema, una herramienta que juega un papel fundamental para conseguir ello son las pruebas de software. Está ampliamente demostrado que una temprana inclusión de las actividades de prueba en el proceso de desarrollo de software, en este caso del desarrollo de software basado en componentes, detecta, previene y permite solucionar los errores de una forma rápida y eficaz con todas las buenas repercusiones que ello conlleva.

Y a pesar de que los beneficios de este proceso de pruebas de software son claros, las compañías desarrolladoras siguen viendo esta práctica como un proceso aparte que no tiene mucha importancia y que sólo produce costos operativos, en gran medida gracias a que no han adoptado un proceso claro y adecuado acorde con sus necesidades, es por ello que con el presente proyecto se pretende proponer una herramienta más que pueda servir a este tipo de compañías y que cada vez más, el uso de buenas prácticas de pruebas de software se implemente en estas empresas.

Habiendo expuesto lo anterior es que el presente proyecto cobra validez, porque propone un procedimiento nuevo para este tipo de desarrollo, que podría ser adoptado fácilmente por una compañía para que de esta manera se haga una buena gestión, seguimiento y control de los errores en un desarrollo de software basado en componentes.

4. ALCANCE Y PRODUCTOS.

El Proyecto tiene como alcance el desarrollo de una nueva metodología para realizar pruebas de software desarrollado a base de componentes sobre aplicativos de software que se hayan desarrollado o que estén en proceso de desarrollo centrándome en la funcionalidad del producto y no en sus otras características ya que las pruebas de funcionalidad o funcionales de software se realizan durante casi todo el ciclo de desarrollo del producto. Además la metodología que se propondrá tendrá especialmente enfoque en las pruebas a realizar sobre la reutilización e integración de componentes la cual es la mayor ventaja y el mayor riesgo respectivamente que se tienen al utilizar éste método de desarrollo marcando así una diferencia con las metodologías de testing tradicionales. Con la aplicación de esta nueva metodología les será posible a las empresas desarrolladoras apoyar las áreas de prueba en forma eficaz y efectiva de manera que podrán notar de manera satisfactoria el mejoramiento y la optimización de sus procesos cuando este método de desarrollo haya sido escogido como el más óptimo para el desarrollo de un aplicativo.
Cabe aclarar que en mi proyecto no se contempla la elaboración o elicitación de los requisitos del software[1], ya que esta actividad concierne a personas especializadas en el tema. Como se mencionó anteriormente, el proyecto tiene funcionalidad y validez a partir de los requisitos ya elicitados.

El producto que se entregará será:

- Documento de la nueva metodología para realizar pruebas de software desarrollado basado en componentes.

[1] La elicitación de requisitos abarca la primera y quizás más importante fase dentro del desarrollo de un software. Su objetivo principal es garantizar que los requisitos del software sean consistentes con las necesidades de la organización donde se utilizará el mismo y con las futuras necesidades de los usuarios.

5. ANTECEDENTES.

Anteriormente, cuando el desarrollo de software apenas comenzaba, se desarrollaban aplicaciones más que todo de manera empírica por encima de una manera metódica. Cuando se creaba un código se realizaban durante el mismo proceso las pruebas de funcionalidad, pero esta labor la realizaba el mismo desarrollador al final y era vista esta actividad como parte de la misma actividad de desarrollo, es decir el término testing aún no tenía connotación alguna.
Apenas hacia finales de los 50's, los desarrolladores empezaron a ver este proceso de testing o de pruebas como algo un poco aparte del mismo desarrollo, es decir ya por lo menos se tenía una metodología establecida, primero se desarrollaba y luego se probaba Sin embargo en la década siguiente, es decir en los años 1960's empezó a cobrar cada vez más importancia la labor de realización de pruebas, esto debido a que se empezaba a entender la importancia y el impacto positivo tanto en costo como en tiempo que tenía el hecho de llevar a cabo un proceso serio de detección de deficiencias en el programa desarrollado

Ya para los años 70's fueron introducidos métodos de pruebas más rigurosos y se empezó a contar con mayores recursos, el término "Ingeniería de Software" fue acuñado. Surgieron conferencias sobre "Pruebas de Software" y cada vez el tema se hacía más indispensable para quienes se dedicaban al desarrollo de aplicaciones como una manera de garantizar que el software creado se desempeñaba realmente de la forma en que se pretendía.
Para los años 80's hubo un vuelco total en la manera de entender el tema, surgió un término que cambiaría la manera de hacer testing de software y de concentrar esfuerzos alrededor de la palabra "Calidad". La calidad se convirtió en el gran objetivo a conseguir, como resultado se empezaron a utilizar mejores prácticas que conllevarían a la creación de estándares de calidad como son los estándares IEEE, ANSI e ISO y a partir de allí el proceso de pruebas de software enfocado a alcanzar "calidad" en el producto final llevó a que cada vez se hicieran mejoras y establecer metodologías de testing que no solo probara el aplicativo en su etapa final sino que por el contrario apoyara el desarrollo del mismo durante todas las etapas del ciclo de vida de desarrollo. Estas metodologías de testing cada vez se fueron adaptando de una mejor manera a los múltiples métodos de desarrollo, entre los cuales encontramos

los más populares RUP, cascada, espiral, etc, perfeccionando así la armonía que debe existir entre el método de desarrollo y su metodología de testing.
En la actualidad han surgido métodos de desarrollo innovadores como ya lo hemos mencionado, es el caso del desarrollo de software basado en componentes que tiene como su gran fortaleza la reutilización de código, es por eso que al igual que lo hicieron los otros métodos de desarrollo en su momento, el desarrollo de software basado en componentes debe ir evolucionando y adoptando una metodología de testing propia que satisfaga sus necesidades en cuanto a calidad se refiere, una metodología de testing que además ataque las debilidades del desarrollo basado en componentes que en este caso bien podría ser la propia especialidad del método, es decir la reutilización de código (componentes). La reutilización tiene muchas ventajas entre las que se destaca la reducción de tiempo en desarrollo y en pruebas si se dan las condiciones, pero a su vez puede implicar la inserción de muchos issues que acarrea la utilización de código ajeno, esto sin contar los problemas funcionales que puede traer consigo la integración de componentes, momento crítico en el desarrollo, y la metodología de testing necesaria para éste método debe saber entender esto y atacar estos problemas de la manera más adecuada.

Hemos visto las metodologías de testing han jugado un papel muy importante en la evolución de los métodos de desarrollo, pero a pesar de ello aún existen opiniones encontradas sobre el Proceso de pruebas, como por ejemplo:

En contra:
”El testing del software puede ser usado para mostrar la presencia de issues, pero nunca su ausencia.” Dijkstra.
"Nunca podemos estar seguros que un sistema de pruebas es correcto" Manna.

A favor:
"Testing es el proceso de evaluación de un sistema o componente para verificar que satisface los requisitos especificados, o identificar diferencias entre lo esperado y los resultados obtenidos". IEEE

Testing es el proceso de ejecutar un programa o sistema con la intención de encontrar issues“. (Myers 1979)

6. CONTEXTO.

Los sistemas de hoy en día son cada vez más complejos, deben ser construidos en tiempo récord y deben cumplir con los estándares más altos de calidad. Para hacer frente a esto, se concibió y perfeccionó lo que hoy conocemos como Ingeniería de Software Basada en Componentes (ISBC), la cual se centra en el diseño y construcción de sistemas computacionales que utilizan componentes de software reutilizables. Esta ciencia trabaja bajo la filosofía de "comprar, no construir", una idea que ya es común en casi todas las industrias existentes, pero relativamente nueva en lo que a la construcción de software se refiere.

Para este momento, ya muchos conocen las ventajas de este enfoque de desarrollo y, de la misma forma, muchos se preguntan día a día el por qué son tan pocos los que realmente alcanzan el éxito siguiendo esta filosofía. La realidad es que aún se ha tanteado poco las posibilidades del software basado en componentes, y es justo hora, en la presente década, que la industria del software despegará y se perfeccionará para estar a la par de cualquier otra industria del medio, este camino hacia el perfeccionamiento de este modelo de desarrollo tan prometedor debe incluir necesariamente la concepción de su propia metodología de testing que apoye y garantice la calidad del software desarrollado con este modelo, haciendo que cada vez más sea tenido en cuenta por esta industria, la del desarrollo de software.

Este trabajo de grado apunta a ello, a proponer una metodología adaptable al desarrollo de software basado en componentes, que al igual que los otros modelos de desarrollo tienen un ciclo de vida del mismo, a través del cual se puede asegurar en un alto porcentaje la calidad del software cuando se ejecutan todas las pruebas necesarias a lo largo de todo este ciclo, a pesar de ello la metodología que se propondrá solo se centrará en la planeación, diseño, ejecución y evaluación de las pruebas funcionales integrales de caja negra en las etapas de implementación con enfoque en la ventaja y en el factor crítico del método de desarrollo que son la reutilización de componentes e integración de componentes respectivamente, esto con el fin de convertir el control de calidad, en un proceso práctico que no presente reacción negativa en las organizaciones que quieran adaptar el desarrollo de software basado en componentes como modelo de desarrollo y que deseen por el contrario que

se convierta en un factor diferenciador para el incremento de la productividad en áreas de Tecnologías de Información.

Esta propuesta metodológica estará estructurada siguiendo el ciclo PHVA (ciclo Demming – Planear, Hacer, Verificar, Actuar)[2] para identificar los actores, las actividades y los resultados de cada proceso. Esto permite focalizar fácilmente las tareas relacionadas con la planeación, la ejecución, el control y el mejoramiento de las pruebas funcionales de software.

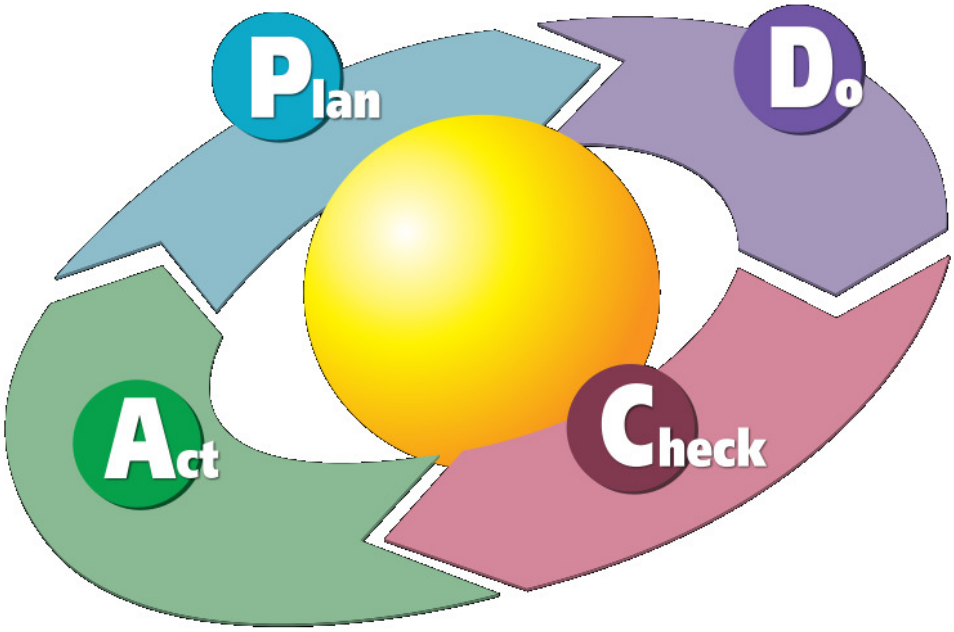

FIGURA 1: Ciclo demming – planear, hacer, verificar, actuar

[2] El ciclo PDCA, también conocido como "Círculo de Deming" (de Edwards Deming), es una estrategia de mejora continua de la calidad en cuatro pasos, basada en un concepto ideado por Walter A. Shewhart. También se denomina espiral de mejora continua

6.1 Pruebas Funcionales de Software.

Se denominan pruebas funcionales o Functional Testing, a las pruebas de software que tienen por objetivo probar que los sistemas desarrollados, cumplan con las funciones específicas para los cuales han sido creados, es común que este tipo de pruebas sean desarrolladas por analistas de pruebas con apoyo de algunos usuarios finales, esta etapa suele ser la última etapa de pruebas y al dar conformidad sobre esta el paso siguiente es el pase a producción.

A este tipo de pruebas se les denomina también pruebas de comportamiento o pruebas de caja negra, ya que los testers o analistas de pruebas, no enfocan su atención a como se generan las respuestas del sistema, básicamente el enfoque de este tipo de prueba se basa en el análisis de los datos de entrada y en los de salida, esto generalmente se define en los casos de prueba preparados antes del inicio de las pruebas.

.

Al realizar pruebas funcionales lo que se pretende en ponerse en los pies del usuario, usar el sistema como él lo usaría sin embargo el analista de pruebas debe ir más allá que cualquier usuario, generalmente se requiere apoyo de los usuarios finales ya que ellos pueden aportar mucho en el desarrollo de casos de prueba complejos, enfocados básicamente al negocio, posibles particularidades que no se hayan contemplado adecuadamente en el diseño funcional, el analista de pruebas debería dar fuerza a las pruebas funcionales y más aún a las de robustez, generalmente los usuarios realizan las pruebas con la idea que todo debería funcionar, a diferencia del analista de pruebas que tiene más bien una misión destructiva, su objetivo será encontrar alguna posible debilidad y si la llega a ubicar se esforzará por que deje de ser pequeña y posiblemente se convertirá en un gran error, cada error encontrado por el analista de pruebas es un éxito, y se debería considerar como tal.

6.1.1 Objetivos de las Pruebas Funcionales de Software.

Básicamente el objetivo general de las pruebas funcionales como mencioné anteriormente es descubrir errores que no han sido detectados hasta entonces en los sistemas desarrollados, y garantizar que cumplan con las funciones específicas para los cuales han sido creados. Sin embargo a continuación se listarán detalladamente objetivos más específicos que ilustrarán de una mejor manera todo lo que se pretende alcanzar cuando se realizan pruebas funcionales:

- Encontrar el mayor número de issues que pueda tener la aplicación antes de que salga a producción, para gestionar la corrección de estos.

- Demostrar que la aplicación cumple con las especificaciones y requerimientos definidos en la documentación.

- Demostrar la adecuada interacción con las demás aplicaciones y su relación.

- Asegurar el correcto funcionamiento de toda la aplicación, frente a un cambio de cualquiera de sus funcionalidades.

- Obtener un conjunto de condiciones de entrada que ejerciten completamente los requisitos funcionales del programa.

- Especificar que las Pruebas Funcionales de Software no son una alternativa a las Pruebas de Caja Blanca.[3]
    - Complementan a las Pruebas de Caja Blanca.

- Detectar:
    - Funcionamiento incorrecto o incompleto.
    - Issues de interface y de accesos a estructuras de datos externas.
    - Problemas de rendimiento.
    - Issues de inicio y terminación.

[3] Las pruebas de caja blanca son pruebas que realizan un seguimiento al código fuente según se van ejecutando los casos de prueba, de manera que se determinan de manera concreta las instrucciones, bloques, etc. en los que existen defectos.

6.1.2 Caso de Prueba Funcional.

Un caso de prueba funcional o también conocido como Test Case es un conjunto de condiciones o variables bajo las cuáles el analista determinará si el requisito de una aplicación es parcial o completamente satisfactorio.
Lo que caracteriza un escrito formal de caso de prueba es que hay una entrada conocida y una salida esperada, los cuales son formulados antes de que se ejecute la prueba. La entrada conocida debe probar una precondición y la salida esperada debe probar una post-condición. A continuación un ejemplo de esto:

Requisito Funcional: El sistema deberá permitir el ingreso a la aplicación al dar click sobre el botón “ok”
Caso de Prueba: CP1 para Requisito Funcional “Entrada a la aplicación”.
Objetivo del caso de prueba: Comprobar que al dar clic en “ok” aparezca un nuevo menú.
Ejecución:

Interfaz Gráfica

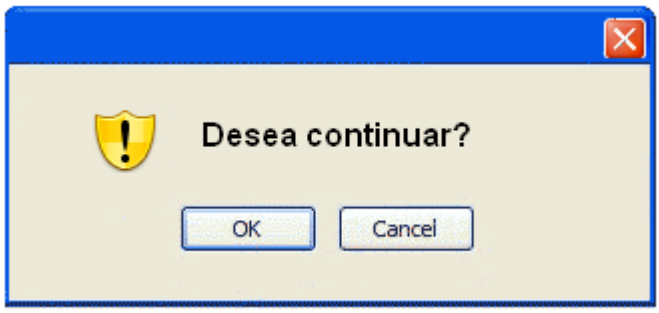

FIGURA 2: Ejemplo de cuadro de diálogo

Entrada: Click en OK
Resultado esperado: Nueva ventana con menú

6.1.2.1 Diseño de casos de prueba.

Un buen diseño de caso de prueba, es aquel diseño que incluye los casos de prueba que tengan la mayor probabilidad de encontrar el mayor número de issues con la mínima cantidad de esfuerzo y tiempo. Es por esto que hay tener criterios para elegir buenos casos de prueba a la hora de elaborar un diseño.

Un caso de prueba funcional es bien elegido si cumple con las siguientes condiciones:

- Especifica cada entrada requerida para ejecutar el caso de prueba (incluyendo las relaciones entre las diversas entradas; por ejemplo, la sincronización de las mismas).

- Especifica todas las salidas y las características requeridas (por ejemplo, el tiempo de respuesta) para los elementos que se van a probar.

- Se especifican necesidades de entorno (hardware, software y otras como, por ejemplo, el personal).

- Se especifican Requisitos especiales de procedimiento (o restricciones especiales en los procedimientos para ejecutar este caso).

- Se especifican dependencias entre casos (por ejemplo, listar los identificadores de los casos que se van a ejecutar antes de este caso de prueba).

- Reduce el número de otros casos necesarios para que la prueba sea razonable. Esto implica que el caso ejecute el máximo número de posibilidades de entrada diferentes para así reducir el total de casos.

- Tiene una probabilidad muy alta de descubrir un nuevo issue.

- No debe ser redundante.

- No debe ser ni muy sencillo ni excesivamente complejo: es mejor realizar cada prueba de forma separada si se quieren probar diferentes casos.

- Cubre un conjunto extenso de otros casos posibles, es decir, no indica algo acerca de la ausencia o la presencia de issues en el conjunto específico de entradas que prueba, así como de otros conjuntos similares.

6.1.2.2 Ejecución de un Caso de Prueba Funcional.

Tomemos como ejemplo el Caso de Prueba CP1 anterior; entonces: ¿cómo probar el caso de prueba y validar si ha tenido éxito?

Ejemplo: Procedimiento de prueba para CP1

1. Ejecutar el programa
2. Hacer clic en el botón “Ok”
3. Verificar que aparezca una nueva ventana con un menú

7. PROCEDIMIENTO PARA REALIZAR PRUEBAS FUNCIONALES DE SOFTWARE BASADO EN COMPONENTES.

La metodología o procedimiento que se desea proponer en el presente proyecto de grado contiene los pasos y etapas que buscan garantizar la calidad funcional en un producto de software, siguiendo como lineamiento la idea de detección temprana de issues del producto a nivel funcional haciéndose necesario que la metodología a proponer vaya alineada con el ciclo de desarrollo del software, desde su etapa más temprana hasta el final del mismo.

Como se ha mencionado a lo largo del proyecto, este procedimiento se encuentra basado en el ciclo de desarrollo del software basado en componentes y en sus mejores prácticas, y a pesar de que no es una de las metodologías de desarrollo más usadas en el momento, tiene un futuro muy prometedor para el análisis, implementación y documentación de aplicativos, lo cual queremos fortalecer aún más con esta propuesta metodológica de pruebas funcionales.

Existen dos modelos de ciclo de vida para el desarrollo de software basado en componentes al menos los más conocidos, el primero es llamado ciclo de vida gemelo y el segundo llamado ciclo de vida Watch (Ver figura 3 y 4 respectivamente).

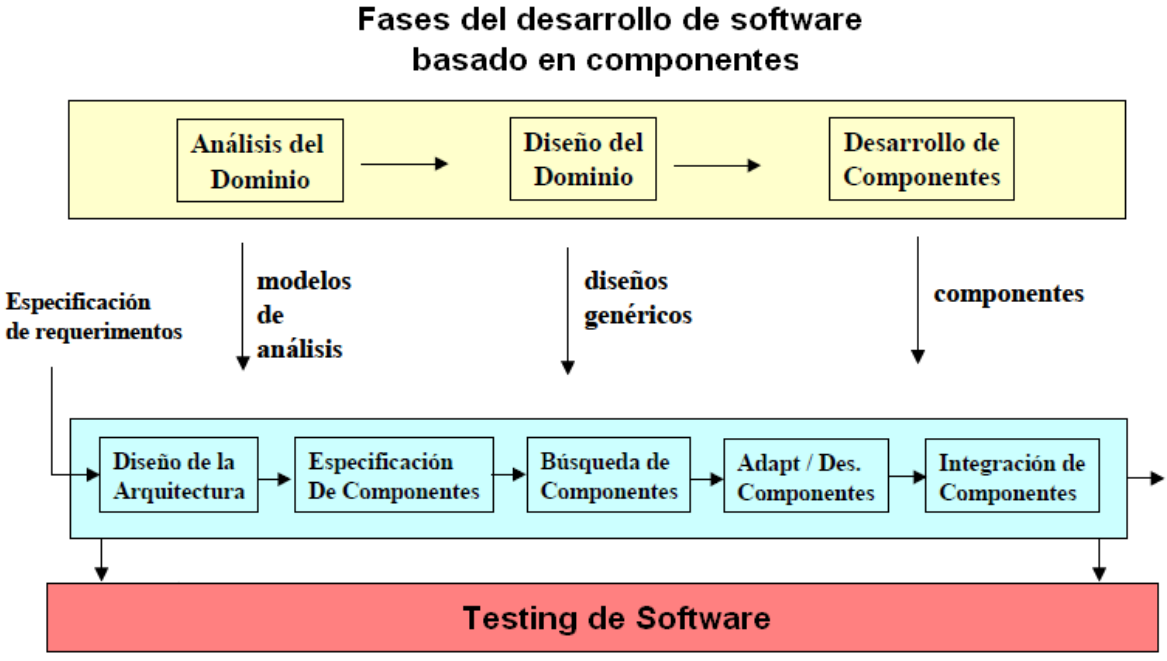

Figura 3. Ciclo de vida gemelo

Este modelo, denominado ciclo de vida gemelo (twin life cycle), divide el proceso de desarrollo de software en dos grandes bloques paralelos, tal como se ilustra en la Figura 3. El primer bloque (amarillo), conocido como Ingeniería de Dominios, contempla los procesos necesarios para desarrollar activos de software reutilizables en un dominio particular. El segundo bloque (azul) es denominado Ingeniería de Aplicaciones. Su propósito es el desarrollo de aplicaciones basado en la reutilización de activos de software producidos a través de la Ingeniería de Dominios. Y el tercer bloque (rojo) correspondiente al testing se software que acompaña el ciclo de vida a lo largo de todas sus etapas (es allí donde centraremos nuestro enfoque).

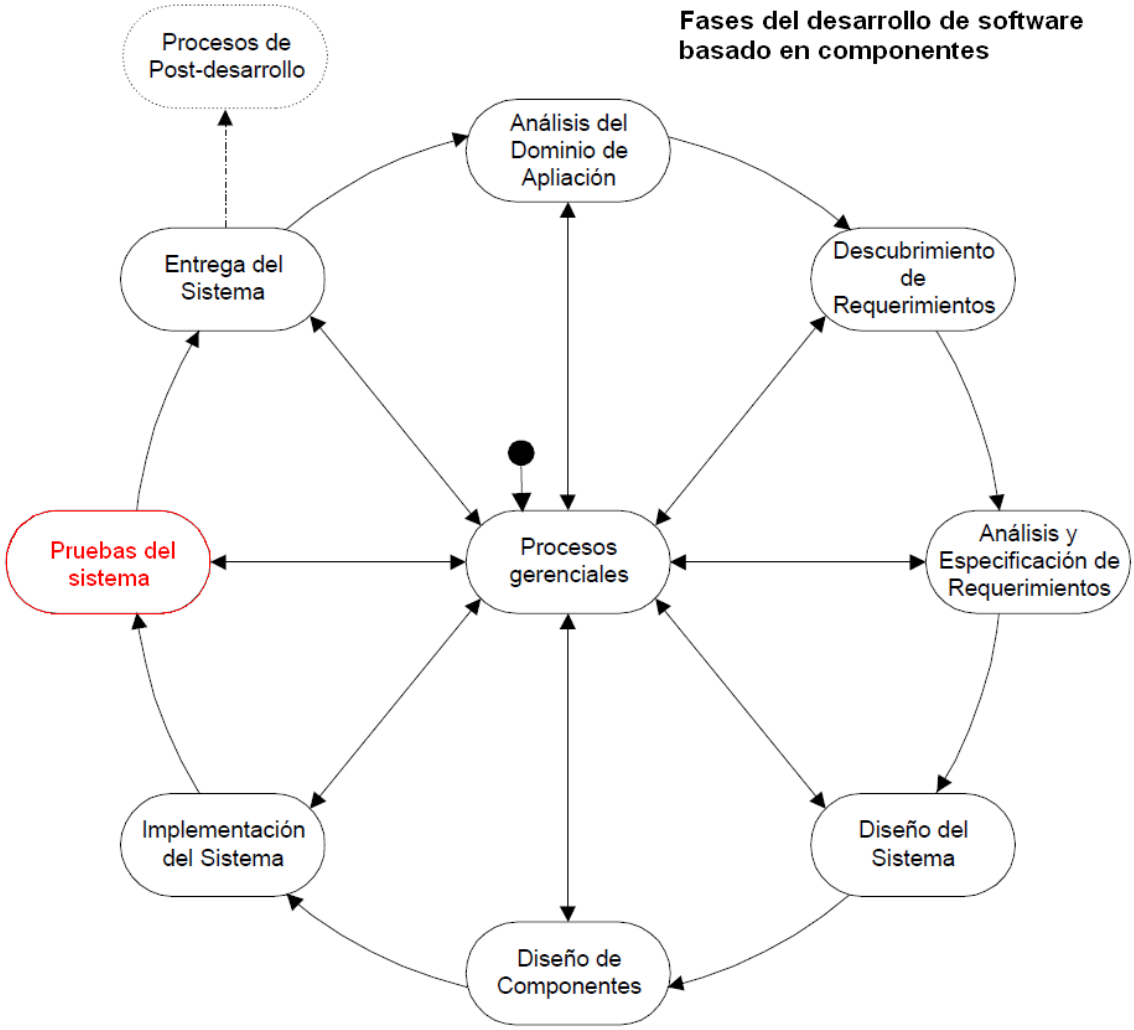

Figura 4. Ciclo de vida Watch

Por otro lado está el modelo de ciclo de vida Watch el cual combina los procesos más relevantes de la Ingeniería de Software Orientada a Objetos con el modelo de Ingeniería de Aplicaciones del ciclo de vida gemelo. Una característica importante de este modelo es la integración de los procesos gerenciales con los procesos técnicos del desarrollo de software basado en componentes. Esta integración facilita la labor del líder del proyecto, al describir cómo se debe llevar a cabo una gestión del proyecto integrada a los procesos técnicos del desarrollo de software. La estructura del método WATCH se ilustra en la Figura 4. Esta estructura emplea la metáfora de un reloj de pulsera para describir el orden de ejecución de los procesos técnicos de desarrollo de aplicaciones, indicando además cómo los procesos gerenciales controlan o coordinan el orden de ejecución de los procesos técnicos. Los procesos gerenciales están ubicados en el centro de la estructura para indicar explícitamente que ellos programan, dirigen y controlan el proceso de desarrollo. Los procesos técnicos están ubicados en el entorno siguiendo la forma que tiene el dial de un reloj. Ello indica que el orden de ejecución de las fases técnicas se realiza en el sentido de las agujas del reloj. Los procesos gerenciales pueden, sin embargo, invertir el orden de ejecución para repetir algunas de las fases anteriores.

Las tres primeras fases del modelo son similares a los modelos de procesos tradicionales. La fase de Análisis del Contexto permite que el grupo de desarrollo adquiera un conocimiento adecuado del dominio o contexto del sistema en desarrollo. Las fases de Descubrimiento, Análisis y Especificación de Requerimientos se encargan de identificar las necesidades y requerimientos de los usuarios, así como analizarlos, especificarlos gráficamente y documentarlos. La fase de diseño del sistema establece y describe la arquitectura del software. Describe cada uno de los componentes que requieren tal estructura y cómo esos componentes se interconectan para interactuar. El grupo de desarrollo debe, a partir de esta arquitectura, determinar cuáles componentes se pueden reutilizar y cuáles requieren ser desarrollados en su totalidad. Los componentes reutilizados deben ser adaptados, para satisfacer los requerimientos del sistema; mientras que los componentes nuevos, deben ser diseñados, codificados y probados separadamente durante la fase de Implementación. Finalmente, la fase de Entrega se encarga de la instalación, adiestramiento de usuarios y puesta en operación del sistema.

Como nuestra área de enfoque será en el flujo de trabajo Testing de software o pruebas de software, la cual hace parte de ambos ciclos de vida. Y como bien mencionaba anteriormente el ciclo de vida Watch es una combinación entre los procesos más relevantes de la Ingeniería de Software Orientada a Objetos con el modelo de Ingeniería de Aplicaciones del ciclo de vida gemelo, tomaremos como referencia las partes coincidentes de ambos ciclos de vida para desarrollar nuestra propuesta metodológica de pruebas de software, es decir tomaremos el ciclo de vida gemelo y sus fases ya que estas a su vez están contenidas en el ciclo de vida Watch haciendo que nuestra propuesta obviamente sea totalmente adaptable al ciclo de vida gemelo pero que también sirva como metodología de pruebas para el ciclo de vida Watch con algunas adaptaciones. Entonces las fases que consideraremos como ciclo de vida del desarrollo de software basado en componentes son:

- Diseño de la arquitectura
- Especificación de componentes
- Búsqueda de componentes
- Adaptabilidad de componentes
- Integración de componentes

Y las etapas que se considerarán dentro de la metodología de las pruebas a proponer son:

- Planeación de pruebas
- Diseño de casos de prueba
- Ejecución de pruebas
- Evaluación de pruebas

Tanto las fases del ciclo de vida del desarrollo de software basado en componentes como las etapas de la metodología de pruebas se pueden observar en la Figura 5.

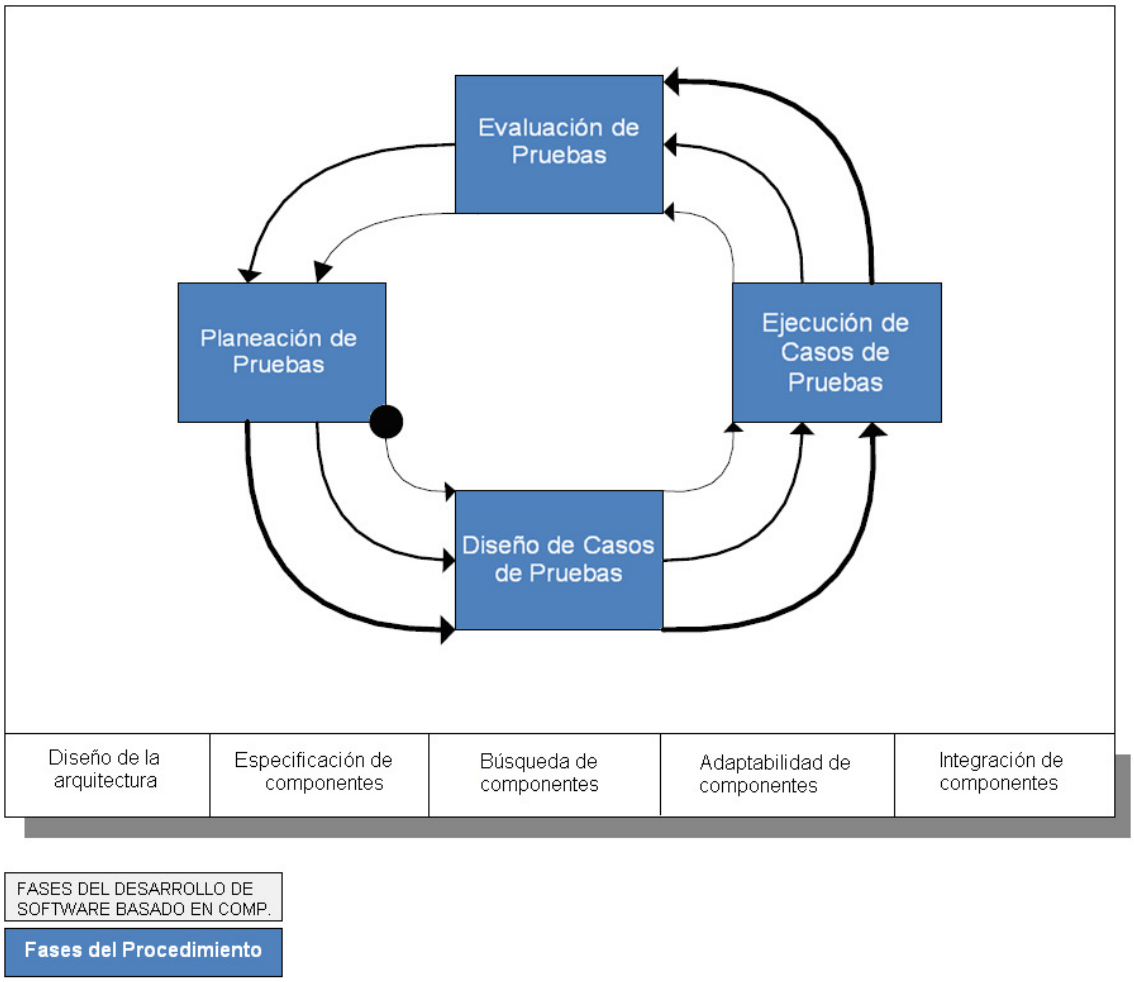

FIGURA 5: Procedimiento para realizar pruebas funcionales de software basado en componentes

7.1 Etapas de la Metodología de Testing para las Fases del Software Desarrollado a Base de Componentes.

A continuación, se pretende proponer la manera como el procedimiento propuesto para realizar pruebas funcionales de software se integra y se adapta a cada uno de los procesos que se deben llevar a cabo por la metodología de desarrollo de software basado en componentes a través de cada una de las fases, diseño de la arquitectura, especificación y búsqueda de componentes, adaptabilidad de componentes e integración de componentes.

7.1.1 Fase de Diseño de la Arquitectura.

7.1.1.1 Descripción.

La fase de diseño describe la arquitectura del software. Describe cada uno de los componentes que requieren tal estructura y cómo esos componentes se interconectan para interactuar, así como el ambiente y los principios que orientan su diseño. El grupo de desarrollo debe, a partir de esta arquitectura, determinar cuáles componentes se pueden reutilizar y cuáles requieren ser desarrollados en su totalidad. Los componentes reutilizados deben ser adaptados, para satisfacer los requerimientos del sistema, mientras que los componentes nuevos, deben ser diseñados, codificados y probados separadamente durante la fase de implementación. En esta fase de diseño de la arquitectura es posible redefinir el alcance del proyecto con los patrocinadores, identificar los riesgos asociados al proyecto.

7.1.1.2 Etapa de la Metodología: Planeación de Pruebas.

Esta etapa comienza con la recepción de la documentación y componentes del aplicativo, donde el cliente si es que lo hay, entrega toda la documentación necesaria para poder planificar y diseñar la prueba, se realiza la lectura y entendimiento de la documentación, así como una reunión con el analista para revisar los temas que no se hayan entendido de la documentación de la aplicación. Además, esta etapa de Planeación de Pruebas permite conocer el alcance de las pruebas definiendo aspectos como las entradas de pruebas (requerimientos para pruebas), la valoración de riesgos, las estrategias, los recursos necesarios, el cronograma y el plan de pruebas. Los resultados de la etapa son el plan de pruebas y el cronograma de pruebas, documentos que contienen todos los aspectos antes descritos.

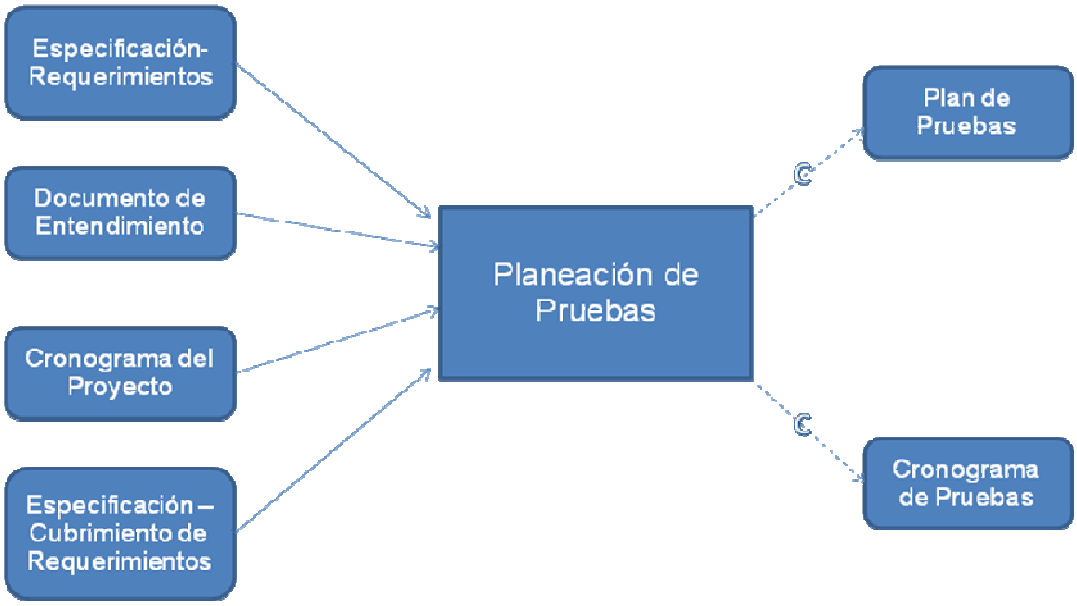

FIGURA 6: Etapa de planeación de pruebas

7.1.1.3 Objetivo.

El propósito del plan de pruebas es explicitar el alcance, enfoque, recursos requeridos, calendario, responsables y manejo de riesgos de un proceso de pruebas. Este último, el manejo de riesgos debe incluir tanto los requerimientos que fueron cubiertos por componentes que fueron reutilizados y aquellos que requerimientos que debieron ser cubiertos por componentes que fueron desarrollados en su totalidad. La razón de hacer esto es que los componentes reutilizados traen consigo un riesgo inherente más alto que los que fueron desarrollados por el mismo desarrollador, esto como consecuencia en ocasiones de no conocer exactamente su procedencia.

7.1.1.4 Cubrimiento.

Va desde el comienzo de la planeación del proyecto, hasta los comienzos del análisis y durante el diseño de la arquitectura del software.

7.1.1.5 Entradas

Proveedor	Insumo Contenido	Requisitos
Cliente	Especificación - requerimientos	• Descripción detallada por cada requisito y para la integración final de los componentes. • Objetivos • Alcance del desarrollo • Recursos de software • Nombre Proveedor • Descripción Proveedor • Restricciones
Cliente	Documento de entendimiento	• Acuerdo entre usuario y desarrolladores • Nombre de los usuarios • Nombres de los desarrolladores del proveedor
Cliente	Cronograma del proyecto	• Tiempo de cada actividad • Fechas de inicio • Fecha de finalización • Porcentajes de avance de acuerdo a la fecha • Responsable
Cliente	Especificación – Cubrimiento de Requerimientos	• Nombre de Requisito • Cubrimiento (Especificar si fue cubierto por componente desarrollado o reutilización de un componente)

7.1.1.6 Actividades.

Responsable	Actividad
PLANEAR	
Analista de Pruebas/Analista líder del proyecto	Revisar toda la documentación, es decir la especificación de requerimientos, aquellos requerimientos que fueron cubiertos por componentes, documento de entendimiento y

	cronograma del proyecto, con el fin de identificar previamente a. Posibles riegos. (Tener en cuenta especialmente aquellos requerimientos que hayan sido cubiertos con la reutilización de componentes y la integración de componentes que siempre supone un momento crítico durante el desarrollo) b. Recursos que sean necesarios tanto de hardware como de software. c. Recursos humanos para el proyecto. d. Estimación de tiempos y esfuerzos del proceso de desarrollo. e. Elaboración de los escenarios de las pruebas.
HACER	
Analista de Pruebas/ Coordinador de Proyecto	En base a los artefactos recibidos, especificación de requisitos, documento de entendimiento, cronograma del proyecto y especificación de cubrimiento de requerimientos hace el documento con los supuestos que se asumirán para el proceso de pruebas.
Analista de Pruebas/ Coordinador de Proyecto	Realiza la matriz de riesgos para cada una de las funcionalidades del aplicativo. Para cada funcionalidad se estima de 1 a 3 (siendo 3 la probabilidad mayor) tanto el impacto y la posibilidad de falla, estos dos factores se multiplican y luego se totaliza el promedio para todas las funcionalidades. Tener muy presente que para aquellas funcionalidades o requerimientos que fueron cubiertos por los componentes reutilizados, el nivel de riesgo debería ser más alto, adicionalmente se debe estimar el riesgo de la funcionalidad final que depende estrictamente de la integración de los componentes, momento crítico del desarrollo con este método, por tanto se supone un alto nivel de riesgo para este caso.
Analista de Pruebas / Coordinador de Proyecto	Define la configuración del ambiente de pruebas para que sea igual o lo más semejante posible al de producción.
	Revisa estimación de tiempos y esfuerzos del proceso de desarrollo realizado.
	Genera el documento de plan de pruebas.

Equipo de control de calidad Funcional	Reunión con el equipo de proyecto para sensibilizar acerca del proceso de control de calidad de software que se va a realizar.
VERIFICAR	
Comité de Cambios	Revisa y aprueba el plan de pruebas y el cronograma de Control de calidad funcional de aplicaciones.
	Registra los cambios que considere necesarios en los documentos.
ACTUAR	
Analista de pruebas/Coordinador de proyecto	Realiza las correcciones necesarias en el documento de plan de pruebas y cronograma de actividades de control de calidad funcional de software.

7.1.1.7 Salidas.

Cliente	**Producto**	**Contenido / Requisitos**
- Analista de Pruebas / Coordinador de proyecto. - Comité de cambios. - Proceso de validación de artefactos	Plan de pruebas	• Propósito. • Alcance del proyecto. • Supuestos. • Elementos software a probar. • Elementos software que fueron abarcados por los componentes reutilizados. • Características a probar. • Enfoque general de la prueba. • Criterios de paso / fallo para cada elemento. • Criterios de suspensión y requisitos de reanudación. • Documentos a entregar. • Responsabilidades de las organizaciones. • Riesgos asumidos por el plan y planes de contingencias. • Aprobaciones y firmas con

		nombre y puesto desempeñado. • Descripción de cada tipo de prueba. • Entregables del proyecto. • Necesidades ambiente. • Recursos humanos. • Datos de acceso y conectividad. • Estadísticas y métricas a obtener.
- Analista de Pruebas / Coordinador de proyecto - Comité de cambios	Cronograma de pruebas.	• Actividades a realizar en el proceso por fases. • Porcentaje de tiempo de cada fase en el proyecto. • Tiempo estimado por actividad. • Fecha inicio y final actividad.

7.1.1.8 Plantilla o Documento para el Plan de Pruebas.

La metodología propuesta tiene para cada una de las etapas dentro de todo el proceso de las pruebas una plantilla o documento, en este caso la plantilla asociada a la etapa del Plan de Pruebas durante la fase del diseño de la arquitectura.

7.1.1.9 Determinación del avance.

El avance de la etapa se determinará principalmente de acuerdo al criterio del analista de pruebas, sin embargo el analista deberá tener en cuenta algunos porcentajes de avance de referencia, como por ejemplo que la etapa estará completo en un 60% solo si ya se ha generado el Plan de Pruebas. Tendrá un avance del 80% si el cronograma de Pruebas se encuentra finalizado y por último se tendrá un 100% de avance si el comité de cambios ha aprobado tanto el plan como el cronograma de pruebas.

7.1.1.10 Definición de Artefactos

7.1.1.10.1 Especificación de Requerimientos.

Este documento contiene cada uno de los requerimientos con las especificaciones de cada uno de ellos y que se esperan sean cumplidos a cabalidad por parte de los componentes. Además es el documento que contiene la información general e inicial sobre el proyecto, tales como el por qué del proyecto, lo que espera obtener después de integrados los componentes.

7.1.1.10.2 Documento de Entendimiento.

Documento donde se establecen los acuerdos y condiciones entre el área de certificación o de pruebas y el área de desarrollo con el fin de establecer una relación de mutuo entendimiento que favorezca al proyecto en general.

7.1.1.10.3 Cronograma del Proyecto.

Cronograma a seguir durante todo el proyecto

7.1.1.10.4 Especificación de Cubrimiento de Requerimientos.

Listado con todos los requerimientos que fueron cubiertos tanto por los componentes reutilizados como por los componentes desarrollados.

7.1.1.10.5 Plan de Pruebas.

Documento donde se define la planeación de las pruebas que se van a realizar.

7.1.1.10.6 Cronograma de Pruebas.

Cronograma que estipula los tiempos a dedicar a cada etapa durante todo el proceso de las pruebas funcionales, registra tanto las horas a dedicar por día a determinada actividad como el porcentaje esperado de avance por día.

7.1.2 Fase de Especificación y Búsqueda de Componentes.

7.1.2.1 Descripción.

La especificación y búsqueda de componentes consta de tres etapas. Primero, la identificación de componentes donde se genera un conjunto inicial de interfaces y especificaciones de componentes conectados entre sí en una primera aproximación a la arquitectura. Segundo, la interacción entre componentes donde se decide como los componentes van a trabajar de modo coordinado para conseguir la funcionalidad deseada. Y finalmente, la especificación de componentes donde se definen las especificaciones concretas de cada uno de los componentes utilizados. Al final se deben haber generado las especificaciones de los componentes.

7.1.2.2 Etapa de la Metodología: Diseño de Casos de Prueba.

El diseño de casos de prueba es una parte de las pruebas de componentes y sistemas en las que se diseñan los casos de prueba (entradas y salidas esperadas) para probar el aplicativo. El objetivo de la etapa de diseño de casos de prueba es plasmar los escenarios que se implementarán en las pruebas del sistema para alcanzar los objetivos que se establecieron en la planeación y que además muestren que el aplicativo satisface los requerimientos. Sin embargo y como sucede a menudo, estos requerimientos han sido cubiertos tanto por el propio desarrollo de componentes como por la reutilización de los mismos, esto hace que sea necesario discriminar en el diseño de casos de prueba aquellas funcionalidades que fueron desarrolladas con componentes propios de aquellas otras que contienen componentes reutilizados, esto con el fin de prevenir al analista de pruebas quien es el ejecutor de dichos casos de prueba del riesgo inherente que trae consigo aquellas funcionalidades desarrolladas con componentes reutilizables ya que en muchos casos por la falta de información y conocimiento de la procedencia de los componentes estos suelen traer consigo riesgos que pueden poner en riesgo el correcto funcionamiento de un aplicativo.
De esta etapa además debe salir el diseño de casos de prueba para el aplicativo final después de la integración de todos los componentes.

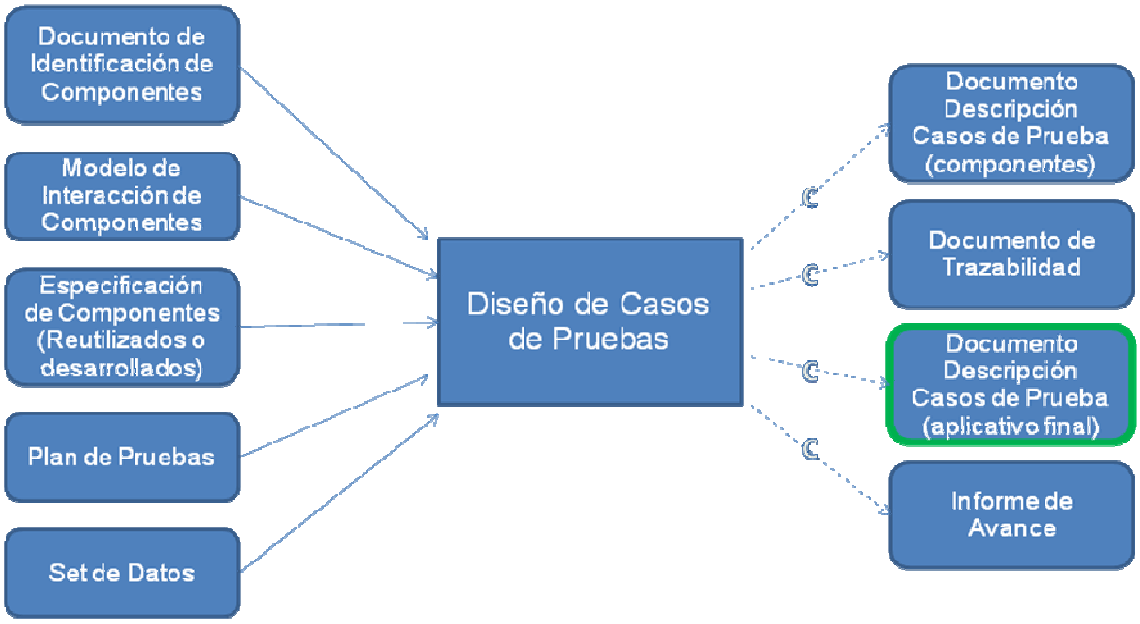

FIGURA 7: Etapa de diseño de casos de pruebas

7.1.2.3 Objetivo.

Especificar los casos de prueba que servirán como escenarios para realizar las pruebas funcionales al aplicativo con el fin de validar la implementación tal cual se consignó en los requerimientos y encontrar la mayor cantidad de issues en el software a evaluar. Esto quiere decir, que el aplicativo desarrollado cumpla a cabalidad con el objetivo principal para el cual fue creado.

7.1.2.4 Cubrimiento

El cubrimiento va específicamente desde el diseño de la arquitectura hasta la integración de componentes.

7.1.2.5 Entradas.

Proveedor	Insumo	Requisitos
Equipo de desarrollo.	Documento de identificación de componentes	• Listado con cada componente utilizado • Revisado y aprobado por el Analista de Pruebas.
	Modelo de Interacción de Componentes	• Modelo que ilustre como los componentes van a trabajar de modo coordinado para conseguir la funcionalidad deseada • Revisados y aprobados por el Analista de Pruebas.
	Especificación de Componentes (Reutilizados o Desarrollados)	Debe contener la definición detallada de las especificaciones concretas de cada uno de los componentes utilizados, las pruebas funcionales realizadas anteriormente sobre cada uno de ellos (si se tiene la información en el caso de los reutilizados) y el requerimiento respectivo al cual le dieron solución.
Analista líder del Proyecto / Analista de Pruebas.	Set de Datos	Revisado y validado por el usuario.
Etapa de planeación de pruebas.	Plan de pruebas.	Deberá estar completo y disponible (ver etapa De planeación de pruebas).

7.1.2.6 Actividades.

Responsable	Actividad
PLANEAR	
Analista de pruebas	Identificar cada uno de los componentes que son la herramienta base para cumplir con los requerimientos del desarrollo y a su vez de insumo para el diseño de los casos prueba.
	Identificar las interacciones entre los componentes que hacen parte del desarrollo y como estos se complementan para cumplir con la funcionalidad del desarrollo. Interacciones que servirán de escenarios para elaborar los casos de prueba.
	Identificar como fueron aplicados los componentes a lo largo del desarrollo validando si restringieron el software construido.
HACER	
Analista de pruebas	*Identificar y describir casos de prueba tanto para cada uno de los componentes como para la integración final de estos:* Cada uno de los casos de prueba debe ser Identificado, especificado y descrito. En el Diseño de Casos de Prueba, se deben especificar los siguientes atributos para cada caso de prueba: 1. **Código**: Este código es el identificador del caso de prueba. 2. **Nombre**: Es el nombre que identifica cada caso de prueba. 3. **Descripción**: Es la descripción del caso de prueba. 4. **Responsable**: Es el nombre del responsable de elaborar el caso de prueba. 5. **Configuración**: Describe en forma detallada el ambiente (tanto de software como de hardware) en el cual se realizarán las pruebas. 6. **Entradas de prueba**: Especifica cada uno de los requisitos y características que conforman dicho caso de prueba. 7. **Precondiciones**: Son los requisitos que se deben dar para poder ejecutar el caso de prueba. 8. **Post-condiciones**: Son los requisitos que se deben dar al terminar la ejecución del caso de prueba. 9. **Criterio de éxito**: Es la meta que se debe cumplir para definir si un caso de prueba tuvo o no éxito al ser ejecutado. 10. **Procedimientos de prueba**: Es la descripción de todos los pasos y puntos de validación que se deben seguir al ejecutar un caso de prueba.

	11. **Procedencia:** Procedencia del componente para el caso de prueba a evaluar. Si el caso de prueba a probar hace parte de una funcionalidad que fue cubierta por un componente reutilizado se debe marcar con una 'R' este caso con el fin de advertir entre los casos de prueba que cubren funcionalidades que fueron desarrolladas con componentes propios y aquellas que fueron desarrolladas con componentes reutilizables.
Analista de pruebas	*Identificar y estructurar procedimientos de prueba:* El procedimiento de prueba es la forma ordenada de ejecutar un caso de prueba. El procedimiento de prueba se compone básicamente de dos elementos: **1. Paso**: Es un procedimiento que se debe implementar utilizando una función de la aplicación. **2. Punto de validación**: Es una comprobación que se realiza para asegurase que un paso o varios están siendo completados satisfactoriamente o no.
	Identificar el Set de datos, es decir el conjunto de datos de prueba que servirán para ejecutar las pruebas funcionales de software. En este proceso se deben identificar datos positivos, que conducen la prueba para identificar que las funciones estén correctas, y datos negativos, que conducen la prueba a encontrar issues. Se deben tener en cuenta los siguientes parámetros en el set de datos. **1. Profundidad**: Cantidad de datos del set de datos de prueba. Es más precisa la ejecución de pruebas funcionales mientras más escenarios de un caso de prueba simulen la realidad. **2. Amplitud**: Es la variación en los valores de los datos de prueba. **3. Alcance**: Importancia de los valores comunes (datos positivos) y no comunes (datos negativos) en el Set de datos.
	Diseñar los casos de prueba de regresión: Las pruebas de regresión son un conjunto de pruebas (ya realizadas antes) que aseguran que los cambios no han dado lugar a cambios colaterales.
	Crear el reporte de informe de avance donde se describen las actividades que se realizan diariamente.
VERIFICAR	
	Verificar que cada caso de prueba se encuentra completo y

Analista de pruebas / Analista líder del proyecto	contiene todos los atributos exigidos, además que se diferencien aquellos casos de prueba correspondientes a las funcionalidades desarrolladas con componentes propios de aquellas funcionalidades desarrolladas con componentes reutilizables.
	Revisar y controlar la cobertura de requisitos: Esta actividad permite mantener un control sobre el cubrimiento de requisitos por los casos de prueba, para identificar si se está abarcando completamente el sistema y trazando de forma adecuada los casos de prueba respecto a los requisitos y/o casos de uso.
ACTUAR	
Analista de pruebas	Completar los atributos faltantes en los casos de prueba. Incorporar nuevos casos de prueba de ser necesario.

7.1.2.7 Salidas.

Cliente	**Producto**	**Requisitos**
- Analista líder del proyecto. - Etapa de ejecución. de pruebas - Analista de pruebas. - Comité de cambios.	Documento descripción de casos de prueba (componentes)	Debe contener cada uno de los casos de prueba con sus atributos y los procedimientos de prueba asociados al mismo para cada uno de los componentes a probar.
- Etapa de ejecución de pruebas. - Comité de cambios	Documento de Trazabilidad.	Deben estar relacionados los requerimientos vs. Casos de prueba.
- Analista líder del proyecto. - Etapa de ejecución. de pruebas - Analista de pruebas. - Comité de cambios.	Documento descripción de casos de prueba (aplicativo final)	Debe contener cada uno de los casos de prueba con sus atributos y los procedimientos de prueba asociados al mismo para cada la certificación del aplicativo final después de

		la integración de los componentes.
- Analista líder del proyecto. - Comité de cambios.	-Informe de avance.	Debe contener un reporte con el avance del proyecto y los problemas encontrados en la etapa en forma diaria.

7.1.2.8 Documentos y Recursos asociados.

- Instructivo para el diseño de casos de prueba.
- Documento de Trazabilidad.

7.1.2.9 Determinación del avance

El avance del diseño de casos de prueba será del 100% solo cuando estén diseñados el 100% de casos de prueba, es decir que todos los requerimientos estén completamente cubiertos implícita o explícitamente en el diseño de dichos casos de prueba con sus respectivos procedimientos de prueba y set de datos siempre y cuando estos apliquen.

7.1.2.10 Definición de Artefactos.

7.1.2.10.1 Documento de Identificación de Componentes

Este documento debe contener los componentes utilizados durante el desarrollo, además el responsable por cada uno de los componentes bien sea que el componente haya sido desarrollado o se haya usado uno reutilizado.

7.1.2.10.2 Modelo de Interacción de Componentes.

Este documento debe ilustrar como los componentes van a trabajar de modo coordinado, las relaciones que van a existir entre ellos y el modo de interactuar para conseguir la funcionalidad deseada.

7.1.2.10.3 Especificación de Componentes (reutilizados o desarrollados).

Este documento debe contener la definición detallada de las especificaciones concretas de cada uno de los componentes utilizados, debe diferenciar entre los componentes que fueron desarrollados de aquellos que son reutilizados, las pruebas funcionales realizadas anteriormente sobre cada uno de ellos (si se tiene la información en el caso de los reutilizados) y el requerimiento respectivo al cual le dieron solución.

7.1.2.10.4 Plan de Pruebas.

Documento donde se define la planeación de las pruebas que se van a realizar.

7.1.2.10.5 Set de Datos.

Este artefacto define una lista de variables y sus posibles valores a introducir para la ejecución de las pruebas, así como también los resultados esperados de la ejecución para propósitos comparativos. Se pueden tomar en cuenta valores específicos o describir rangos de valores. Los Datos de Pruebas se utilizan como fuente de engaño al objeto de prueba y así encontrar issues. Cabe destacar que cada caso de prueba deberá ser ejecutado una vez por cada combinación de valores.

7.1.2.10.6 Documento Descripción de Casos de Prueba (componentes).

Documento donde se definen los casos de prueba que se van a ejecutar para cada uno de los componentes a probar.

7.1.2.10.7 Documento de Trazabilidad.

Representación grafica de las relaciones existentes entre los requisitos y los casos de prueba, y la manera como se encuentran estas relaciones.

7.1.2.10.8 Documento Descripción de Casos de Prueba (aplicativo final).

Documento donde se definen los casos de prueba que se van a ejecutar para la fase de integración de los componentes que da como resultado el aplicativo final.

7.1.2.10.9 Informe de Avance.

Informe con los avances que se han realizado en el proceso de pruebas funcionales.

7.1.3 Fase de Adaptabilidad de Componentes.

7.1.3.1 Descripción.

El propósito de esta fase es adaptar el componente o los componentes de manera tal que cumplan con la funcionalidad especificada, esta adaptación debe realizarse de manera independiente por cada una de las funcionalidades que se requieren, para ello se deben clarificar en primer lugar los requerimientos pendientes, administrar el cambio de los componentes seleccionados, y ejecutar el plan de administración de recursos y mejoras en el proceso de desarrollo para el proyecto.

7.1.3.2 Etapa de la Metodología: Ejecución de Casos de Pruebas.

En esta etapa se realiza la ejecución de los casos de prueba que se planearon y diseñaron, los cuales son ejecutados a través de la interfaz de usuario y se conocen como pruebas de caja negra. La ejecución de los casos de prueba se debe realizar siguiendo la descripción de cada caso de prueba, haciendo énfasis tanto en aquellos casos de prueba cuya funcionalidad haya sido desarrollada con componentes reutilizados (estos casos debieron ser marcados con una 'R' desde la anterior etapa), así como en los componentes desarrollados cuyas pruebas funcionales realizadas anteriormente en otro proyecto hayan presentado algún tipo de inconveniente, además se debe registrar el resultado arrojado al ejecutar cada caso de prueba, así como los issues encontrados para cada componente. En esta etapa además se debe llevar a cabo la ejecución de los casos de prueba realizados para la integración de los

componentes que da como resultado el aplicativo final, así mismo como el registro de los issues encontrados.

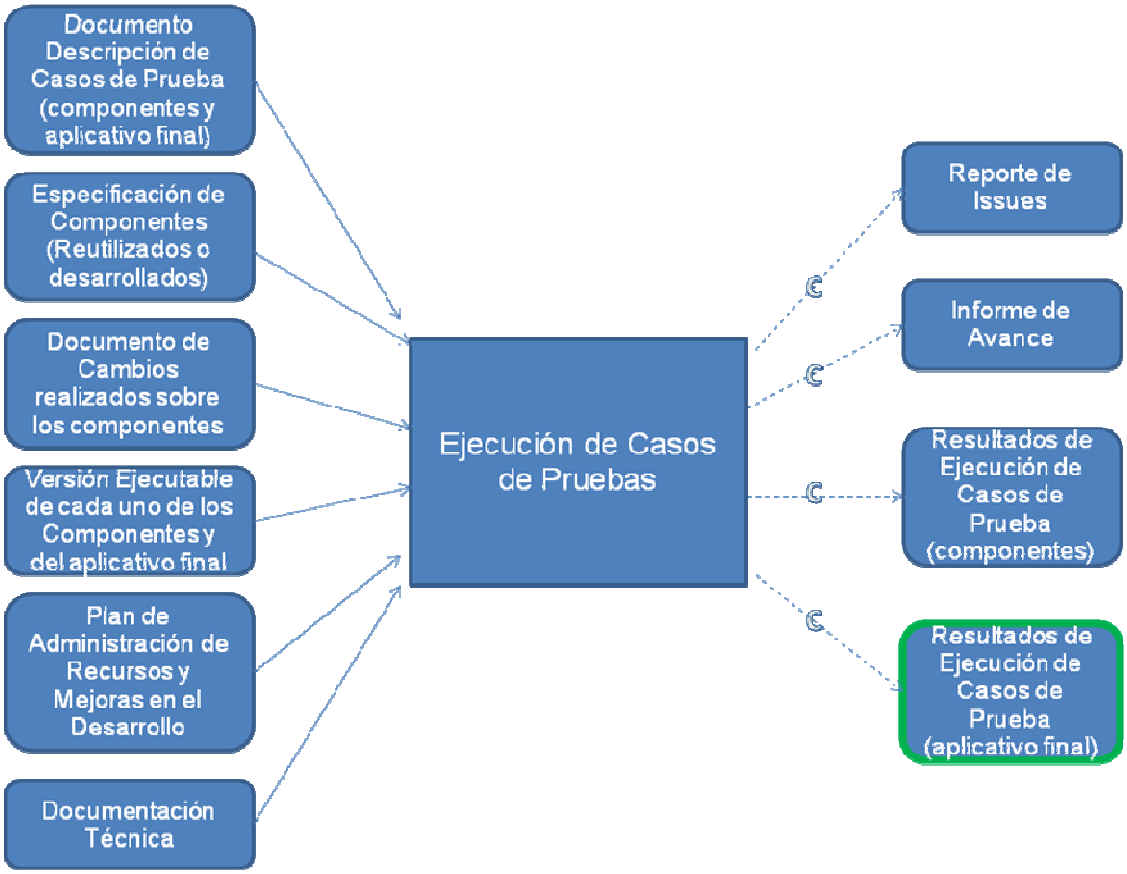

FIGURA 8: Etapa de ejecución de casos de pruebas

7.1.3.3 Objetivo.

Ejecutar cada uno de los casos de prueba y reportar los issues encontrados en el software.

7.1.3.4 Cubrimiento.

Va desde la primera liberación del aplicativo en pruebas, hasta que el producto alcanza estabilidad y puede realizarse una prueba piloto antes de ser desplegado a producción.

7.1.3.5 Entradas.

Proveedor	Insumos	Requisitos
Etapa Diseño de Casos de Prueba.	Documento descripción de Casos de Prueba (componentes y aplicativo final)	Cada caso de prueba debe tener completos sus atributos (ver Etapa Diseño de Casos de Prueba).
Etapa Diseño de Casos de Prueba.	Especificación de Componentes (reutilizados o desarrollados)	Cada componente debe tener sus especificaciones (ver Etapa Diseño de Casos de Prueba).
Equipo de Desarrollo	Documento de Cambios realizados sobre los componentes	Si los componentes utilizados fueron modificados, todos los cambios deben estar registrados en un documento.
Equipo de Desarrollo	Versión ejecutable de cada uno de los componentes y del aplicativo final	Los componentes deben contener un número de versión y fecha de creación. Deben estar aislados en un ambiente de pruebas. Lo mismo aplica para la versión ejecutable del aplicativo final.
Equipo de Desarrollo	Plan de administración de recursos y mejoras en el desarrollo	Debe haber sido revisado por equipo de desarrollo.
Equipo de Desarrollo	Documentación técnica - Manual de instalación y configuración. -Manual de mantenimiento y soporte. - Manual de administración	Debe haber sido revisada y validada por el equipo de desarrollo.

	y seguridad. -Manual de uso y ayudas en línea.	

7.1.3.6 Actividades.

Responsable	Actividad
PLANEAR	
Equipo de plataforma	Configuración de la ambientación de equipos de pruebas: En este paso se debe adaptar el ambiente de pruebas donde será probado la versión de la aplicación. Se deben configurar tanto el software (sistema operativo, base de datos, servicios y protocolos, entre otros) como el hardware (Procesador, memoria, disponibilidad de espacio en disco, entre otros).
	Configuración de las herramientas de pruebas: Si se van a utilizar herramientas para automatización de pruebas, estas deben ser configuradas en el servidor y/o en el (los) cliente (s).
Analista de Pruebas	Conocer y explorar el componente, antes de ejecutar un caso de prueba el analista deberá realizar una prueba exploratoria (smoke test) para conocer el componente y cada uno de los módulos o funciones que se van a probar. También se hace con el fin de saber las acciones necesarias para suspender o detener la prueba (cuando los acontecimientos no previstos lo obliguen). El mismo procedimiento se debe seguir al momento de ejecutar un caso de prueba del aplicativo final.
	Verificar que cada módulo de la aplicación que vaya a ser probado haya sido verificado: Cada módulo que se vaya a probar debe haber sido verificado por un miembro del equipo de desarrollo, para lo cual debe entregar la lista de chequeo correspondiente.
HACER	
	Ejecución de los casos de prueba para los componentes: En esta actividad se realiza la ejecución de los casos de

Analista de Pruebas	prueba que se planearon y diseñaron, los cuales son ejecutados a través de la interfaz de usuario y se conocen como pruebas de caja negra. Esta ejecución se debe hacer teniendo en cuenta el cronograma, los diseños de casos de prueba tanto los normales como los que fueron marcados con 'R' ya que sus componentes son aquellos componentes reutilizados y las pruebas funcionales realizadas anteriormente sobre cada uno de los componentes (tanto los desarrollados como los reutilizados). A continuación entonces las condiciones a tener en cuenta durante la ejecución de los casos de prueba: • Si el diseño de caso de prueba no está marcado, es decir corresponde al caso de prueba de una funcionalidad cuyos componentes utilizados fueron desarrollados, entonces se utilizan como fundamento las pruebas funcionales unitarias realizadas por el desarrollador y se prueba el caso solo una vez sin tener mayores consideraciones. • Si el diseño de caso de prueba está marcado con una 'R' es altamente recomendable hacer una pequeña regresión de este caso en particular para asegurarnos que el componente reutilizado que fue usado para cumplir con la funcionalidad de dicho caso de prueba es confiable y no representa ningún riesgo para el aplicativo. Para ello se recomienda revisar las pruebas funcionales realizadas anteriormente en otro proyecto sobre el componente o los componentes que han sido reutilizados para el desarrollo de la funcionalidad de la cual hace parte el caso de prueba en mención. Si en estas pruebas funcionales se evidencia complejidad y se detecta que el caso es un caso de prueba que en su momento fue crítico entonces: - Si fue crítico y paso las pruebas realizadas sin ningún problema, se aprovecha una de las cualidades de

Analista de Pruebas	este método de desarrollo y no se prueba nuevamente. - Pero si fue crítico y se evidenciaron problemas durante las pruebas realizadas, se convierte en una desventaja el uso de componentes reutilizables y se debe probar el caso de prueba no solo una vez sino también hacerle una regresión en particular para descartar issues inherentes a la reutilización. • Un tercer caso que aplica tanto para los casos de prueba que no están marcados con 'R' como los que si están es cuando simplemente por criterio del analista de pruebas, este considera que se trata de un componente de alto riesgo, en este caso también se recomienda hacer una pequeña regresión en particular del caso de prueba con el fin de descartar la aparición de un posible issue. Ejecución de los casos de prueba para el aplicativo final (después de la integración de componentes): En esta actividad se realiza la ejecución de los casos de prueba que se planearon y diseñaron para el aplicativo final, los cuales son ejecutados a través de la interfaz de usuario y se conocen como pruebas de caja negra. Esta ejecución se debe hacer teniendo en cuenta el cronograma y los diseños de casos de prueba
	• Reportar issues: - Todos los issues que surjan en la etapa de ejecución de casos de prueba, deben ser reportados. Para que se genere mayor facilidad en el proceso de pruebas, los issues son reportados en una aplicación que permita el seguimiento de dichos issues y permita obtener un registro e historial de los mismos. La herramienta es llamada bugtracker.
	• Registrar la ejecución de pruebas: Cada que se realice un ciclo de pruebas se debe

	registrar en el documento llamado Resultados de Ejecución, todos los eventos que sucedan durante la ejecución de cada escenario de cada caso de prueba. Se debe especificar en el Registro de Pruebas: - Código del caso de prueba. - Nombre del caso de prueba. - Módulo al que corresponde el caso de prueba - Resultado Obtenido. - Estado de ejecución (Pasó, Falló, No se pudo ejecutar). - Issues Asociados durante ese ciclo. - Fecha de ejecución. - Analista de Pruebas. Este documento permitirá obtener métricas y estadísticas comparativas.
VERIFICAR	
Equipo de Desarrollo	Investigar los resultados inesperados: Una vez reportados los issues, éstos son analizados por el equipo de desarrollo quien debe resolver los nuevos issues que aparecen reportados, en el orden de prioridad e impacto, que se les haya asignado.
Analista de Pruebas	Evaluación de la ejecución de pruebas: La evaluación de la ejecución de pruebas permite determinar si los criterios de completitud de los casos de prueba se han alcanzado satisfactoriamente, si se han encontrado issues nuevos o reincidentes o se han detenido casos de prueba que se estaban ejecutando debido a problemas externos o a issues que paralizan el funcionamiento; determinando así, si hubo una terminación normal o anormal. Una terminación normal, es aquella donde los casos de prueba dispuestos para un ciclo en particular, pudieron ser ejecutados. Una terminación anormal, es donde varios casos de prueba, no se pudieron completar o cuando hubo una falla del sistema y se debió detener la ejecución de las pruebas debido a un issue stopper. En este caso, toda la

	ejecución de las pruebas se debe volver a realizar y no será considerada como ciclo normal de pruebas.
Equipo de Desarrollo	Implementar la solución para cada issue reportado por el Analista de Pruebas.
Analista de Pruebas	Verificar solución de issues (también llamado Test de Regresión) esto independientemente de las regresiones particulares que se hayan hecho para algunos casos de prueba como se indicó anteriormente: Una vez implementada la solución de un issue, se deben realizar la evaluación y la ejecución de los casos de prueba respectivos para verificar lo siguiente: a. Que la solución del issue ha sido incorporada correctamente. b. Que no se generaron issues adicionales después de implementar la solución. Hasta que la solución de todos los issues no sea verificada, no se considerará finalizado.
ACTUAR	
Coordinador de proyectos / Analista de Pruebas	Ejecutar casos de prueba con issues reincidentes: Si existen issues reincidentes, es decir issues que han sido reportados durante varios ciclos de prueba, el equipo de prueba deberá elaborar un informe en el cual se especifique el alcance del issue y el impacto que tiene en la aplicación, en caso de que el issue sea reincidente durante más de 2 ciclos se levantará una acción correctiva y será escalado al analista líder del proyecto.
Analista de Pruebas	Identificar y describir nuevos casos de prueba: Si durante la etapa de ejecución se llega a la conclusión que existe un escenario de la aplicación, que no se había identificado, el cual aporta un nuevo caso de prueba, este se debe adicionar al gestor de pruebas, e identificar los atributos correspondientes, (ver Etapa Diseño de casos de prueba) para ese nuevo caso.

7.1.3.7 Salidas.

Cliente	Producto	Requisitos
- Equipo de desarrollo. - Analista de Pruebas. - Analista de Pruebas / Analista líder del Proyecto. - Etapa de Evaluación de pruebas.	Reporte de Issues	Debe contener cada uno de los issues encontrados donde se especifique: • Prioridad. • Impacto. • Severidad. • Resumen del issue. • Descripción. • Pasos para reproducirlo. • Archivos adicionales.
- Analista líder del Proyecto. - Comité de cambios.	Informe de Avance	Debe contener un reporte con el avance del proyecto y los problemas encontrados en la etapa en forma diaria.
- Ejecutor de Pruebas. - Analista líder del Proyecto. - Etapa de Evaluación de pruebas.	Resultados de ejecución de casos de prueba (componentes)	Cada caso de prueba ejecutado debe tener: • Código del caso de prueba. • Nombre del caso de prueba. • Resultado esperado. • Resultado Obtenido. • Estado de ejecución. (Pasó, Falló, No se pudo ejecutar). • Ciclo correspondientes. • Issues Asociados. • Fecha de ejecución. • Analista de Pruebas.
- Ejecutor de Pruebas.		Cada caso de prueba

- Analista líder del Proyecto. - Etapa de Evaluación de pruebas.	Resultados de ejecución de casos de prueba (aplicativo final)	ejecutado debe tener: • Código del caso de prueba. • Nombre del caso de prueba. • Resultado esperado. • Resultado Obtenido. • Estado de ejecución. (Pasó, Falló, No se pudo ejecutar). • Ciclo correspondientes. • Issues Asociados. • Fecha de ejecución. • Analista de Pruebas.

7.1.3.8 Documentos y Recursos Asociados.

• Instructivo de pruebas del sistema.
• Resultados de Ejecución.
• BugTracker: Aplicación para control y seguimiento de issues.

7.1.3.9 Determinación del Avance.

Las pruebas del sistema estarán completas en un 100% cuando todos los casos de prueba se hayan alcanzado y todos los issues reportados hayan sido solucionados y no se encuentren issues reincidentes o nuevos. Para obtener un valor de porcentaje más exacto, se recomendaría dividir el total del 100% entre la cantidad de casos de prueba que se tienen y de esta manera obtener un valor para cada caso individualmente.

7.1.3.10 Definición de Artefactos.

7.1.3.10.1 Descripción de Casos de Prueba (componentes y aplicativo final).

Documento donde se definen los casos de prueba que se van a ejecutar tanto para los componentes como para el aplicativo final que viene después de la integración de los componentes.

7.1.3.10.2 Especificación de Componentes (reutilizados o desarrollados).

Este documento debe contener la definición detallada de las especificaciones concretas de cada uno de los componentes utilizados, debe diferenciar entre los componentes que fueron desarrollados de aquellos que son reutilizados, las pruebas unitarias realizadas sobre cada uno de ellos (si se tiene la información en el caso de los reutilizados) y el requerimiento respectivo al cual le dieron solución.

7.1.3.10.4 Documento de Cambios realizados sobre los componentes.

Documento que contiene los cambios realizados en caso tal de que los componentes utilizados hubieran sido modificados. Por más mínima que sea la actualización debe haber sido reportada en este documento.

7.1.3.10.4 Versión ejecutable de cada uno de los Componentes y del Aplicativo Final.

Versión ejecutable de los componentes sobre los cuales se van a realizar las pruebas y posteriormente integrados para cumplir con la funcionalidad final del aplicativo.

7.1.3.10.5 Plan de administración de recursos y mejoras en el desarrollo.

Este artefacto define los posibles cambios o mejoras que se le pueden realizar a cada uno de los componentes. Este documento es importante ya que al leer estas posibles sugerencias al desarrollo puede ser posible en ocasiones descifrar algunas

debilidades de los componentes utilizados que seguramente serán muy útiles al momento de la ejecución de los casos de prueba.

7.1.3.10.6 Documentación Técnica.

Manual de instalación y configuración, Manual de mantenimiento y soporte, Manual de administración y seguridad, Manual de uso y ayudas en línea Modelo y diccionario de datos refinado. Estos manuales pueden ser opcionales.

7.1.3.10.7 Resultados de Ejecución de Casos de Prueba (componentes).

Representa los resultados obtenidos al probar cada uno de los componentes.

7.1.3.10.8 Resultados de Ejecución de Casos de Prueba (aplicativo final).

Representa los resultados obtenidos al probar la aplicación final (después de la integración de los componentes). Este resultado debe coincidir con el objetivo principal propuesto por el aplicativo porque no necesariamente por haber certificado todos los componentes que componen el aplicativo esto se traduce en un correcto funcionamiento después de la integración de componentes.

7.1.3.10.9 Reporte de Issues.

Es un reporte sobre los issues encontrados cuando se probó la aplicación y que se envía a los desarrolladores para que los corrijan.

7.1.3.10.10 Informe de Avance.

Informe con los avances que se han realizado en el proceso de pruebas funcionales.

7.1.4 Fase de Integración de Componentes.

7.1.4.1 Descripción.

La integración de componentes o elementos de software no es un concepto nuevo. Desde hace muchos años el concepto de integración de código de dos o más componentes ha sido crítico en el éxito de las organizaciones de desarrollo lo que ha llevado a la integración de software a límites mucho más extensos. La integración de este tipo de componentes ya sean reutilizables o desarrollados, por su misma naturaleza ofrece casi todas las ventajas para construir un aplicativo de calidad, aplicativos que estén a la medida del cliente mediante la integración de elementos que más se adaptan a sus necesidades evitando en la medida de lo posible la modificación de las aplicaciones a integrar haciendo que los componentes que fueron identificados para el desarrollo interactúen entre sí de tal modo que conjuntamente logren cumplir con la funcionalidad propuesta del aplicativo especificando exactamente la manera como estos se complementan entre sí.

Sin embargo no son solo ventajas las que ofrece la integración de componentes, se debe mencionar además que este modo de desarrollo tiene también sus desventajas y propios riesgos. El punto crítico en la integración de componentes es precisamente esa, la integración, lograr integrar los componentes usados de manera tal que logren cumplir con la funcionalidad propuesta en ocasiones no es tarea sencilla, esto precisamente no se garantiza validando que cada uno de los componentes funcione individualmente como debiera, el resultado del trabajo conjunto de todos los componentes a pesar de que todos hayan sido probados por separado y funcionen como debieran, es inesperado y pueden presentarse incompatibilidades.

7.1.4.2 Etapa de la Metodología: Evaluación de Pruebas.

Esta etapa entrega un documento de certificación y el resumen de pruebas a partir de los registros de ejecución de las pruebas y el reporte de issues. La evaluación de la ejecución de pruebas funcionales permite determinar si los criterios de completitud de los casos de prueba han terminado a satisfacción o si se han encontrado issues o se han detenido casos de prueba que se estaban ejecutando, determinando así si hubo una terminación normal o anormal. Una terminación normal, es aquella donde todos

los casos de prueba fueron ejecutados y todos los datos son válidos. Una terminación anormal, es donde uno o varios casos de prueba, no se pudieron completar o cuando hubo una falla del sistema y se debió detener la ejecución de las pruebas. En este caso toda la ejecución de las pruebas se debe volver a realizar. De lo contrario si hubo una terminación normal, se debe evaluar que el aplicativo en su conjunto funcione como se especificó inicialmente, es decir se evalúa que el hecho de haber probado las funcionalidades con sus componentes por separado se vea reflejado en la integración final de los componentes que cumplan con la funcionalidad principal para la cual fue pensado el aplicativo en un primer lugar.

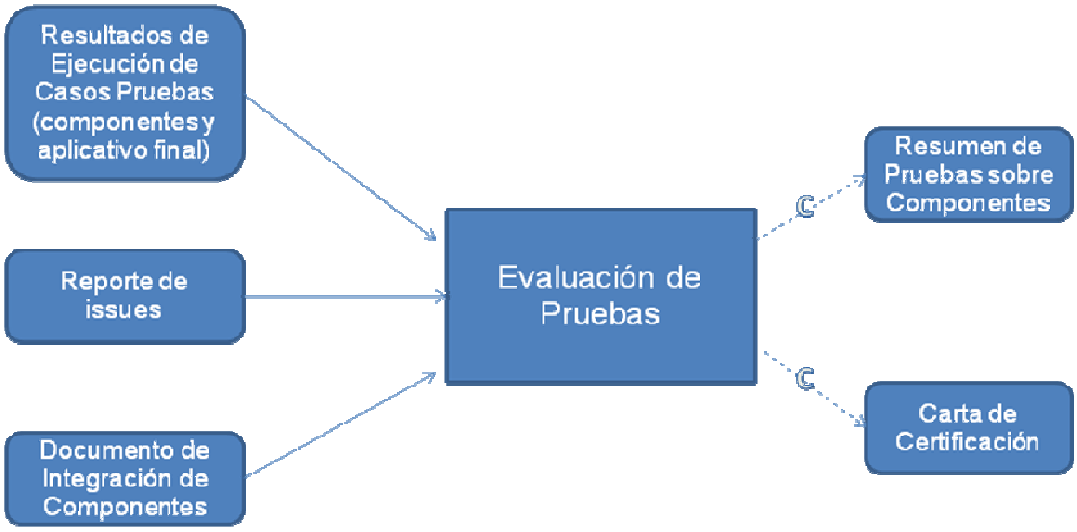

FIGURA 9: Etapa de evaluación de pruebas

Para que se genere mayor facilidad en el proceso de pruebas, los issues son reportados en una aplicación que permita el seguimiento de estos issues y permita obtener un registro e historial de los mismos. Esta actividad forma la metodología conocida como Change Control Management (CCM)[4].

Los reportes de issues se llevan a una matriz de control llamada “Bug Tracker” en donde se hace seguimiento del estado de cada uno de los problemas reportados.

[4] Metodología con la cual se administra todo el proceso de control de cambios y seguimiento de issues en el software.

Dependiendo de la gravedad del issue y los criterios de salida de la prueba, se debe dar prioridad a estos para una acción correctiva. Los de menor gravedad deberán ser documentados y manejarse según los procedimientos de informes y seguimiento del problema para tener la seguridad de una corrección posterior.
La idea es conformar un comité de cambios el cual se debe reunir periódicamente (máximo cada semana) con el fin de asignar los issues al desarrollador o grupo de desarrolladores, definir prioridad, severidad, nivel de impacto y establecer compromisos concretos de solución.
Se evalúan los resultados de las pruebas funcionales, analizando las incidencias recibidas y comprobando que se han llevado a cabo todos los casos de prueba establecidos en el Plan de Pruebas.
La evaluación consiste en:

- Comparar los resultados obtenidos con los esperados.
- Identificar el origen de cada problema para poder remitirlo a quien corresponda y determinar qué acciones o medidas correctivas se deben llevar a cabo para resolverlo de forma satisfactoria.
- Indicar qué pruebas se deben volver a realizar, o si será necesario contemplar nuevos casos de prueba.

Durante la etapa de evaluación de pruebas se obtienen las métricas de todo el proceso de pruebas del proyecto.

Las actividades de esta etapa son las siguientes:

- Análisis de los resultados.
- Evaluación de la cobertura de los requerimientos.
- Análisis de issues.
- Certificación de la aplicación.
- Creación del resumen de pruebas.

7.1.4.3 Objetivo.

Después de evaluar que la integración de los componentes cumple con la funcionalidad requerida del aplicativo, se procede a recopilar y organizar todas las métricas y estadísticas arrojadas durante la etapa de ejecución de casos de prueba, para emitir un certificado de calidad que garantice que el producto se encuentra conforme a las especificaciones del usuario y con la ausencia máxima o total de issues.

7.1.4.4 Cubrimiento.

Se inicia al finalizar la ejecución de los casos de prueba y finaliza cuando se emita el certificado de la aplicación. Llamada también Carta de Entrega. Este documento también es aportado por este procedimiento propuesto.

7.1.4.5 Entradas.

Proveedor	Insumo	Requisitos
Etapa de Ejecución de Pruebas.	Resultados de ejecución de casos de prueba (componentes y aplicativo final) Reporte de issues	Ver etapa de ejecución de Pruebas
- Equipo de desarrollo.	Documento de integración de componentes.	Documento con la especificación de la interacción de los componentes utilizados en el desarrollo. Debe contener: • Nombre del componente • Relación con otros componentes. • Resultado esperado. • Resultado Obtenido.

7.1.4.6 Actividades.

Responsable	Actividad
PLANEAR	
Analista de Pruebas.	Recolección de resultados arrojados en la etapa de Ejecución de Pruebas.
HACER	
Analista de Pruebas	Análisis de los resultados: Es el análisis de las diferencias entre los resultados esperados y los resultados obtenidos, se documenta toda la etapa de ejecución de casos de prueba y los resultados obtenidos.
	Evaluación de la cobertura de los requisitos: El objetivo de esta actividad, es el registro de medidas con el fin de determinar cuántos y cuáles requisitos se encuentran cubiertos, es decir, cuantos están asociados con casos de prueba para verificar que están correctamente implementados. También se mide la cantidad de issues por requisito y el impacto en los cambios. Las medidas claves están dadas por 2 factores: **Cobertura:** La cobertura identifica que tan completas han sido las pruebas funcionales del sistema, indicando el porcentaje de casos de prueba que fueron ejecutados, para determinar qué porción de requisitos fueron cubiertos. **Calidad:** Identifica si las pruebas realizadas han alcanzado los criterios de completitud definidos en los casos de prueba. De esta manera, se puede saber que tan coherente es el sistema y si en verdad cumple con los propósitos consignados en los requisitos. Ejecución del aplicativo final: Después de haber probado satisfactoriamente cada uno de los componentes en la etapa de ejecución de casos de prueba, se procede a probar la integración de estos de tal modo que conjuntamente cumplan con el objetivo principal

	que tenía propuesto el aplicativo. Lo anterior se debe realizar ya que las pruebas individuales realizadas para cada funcionalidad sobre cada uno de los componentes no garantiza que a pesar de que estos hubieran pasado las pruebas al trabajar conjuntamente no presenten ningún tipo de incompatibilidad, por tanto se hace necesario que después de la integración de componentes cumplan con la funcionalidad principal del aplicativo. Estas pruebas realizadas deben quedar registradas en el documento Resultado de ejecución del aplicativo final.
	Análisis de issues: En esta actividad se analizan, describen y miden todos los issues detectados. El reporte de medida de un issue está dado por tres factores: **Densidad del issue:** Muestra el número de issues en una categorización dada, ejemplo, por prioridad, severidad, módulos. **Antigüedad del issue:** Los reportes de antigüedad del issue demuestran cuánto tiempo un issue se ha encontrado en un estado particular, por ejemplo, cuánto tiempo se ha encontrado en el estado abierto sin que se le haya tratado. **Tendencia del issue:** Muestra el número de issues en una categoría dada sobre un periodo de tiempo, por ejemplo, número de issues abiertos en 6 semanas.
Analista de Pruebas	Creación del resumen de pruebas: El resumen de pruebas es un reporte objetivo de todo el proceso de pruebas funcionales de un proyecto determinado, en él se consignan: • Los resultados de pruebas • Las medidas de la pruebas • Los reportes de medidas • Recomendaciones o acciones sugeridas

	Seguimiento del cronograma de actividades: Se verifica que se hayan cumplido con los tiempos y la ejecución de las actividades.
VERIFICAR	
Comité de cambios.	Revisar el resumen de pruebas: verificar que el resumen de pruebas esta correcto y cumple con los objetivos.
ACTUAR	
Analista de Pruebas.	Realizar las correcciones necesarias al documento resumen de pruebas.

7.1.4.7 Salidas.

Cliente	Producto	Requisitos
- Analista líder del Proyecto. - Equipo de desarrollo. - Comité de cambios.	Resumen de Pruebas sobre Componentes y Aplicativo Final.	El documento debe contener: •Los resultados de pruebas sobre componentes y aplicativo final. •Las medidas de las pruebas. • Los reportes de medidas. •Recomendaciones o acciones sugeridas.
- Analista líder del Proyecto. - Comité de cambios.	Carta de certificación.	Todos los issues debieron haber sido tratados y solucionados, toda la documentación de pruebas está consolidada y todas las estadísticas y métricas han sido obtenidas.

7.1.4.8 Documentos y Recursos Asociados.

Se cuenta con los siguientes documentos y recursos de apoyo:
• Instructivo de evaluación de pruebas.
• Reporte de resumen de pruebas o estadísticos.
• Documento de Trazabilidad clasificado según los Resultados de Pruebas.
• Bugtracker.

7.1.4.9 Determinación del Avance.

Al recolectar todos los datos estadísticos (métricas de calidad) del proceso de pruebas funcionales y haber generado el resumen de pruebas, se puede decir que se ha completado en un 100%.

7.1.4.10 Definición de Artefactos.

7.1.4.10.1 Resultados de Ejecución de Casos de Prueba (componentes y aplicativo final).

Representa los resultados obtenidos al probar la aplicación.

7.1.4.10.2 Reporte de Issues.

Es un reporte sobre los issues encontrados cuando se probó la aplicación y que se envía a los desarrolladores para que los corrijan.

7.1.4.10.3 Documento de Integración de Componentes.

Documento con la especificación de la interacción de los componentes utilizados en el desarrollo.

7.1.4.10.4 Resumen de Pruebas sobre Componentes y Aplicativo Final.

Resumen general con los resultados del proceso de pruebas sobre los componentes y el aplicativo final.

7.1.4.10.5 Carta de Certificación.

Carta de certificación donde se dice que las pruebas se realizaron y la aplicación está libre de issues.

7.2 Definición de Actores para la Metodología Propuesta.

A continuación se definirán los roles para la metodología propuesta, es decir los actores que se verán involucrados a lo largo de la implementación de la metodología de pruebas para los desarrollos de software basado en componentes. Se debe aclarar que en cuanto a los nombres que daremos a los actores que intervienen en la metodología se refiere se puede ser algo flexible, es decir lo que interesa realmente son los roles que estos actores ejecutan y es de libre escogencia los nombres que se le quiera dar a los actores por parte de quienes utilicen mi propuesta metodológica de pruebas.

7.2.1 Analista de Pruebas.

Es responsable de identificar las pruebas requeridas, monitorear el progreso de las pruebas y los resultados de ellas en cada ciclo. Este rol normalmente lleva la responsabilidad de representar apropiadamente las necesidades de los beneficiarios del proyecto que no tienen directa o indirectamente representación.

7.2.2 Analista Líder de Pruebas.

Tiene la responsabilidad de planear y ejecutar los esfuerzos de prueba. Este rol involucra la calidad y pruebas, además de planeación de recursos y gestión de los temas o aspectos a probar.

7.2.3 Diseñador de Pruebas.

Es el responsable de definir los procedimientos para las pruebas y garantizar su correcta implementación. Además de identificar las técnicas, herramientas y guías para implementar las pruebas requeridas.

7.2.4 Ejecutor de Pruebas.

Este rol es responsable de ejecutar las actividades básicas para las pruebas, las cuales involucran las pruebas correspondientes e informes de éstas.

7.3 Diagrama de Estados del Issue.

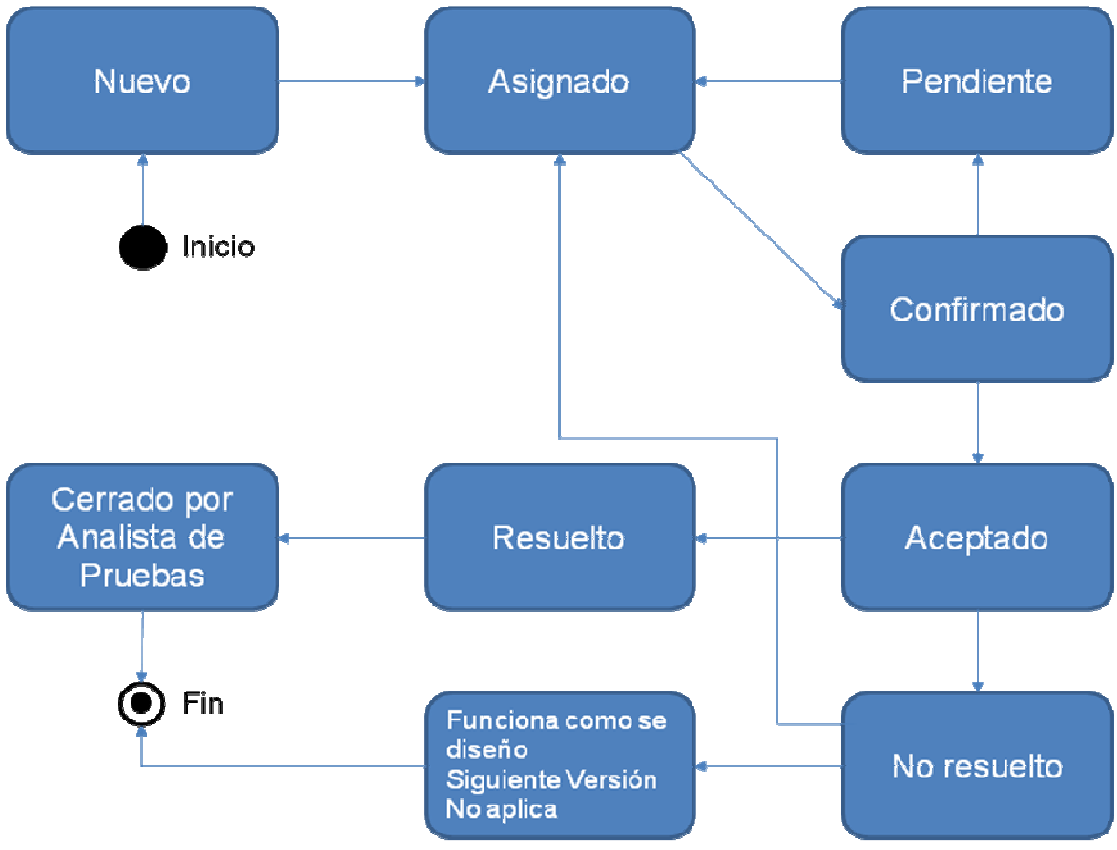

FIGURA 10: Diagrama estados del issue

7.3.1 Flujo de los Estados de los Issues

- **Nuevo:** Cuando el Ejecutor de Pruebas encuentra un nuevo issue, éste se ingresa en el bugtracker y se le coloca el estado de "nuevo". De este estado solo se puede pasar al estado "asignado"

- **Asignado:** Luego que el issue es encontrado, se asigna al desarrollador para su corrección. De este estado solo puede pasar al estado confirmado.

- **Confirmado:** Una vez el issue esté asignado, el desarrollador debe confirmar dicha asignación y dar conocimiento de esto. De este estado, solo puede pasar a los estados "aceptado" o "pendiente".

- **Aceptado:** El desarrollador, una vez confirmado el issue, acepta tener conocimiento del issue y comienza a trabajar en su corrección. De este estado, el ejecutor de pruebas puede pasar el issue ya sea al estado "resuelto" o al estado "no resuelto".

- **Pendiente:** El issue queda en estado pendiente cuando hace falta algún tipo de información o recurso para su corrección. De este estado, solo puede regresar el estado "asignado".

- **Resuelto:** Cuando el desarrollador informa que el issue ya fue solucionado, queda pendiente por revisión por parte del Ejecutor de Pruebas quien tiene el poder para pasar el estado del issue a "cerrado" en caso de haberse corregido el issue de forma efectiva.

- **No Resuelto:** Cuando el desarrollador informa que el issue no fue solucionado, entonces es responsabilidad del Ejecutor de Pruebas catalogar el estado en que quedará el issue de acuerdo a la explicación dada por el desarrollador del por qué no fue solucionado dicho issue. Habiendo dicho esto, el ejecutor puede poner el estado del issue en los siguientes estados: "Funciona como se diseñó", "Siguiente versión", "No aplica" "Re- Abierto" o en caso de querer asignar el issue nuevamente al desarrollador por que definitivamente el issue debe ser solucionado se puede pasar nuevamente al estado "asignado".

- **Funciona como se Diseñó:** Cuando la explicación del desarrollador para la no solución del issue reportado es que la funcionalidad evaluada no tiene precisamente un error sino que por el contrario funciona así debido a factores externos como por ejemplo el ambiente de pruebas que no refleja exactamente el ambiente de producción entonces se dice que el issue funciona como se diseñó. Luego, se informa en el comité de cambios que el issue no fue solucionado debido a la razón descrita.

- **Siguiente Versión:** Cuando la explicación del desarrollador para la no solución del issue reportado es que la funcionalidad defectuosa no afecta gravemente el aplicativo y por tanto será corregida en la próxima que se desarrolle del aplicativo, se dice que el issue es de siguiente versión. Luego, se informa en el comité de cambios que el issue no fue solucionado debido a la razón descrita.

- **No Aplica:** Cuando la explicación del desarrollador para la no solución del issue reportado es que la funcionalidad que el ejecutor de pruebas cree defectuosa en realidad no presenta ningún tipo de error, y esta explicación es lo realmente clara como para convencer al ejecutor de pruebas entonces este debe catalogar el issue en estado No aplica. Luego, se informa en el comité de cambios que el issue en realidad no era un error y que simplemente fue malinterpretado por el ejecutor de pruebas.

- **Cerrado por Analista de Pruebas:** El issue pasa al estado cerrado cuando en el proceso de pruebas o en el test de regresión no se reproduce el issue. Luego, se informa en el comité de cambios que el issue está totalmente solucionado.

Prioridad del Issue.

Prioridad: Rapidez con que se necesita resuelto un reporte.

- **Urgente:** La solución de software no puede ser usada o las pruebas no pueden continuar hasta que el issue sea resuelto.

- **Alto:** El issue debe de ser resuelto tan pronto como sea posible porque es imperativo para las actividades de pruebas. El issue no detiene la prueba, pero ésta es severamente afectada hasta que el issue sea arreglado.

- **Normal:** El issue debería ser resuelto en un curso normal de actividades de desarrollo.

- **Bajo:** El issue puede ser resuelto después de resolver issues más serios.

7.4 Informe de Incidentes.

El objetivo del informe de incidentes es documentar cada incidente (por ejemplo, una interrupción en las pruebas debido a un corte de electricidad, bloqueo del teclado, etc.) ocurrido en la prueba y que requiera una posterior investigación.
Este procedimiento propuesto para realizar pruebas de software contiene una plantilla de Informe de Incidentes.

7.4.1 Estructura Fijada en el Estándar.

- Identificador.
- Resumen del incidente.
- Descripción de datos objetivos (fecha/hora, entradas, resultados esperados, etc).
- Impacto que tendrá sobre las pruebas.

7.5 Buenas Prácticas de Pruebas Exitosas.

- En general, es una buena práctica contar con el manual de uso del aplicativo a probar, y en este caso, para pruebas de software basado en componentes sería aún mejor si se puede contar con el manual de uso propio de cada componente usado durante el desarrollo del aplicativo, de este modo se puede obtener una idea más concreta de las funcionalidades clave del aplicativo

- En la medida de lo posible se debe contar con un deck de pruebas elaborado por el desarrollador del aplicativo. El deck básicamente son pruebas unitarias realizadas por el desarrollador donde se reflejan las evidencias del smoke test sobre la prueba, es decir la evidencia sobre la funcionalidad principal del aplicativo, esto con el fin de que el ejecutor de pruebas quién es un poco más ajeno al aplicativo le sea más fácil entender la funcionalidad principal y a partir de allí elaborar los demás casos de prueba para el resto de funcionalidades a evaluar.

- Se deben evitar los casos desechables, es decir, los no documentados ni diseñados con cuidado, ya que suele ser necesario probar muchas veces el software y por tanto hay que tener claro qué funciona y qué no.

- Indagar siempre en caso de duda. Es muy normal que al principio de las pruebas a realizar se presenten inquietudes respecto al aplicativo. En este caso no dudar de preguntar para resolver dichas inquietudes, por lo general la persona más indicada para aclarar este tipo de preguntas es el usuario final quién conoce y sabe cómo debe funcionar correctamente el aplicativo.

- Nunca asumir o suponer durante el proceso de pruebas. Es mucho más recomendable preguntar en caso de dudar en lugar de hacer suposiciones.

- La experiencia parece indicar que donde hay un issue hay otros, es decir, la probabilidad de descubrir nuevos issues en una parte del software es proporcional al número de issues ya descubierto.

- Nunca pensar que una funcionalidad es muy obvia para ser probada. Las mejores pruebas realizadas sobre un aplicativo son aquellas que evalúan incluso las funcionalidades más obvias y no suponen ni asumen ningún resultado antes de ser realizadas.

- No se puede ser confiado del aplicativo, por qué por muy remoto que parezca, esa funcionalidad que pensamos que no debe fallar y por tanto no le prestamos la suficiente atención a las pruebas que le realizaremos puede presentar resultados inesperados, ninguna funcionalidad está exenta de este tipo de sucesos.

8. GLOSARIO DE TÉRMINOS.

- **Bugtracker:** Sistema de seguimiento de errores diseñado para asistir a los programadores y otras personas involucradas en el desarrollo y uso de sistemas informáticos en el seguimiento de los defectos de software.

- **Caso de prueba (test case):** Un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular.

- **Issue:** Resultado de un fallo o deficiencia durante el proceso de creación de software. Los defectos pueden suceder en cualquier etapa de la creación del software.

- **Fallo (failure):** La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados.

- **Pruebas (test):** Una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de algún aspecto. Proceso de ejecutar un programa con el fin de encontrar issues

- .**Release de la aplicación:** Distribución, pública o privada, de una versión inicial o actualizada de un producto software.

- **Stakeholders:** Son todas las personas que pueden afectar o verse afectadas por las actividades realizadas durante el proceso.

- **SmokeTest:** Son las pruebas iniciales que se deben realizar sobre un aplicativo con el fin de asegurar que este sea estable de este modo garantizando su mínima funcionalidad para poder llevar a cabo el proceso completo de pruebas.

- **Deck de Pruebas:** Son las pruebas unitarias que realiza el desarrollador sobre su aplicativo desarrollado con el fin de ilustrar el básico funcionamiento de este,

ayudando de este modo al analista de pruebas a entender de una mejor manera la funcionalidad principal del aplicativo en sí, y elaborar a partir de allí el resto de casos de prueba.

9. CONCLUSIONES.

- La metodología de testing para el desarrollo de software basado en componentes debe ser usada de manera estricta y juiciosa ya que como pudimos observar a lo largo de todo el presente escrito, el testing de un aplicativo es una labor que requiere de disciplina y sincronización entre los diferentes actores que participan durante el proceso de pruebas, de no seguirse el procedimiento y pasar pasos por alto se pone en riesgo el aseguramiento de la calidad funcional del software.

- Entre más temprana sea la detección de errores durante el proceso de pruebas mayor puede ser el ahorro en dinero para el proyecto, ya que es muchísimo más económico corregir un error durante el proceso de desarrollo que una vez que se encuentra en producción.

- La metodología de testing propuesta proporciona un enfoque práctico y cualitativo para la evaluación de la calidad funcional de los aplicativos que han sido desarrollados con el método de desarrollo de software basado en componentes.

- Las pruebas funcionales son un proceso para procurar encontrar discrepancias entre el programa y la especificación funcional más no son pruebas para evaluar la calidad del método de desarrollo que fue utilizado para crear el programa, en nuestro caso no son pruebas para evaluar el método de desarrollo de software basado en componentes, sino exclusivamente su funcionalidad.

- Por último, los métodos más usados de desarrollo son aquellos que cuentan con una metodología ya establecida para todas sus etapas del ciclo de desarrollo, incluyendo la metodología de testing a lo largo de dicho ciclo. Por tanto, es de esperarse que el desarrollo de software basado en componentes que aún sigue siendo un método relativamente nuevo de desarrollo de aplicativos siga tomando cada vez más fuerza entre el gremio desarrollador ya que al tener una metodología de testing, como la propuesta en este proyecto,

que se adapte exactamente al ciclo de desarrollo de este método dará un nuevo impulso a la expansión de este método.

10. BIBLIOGRAFÍA.

Component software, Beyond Object-Orientes Programming, Second Edition, by Clemens Szyperski with Dominik Gruntz and Stephan Murer, 2002.

Theory of software reliability based on components: Paper Software Engineering, 2001. ICSE 2001.
http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?reload=true&arnumber=919109

Domain-Driven Design Quickly, A Summary of Eric Evans`Domain-Driven Design, by Abel Avram & floyd Marinescu, 2007.

Desarrollo de Software Basado en Componentes: Paper Universidad de los Andes, Mérida – Venezuela.
http://ws-01.ula.ve/ingenieria/jonas/Productos/Publicaciones/Congresos/CAC03%20Desarrollo%20de%20componentes.pdf

Domain Driven Design, Step by Step Guide, by Casey Charlton
http://dddstepbystep.com/

Domain-Driven Design, Tackling Complexity in the Heart of Software, by Eric Evans

Proyecto de Grado: Procedimiento para realizar pruebas de software basado en RUP. Juan David Álvarez, Juan Felipe Zuluaga, Luis Fernando Macías. Universidad EAFIT 2008.

Component-based software engineering
http://en.wikipedia.org/wiki/Component-based_software_engineering

The Complete Guide to Software Testing - Second Edition, Hetzel, B., QED Information Sciences Inc, Massachusetts, 1988.

The Software Testing Life-Cycle. Andrew Ireland. School of Mathematical and Computer Sciences. Heriot-Watt University. Edinburgh.

Software Testing Best Practices. Ram Chillarege. Center for Software Engineering. IBM Research.

Aspectos de Calidad en el Desarrollo de Software Basado en Componentes: Paper Universidad de Málaga-Depto. Lenguajes y Ciencias de la Computación. http://www.lcc.uma.es/~av/Publicaciones/02/CalidadDSBC.pdf

Pruebas de Software: Paper CETIC (Comité Estatal de Tecnologías de Información y Comunicaciones), México. http://www.cetic.guerrero.gob.mx/pics/art/articles/113/file.TiposPruebasSoftware.pdf

Testing and Quality Assurance for Component-Based Software. Jerry Gao,H.-S. J. Tsao,Ye Wu, Estados Unidos de América.

11. ANEXOS.

Anexo 1 – Articulo: Aspectos de Calidad en el Desarrollo de Software Basado en Componentes.

Por: Manuel F. Bertoa, José M. Troya y Antonio Vallecillo.

El desarrollo de software basado en componentes se ha convertido actualmente en uno de los mecanismos más efectivos para la construcción de grandes sistemas y aplicaciones de software.

Una vez que la mayor parte de los aspectos funcionales de esta disciplina comienzan a estar bien definidos, la atención de la comunidad científica comienza a centrarse en los aspectos extrafuncionales y de calidad, como un paso hacia una verdadera ingeniería. En este artículo se discuten precisamente los aspectos de calidad relativos a los componentes software y a las aplicaciones que con ellos se construyen, con especial énfasis en los estándares internacionales que los definen y regulan, y en los problemas que se plantean en este tipo de entornos.

Calidad en el Desarrollo Software Basado en Componentes

La palabra calidad tiene varios significados, aunque dentro de la Ingeniería del Software podemos adoptar la definición de la norma ISO-8402, que luego se repite en otras (por ejemplo en ISO-14598): “La totalidad de aspectos y características de un producto o servicio que tienen que ver con su habilidad para satisfacer las necesidades declaradas o implícitas”.

Aunque la calidad es un objetivo importante para cualquier producto, no debemos olvidar que los productos, y también los productos software, se construyen para ser utilizados. Por tanto, el principal objetivo de un producto es satisfacer una necesidad (o varias) de un usuario y, por consiguiente, ofrecer al usuario algún beneficio por su utilización. Es decir, la calidad no es el objetivo último del producto, sino satisfacer las necesidades de un cliente.

También es importante señalar que esto implica que la calidad de un producto software no se puede referir únicamente a obtener un producto sin errores. La especificación de la calidad del software debe ser más detallada y exacta, y el camino para ello es la formalización de la calidad mediante un modelo de calidad que, como veremos a continuación, define las características de un producto que influyen a la hora de medir su calidad.

Características de Calidad en Componentes

En general, no existe un consenso a la hora de definir y clasificar las características de calidad que debe presentar un producto software. Por tanto, utilizaremos los estándares internacionales, fundamentalmente el ISO-9126. Aunque actualmente en proceso de revisión, ha sido el primero en definir y concretar este tipo de características.

Siguiendo su terminología, entendemos por característica de calidad de un producto software a un conjunto de propiedades mediante las cuales se evalúa y describe su calidad. Una característica se puede refinar en múltiples niveles de sub-características. Ejemplos de características son la funcionalidad, la fiabilidad o la facilidad de uso. A su vez, la característica funcionalidad se puede descomponer en sub-características como corrección, interoperatividad y seguridad, entre otras.

Llamaremos atributo a una propiedad de calidad a la que puede asignársele una métrica, entendiendo por métrica un procedimiento que examina un componente y produce un dato simple, un símbolo (p.e. Excelente, Sí, No) o un número. Hay que tener en cuenta que no todas las propiedades son medibles (p.e. la “demostrabilidad”).

Con todo esto, un modelo de calidad se define como “un conjunto de características y sub-características, junto con las relaciones que existen entre ellas”. Por supuesto, el modelo de calidad a utilizar va a depender del tipo de producto a probar. En este sentido, los estándares y propuestas existentes definen modelos de calidad generales, válidos para grandes familias de productos. Sin embargo, su alto grado de generalización y abstracción hacen difícil su aplicación a temas concretos. También, es importante tratar de “refinar” esos modelos generales, particularizándolos a casos concretos. Por ello, resultaría de gran interés y utilidad poder contar con un modelo de calidad particularizado para componentes. Desde nuestro punto de vista, dicho modelo de calidad ha de ser simple y concreto si queremos que pueda ser utilizado con ciertas garantías de éxito.

Uno de los principales objetivos de tal modelo de calidad para componentes es el de detectar los atributos que pueden describir los proveedores (externos o internos) de componentes COTS (Commercial-Off-The-Shelf) en la información que suministran acerca de los mismos y que, por tanto, permitirían facilitar su valoración y selección por parte de los diseñadores y desarrolladores de productos software.

Clasificación de los atributos de calidad

Para los componentes, y teniendo en cuenta como posible objetivo la definición de un modelo de calidad específico, es fundamental primero realizar una taxonomía, tratando de clasificar las características de calidad de acuerdo a su naturaleza y a distintos parámetros que intervienen en su medida. Se pueden realizar distintos tipos de clasificaciones.

1 En primer lugar, necesitamos discriminar entre aquellas características que tienen sentido para los componentes aislados (características locales) o bien deben ser valoradas a nivel de la arquitectura software de la aplicación (que llamaremos características globales). Por ejemplo, la "tolerancia a fallos" es una típica característica que va a depender de la arquitectura de la aplicación, mientras que la "madurez" es más propia de los componentes.

2 El instante en el cual una característica puede ser probada o medida, permite establecer otra clasificación de las características de un producto. Así, tenemos dos posibles categorías dependiendo de si la característica se puede probar en tiempo de ejecución (p.e. el rendimiento) o durante el ciclo de vida del producto (p.e. la mantenibilidad).

3 Como se menciona en los estándares de ISO, es importante identificar los usuarios a los que se dirige el modelo. En el ámbito del DSBC, los usuarios son fundamentalmente los arquitectos software, que necesitan evaluar los componentes COTS que van a formar parte de su aplicación. Así, las interfaces de los componentes objeto de nuestro estudio son más las interfaces programáticas (es decir, las APIs que definen las formas de acceder desde otros programas a los servicios que ofrecen los componentes), que las interfaces de usuario.

4 Para componentes COTS, es fundamental distinguir entre métricas internas y externas. Las internas miden los atributos internos del producto final o de los productos intermedios (p.e. la especificación o el código fuente) durante el diseño y la codificación. Las externas son las que realizan las mediciones en función del comportamiento del sistema durante las pruebas y la operación del componente. Por tanto, debido al carácter de caja negra de los componentes COTS, son las métricas externas las que interesan. Esto no quita que algunas de las características internas den una medida indirecta de las externas, e incluso que

puedan tener efectos sobre la arquitectura final. Así, por ejemplo, el tamaño de un componente puede ser importante a la hora de tener en cuenta los requisitos de espacio de la aplicación.

Anexo 2 – Articulo: Pruebas de Software.

Por: CETIC - Comité Estatal de Tecnologías de Información y Comunicaciones de México.

PRUEBAS DE SOFTWARE

La prueba del software es un elemento crítico para la garantía de la calidad del software. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Además, esta etapa implica:

- Verificar la interacción de componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
- Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

La prueba es un proceso que se enfoca sobre la lógica interna del software y las funciones externas. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. Un buen caso de prueba es aquel que tiene alta probabilidad de mostrar un error no descubierto hasta entonces. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

TECNICAS DE PRUEBA DE SOFTWARE

Una vez generado el código fuente, es necesario probar el software para descubrir (y corregir) la mayor cantidad de errores posible antes de entregarlo al cliente. Su objetivo es diseñar una serie de casos de prueba que tengan una alta probabilidad de encontrar errores. Aquí es donde entran las técnicas de prueba del software. Estas técnicas proporcionan directrices sistemáticas para pruebas de diseño que 1) comprueben la lógica interna y las interfaces de todo componente del software y 2) comprueben los dominios de entrada y salida del programa para descubrir errores en su función, comportamiento y desempeño.

Durante las etapas iniciales del proceso, el desarrollador realiza todas las pruebas. Sin embargo, a medida que avanza este proceso se irán incorporando especialistas en pruebas.
Las pruebas de caja negra son las que se aplican a la interfaz del software. Una prueba de este tipo examina algún aspecto funcional de un sistema que tiene poca relación con la estructura interna del software. La prueba de caja blanca del software se basa en un examen cercano al detalle procedimental. Se prueban rutas lógicas del software y la colaboración entre componentes, al proporcionar casos de prueba que ejerciten conjuntos específicos de condiciones, bucles o ambos.

Prueba de caja blanca:

Permiten examinar la estructura interna del programa. Se diseñan casos de prueba para examinarla lógica del programa. Es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar casos de prueba que garanticen que:

- Se ejercitan todos los caminos independientes de cada módulo.
- Se ejercitan todas las decisiones lógicas.
- Se ejecutan todos los bucles.
- Se ejecutan las estructuras de datos internas.

Prueba de caja negra:

Las pruebas se llevan a cabo sobre la interfaz del software, y es completamente indiferente el comportamiento interno y la estructura del programa. Los casos de prueba de la caja negra pretende demostrar que:

- Las funciones del software son operativas.
- La entrada se acepta de forma adecuada.
- Se produce una salida correcta.
- La integridad de la información externa se mantiene.

Se derivan conjuntos de condiciones de entrada que ejerciten completamente todos los requerimientos funcionales del programa.

La prueba de la caja negra intenta encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Los casos de prueba deben satisfacer los siguientes criterios:

- Reducir, en un coeficiente que es mayor que uno, el número de casos de prueba adicionales.
- Que digan algo sobre la presencia o ausencia de clases de errores.

ESTRATEGIAS DE PRUEBA DEL SOFTWARE

Una estrategia de prueba del software integra los métodos de diseño de caso de pruebas del software en una serie bien planeada de pasos que desembocará en la eficaz construcción del software. La estrategia proporciona un mapa que describe los pasos que se darán como parte de la prueba, indica cuándo se planean y cuándo se dan estos pasos, además de cuánto esfuerzo,
tiempo y recursos consumirán. Por tanto, en cualquier estrategia de prueba debe incorporar la planeación de pruebas, el diseño de casos de pruebas, la ejecución de pruebas y la recolección y evaluación de los datos resultantes.
Una estrategia de prueba del software debe ser lo suficientemente flexible como para promover un enfoque personalizado. Al mismo tiempo, debe ser lo adecuadamente rígido como para promover una planeación razonable y un seguimiento administrativo del avance del proyecto. Si se considera el proceso desde un punto de vista procedimental, la prueba consiste en una serie de pasos que implementan de manera secuencial. Estos pasos se describen a continuación:

Pruebas de unidad:

La prueba de unidad se centra en el módulo. Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de

descubrir errores dentro del ámbito del módulo. La prueba de unidad hace uso intensivo de las técnicas de prueba de caja blanca.

Prueba de integración:

El objetivo es coger los módulos probados en la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.
Hay dos formas de integración:

- Integración no incremental: Se combinan todos los módulos por anticipado y se prueba todo el programa en conjunto.
- Integración incremental: El programa se construye y se prueba en pequeños segmentos.

En la prueba de integración el foco de atención es el diseño y la construcción de la arquitectura del software.

Prueba de Arquitectura y Aplicaciones:

La arquitectura cliente/servidor representa un importante desafío para quienes prueban el software. La naturaleza distribuida de los entornos cliente/servidor, los aspectos de desempeño relacionados con el proceso de transacción, la posible presencia de varias plataformas de hardware diferentes, la complejidad de la comunicación en red, la necesidad de servir varios clientes desde una base de datos centralizada (o, en algunos casos, distribuida) y los requisitos de coordinación impuestos al servidor se combinan para que la prueba de las arquitecturas de software cliente/servidor resulte considerablemente más difícil que la prueba de aplicaciones independientes.
Pruebas de funcionalidad de la aplicación: la aplicación se prueba de manera independiente.

- Pruebas de servidor: se prueban funciones de coordinación y manejo de datos del servidor. También se considera el desempeño del servidor (tiempo de respuesta y procesamiento de los datos).
- Pruebas de base de datos: se prueba la exactitud e integridad de los datos almacenados en el servidor. Se examinan las transacciones que realizaron las

aplicaciones de cliente para asegurar que los datos se almacenan, actualizan y recuperan apropiadamente. También se prueba la función de archivado.

- Pruebas de transacción: se crea una serie de pruebas para asegurar que cada clase de transacciones se procesa de acuerdo con sus requisitos. Las pruebas se concentran en determinar si es correcto el procesamiento y en aspectos de desempeño.
- Pruebas de comunicaciones de red: con estas pruebas se verifica que la comunicación entre los nodos de la red ocurre de manera correcta y que el paso de mensajes, transacciones y el tráfico de la red relacionado se realiza sin errores. También es posible realizar pruebas de seguridad de la red como parte de estas pruebas.

Prueba del sistema:

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

- Prueba de validación: Proporciona una seguridad final de que el software satisface todos los requerimientos funcionales y de rendimiento. Además, valida los requerimientos establecidos comparándolos con el sistema que ha sido construido. Durante la validación se usan exclusivamente técnicas de prueba de caja negra.
- Prueba de recuperación: Fuerza un fallo del software y verifica que la recuperación se lleva a cabo apropiadamente.
- Prueba de seguridad: Verificar los mecanismos de protección.
- Prueba de resistencia: Enfrenta a los programas a situaciones anormales.
- Prueba de rendimiento: Prueba el rendimiento del software en tiempo de ejecución.
- Prueba de instalación: Se centra en asegurar que el sistema software desarrollado se puede instalar en diferentes configuraciones hardware y software y bajo condiciones excepciones, por ejemplo con espacio de disco insuficiente o continuas interrupciones.

Pruebas de regresión:

Las pruebas de regresión son una estrategia de prueba en la cual las pruebas que se han ejecutado anteriormente se vuelven a realizar en la nueva versión modificada, para asegurar la calidad después de añadir la nueva funcionalidad. El propósito de estas pruebas es asegurar que:

- Los defectos identificados en la ejecución anterior de la prueba se ha corregido.
- Los cambios realizados no han introducido nuevos defectos o reintroducido defectos anteriores.

La prueba de regresión puede implicar la re-ejecución de cualquier tipo de prueba. Normalmente, las pruebas de regresión se llevan a cabo durante cada iteración, ejecutando otra vez las pruebas de la iteración anterior.