FRAMEWORK PARA EL PROCESO DE TESTING

MARIA FERNANDA ESTRADA MARTÍNEZ

ESCUELA DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA Y SISTEMAS
UNIVERSIDAD EAFIT
MEDELLÍN
2010

FRAMEWORK PARA EL PROCESO DE TESTING

MARIA FERNANDA ESTRADA MARTÍNEZ

Proyecto de grado para optar al título de
Ingeniero de Sistemas

ASESOR
RAFAEL DAVID RINCÓN BERMÚDEZ
Profesor Departamento de Informática y Sistemas
Universidad EAFIT, Medellín

ESCUELA DE INGENIERÍA
DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS
UNIVERSIDAD EAFIT
MEDELLÍN
2010

Nota de Aceptación

_____

_____

_____

_____

_____

_____

_____

_____
Firma del presidente del jurado

_____
Firma del jurado

_____
Firma del jurado

Medellín, Noviembre 9 de 2010

AGRADECIMIENTOS

En primer lugar quiero agradecer a Rafael Rincón la oportunidad que me ha brindado para realizar este proyecto y aprender de él, por su apoyo y entusiasmo y por hacer de este proyecto una gran meta cumplida.

A mi familia porque siempre han creído en mí y en todo lo que puedo llegar a ser.

A todos mis maestros, por todo lo que se atrevieron a enseñarme.

A todos mis amigos de siempre por su apoyo en los momentos cruciales.

Y a los amigos de ahora por animarme constantemente en este largo proceso.

¡Gracias totales!

TABLA DE CONTENIDOS

TABLA DE FIGURAS

RESUMEN

Este documento sirve como base de conocimientos para la implementación de un proceso de testing que asegure la calidad de los productos de software. El centro del proyecto es el framework, que propone el proceso de testing desde el punto de vista teórico y práctico y además funciona como un repositorio de conocimientos para permitir a quienes estén interesados en este proceso acceder a información concisa para ser llevada a la realidad de los proyectos que incluyan testing de software, tanto en el ámbito empresarial como académico.

Por medio de la utilización de este framework se podrá tener una amplia visión de todo lo que el testing de software implica. Como se plantea en el transcurso de este documento, la idea central es tener una amplia gama de conocimientos que permitan el aprendizaje y la realización de los procesos involucrados en el testing de software, logrando así la uniformidad de la información, la claridad de la referenciación y el entendimiento de las posibles soluciones planteadas por diferentes expertos en la materia.

INTRODUCCIÓN

En la actualidad, el testing ha adquirido mayor relevancia en las organizaciones en lo que al desarrollo de software se refiere, ya que se ha hecho evidente su importancia por el ahorro que representa detectar tempranamente los errores del software.

"El National Institute of Standards & Technology (NIST) informa que los errores de software están costando a la economía de Estados Unidos aproximadamente $59,5 billones de dólares cada año, con más de la mitad de los gastos sufragados por los usuarios finales y el resto por los desarrolladores y proveedores, y que las mejoras en las pruebas pueden reducir este costo en cerca de un tercio, o $22,5 billones de dólares, pero aun así no se puede garantizar que se eliminen todos los errores del software"[1].

"De acuerdo con Gartner, Estados Unidos gasta US $59 billones de dólares en testing de software. De este total, US $13 billones de dólares se invierten en *outsourcing* en los países asiáticos. Dada la reputación de India, como país TI, un 70%, representado en US $9.1 billones de dólares, se obtuvo por servicios de TI en pruebas de software externalizadas. Además, se estima que la India necesitará cerca de 18.000 profesionales en testing cada año durante los próximos tres años para satisfacer la demanda del mercado de las pruebas de software"[2].

Los errores son caros y cuando más tarde se descubran mayor será el coste de la solución. La manera más eficaz de evitar que los errores se produzcan y, sobre todo que se propaguen es disponer de un proceso eficaz de calidad y pruebas, por lo que las pruebas suponen un coste en ocasiones muy elevado, pero que son imprescindibles y suponen un beneficio que se retorna de manera casi inmediata.

[1] The Economic Impacts of Inadequate Infrastructure for Software Testing - National Institute of Standards & Technology (NIST). Estudio para NIST realizado por Research Triangle Institute en Mayo de 2002.

[2] http://www.siliconindia.com/magazine_articles/Software_Testing_The_Next_Big_Employment_Wave-QSUG917969199.html, artículo por Pradeep Chennavajhula, Julio de 2010.

La complejidad propia del software hace que este sea susceptible de tener errores y un software que no pase por un proceso de testing puede causar importantes pérdidas a nivel económico, daños ambientales e incluso puede costar la vida de las personas; puede resultar en un software no usable (en términos de usabilidad o con múltiples defectos), teniendo como resultado procesos inestables o con resultados inútiles. Sin embargo, a pesar de la importancia del testing en la calidad del software, y de los beneficios que tiene, no existe en la actualidad un framework que reúna los procesos, las actividades, los roles involucrados, los formatos requeridos, etc. que cubra el aspecto teórico y práctico del testing, y que identifique los distintos entregables o productos de trabajo en cada fase, que permitan asegurar un proceso efectivo de testing.

Con este proyecto se pretende tener una base de conocimientos que sirva de referente para la implementación de un proceso de testing que asegure la calidad de los productos entregados. Este framework propone el proceso de testing desde el punto de vista teórico y práctico y además funciona como un repositorio de conocimientos para permitir a quienes estén interesados en este proceso acceder a información concisa para ser llevada a la realidad de los proyectos que incluyan testing de software, tanto en el ámbito empresarial como académico.

Por medio de la utilización de esta herramienta se podrá tener una amplia visión de todo lo que el testing de software implica. Como se plantea en el transcurso de este documento, la idea central es tener una amplia gama de conocimientos que permitan el aprendizaje y la realización de los procesos involucrados en el testing de software, logrando así la uniformidad de la información, la claridad de la referenciación y el entendimiento de las posibles soluciones planteadas por diferentes expertos en la materia.

OBJETIVOS

OBJETIVO GENERAL

Elaborar un framework que permita centralizar el conocimiento relacionado con el testing de software y que sirva de referencia al proceso de testing de software.

OBJETIVOS ESPECÍFICOS

1. Definir los conceptos fundamentales del testing de software.

2. Entender el proceso de testing dentro del contexto de la calidad del software.

3. Detallar los elementos críticos para la gestión del proceso de testing de software.

4. Identificar las etapas, procesos, roles, actividades, entregables, formatos y demás elementos relacionados con el proceso de testing de software.

5. Construir un framework para el proceso de testing a modo de mapa sensitivo, que constituya una guía de aprendizaje y ejecución del proceso de testing de software.

1. ¿QUÉ ES CALIDAD?

En el mundo actual en el que los clientes evalúan con una mayor rigurosidad los productos y/o servicios que reciben, y la satisfacción que éstos les provocan, ha aparecido una palabra clave que se ha convertido en un factor estratégico y diferenciador para las organizaciones en búsqueda de esa tan valorada y esperada satisfacción de las necesidades y expectativas de sus clientes: *la calidad*. Las organizaciones se han dado cuenta de los importantes beneficios que conlleva ofrecer productos y servicios de excelente calidad, reflejados en su productividad, costos, participación en el mercado, competitividad, motivación del personal y lealtad de los clientes.

El término calidad ha tomado tal importancia en estas últimas décadas que su aplicación está más allá de algún tipo de organización en particular o de algún producto específico; el concepto de calidad es una filosofía aplicable en todos los tipos de organizaciones y alcanza a tocar a cada persona perteneciente a la organización; así pues, que la calidad tiene que ver con todo y con todos, y es el factor que permite a las organizaciones asegurar su permanencia en el mercado.

Algunos reconocidos autores definen la calidad de la siguiente manera:

"La calidad no es otra cosa más que una serie de cuestionamientos hacia una mejora continua".

Edwards Deming

“La calidad es la adecuación para el uso, satisfaciendo las necesidades del cliente".

Dr. J. Juran

“La primera hipótesis errónea es que la calidad significa la bondad o elegancia. La palabra "calidad" se utiliza a menudo para indicar el valor relativo de algo en frases tales como "buena calidad", "mala calidad" y "calidad de vida", lo que significa algo diferente para cada persona. Por lo tanto, la calidad debe ser definida como "la conformidad con los requisitos", si hemos de gestionarla. En consecuencia, la no conformidad detectada es la falta de calidad, los problemas en la calidad se convierten en problemas de no conformidad, y la calidad se convierte en definible.”

Philip B. Crosby

1.1. CALIDAD DEL PRODUCTO SOFTWARE

La calidad que pueden alcanzar los productos software, y en general cualquier producto, está sometida a cómo se desarrolla cada una de las etapas de la vida del producto, partiendo por la definición de la idea del producto hasta la entrega y mantención del mismo. Es importante hacer énfasis en que la calidad se construye durante el proceso y no al final cuando el producto ya está construido, y que el testing se encarga de verificar y validar que el producto final satisface los requerimientos de usuarios y uso deseado. Es decir, el testing no agrega calidad, ésta se construye durante el proceso de desarrollo.

Así, la entrega de calidad a un producto considera actividades tales, como:

- Administración de los recursos, asegurando minimizar las diferencias entre los recursos presupuestados y los recursos realmente utilizados en las distintas etapas. Dichos recursos incluyen el staff, el equipamiento y tiempo de desarrollo.
- Uso de tecnología de Ingeniería de Software eficiente, considerando métodos de desarrollo y herramientas.
- Aplicación de técnicas o metodologías formales de ingeniería de software a lo largo de todo el proceso.

- Minimización de las variaciones entre los productos, disminuyendo las diferencias y defectos entre versiones.
- Testeo exhaustivo en diferentes etapas del desarrollo.
- Control de la documentación, tanto de apoyo al desarrollo como la entregada al usuario final, generada en cada etapa, y verificación de los posibles cambios y modificaciones que pudiera sufrir.
- Correcta mantención y servicios de post-venta.

Existen varios modelos reconocidos mundialmente que sirven de guía para la gestión de la calidad en relación con el software. Cada uno de ellos presenta un conjunto de buenas prácticas para el ciclo de vida del software, enfocándose en los procesos de gestión y desarrollo de proyectos.

Entre los más conocidos se encuentra el modelo CMMI®, que en una de sus últimas versiones tiene el modelo CMMI-DEV V1.2 (CMMI for Development – CMMI para Desarrollo), el cual sirve de apoyo a una organización o proyecto que desarrolle productos o servicios, enfocándose en la administración, medición y monitoreo de los procesos de desarrollo.

Por otro lado, se encuentra la norma ISO/IEC 15504, que es un modelo para la mejora y evaluación de los procesos de desarrollo y mantenimiento de sistemas y productos de software.

Además, el estándar ISO/IEC 9126 es un estándar internacional para la evaluación del software. Está dividido en cuatro partes las cuales dirigen, respectivamente, lo siguiente: modelo de calidad, métricas externas, métricas internas y métricas de calidad en uso.

El estándar ISO/IEC 29119 para Testing de Software, aún no publicado, tiene como objetivo proveer un estándar que capture vocabulario, procesos, documentación y técnicas para todo el ciclo de pruebas de software.

Finalmente, ISO/IEC 12207 establece un sistema para los procesos del ciclo de vida del software. Contiene procesos, actividades y tareas que se deben aplicar durante la adquisición de un software.

2. TESTING DE SOFTWARE

El testing de software es una investigación realizada para informar sobre la calidad del producto o servicio bajo prueba. El testing de software también proporciona una visión objetiva e independiente del software para conocer y comprender los riesgos cuando se desarrolla software.

El testing de software se aplica a lo largo del proceso de desarrollo de software, su objetivo es asegurar que el software cumpla con las especificaciones requeridas y eliminar los posibles defectos que pudiera tener. Este proceso es llevado a cabo por un equipo de testers.

El testing nunca podrá identificar todos los defectos en el software. En su lugar, realiza una comparación entre el estado y el comportamiento del producto frente a los principios o mecanismos por los cuales alguien podría reconocer que el software tiene un problema.

El testing de software es un proceso de verificación y validación del mismo. Tiene tres propósitos principales: verificar, validar y encontrar defectos:

- El proceso de verificación confirma que el software cumple con sus especificaciones técnicas. Una especificación es una descripción de una función en términos de un valor de salida medible dado un valor de entrada específico bajo condiciones específicas.
- El proceso de validación confirma que el software cumple con los requisitos del negocio y con el uso previsto por el usuario.
- Y finalmente, un defecto es una variación entre el resultado previsto y el real.

En un principio, la mayoría de empresas de desarrollo contaban con una etapa de pruebas demasiado informal, pero en la actualidad el testing de software se ha convertido en una de las etapas más críticas del ciclo de vida del desarrollo de software y esto se ha traducido en el origen de diversas metodologías.

En la actualidad el testing se hace más complicado ya que debe hacer frente a una gran cantidad de metodologías de desarrollo, lenguajes de programación, sistemas operativos, hardware etc.

Para las pruebas del producto lo recomendable es que el software se mantenga en un ambiente aislado o separado del ambiente de desarrollo o de producción: lo ideal es preparar un ambiente de pruebas lo más parecido a los ambientes que existen en producción para asegurar su correcto funcionamiento en esa futura etapa. Se debe considerar adquirir un equipo de pruebas especializado, conformado por “testers” o analistas de pruebas con experiencia, aunque en ocasiones, algunas empresas más informales utilizan a los futuros usuarios del sistema como testers, situación que puede traer una serie de problemas debido a la poca experiencia que pueden tener los usuarios en la detección de errores y la falta de formación en testing, además se obvian pruebas importantes como las pruebas de esfuerzo o “de stress”; también se dejan de lado las pruebas unitarias o pruebas modulares, las que deberían asegurar que cada módulo del sistema trabaje correctamente de manera independiente.

En el software la confianza es un elemento importante ya que ciertas fallas pueden tener consecuencias indeseables como pérdidas de dinero, de negocios, e incluso de vidas, dependiendo de qué tan crítico sea el dominio en el cual el software interactúa. Las pruebas le dan valor agregado a cada proyecto brindando confianza a los distintos actores.

Algunas definiciones de testing de software:

"El testing es una habilidad. Si bien esto puede ser una sorpresa para algunas personas es un hecho simple."

Graham Fewster

"Realizarle pruebas a un producto es un proceso de aprendizaje."

Brian Marick

“Una investigación técnica de un producto bajo prueba con el fin de brindar información relativa a la calidad del software, a los diferentes actores involucrados en un proyecto.”

Cem Kaner

"El testing puede probar la presencia de errores pero no la ausencia de ellos."

E. W. Dijkstra

"Software Testing es el proceso de evaluar un Sistema o Componente de un Sistema de forma manual o automática para verificar que satisface los requisitos esperados, o para identificar diferencias entre los resultados esperados y los reales."

IEEE

3. EVOLUCIÓN DEL TESTING

Dave Gelperin y William C. Hetzel describieron en 1988 la evolución de las pruebas de software como se puede observar en la Figura 1:

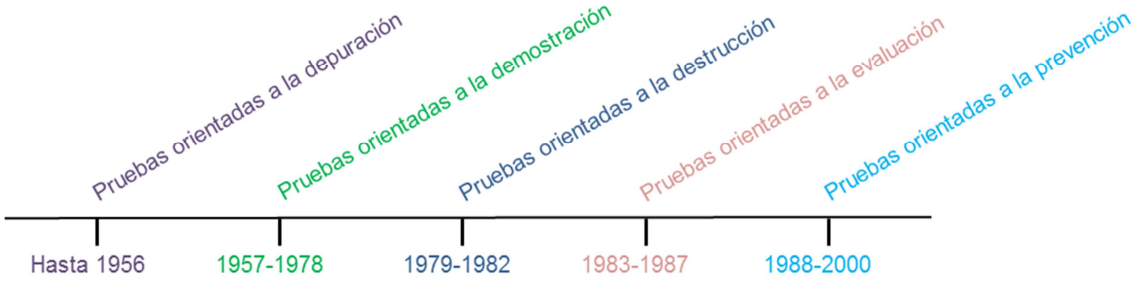


Figura 1. Evolución del Testing[3]**.**

Hasta 1956 fue el período orientado a la depuración, donde el testing era asociado a menudo con la depuración, no había una clara diferencia entre uno y otro.

De 1957 a 1978 fue el período orientado a la demostración, en la cual la depuración y el testing se distinguían uno del otro. En este período se empezó a mostrar que el software debía cumplir con los requerimientos.

Entre 1979 y 1982 comienza el período orientado a la destrucción, donde la meta principal era encontrar errores.

De 1983 a 1987 es el período orientado a la evaluación, donde la intención era que durante el ciclo de vida del software se hacía una evaluación del producto y una medición de la calidad.

[3] http://buildtesting.blogspot.com/2008/01/history-software-testing.html

Desde 1988 comenzó un período orientado a la prevención, donde las pruebas se hacen para demostrar que el software satisface las especificaciones, para detectar fallas y prevenir errores.

4. PROCESO DE TESTING

El testing es, a veces, incorrectamente pensado como una actividad a realizar después de que el desarrollo está terminado. Lo correcto sería pensar que las pruebas sean realizadas en cada etapa de desarrollo del producto.

Si dividimos el ciclo de vida del desarrollo del software en "Análisis de Requerimientos", "Diseño", "Programación/Construcción" y "Operación y Mantenimiento", entonces las pruebas deben acompañar a cada una de las fases anteriores. Si las pruebas están aisladas en una única fase al final del ciclo, los errores en el planteamiento del problema o en el diseño pueden generar costos exorbitantes. No sólo se debe corregir el error original, sino cambiar toda la estructura construida sobre él. Por lo tanto, las pruebas no deberían estar aisladas como una actividad de control. Más bien, las pruebas deben estar presentes en todo el ciclo de desarrollo del software para llegar a un producto de calidad.

El Modelo en V ayuda a visualizar las actividades del proceso de desarrollo y la correspondencia con las actividades del proceso de pruebas durante el ciclo de vida del desarrollo del software. Este modelo representa la secuencia de pasos del desarrollo describiendo las actividades y resultados que se producen. El lado izquierdo representa la descomposición de los requerimientos y la creación de las especificaciones del sistema. El lado derecho representa la integración de las partes y su verificación. En la Figura 2 se puede observar el esquema de este modelo.

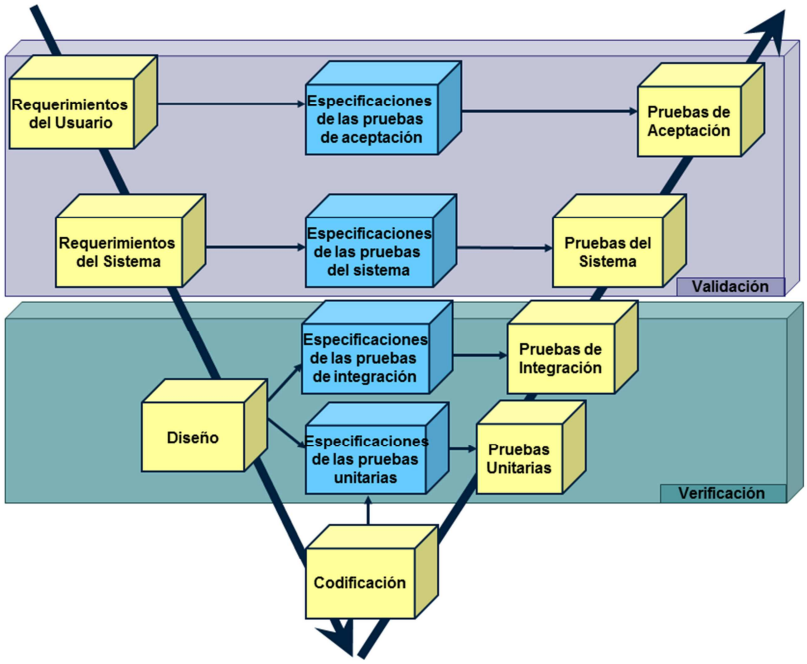


Figura 2. Modelo en V[4].

Se pueden identificar las siguientes etapas dentro del proceso de las pruebas:

1. **Estrategia y planeación de las pruebas:** En esta fase, el test manager prepara la estrategia de pruebas y el plan de pruebas.

   Las actividades que se realizan en esta fase, son:

   a) Identificar los modelos a utilizar.
   b) Definir los objetivos de las pruebas, el alcance de las pruebas, las fases y las actividades.
   c) Identificar los recursos a utilizar, tanto los físicos como los humanos.
   d) Definir los roles y las responsabilidades de cada individuo en la organización.
   e) Revisar los requisitos del negocio y los requisitos del sistema para identificar los elementos de las pruebas.

[4] http://www.tkomarek.com/category/embedded-software/

f) Definir el proceso y los procedimientos de las pruebas.
g) Establecer los estándares de configuración, la gestión de los defectos y los procedimientos de gestión del cambio.
h) Identificar las herramientas, técnicas y prácticas a utilizar en las pruebas.
i) Definir las dependencias entre las pruebas, el criterio de finalización de las pruebas y los criterios de aceptación del usuario.
j) Identificar los componentes que tienen un mayor riesgo y que pueden requerir niveles más elevados de esfuerzo en las pruebas.
k) Evaluar la viabilidad para automatizar algunas de las pruebas.

La información generada se documentará en el plan de pruebas.

2. **Configuración del ambiente de pruebas:** Cada configuración del ambiente de pruebas proporciona un ajuste apropiado y controlado en el cual se conducen las actividades requeridas de las pruebas y de la evaluación, proporcionando un ambiente conocido y controlado. Esto es de utilidad para asegurar que los resultados de estos esfuerzos sean exactos, válidos, y tengan una alta probabilidad de ser reproducidos. Una configuración del ambiente de pruebas bien controlada es un aspecto importante del análisis eficiente de fallos y resolución de defectos.

   Cada configuración del ambiente de prueba debe considerar varios aspectos incluyendo lo siguiente:

   a) Los requisitos básicos del hardware; por ejemplo, procesadores, almacenaje de memoria, almacenaje de disco duro, dispositivos del interfaz de entrada-salida.
   b) El ambiente de software subyacente básico; por ejemplo, sistema operativo y las herramientas básicas de productividad, tales como E-mail, sistema de calendario, entre otros.

c) Periféricos adicionales de hardware de entrada/salida especializados; por ejemplo, los exploradores de código de barras, impresoras, dispositivos con sensor, etcétera.
d) El software requerido para los periféricos de hardware de entrada/salida; por ejemplo, controladores de interfaces y dispositivos de entrada.
e) El sistema mínimo de herramientas del software necesarias para facilitar las pruebas, evaluación, y actividades de diagnóstico; por ejemplo, diagnóstico de la memoria, ejecución automatizada de pruebas, bases de datos y así sucesivamente.
f) Los ajustes requeridos de la configuración de las opciones del hardware y de software; por ejemplo, resolución del video, asignación de recursos, variables de entorno, entre otros.
g) Los materiales consumibles "pre-existentes" requeridos; por ejemplo, módems, puertos de recepción de la impresora y similares.

3. **Diseño de los casos de pruebas:** Los objetivos de las pruebas establecidos en el plan de pruebas deben fragmentarse en casos de prueba individuales hasta que sean definidos los objetivos individuales de las pruebas funcionales y estructurales.

   Para la creación y el uso de casos de prueba, se podrían seguir los siguientes pasos:

   a) Identificar los recursos que se van a utilizar en las pruebas.
   b) Identificar las condiciones que van a ser probadas.
   c) Priorizar las condiciones de las pruebas.
   d) Determinar los datos de prueba.
   e) Determinar los resultados correctos esperados.
   f) Crear los casos de prueba.
   g) Documentar las condiciones de las pruebas.
   h) Realizar las pruebas.

i) Verificar los resultados obtenidos.

Para la creación de los casos de prueba, el flujo a seguir es el siguiente:

a) Una vez se tengan los casos de uso y los requerimientos de los clientes, se debe proceder con su análisis.
b) Validar el entendimiento de los requerimientos con el cliente mediante reuniones.
c) Definir el enfoque y el número de test cases.
d) Identificar los escenarios de alto nivel para cada test case.
e) Enviar los escenarios para revisión.
f) Preparar la matriz de trazabilidad.
g) Desarrollar los procedimientos detallados de las pruebas para los test cases.
h) Revisar los test cases.
i) Actualizar los test cases de acuerdo con los resultados de la revisión.
j) Entregar al cliente para su aprobación final.

4. **Ejecución de las pruebas:** Este es el proceso central del testing. Aunque el proceso y el flujo de ejecución de pruebas son extremadamente dependientes de las herramientas, el ambiente y contexto del proyecto, hay algunas tareas y consideraciones a tener en cuenta cuando se ejecutan pruebas.

   Una vez que se haya determinado que la aplicación está en un estado apropiado para ser probada, las pruebas suelen comenzar con los test cases de mayor prioridad que, razonablemente, puedan llevarse a cabo en función del estado actual del proyecto y de la aplicación. Después de cada prueba, se prepara un breve resumen de lo sucedido durante la prueba y se agregan comentarios para futuras referencias. Estos comentarios pueden referirse a fallas del equipo, a excepciones de la aplicación y a los errores, problemas de red o de espacio en disco, entre otros.

Con cada nueva versión del producto se realizan alguna o todas las tareas asociadas a las pruebas, a esto se le llama un ciclo de prueba. Durante el ciclo de vida de un producto, sin importar cuál sea el proceso de desarrollo, se van generando distintas versiones de la aplicación. Las actividades de la prueba se realizan para una determinada versión del producto, sobre la cual se ejecutan las pruebas y se reportan los incidentes encontrados. Cada ciclo de prueba está asociado a una versión del producto a probar, cada nuevo ciclo de prueba implica una nueva versión de uno o más componentes del sistema. Uno de los principales desafíos es estimar cuantos ciclos de prueba se requieren, ya que no todas las versiones que genera desarrollo llegan a ser probadas por el equipo de pruebas, entre dos ciclos de prueba podrían existir más de dos versiones del producto generadas por el equipo de desarrollo.

Las 4 etapas podrían explicarse de forma más detallada a manera de flujo; una vez se tengan los test cases aprobados, los pasos a seguir son los detallados en la Figura 3.

Figura 3. Flujo de ejecución de pruebas.

(Adaptación propia)

4.1. Reporte de defectos y monitoreo: Los defectos son brechas entre un comportamiento esperado y el comportamiento actual de la aplicación.

Pueden generarse en errores cometidos por alguna persona ya sea en el código o en el diseño de una aplicación y pueden surgir en cualquier fase del ciclo de desarrollo de un software.

Los defectos son útiles para:

a) Corregir el software.
b) Predecir la calidad del software.
c) Recopilar estadísticas que se pueden usar para el análisis de expectativas de defectos en futuras aplicaciones.
d) Para tomar medidas preventivas como capacitaciones, establecimiento de métodos y procedimientos.
e) Para reducir el CoQ (Costo de la Calidad, CoQ = Esfuerzo Preventivo + Esfuerzo de Revisión + Esfuerzo de Corregir Fallas).
f) Para mejorar el proceso de desarrollo de software.

5. ¿POR QUÉ HACER TESTING?

El testing se necesita, porque:

- La complejidad propia del software lo hace susceptible de tener errores.
- Un software que no ha tenido un proceso de pruebas o que tiene un proceso de pruebas débil puede causar pérdidas en la vida real.
- Los errores de software son costosos, en términos de retrabajo, pérdida de productividad y daños a terceros.
- La carencia de un proceso de pruebas puede resultar en un software inutilizable (en términos de usabilidad o porque está lleno de errores, generando un software inestable o que no es útil).

Por otro lado, el testing aporta:

- Calidad durante todo el proceso.
- Disminución de costos.
- Reducción de riesgos.
- Optimización de recursos.
- Verificación del seguimiento de estándares.

6. TIPOS DE TESTING

6.1. DINÁMICO VS ESTÁTICO

6.1.1. DINÁMICO

Proporciona información en tiempo de ejecución sobre el funcionamiento de un software. Se logra a través de la instrumentación del programa de prueba.

En el testing dinámico se hacen pruebas al comportamiento de la aplicación. Esto es, se revisa la respuesta física del sistema a variables que no son constantes y cambian con el tiempo. En el testing dinámico el software debe ser ejecutado basándose en unos casos de prueba. El testing dinámico implica trabajar con el software, dada unas entradas iniciales se valida si la salida es la esperada.

Las pruebas unitarias, las pruebas de integración, las pruebas de sistema y las pruebas de aceptación son algunas de las metodologías de testing dinámico.

6.1.2. ESTÁTICO

Se lleva a cabo sin ejecutar el software para testear y pueden ser detectados los siguientes errores:

- Código inaccesible.
- Variables no usadas.
- Tipo de parámetro inconsistente.
- Funciones y procedimientos que no son ejecutados.
- Posible violación a los índices de los arreglos.

Los siguientes son los tipos de testing estático:

6.1.2.1. INSPECCIÓN MANUAL

Se examina el producto para comprobar el cumplimiento de normas específicas y la inspección contra el historial de posibles errores comunes.

6.1.2.2. REVISIÓN

El producto es analizado por un grupo de personas y se reevalúa o se reexamina buscando posibles arreglos o correcciones.

6.1.2.3. WALKTROUGH

Se realiza sobre el código desarrollado, donde el mismo es trazado manualmente para poder monitorear las variables del programa como una manera de analizar la lógica.

6.2. FUNCIONAL VS NO FUNCIONAL

6.2.1. PRUEBAS FUNCIONALES

Las pruebas funcionales son un proceso para procurar encontrar discrepancias entre el programa y la especificación funcional. Una especificación funcional es una descripción exacta del comportamiento del programa desde el punto de vista del usuario final.

Las pruebas funcionales son una actividad de caja negra. Para realizar una prueba funcional, la especificación se analiza para derivar un conjunto de casos de prueba utilizando las diferentes técnicas de caja negra.

6.2.2. PRUEBAS NO FUNCIONALES

En este tipo de pruebas se dejan de lado los aspectos funcionales del sistema. Se enfocan a tipos de prueba como las de performance.

Se distinguen los siguientes tipos de pruebas no funcionales:

6.2.2.1. PRUEBAS DE VOLUMEN

(Volume Testing) Las pruebas de volumen pertenecen al grupo de pruebas no funcionales. Las pruebas de volumen se refieren a pruebas hechas a una aplicación de software para un cierto volumen de datos. Este volumen puede ser en términos generales, el tamaño de la base de datos o también el tamaño de un archivo de interfaz. Por ejemplo, si se desea probar la aplicación con respecto al volumen de datos con un tamaño de base de datos específico, se debe llevar la base de datos a ese tamaño y luego ejecutar pruebas de rendimiento sobre la aplicación.

Otro ejemplo podría ser cuando es un requisito para la aplicación interactuar con una interfaz de archivo (puede ser cualquier archivo como *.dat* o *.xml*); esta interacción podría ser la lectura y/o escritura hacia o desde el archivo. Se crea un archivo de muestra del tamaño que se desee y luego se prueba el funcionamiento de la aplicación con ese archivo para probar el rendimiento.

6.2.2.2. PRUEBAS DE USABILIDAD

(Usability Testing) Según la norma ISO/IEC 9126-1, la usabilidad es la capacidad de un producto software de ser entendido, aprendido, usado y atractivo para el usuario, cuando es usado bajo unas condiciones específicas. Por lo tanto, las pruebas de usabilidad consisten en comprobar la adaptabilidad del sistema a las

necesidades de los usuarios, tanto para asegurar que se acomoda a su modo habitual de trabajo, como para determinar las facilidades que aporta al introducir datos en el sistema y obtener los resultados. Las pruebas de usabilidad están dentro de las pruebas de caja negra y en la categoría de las pruebas no funcionales.

El objetivo es observar a las personas usar el producto para descubrir errores y áreas de mejora. Normalmente se mide la respuesta del sujeto que hace las pruebas en 4 áreas: Eficiencia, exactitud, recordación y respuesta emocional. Los resultados de la primera prueba se usan como base y las siguientes se comparan con esta para ver los niveles de mejora:

- Desempeño: ¿Cuánto tiempo y cuántos pasos se requieren para que la persona complete una tarea básica?
- Exactitud: ¿Cuántos errores cometió la persona?
- Recordación: ¿Qué tanto recuerda la persona después de un tiempo sin usar la aplicación?
- Respuesta Emocional: ¿Cómo se sintió la persona con la tarea finalizada?, ¿Se siente confiada o estresada?, ¿El usuario le recomendaría este sistema a un amigo?

6.2.2.3. PRUEBAS DE SEGURIDAD

(Security Testing) Este tipo de pruebas son no funcionales y consisten en verificar los mecanismos de control de acceso al sistema para evitar alteraciones indebidas en los datos.

Las Pruebas de Seguridad pretenden medir la Confidencialidad, Integridad y Disponibilidad de los datos, desde la perspectiva del aplicativo, es decir, partiendo de identificar amenazas y riesgos desde el uso o interface de usuario final. Las pruebas de seguridad pretenden medir y cuantificar los riesgos a los cuales se ven

expuestos los aplicativos, tanto en la infraestructura interna como externa, valiéndose de la filosofía del Hacking ético. Las pruebas de seguridad simulan un ataque informático desde cualquier perspectiva (Internet, red interna, redes asociadas, acceso remoto, etc.) para establecer qué tan posible sería para un atacante, comprometer la seguridad de la información y validar la posible ocurrencia de un fraude.

Las pruebas de seguridad para aplicativos buscan garantizar:

- Que la información debe ser accesible únicamente a las personas autorizadas (Confidencialidad).
- La fiabilidad de la información (Integridad).
- Que la información esté disponible cuando se requiera y durante el tiempo que sea necesario (Disponibilidad).

Incluye las siguientes actividades:

- Autenticación.
- Autorización.
- Ataques Lógicos.
- Ataques del lado del cliente.
- Ejecución de código.
- Divulgación de información.

6.2.2.4. PRUEBAS DE RENDIMIENTO

(Performance Testing) El rendimiento del sistema suele ser evaluado en términos de tiempo de respuesta y tasas de rendimiento bajo diferentes condiciones de procesamiento y de configuración.

Los problemas de rendimiento son a menudo el resultado de una configuración del cliente o del servidor, inadecuada o un problema de construcción de la aplicación. La mejor estrategia para mejorar el rendimiento es un proceso de tres pasos. En primer lugar, ejecutar pruebas de rendimiento controladas que recopilen datos sobre el volumen, el estrés y las pruebas de carga. En segundo lugar, analizar los datos recogidos. En tercer lugar, examinar y resolver los problemas encontrados, por ejemplo si es en las consultas de la base de datos, se puede proporcionar almacenamiento temporal de datos en el cliente mientras la aplicación se está ejecutando. Luego de esto se ejecutan de nuevo las pruebas.

Las pruebas de rendimiento son el proceso para determinar la velocidad o la eficacia de un computador, red, programa o dispositivo. Este proceso puede incluir pruebas cuantitativas realizadas en un laboratorio, como la medición del tiempo de respuesta o el número de MIPS (millones de instrucciones por segundo) del sistema. Atributos cualitativos tales como fiabilidad, escalabilidad e interoperabilidad también pueden ser evaluados. Las pruebas de rendimiento se hacen a menudo en conjunto con pruebas de estrés.

Las pruebas de rendimiento pueden verificar que el sistema cumple con las especificaciones solicitadas por el fabricante o el distribuidor de un aplicativo. El proceso puede comparar dos o más dispositivos o programas en función de parámetros como la velocidad, velocidad de transmisión, ancho de banda, rendimiento, eficiencia o confiabilidad.

Las pruebas de rendimiento también se pueden utilizar como una ayuda diagnóstica en la localización de cuellos de botella. A menudo un sistema funcionará mucho mejor si un problema se resuelve en un sólo punto o en un único componente.

Una baja tasa de transferencia de datos puede ser inherente al hardware, pero también puede ser resultado de problemas relacionados con el software, tales como:

- Demasiadas aplicaciones corriendo al mismo tiempo.
- Un archivo dañado en un explorador Web.
- Un fallo en la seguridad.
- Un antivirus mal configurado.
- La existencia de malware activo en el disco duro.

Las pruebas de rendimiento efectivas pueden identificar rápidamente la naturaleza o ubicación de un problema de rendimiento relacionado con el software.

Las pruebas de rendimiento se dividen en los siguientes géneros:

6.2.2.4.1. PRUEBAS DE CARGA

(Load Testing) Consisten en comprobar el funcionamiento del sistema en el umbral límite de los recursos, sometiéndole a cargas masivas. El objetivo es establecer los puntos extremos en los cuales el sistema empieza a operar por debajo de los requisitos establecidos.

6.2.2.4.2. PRUEBAS DE ESTRÉS

(Stress Testing) Las pruebas de estrés tratan de la calidad de la aplicación en el ambiente. La idea es crear un ambiente más exigente del que la aplicación tendría bajo cargas normales de trabajo. Esta es la categoría más difícil y más compleja del testing ya que requiere un esfuerzo conjunto de todos los equipos.

El ambiente de pruebas se establece con varias estaciones de prueba o con usuarios “virtuales”. En cada uno, un script está siendo ejecutado en el sistema. Estos scripts se basan normalmente como una suite de pruebas de regresión. Cada vez más estaciones o usuarios se añaden simultáneamente en el sistema, hasta que el sistema llegue a su límite. El sistema se repara y la prueba de esfuerzo se repite hasta un nivel de estrés mayor, que es normalmente mucho más alto de lo que se espera en el sitio del cliente.

6.2.2.4.3. PRUEBAS DE RESISTENCIA

(Endurance Testing) Las pruebas de resistencia se hacen para determinar si la aplicación puede mantener una carga continua. Durante estas pruebas, se monitorea la utilización de la memoria para detectar las posibles fallas. El objetivo es asegurar que el rendimiento o los tiempos de respuesta después de un largo periodo de actividad son tan buenos como al inicio de la prueba o mejores.

6.2.2.4.4. PRUEBAS DE AISLAMIENTO

(Isolation Testing) Pruebas de los componentes individuales de forma aislada a partir de componentes circundantes para repetir la ejecución de una prueba que dio lugar a un problema en la aplicación. A menudo se utiliza para aislar y confirmar el fallo.

6.2.2.4.5. PRUEBAS DE CONFIGURACIÓN

(Configuration Testing) Las pruebas de configuración son otra variante de las pruebas de rendimiento tradicional. En lugar de hacer las pruebas de rendimiento desde la perspectiva de la carga, lo que se prueba son los efectos de los cambios de configuración en el rendimiento y el comportamiento de las aplicaciones.

6.2.2.4.6. PRUEBAS DE COMPORTAMIENTO CRÍTICO

(Spike Testing) Estas pruebas se realizan aumentando el número de usuarios para entender el comportamiento de la aplicación, para saber si el rendimiento se verá afectado, si la aplicación fallará o no, o si será capaz de manejar cambios drásticos en la carga.

6.2.2.5. PRUEBAS DE COMPATIBILIDAD

(Compatibility Testing) Programas tales como sistemas operativos, sistemas de gerencia de base de datos, y programas de conmutación de mensajes soportan una variedad de configuraciones de hardware, incluyendo varios tipos y números de dispositivos de entrada - salida y líneas de comunicaciones, o diversos tamaños de memoria. A menudo el número de configuraciones posibles es demasiado grande para probar cada uno, pero en lo posible se debe probar el programa con cada tipo de dispositivo de hardware y con la configuración mínima y máxima. Si el programa por sí mismo se puede configurar para omitir componentes, o si puede funcionar en diversas computadoras, cada configuración posible debe ser probada.

6.2.2.6. PRUEBAS DE INSTALACIÓN

(Installation Testing) Algunos tipos de sistemas de software tienen complicados procedimientos de instalación. Las pruebas de los procedimientos de instalación son una parte importante del proceso de pruebas del sistema. Al funcionar incorrectamente el programa de instalación podría generar una experiencia desagradable al usuario con el sistema. La primera experiencia de un usuario es cuando instala la aplicación. Si esta fase se realiza mal, entonces el usuario puede perder confianza en la validez de la aplicación o incluso se pueden generar errores en la aplicación debido a una mala instalación.

6.2.2.7. PRUEBAS DE COMUNICACIÓN

(Communication Testing) Pruebas no funcionales que determinan que las interfaces entre los componentes del sistema funcionan adecuadamente, tanto a través de dispositivos remotos, como locales. Así mismo, se realizan pruebas sobre las interfaces hombre - máquina.

6.2.2.8. PRUEBAS DE RECUPERACIÓN

(Recovery Testing) Pruebas no funcionales que fuerzan el fallo del software de muchas formas y verifican que la recuperación se lleva a cabo apropiadamente.

6.2.2.9. PRUEBAS DE ALMACENAMIENTO

(Storage Testing) Los programas tienen de vez en cuando objetivos de almacenamiento que indican, por ejemplo, la cantidad de memoria principal y secundaria que el programa usa y el tamaño de los archivos temporales. Se diseñan casos de prueba para demostrar que estos objetivos de utilización de recursos se satisfacen.

6.2.2.10. PRUEBAS DE DOCUMENTACIÓN

(Documentation Testing) Estas pruebas verifican la calidad de la documentación, por ejemplo las guías de usuario o los manuales de instalación. La documentación del usuario debe ser el tema de una inspección, comprobándola para saber si hay exactitud y claridad. Cualquiera de los ejemplos ilustrados en la documentación se debe probar.

6.2.2.11. PRUEBAS DE IMPLANTACIÓN

(Resilience Testing) El objetivo de las pruebas de implantación es comprobar el funcionamiento correcto del sistema integrando el hardware y software en el entorno de operación, y permitir al usuario que, desde el punto de vista de operación, realice la aceptación del sistema una vez instalado en su entorno real y con base en el cumplimiento de los requisitos no funcionales especificados.

7. ENFOQUES

Los métodos para pruebas de software se dividen tradicionalmente en caja blanca y caja negra. Estos dos enfoques se usan para describir el punto de vista usado para diseñar los test cases. Adicionalmente, la caja gris, es una combinación entre testing de caja negra y caja blanca.

7.1. CAJA BLANCA

(White Box) Son pruebas que se realizan a bajo nivel de los programas para corroborar la ejecución de las opciones del código a probar y la implementación de las tareas funcionales.

Se basa en identificar estructuras del software o del sistema, por ejemplo:

- Nivel de Componente: Estructura del código en sí, por ejemplo: declaraciones de variables.
- Nivel de Integración: Puede considerarse como una estructura de árbol (módulos dependientes de otros).
- Nivel de Sistema: Estructura de menú, procesos de negocio o estructura de página web.

7.2. CAJA NEGRA

(Black Box) Se ignora el funcionamiento interno del sistema o componente enfocándose sólo en la salida generada como reacción de ciertos datos de entrada.

Las siguientes son las técnicas de Caja Negra:

- Partición equivalente:
    - Técnica para llevar a cabo la cobertura de entradas y salidas.
    - Se utiliza para encontrar datos válidos e inválidos.
    - Las particiones pueden ser identificadas como salidas, valores internos, relación de tiempo (Por ejemplo, antes o después de un evento) y parámetros de interfaz (Testing de Integración).
    - Las pruebas pueden ser diseñadas para cubrir particiones.

- Análisis de Valor Límite:
    - El valor máximo y el valor mínimo son los valores límites. Un valor límite de una partición válida es un valor límite válido, el valor de una partición inválida es un valor límite inválido.
    - Las pruebas diseñadas cubren los valores límites válidos e inválidos.
    - Es aplicable para todos los niveles de testing.
    - Es considerada una extensión de la Partición Equivalente.
    - Es usada para las entradas, por ejemplo, una tabla de límites.
    - Puede ser utilizada para la prueba de selección de datos.

- Tabla de Decisiones:
    - Sirven para capturar los requerimientos del sistema que contienen las condiciones lógicas y documentar el diseño interno del sistema.
    - Pueden ser utilizadas para grabar reglas de negocio complejas que se crean en un sistema a aplicar. La especificación se analiza, y las condiciones y acciones del sistema se identifican.
    - Muchas veces, condiciones y acciones de entrada son estados que pueden ser Verdaderos o Falsos.
    - Cada columna corresponde a una regla de negocio y es una única combinación de condiciones y el resultado de la ejecución se asocia con esta regla.

    - La cobertura estándar es tener al menos una prueba por columna, lo que implica cubrir todas las combinaciones de pruebas.

- Transición de Estado:
    - Un sistema puede mostrar una respuesta diferente en función de las condiciones actuales o de su historial (su estado).
    - Permite al tester poder ver el software en estados, las transiciones en los estados, la entrada o eventos que provocan cambios de estado (transiciones) y las acciones que puedan resultar de estas transiciones. Los estados del sistema (u objeto) bajo prueba son separados, identificados y limitados en número.
    - Se muestran las relaciones entre: entradas y estados, y pueden destacarse posibles transiciones que son inválidas. Las pruebas pueden ser diseñadas para cubrir una secuencia típica de estados, para poder cubrirlos todos, para ejercer cada transición, ejercer secuencias específicas de transiciones o pruebas de transiciones inválidas.
    - Técnica muy útil para modelar un objeto de negocio que tenga estados específicos o pruebas de flujo.

El Testing Basado en Riesgos *(Risk Based Testing)* es un tipo de pruebas de software que prioriza las características y funciones que se van a probar basándose en los riesgos que representan, en función de su importancia, probabilidad de ocurrencia o impacto. En teoría, ya que existe un número ilimitado de posibles pruebas, cualquier conjunto de pruebas debe ser un subconjunto de todas las posibles pruebas existentes. Las técnicas de Análisis de Valor Límite y Transición de Estado buscan las áreas donde es más probable encontrar defectos.

7.3. CAJA GRIS

(Grey Box) Las pruebas de caja gris son una mezcla entre pruebas de caja negra y pruebas de caja blanca. No son pruebas de caja negra, porque el tester conoce parte del funcionamiento interno del software bajo prueba. En las pruebas de caja gris el tester usa un número limitado de test cases para probar el funcionamiento interno del software y el resto de pruebas las enfoca por medio de caja negra. El concepto es simple; si la persona conoce cómo trabaja el software por dentro, puede probarlo mejor desde afuera, ya que se toman mejores decisiones sobre cómo enfocar una prueba de determinado componente.

8. NIVELES DE TESTING

Las pruebas también se pueden agrupar según el lugar en dónde se ejecutan en el proceso de desarrollo de software, o por el nivel de especificidad de la prueba.

8.1. PRUEBAS UNITARIAS

(Unit Testing) En programación, las pruebas unitarias son un procedimiento utilizado para validar que las unidades individuales de código fuente están funcionando correctamente. Una unidad es la parte más pequeña de una aplicación que puede ser probada. En la programación procedimental puede ser un programa, una función, un procedimiento, etc., mientras que en la programación orientada a objetos, la unidad más pequeña es un método; que puede pertenecer a una superclase, a una clase abstracta o a una clase hija.

Lo ideal sería que cada caso de prueba sea independiente de los demás. Las pruebas unitarias son realizadas normalmente por los desarrolladores y no por los testers o por los usuarios finales.

8.2. PRUEBAS DE INTEGRACIÓN

(Integration Testing) El objetivo de las pruebas de integración es verificar el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente con el fin de comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes.

Los tipos fundamentales de integración son los siguientes:

- Integración Incremental: Se combina el siguiente componente que se debe probar con el conjunto de componentes que ya están probados y se va incrementando progresivamente el número de componentes a probar.
- Integración No Incremental: Se prueba cada componente por separado y posteriormente se integran todos de una vez realizando las pruebas pertinentes.

Tipos de pruebas de integración:

- Pruebas *Top-Down*: El primer componente que se desarrolla y prueba es el primero de la jerarquía. Los componentes de nivel más bajo se sustituyen por componentes auxiliares para simular a los componentes invocados. Una de las ventajas de aplicar esta estrategia es que las interfaces entre los distintos componentes se prueban en una fase temprana y con frecuencia.
- Pruebas *Bottom-Up*: En este caso se crean primero los componentes de más bajo nivel y se crean componentes conductores para simular a los componentes que los llaman. A continuación se desarrollan los componentes de más alto nivel y se prueban. Por último dichos componentes se combinan con el primero de la jerarquía. Este tipo de enfoque permite un desarrollo más en paralelo que el enfoque de "*top-down"*, pero presenta mayores dificultades a la hora de planificar y de gestionar.
- Estrategias combinadas: A menudo es útil aplicar las estrategias anteriores conjuntamente. De este modo, se desarrollan partes del sistema con un enfoque "*top-down*", mientras que los componentes más críticos en el nivel más bajo, se desarrollan siguiendo un enfoque "*bottom-up*". En este caso es necesaria una planificación cuidadosa y coordinada de modo que los componentes individuales se encuentren en el centro.

8.3. PRUEBAS DE HUMO

(Smoke Testing) De acuerdo con Roger Pressman, la prueba de humo (más conocida como Smoke Test) es un método de prueba que es comúnmente utilizada cuando se ha desarrollado un producto de software "empaquetado". Es diseñado como un mecanismo para productos críticos en tiempo, permitiendo que el equipo de software valore el proyecto sobre una base sólida antes de liberarlo a un proceso formal de pruebas. Estas pruebas que se ejecutan sobre un subconjunto de todos los test cases definidos inicialmente, que cubre la funcionalidad principal de un componente o sistema, para determinar que las funciones cruciales del programa funcionan, sin detenerse en los detalles.

Algunas de las actividades de la prueba de humo son las siguientes:

- Los componentes que han sido traducidos a código se integran en una aplicación.
- Se diseña una serie de pruebas para descubrir errores que impiden al programa realizar su función adecuada.
- En la prueba de humo es habitual que la aplicación se integre con otras aplicaciones y que se aplique una nueva prueba de humo al producto completo. La integración puede hacerse bien de forma descendente o ascendente.

8.4. PRUEBAS DEL SISTEMA

(System Testing) Las pruebas del sistema buscan discrepancias entre el programa y sus objetivos o requerimientos. El testing del sistema cae dentro del alcance de las pruebas de caja negra, y como estas, no requiere de un conocimiento interno del código de la aplicación.

Las pruebas del sistema prueban el sistema completamente para verificar que cumple con los requisitos y su propósito particular es comparar el sistema o el programa con sus objetivos originales (Requerimientos funcionales y no funcionales). Dado este propósito, se presentan dos implicaciones:

1. Las pruebas de sistema no se limitan a los sistemas. Si el producto es un programa, la prueba del sistema es el proceso de procurar demostrar cómo el programa, en su totalidad, resuelve sus objetivos o requerimientos.

2. Las pruebas de sistema, por definición, son imposibles si no están los requerimientos por escrito del producto.

Las pruebas del sistema tienen como objetivo ejercitar profundamente el sistema comprobando el funcionamiento del sistema de información globalmente.

Son pruebas sobre el sistema de información completo, y permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y técnicas se cumplen. Dan una visión muy similar a su comportamiento en el entorno de producción.

8.5. PRUEBAS DE REGRESIÓN

(Regression Testing) Las pruebas de regresión son cualquier tipo de pruebas de software que tratan de descubrir los errores de regresión. Los errores de regresión se producen cuando una funcionalidad del software que antes funcionaba como se deseaba, deja de funcionar o deja de trabajar de la forma como fue planeado con anterioridad. Normalmente se producen errores de regresión como una consecuencia no deseada de los cambios al programa.

Los métodos comunes de pruebas de regresión incluyen volver a ejecutar con anterioridad pruebas ya hechas y comprobar si los defectos previamente arreglados vuelven a surgir.

Estas pruebas se ejecutan mediante pruebas automatizadas cuando la aplicación lo permite y cuando la cantidad y complejidad de escenarios así lo amerita.

Tipos de regresión:

- Local: Cambios que introducen nuevos errores.
- Desenmascarada: Cambios que desenmascaran errores existentes con anterioridad.
- Remota: Un cambio en una parte rompe otra parte del programa.

Las pruebas de regresión pueden incluir:

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

8.6. PRUEBAS DE ACEPTACIÓN

(Acceptance Testing) El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento.

Las pruebas de aceptación son definidas por el usuario del sistema y preparadas por el equipo de desarrollo, aunque la ejecución y aprobación final corresponden al usuario.

La validación del sistema se consigue mediante la realización de pruebas de caja negra que demuestran la conformidad con los requisitos, y son realizadas por los usuarios mediante la ejecución de algunos escenarios de negocio. Estas pruebas están diseñadas para asegurar que se satisfacen todos los requisitos funcionales especificados por el usuario teniendo en cuenta también los requisitos no funcionales relacionados con el rendimiento, seguridad de acceso al sistema, a los datos y procesos, así como a los distintos recursos del sistema.

Cuando se construye software a medida para un cliente se lleva a cabo una serie de pruebas de aceptación para permitir que el cliente valide todos los requisitos. La mayoría de los desarrolladores de productos de software llevan a cabo un proceso denominado pruebas alfa y beta para descubrir errores que parezca que sólo el usuario final puede descubrir.

8.6.1. PRUEBA ALFA

Se lleva a cabo por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y problemas de uso. Las pruebas alfa se llevan a cabo en un entorno controlado.

8.6.2. PRUEBA BETA

Se llevan a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador. El cliente registra todos

los problemas que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador.

8.7. PRUEBAS ÁGILES

(Agile Testing) En los enfoque ágiles las pruebas son el centro de la metodología y, por lo tanto son ellas las que dirigen el proceso de desarrollo. Las metodologías ágiles plantean que el desarrollo no es un conjunto de fases en las que las pruebas son una fase más, sino que permiten que las pruebas y el desarrollo estén completamente integrados.

Hay muchos métodos de desarrollo ágil, la mayoría minimiza el riesgo mediante el desarrollo de software en cortos períodos de tiempo. El software desarrollado en una unidad de tiempo se conoce como una iteración, que puede durar de una a cuatro semanas. Cada iteración es un proyecto de software completo, que incluye la planificación, el análisis de requerimientos, el diseño, la codificación, las pruebas y la documentación. Una iteración no puede añadir funcionalidad suficiente como para justificar la liberación del producto en el mercado pero el objetivo es tener una versión disponible (sin errores) al final de cada iteración. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto.

Los métodos ágiles hacen hincapié en la comunicación cara a cara en vez de la comunicación escrita. Los equipos ágiles se encuentran en una única oficina abierta y como mínimo tienen programadores y "clientes" (clientes que definen el producto; pueden ser gerentes de producto, analistas de negocio, o los clientes). También se incluyen testers, diseñadores, escritores técnicos y gerentes.

Los métodos ágiles también hacen hincapié en el software como la principal medida de progreso. Adicional al tipo de comunicación, los métodos ágiles producen muy poca documentación escrita en relación con otros métodos.

Las metodologías ágiles no consideran las pruebas como un conjunto de niveles que hay que ir superando para alcanzar la validación final del sistema que se está desarrollando. Las metodologías ágiles presentan distintos enfoques del proceso de desarrollo que vienen determinados por los tipos de pruebas que se realizan.

9. FRAMEWORK

El propósito de esta sección es ofrecer una perspectiva en profundidad sobre el proceso y los procedimientos que debe seguir un equipo de testing en un proyecto de software.

El proceso de testing describe y detalla los procesos y los procedimientos que seguirá el equipo de pruebas, definiendo los roles y responsabilidades desde el momento que inicia la planeación del proyecto hasta la entrega del producto. Adicionalmente, se presentan todos los entregables que genera un equipo de testing durante el proceso de pruebas.

El énfasis se hace en pruebas manuales y automatizadas, y podría considerarse como una futura línea de trabajo el planteamiento para el resto de tipos de pruebas.

9.1. PROCESO DE TESTING

La figura 4 ilustra el diagrama del proceso de pruebas, proporcionando el flujo de las tareas que se deben completar.

En las secciones siguientes se describen las principales actividades del proceso de pruebas y los roles de quienes realizan las tareas asociadas con cada actividad. Para cada actividad se detallan los criterios de entrada y de salida y los entregables de la misma. Cada actividad será detallada en las siguientes secciones.

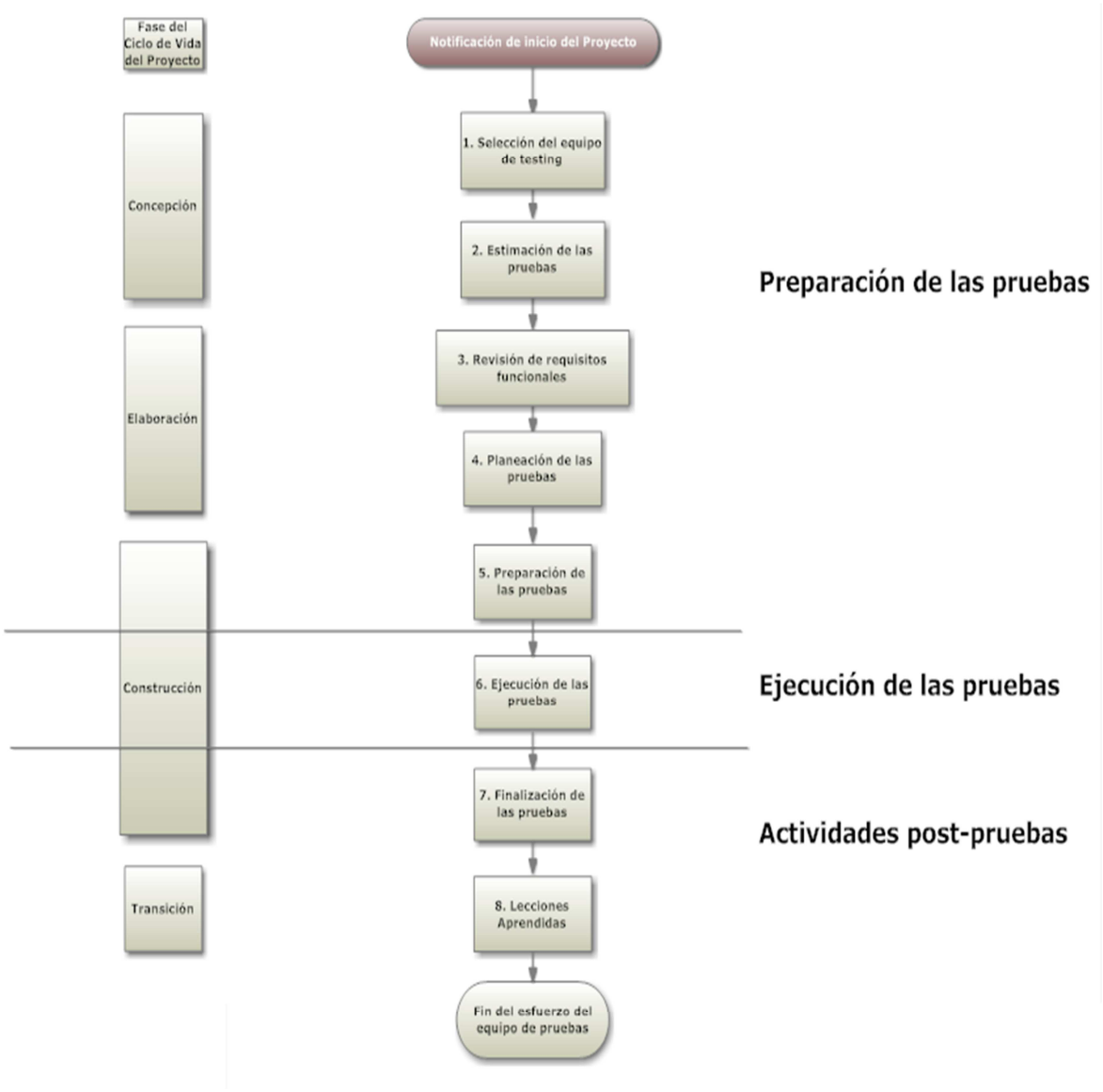


Figura 4. Proceso de pruebas.

(Adaptación propia)

9.2. SELECCIÓN DEL EQUIPO DE TESTING

Esta actividad se realiza al inicio del proyecto para seleccionar al personal adecuado según el tipo de proyecto, aunque también sirve como base cuando se va a contratar personal para el área de pruebas.

9.2.1. ROLES

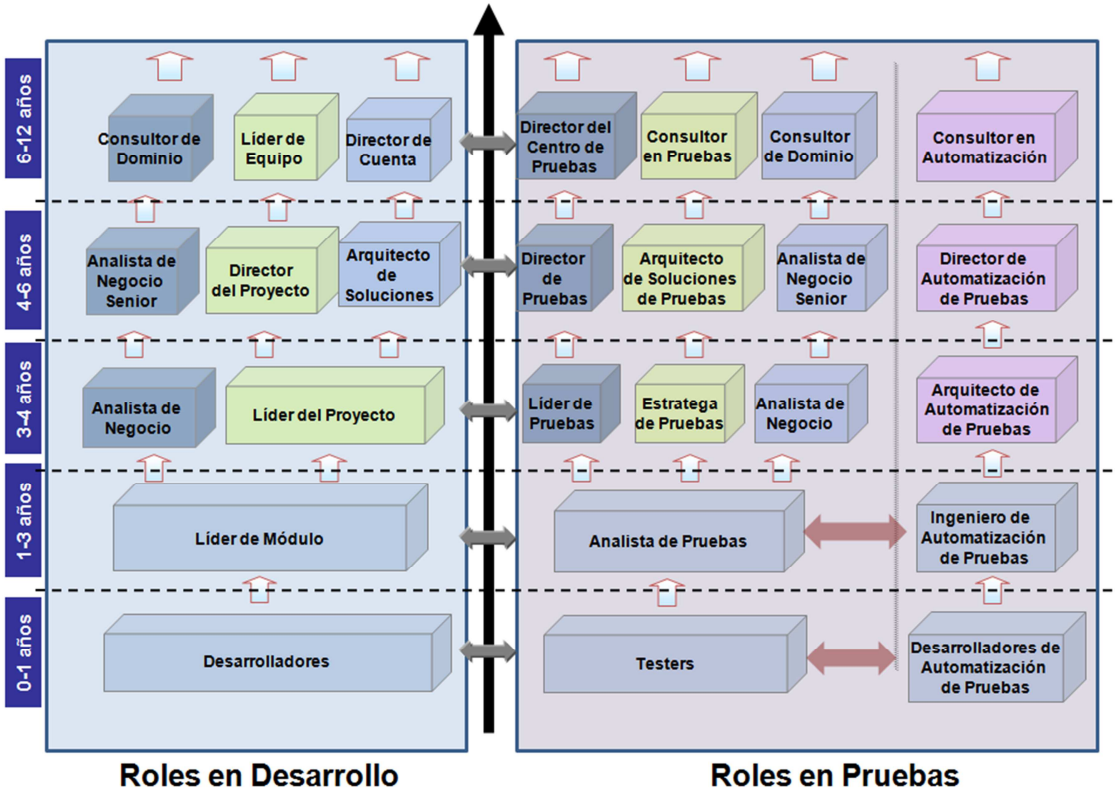


Figura 5. Flujo de roles para Desarrollo y Pruebas.[5]

En la Figura 5 se puede ver una posible clasificación de los roles por los que puede pasar una persona dentro de una organización de software, tanto en la parte de desarrollo como en la correspondiente a pruebas.

A continuación se describen los roles en el proceso de pruebas y sus principales funciones:

a. Ingenieros de Pruebas/Testers

✓ Ejecuta los test cases según el plan de pruebas y registra los defectos que encuentra.

[5] Tomado de Testing Induction - Day 01 – Introduction – TCS.

- ✓ Trabaja cercanamente con el Líder de Pruebas o el Analista de Pruebas para lograr los objetivos planeados.

b. Desarrolladores de Automatización de Pruebas

- ✓ Diseña, desarrolla e implementa software que hace pruebas sobre la aplicación bajo análisis.
- ✓ Es responsable del desarrollo, documentación y mantenimiento de las pruebas automatizadas.

c. Analista de Pruebas

- ✓ Desarrolla los casos de prueba, los scripts de prueba y los datos de prueba.
- ✓ Prepara la matriz de trazabilidad.
- ✓ Trabaja mano a mano con el equipo de pruebas para conseguir el mejor resultado.

d. Ingeniero de Automatización de Pruebas

- ✓ Prepara los diseños de las pruebas que se van a automatizar basándose en la arquitectura de automatización y el framework definido para el proyecto.
- ✓ Define los estándares y lineamientos de automatización.

e. Líder de Pruebas

- ✓ Desarrolla y gerencia los planes de prueba.
- ✓ Monitorea la ejecución de las pruebas y le hace seguimiento a los defectos encontrados.
- ✓ Establece el ambiente de pruebas, los procesos a seguir e implementa las metodologías de pruebas.
- ✓ Reporta el estado de las pruebas y es quien coordina a los usuarios finales, a los testers y al director de pruebas.

f. Estratega de Pruebas

- ✓ Diseña el framework de pruebas incluyendo las pruebas automatizadas.

- ✓ Revisa los entregables de las pruebas.
- ✓ Analiza las herramientas de pruebas y hace recomendaciones con respecto a ellas.
- ✓ Define la estrategia de pruebas.
- ✓ Define las métricas.

g. Analista de Negocio

- ✓ Prepara los escenarios de prueba de alto nivel basados en los escenarios y los requisitos del negocio.
- ✓ Analiza que los requisitos sean exactos y estén completos desde la perspectiva del negocio.
- ✓ Revisa los escenarios de prueba de alto nivel.

h. Arquitecto de Automatización de Pruebas

- ✓ Procura obtener una mezcla adecuada de herramientas, técnicas y personal para asegurar unas pruebas automatizadas exitosas.
- ✓ Se encarga del mejoramiento del ambiente de pruebas para automatizar.

i. Director de Pruebas

- ✓ Desarrolla la estrategia de pruebas.
- ✓ Se encarga de la planeación, de la estimación del esfuerzo, de la gestión de recursos y del entrenamiento.
- ✓ Gestiona las actividades del equipo de pruebas.
- ✓ Se encarga de establecer, monitorear y mejorar los procesos.
- ✓ Identifica los riesgos y formula estrategias de mitigación.

j. Arquitecto de Soluciones de Pruebas

- ✓ Evalúa el proceso de pruebas y recomienda el plan de acción de mejoramiento.
- ✓ Evalúa la viabilidad de hacer pruebas automatizadas.

- ✓ Prepara el plan de evaluación y coordina con las partes interesadas del proyecto.

k. Analista de Negocio Senior

- ✓ Su función principal es comprender el negocio y los retos de la organización del cliente y de la industria.
- ✓ Identifica fortalezas y debilidades de la organización del cliente y sugiere áreas de mejora.
- ✓ Revisa los requisitos, especificaciones, procesos de negocio y hace recomendaciones en donde sea necesario.

l. Director de Automatización de Pruebas

- ✓ Se encarga del entrenamiento y desarrollo de competencias del personal.
- ✓ Se encarga de los estándares de los procesos y las metodologías utilizadas en automatización.
- ✓ Mantiene el repositorio de conocimiento.
- ✓ Realiza el reporte de métricas a la alta gerencia y al cliente.

m. Director del Centro de Pruebas

- ✓ Establece y gestiona las expectativas del cliente.
- ✓ Gestiona las pruebas para varios proyectos.
- ✓ Identifica y monitorea los problemas y riesgos a nivel de centro de pruebas.
- ✓ Se encarga de reportarle directamente a los clientes el avance de las pruebas.

n. Consultor de Dominio

- ✓ Evalúa el proceso de pruebas y recomienda el plan de acción de mejoras.
- ✓ Prepara el plan de evaluación y coordina con las partes interesadas del proyecto.
- ✓ Participa del proceso de preventa y desarrollo del negocio.

o. Consultor en Automatización

- ✓ Realiza el análisis de la viabilidad de las pruebas automatizadas y la evaluación de herramientas.
- ✓ Realiza el plan de trabajo y define el enfoque de las pruebas automatizadas.
- ✓ Participa del proceso de preventa y desarrollo del negocio.
- ✓ Se encarga de la revisión de productos y de los avances en nuevas herramientas y técnicas.

9.2.2. MENTALIDAD DE UN TESTER

La mentalidad que se utiliza para hacer testing es diferente a la que se usa mientras se desarrolla software. Con una mentalidad adecuada, un desarrollador es capaz de probar su propio código, pero lo más adecuado es asignar esta responsabilidad a un tester, lo cual ayuda a no perder el enfoque y a tener una visión independiente.

Buscar fallas en un sistema requiere curiosidad, un ojo crítico, atención al detalle, buena comunicación con el resto del equipo y experiencia para saber enfocar los errores y las posibles causas que los generan, además de formación en técnicas de testing.

A continuación se listan algunas de las características que debe tener un buen tester, según Boris Beizer, reconocido autor de testing de software[6]:

> ¿Qué tiene un buen tester de software? Hay muchos mitos alrededor, como decir que son personas creativamente sádicas o capaces de manejar un trabajo aburrido y repetitivo. Este es un breve resumen de las lecciones aprendidas.
>
> 1. Conocimiento de programación

[6] Tomado, adaptado y traducido de http://www.sqatester.com/qateam/qualitiesofagoodtester.htm

Este es el ítem más controvertido. Hay un mito popular de que las pruebas pueden ser desarrolladas por personas que tienen pocos o nulos conocimientos de programación. Esto no funciona, a pesar de que desafortunadamente es un enfoque muy común. Hay dos motivos principales por los que no funciona:

a) Son pruebas de software. Sin conocimientos de programación, no se puede tener una percepción real de los tipos de errores que tiene el software y el lugar más probable para encontrarlos. Nunca hay tiempo suficiente para probar "por completo" una aplicación, por lo que todas las pruebas de software son un compromiso entre los recursos disponibles y el rigor con el que se realiza la prueba. El tester debe optimizar sus escasos recursos y esto significa centrarse en los errores más probables. Si no sabe programar, es poco probable que tenga la intuición necesaria para saber dónde buscar.
b) Todos los métodos de pruebas, excepto los más simples (y, por tanto, ineficaces) usan herramientas tecnológicas. Las herramientas, tanto como los productos para hacer pruebas, asumen que se tienen los conocimientos de programación necesarios para utilizarlos. Sin una formación como desarrollador, la mayoría de estas técnicas de pruebas (y las herramientas basadas en esas técnicas) no son fácilmente entendibles. El tester que no sepa de programación está limitado a la utilización de técnicas ad-hoc y a las herramientas más simples.

¿Significa esto que los testers deben tener una formación formal como desarrolladores, o haber trabajado como desarrolladores? El entrenamiento formal y la experiencia son normalmente la manera más fácil de cumplir con el requisito de “conocer de programación”. Es deseable un entrenamiento formal en programación como un grado universitario en Ciencias de la Computación o Ingeniería de Software, seguido de 2 ó 3 años de experiencia trabajando como desarrollador en alguna compañía.

No es bueno tomar desarrolladores junior y ponerlos a hacer pruebas, por:

a) Pérdida de imagen: Pocas universidades ofrecen formación en testing. Además los desarrolladores junior esperan tener un trabajo como desarrolladores y si se les ofrece un empleo como parte de un equipo de testing, lo tomarán como un fallo de su parte, es decir, creerán que no tiene lo necesario para ser desarrollador en esa organización. Esa percepción existe aún en organizaciones que valoran el trabajo de los testers.
b) Credibilidad con los desarrolladores: Los testers normalmente tienen que tratar con desarrolladores senior. A menos que hayan participado en alguna pasantía, toda su experiencia en programación ha sido en el ámbito académico; normalmente no tienen una idea real sobre la programación en un ambiente profesional, y por lo tanto, carecen de credibilidad frente a sus contrapartes en desarrollo.

c) Sobre el *Know-How*: El desarrollador tiene la razón. Un novato no sabe cómo desarrollar software. Si el nuevo integrante es un desarrollador "real", entonces el desarrollador senior normalmente se tomará el tiempo de entrenarlo, pero no hará lo mismo con un "simple" tester. Es más fácil para un tester nuevo haber aprendido cómo generar versiones, cómo hacer procedimientos, procesos, etc. como desarrollador que como un tester.

2. Conocimiento de la aplicación

Este es el otro lado de la moneda del conocimiento. El tester ideal tiene un profundo conocimiento de cómo los usuarios manipularán la aplicación y el tipo de errores que son más probables de surgir. En algunos casos, es virtualmente imposible, o al menos no es práctico para un tester, conocer la aplicación y la programación. Si la aplicación necesita de un profundo conocimiento, entonces es más fácil entrenar a un conocedor experto en la aplicación en programación, que a un desarrollador entrenarlo en la aplicación.

3. Inteligencia

La cualidad más importante es la inteligencia. Los buenos desarrolladores y los buenos testers son personas realmente inteligentes.

4. Hipersensibilidad a las cosas pequeñas

Los buenos testers notan aquellas pequeñas cosas que los demás, incluso los desarrolladores, no ven o ignoran. Los testers ven síntomas, no defectos. Se sabe que un defecto puede tener diferentes síntomas, desde los inocuos a los catastróficos. Y también sabemos que los síntomas de un defecto están relacionados en severidad con su causa. En consecuencia, no hay un síntoma menor porque un síntoma no es un error. Es sólo después de que el síntoma se ha explicado completamente (es decir, completamente depurado) que se tiene el derecho de decir si el error es menor o mayor. Por lo tanto, cualquier cosa fuera de lo común vale la pena ser revisada. Los buenos testers notan esas cosas y son capaces de usarlas para encontrar un conjunto de entradas relacionadas que causarán una falla catastrófica y llamarán la atención del desarrollador. Afortunadamente, este atributo se puede aprender con entrenamiento.

5. Tolerancia al caos

Cada persona reacciona frente al caos y a la incertidumbre de manera diferente. Algunos se acobardan y se rinden mientras otros tratan de crear algo de orden en el caos. Si el tester espera a que todos los problemas se resuelvan completamente antes de comenzar a diseñar o a ejecutar las pruebas, entonces no empezará hasta que el software haya sido entregado al cliente. Los testers tienen que ser flexible y dejar de lado las cosas que no se

pueden hacer y moverse a otra actividad que no esté “bloqueada”. En este sentido, los buenos testers difieren de los desarrolladores, ya que su mundo es inherentemente más caótico que el de los desarrolladores.

6. Habilidades interpersonales

Esta es otra característica en la que los testers y los desarrolladores pueden diferir. Se puede tener un desarrollador muy eficiente, incluso si es una persona hostil y antisocial, lo cual no funcionaría para un tester. Los tester pueden soportar mucho de los abusos de los indignados desarrolladores. El sentido del humor y fuerte carácter ayudarán al tester a sobrevivir. Por lo tanto deben ser diplomáticos cuando se enfrentan a un desarrollador senior. Diplomacia, tacto, una sonrisa, todo funciona a favor de un tester. Esto podría explicar una de las razones por las que hay tantas mujeres que hacen testing. Las mujeres son normalmente reconocidas por tener más habilidades interpersonales que los hombres.

7. Tenacidad

La habilidad de lograr compromiso y consenso puede ser una muestra de tenacidad. Esa es la otra parte de las habilidades interpersonales. Ser socialmente inteligente y diplomático no significa ser indeciso o débil. Los mejores testers son sociales y al mismo tiempo tenaces, según el caso.

8. Orden

No es posible imaginar un tester desordenado. Hay muchas cosas para registrar como para confiarle eso a la memoria. Los buenos testers usan archivos, bases de datos y otras herramientas de una mente organizada. Hacen listas de chequeo para mantener registro de sus actividades. También reconocen que pueden cometer errores, por lo cual revisan lo que encuentran dos veces. Tienen hechos y figuras para apoyar su posición. Es decir, cuando existe la sospecha de un defecto y el desarrollador no lo cree, el tester lo inundará con evidencia muy bien organizada y abrumadora.

Una de las consecuencias de una mente ordenada es la facilidad que tiene para las habilidades escritas y orales. Normalmente la inhabilidad de expresarse claramente de manera escrita es un síntoma de una mente desordenada. La buena escritura es ordenada, clara y va directo al punto. Y normalmente una escritura clara puede transformarse, mediante entrenamiento, en habilidad para hacer presentaciones orales.

9. Escepticismo

Esto no significa ser hostil. Cuando se habla de escepticismo es en el sentido de que nada se puede dar por sentado y que todo es susceptible de ser

cuestionado. Sólo la evidencia tangible, como documentos, especificaciones, código y resultados de las pruebas importan.

10. Autosuficiencia y resistencia

Si un tester necesita amor, no está esperando obtenerlo en su empleo. No andan buscando interacción entre ellos y los desarrolladores como fuente de estímulo para su ego. Su ego se alimenta encontrando errores, sin preocuparse por el dolor que pueda sentir el desarrollador.

11. Astucia

“Inteligencia callejera” es un buen descriptor de esta característica. Las técnicas de pruebas sistematizadas como las pruebas de sintaxis y las pruebas automatizadas han reducido la necesidad de testers astutos, sin embargo, esta necesidad seguirá existiendo porque no será posible sistematizar algún día todos los aspectos del testing. Siempre habrá espacio para ese tipo de pensamiento poco convencional que dará lugar a un caso de prueba que exponga un error realmente grave.

12. Hambre de tecnología

Un tester no disfruta del trabajo aburrido, repetitivo; lo haría si le toca, pero por muy poco tiempo. Un tester sabe que el testing manual puede introducir errores, y con el objeto de mejorar la calidad de su trabajo siempre está dispuesto a encontrar formas de eliminar los pasos que generen errores.

13. Honestidad

Los testers son fundamentalmente honestos e incorruptibles. Esta honestidad se extiende a un entendimiento brutalmente realista de sus propias limitaciones como ser humano. Aceptan la idea de que no son mejores o peores, y por lo tanto también saben que no son ajenos a cometer errores.

9.2.3. ENTRENAMIENTO

Cada nuevo recurso que ingrese al equipo de pruebas debe tener al menos un entrenamiento de 2 semanas, que incluya capacitaciones en testing y en la metodología que se usa en el equipo, en el producto que va a probar y en las herramientas que se utilizan durante las pruebas.

El proceso a seguir puede ser el siguiente:

1. Se debe iniciar con una orientación para explicar el rol que la persona va a tener y su posición dentro de la organización.
2. Se explica el funcionamiento del equipo de testing.
3. Se explican las políticas y procedimientos de la organización.
4. Se asignan permisos, accesos e identificaciones necesarias.
5. Se asignan los documentos, cursos o programas necesarios para el buen desempeño de las labores, es decir, aquellos relacionados con calidad, seguridad, testing y con el negocio en el cual se va a desempeñar, es decir, si la aplicación es de banca, salud, servicios, etc. debe tener una formación adecuada en ese campo. Adicionalmente se asignan cursos en *softskills*, como trabajo en equipo, habilidades comunicativas, habilidades analíticas, innovación y creatividad, aprendizaje continuo, orientación al cliente, planeación, entre otros.
6. Después de esto se entrega información del producto o proyecto al que está asignado.
7. Explicación del entorno laboral, de la empresa y de las funciones de los diferentes departamentos.
8. Entrenamiento en la herramienta de gestión de defectos.
9. Evaluación del plan de orientación.

9.3. ESTIMACIÓN DE PRUEBAS

El objetivo de utilizar varias técnicas de estimación es lograr que los resultados converjan a un valor de esfuerzo determinado, disminuyendo los problemas de estimación puntual. Para realizar una estimación se debe contar con un documento donde se especifique el alcance y los requisitos del sistema, subsistema o funcionalidad a estimar y el resultado se debe complementar con el cronograma de desarrollo.

Se pueden utilizar algunas de las siguientes técnicas para realizar la estimación:

1. **Estimación Juicio de Experto:** La estimación por Juicio de Expertos consiste en enumerar las funcionalidades que el sistema contendrá y valorar las diferentes actividades de pruebas que se deben ejecutar.

   Para cada funcionalidad a probar, el experto define para cada actividad del ciclo de pruebas la cantidad de recursos que se deben involucrar, el número de veces que dicha actividad debe ejecutarse y el esfuerzo estimado para cada actividad, definiendo el número de horas para el mejor caso, el peor caso y el caso esperado.

2. **Estimación por Tallas:** La estimación por tallas consiste en asignar una talla (Extragrande, Grande, Mediana, Pequeña), por disciplina, a cada caso de uso. Según la talla asignada será el esfuerzo requerido para ejecutar todas las tareas asociadas a la ejecución de las pruebas de la funcionalidad expresada en el caso de uso.

   Según la aplicación se clasifican los casos de uso por tipo, por ejemplo: Consulta, Creación, Visualización, Eliminación, Modificación, Reporte, etc. Y por cada tipo se definen los criterios bajo los cuales se clasifican los casos de uso en cada tipo de talla.

   Por ejemplo, si es un caso de uso de generación de reportes, los criterios podrían ser:
   - Pequeño: Los datos son traídos de manera directa sin demasiado procesamiento o vienen de una sola fuente de datos.
   - Mediano: Hay muchos campos en el reporte, los cuales hacen que el resultado de la información no sea fácil de interpretar.
   - Grande: La información proviene de diversas fuentes de datos con mucha integridad referencial y la lógica del negocio es compleja y se involucran varios aplicativos.

- Extragrande: Altísima complejidad en campos, cálculos y fuentes de datos.

3. **Estimación por Puntos de Caso de Uso:** La estimación por puntos de caso de uso consiste en asignar una complejidad (Complejo, Medio, Simple) a cada caso de uso y a los actores del sistema. Según esta complejidad, la técnica calcula los puntos de caso de uso que luego son ajustados de acuerdo con factores de ambiente, técnicos y de pruebas.

    - Para los factores de ambiente se tiene en cuenta la familiaridad con el proyecto, la motivación, los riesgos del proyecto, etc.
    - Para los factores técnicos se revisa la facilidad de instalación, la facilidad de uso, la portabilidad, etc.
    - Y desde la perspectiva de pruebas se inspecciona el uso de herramientas de pruebas, la documentación, el ambiente de pruebas, la complejidad de las interfaces, etc.

4. **Estimación por Puntos de Función:** La estimación por Puntos de Función consiste en indicar por cada requisito a implementar (o Caso de Uso) sus valores con relación a Entradas Externas, Salidas Externas, Consultas Externas, Archivos Lógicos Internos, Interfaces Externas, etc. Basándose en los valores de cada requisito, se calculan los puntos de función.

    Una vez se tiene la estimación para las pruebas, esta se envía al gerente del proyecto para que la revise y la incluya dentro del cronograma del proyecto.

9.4. REVISIÓN DE REQUISITOS FUNCIONALES

El propósito de esta actividad es involucrar al equipo de testing en la revisión de los requisitos de la aplicación. El equipo debe participar en las reuniones de revisión de los requisitos, en la revisión de los documentos generados y con base

en estos elementos comenzar a identificar los ítems a los que se les van a realizar pruebas.

Se debe examinar el plan de cada iteración e identificar los ítems específicos que lo conforman, así mismo, los entregables que durante la ejecución de la misma serán validados. Los elementos que se deben examinar incluyen la matriz de riesgos, solicitudes de cambios, requisitos, casos de uso, modelado UML, entre otros.

Para cada recurso o elemento clave es importante determinar los factores críticos. Si no se conoce el recurso o no se tiene el detalle suficiente, es importante discutir los ítems con el arquitecto o el gerente de proyecto.

La importancia de esta actividad radica en el valor que genera para el proyecto que los testers conozcan el negocio del cliente y que se familiaricen con la aplicación desde su concepción.

9.5. PLANEACIÓN DE LAS PRUEBAS

El objetivo de esta actividad es ajustar el enfoque, cronograma, recursos y metas del equipo de pruebas para el proyecto.

El líder de pruebas debe preparar la estrategia de pruebas como parte de las actividades de planeación.

Estas actividades son:

1. Examinar el plan del proyecto e identificar el alcance y los objetivos del mismo. Como forma complementaria a la lectura de la documentación se recomiendan reuniones informales con el equipo del proyecto.

2. Definir el objetivo de las pruebas, el alcance, las fases y las actividades para asegurar la cobertura de los requisitos. Es importante discutir con los interesados en el proyecto el alcance de las pruebas, presentarles a ellos lo que se ha identificado con el fin de clarificar los objetivos y requerimientos de las pruebas y obtener la respectiva realimentación.
3. Identificar el tipo de pruebas que se van a realizar.
4. Definir la organización del equipo de pruebas, los roles y las responsabilidades de cada miembro del equipo. En este punto se define usualmente el esquema de comunicación del equipo, alineado con el plan de comunicaciones del proyecto. Adicionalmente se definen los mecanismos de seguimiento al equipo, es decir, las reuniones, los informes, el seguimiento al cronograma, etc. y su periodicidad.
5. Revisar los requerimientos del negocio con el equipo de pruebas.
6. Definir el enfoque, los estándares y procedimientos. Se deben identificar los mecanismos potenciales para cubrir la estrategia de pruebas utilizando el conocimiento de la arquitectura de software y el entorno del sistema definido y se deben documentar como parte del plan de pruebas.
7. Identificar los entregables de las pruebas. Se debe dejar claro cuáles documentos serán el resultado de las pruebas. Los distintos ítems que genera el equipo de pruebas pueden ser importantes como entregables para distintos clientes y algunos son importantes para el equipo de desarrollo. Estos deben estar documentados en el plan de pruebas.
8. Identificar las herramientas y técnicas que van a utilizarse.
9. Identificar los escenarios de prueba de alto nivel para cada test case.
10. Revisar los escenarios de prueba con el arquitecto o equipo de desarrollo.
11. Identificar el criterio de finalización de pruebas y los criterios de aceptación por parte del cliente.
12. Preparar la matriz de trazabilidad. La matriz de trazabilidad se utiliza para saber cuáles requerimientos quedan cubiertos por una prueba, es decir, qué partes o módulos del software no están cubiertos y deberían probarse por otras pruebas o identificar los requerimientos más críticos para saber si están

cubiertos. También se puede identificar los casos de prueba que han fallado y a partir de ahí ver qué requerimiento está en riesgo, para poder evaluar su criticidad y riesgo.

13. Identificar los componentes que tengan más riesgos para la calidad del producto y realizar el plan de gestión de riesgos.
14. Si aplica, se identifican las partes del sistema en dónde se pueden realizar pruebas automatizadas, de performance, de seguridad, etc.
15. Detallar las pruebas para cada escenario.
16. Definir, de acuerdo con los objetivos de la prueba, la naturaleza de los datos que se van a utilizar: generados, capturados o creados manualmente.
17. Realizar y enviar las estimaciones para que sean aprobadas.
18. Realizar y enviar el plan de pruebas para que sea aprobado.

9.6. PREPARACIÓN DE LAS PRUEBAS

La preparación de las pruebas incluye todas las tareas involucradas en la creación, revisión, preparación y automatización de test cases, si es el caso. Esto incluye las pruebas funcionales y los otros tipos de pruebas que se vayan a realizar. Adicionalmente, esta actividad incluye las tareas de creación de los test cases y la verificación de cobertura de los requerimientos, además de la preparación del ambiente y datos de pruebas.

9.6.1. CREACIÓN DE TEST CASES

Es importante seleccionar la técnica más apropiada para diseñar e implementar un caso de prueba. Algunas veces la implementación de un caso de prueba puede cubrir varios casos de prueba y esto en la práctica sería lo ideal.

Normalmente los métodos para implementar casos de prueba incluyen la descripción textual de los pasos que se deben seguir (para pruebas manuales) y la

programación, captura o grabación que generan dichos pasos en pruebas automatizadas.

Muchas pruebas son llevadas a cabo manualmente, así como también ciertos casos de prueba quedan mejor cubiertos por pruebas manuales que por las pruebas automatizadas. Un ejemplo de esto son las pruebas de Usabilidad, un área en el cual una aproximación manual da mucho más resultado que una prueba automatizada.

Debido a esto es recomendable realizar primero un acercamiento al objeto de prueba con un caso de prueba manual; esto permite al analista de prueba aprender más del objeto de prueba e ir adaptando las respuestas o comportamientos en uno o más casos de prueba. Algunas veces los casos de prueba manuales son llevados a la automatización como parte de las pruebas de regresión. No es necesario, ni deseable automatizar todos los casos de prueba manuales. Aunque la automatización reduce tiempos, da mayor cobertura, y permite una mayor eficiencia al mantener complejos sets de pruebas, no es la solución de todas las pruebas.

Programar los casos de prueba no es la forma más pura de implementar la automatización de los mismos, esta práctica se realiza de la misma manera y utiliza los mismos principios generales de la programación de software. De tal forma que las mismas herramientas de desarrollo pueden ser aplicables a la programación de casos de prueba programados.

Usando cualquier ambiente de desarrollo (como Microsoft Visual Studio) o utilizando herramientas especializadas para pruebas (como Rational Functional Tester, QTP, entre otras), el analista de pruebas tiene la libertad de utilizar las características del ambiente para alcanzar un mejor resultado.

El aspecto negativo de las pruebas automatizadas está directamente relacionado con los aspectos negativos de la programación como tal. Para que la programación sea efectiva, algunas consideraciones deben ser realizadas con un apropiado diseño, sin esto es posible que la implementación falle.

Un riesgo al que se enfrenta el analista de pruebas es a construir la solución con complejas características o soluciones elegantes y sofisticadas a problemas que se podrían alcanzar de formas simples. El resultado es que el analista de pruebas pierde mucho tiempo en tareas de programación y no en tareas efectivas de pruebas o evaluación de resultados.

Otro riesgo es que el programa que se implementó para probar incluya por sí mismo errores. Algunos de estos son fáciles de identificar y corregir durante el curso de la implementación de la automatización, otros no. Más aún, algunos errores son introducidos al utilizar el mismo algoritmo de la implementación de software; el resultado es que dichos errores se convierten en indetectables. La forma de mitigar este riesgo es no basar los casos de prueba programados en la solución de software y utilizar distintos algoritmos cuando sea posible.

9.6.2. CONFIGURACIÓN DEL AMBIENTE DE PRUEBAS

Inicialmente se deben definir las configuraciones del ambiente de pruebas necesarias para soportar el esfuerzo de pruebas. Para esto se deben realizar las siguientes tareas:

1. Revisar la estrategia de pruebas y el cómo se ve afectada por la arquitectura de la solución: Revisar la estrategia de pruebas, caracterizando los ítems y los aspectos claves de la misma. Con base en esta información, revisar la arquitectura del sistema y definir las necesidades de ambiente para las pruebas.
2. Identificar y diferenciar los distintos ambientes: Usando la arquitectura del software como punto de partida, localizar y revisar el modelo de despliegue y la

información relacionada. También, identificar cada ambiente en el que se haría despliegue para entender las características de los mismos.

3. Configuración del ambiente de pruebas: Se deben identificar y definir los siguientes detalles:
   3.1. Identificar las necesidades específicas del ambiente por técnica de prueba. Usando el plan de pruebas, identificar cada técnica que va a ser utilizada en la estrategia de pruebas. Para cada una de ellas, listar los requisitos específicos que se deben satisfacer para llevar a cabo la prueba.
   3.2. Definir los recursos de hardware y software. Usando los requisitos que se identificaron, se debe construir una lista de hardware o software que son necesarios para el desarrollo de la prueba.
   3.3. Definir la administración del ambiente de pruebas. Algunas veces los procesos que deben llevarse a cabo para inicializar los ambientes son tan complejos que no es necesario concentrar el esfuerzo del equipo de pruebas en la administración de los mismos. En caso de ser así esto debe dejarse por escrito en el plan de pruebas.
   3.4. Se debe suministrar al equipo de pruebas una descripción del ambiente, y los pasos para configurarlo (Incluso aunque el equipo de pruebas no sea quien lo administre).

En este punto también se inicia la creación del set de datos de pruebas. El éxito de las pruebas realizadas a un sistema depende en gran parte de los datos utilizados durante el proceso. La administración de grandes cantidades de datos consume tiempo y recursos. Para lograr cumplir con estándares de calidad es necesario no utilizar datos reales que pudieran ser objeto de mal uso durante el proceso de pruebas.

Las herramientas de generación de datos sirven para generar datos adecuados que no estén relacionados con los datos de producción, resultando útil si es importante proteger la privacidad o la seguridad de los datos de producción. Los datos de prueba expresan todas las variables del caso de uso, su estructura si son

tipos complejos, las restricciones que puedan existir entre ellos y las particiones de sus respectivos dominios.

Para generar datos es importante crear un plan de generación de datos donde se defina lo siguiente:

1. Qué tablas desea poblar.
2. Qué datos necesita.
3. Cuántos datos necesita en cada tabla.
4. Seleccionar las columnas que desea llenar de datos y establecer la configuración de cada columna.
5. Método de generación de los datos, es decir, directamente en la base de datos o mediante la utilización de la aplicación.

9.6.3. DESPLIEGUE DEL PRODUCTO PARA PRUEBAS

Se deben utilizar los artefactos de despliegue y el manual de administración para instalar y configurar el producto y dejarlo apto para el inicio de las pruebas. En esta etapa es donde se identifican incidentes en el manual y la configuración por defecto de la solución. Utilizando los artefactos construidos y siguiendo el manual de instalación, se debe realizar un paso a paso en el ambiente de pruebas. Es importante verificar que se encuentren todos los artefactos y los recursos definidos en el plan de pruebas.

Además, se debe validar la estabilidad de la solución, identificando si la construcción está estable y puede ser objeto de pruebas confiable. A partir de pruebas cortas o pruebas de humo, se busca establecer si la solución cumple con las expectativas de los objetos a revisar en el ciclo de pruebas. Tradicionalmente, se realiza un recorrido por la aplicación para validar que la instalación por medio de los artefactos de despliegue ha sido exitosa. Durante esta validación el Analista de Pruebas y/o el Tester deben hacer una revisión de las listas de chequeo.

9.7. EJECUCIÓN DE LAS PRUEBAS

En esta actividad, el propósito principal es la realización de las pruebas definidas en el producto de software. Incluye la ejecución de las pruebas manuales y automatizadas.

Basándose en los criterios de validación establecidos, se identifican los productos y componentes que no se desempeñan adecuadamente en su ambiente de operaciones y se identifican problemas con los métodos, criterios, y/o ambiente, a través de la comparación entre los resultados actuales y los resultados esperados, los cuales están detallados en los casos de prueba diseñados.

Es importante dejar registrada la ejecución de los test cases; así mismo, el identificar posibilidades de creación y diseño de nuevos casos de prueba.

Los pasos para la ejecución, son:

1. Ejecutar las pruebas. En este paso normalmente se inicia con las pruebas manuales y luego se procede con las pruebas automatizadas, o con cualquier otro tipo de prueba. La idea es garantizar que algunos requisitos básicos del software se cumplen antes de gastar esfuerzo de desarrollo.
2. Registrar los resultados de las pruebas en alguna herramienta o archivo, donde se identifiquen los test cases que han sido ejecutados, quién los ejecutó, en qué fecha y cuál fue el resultado obtenido.
3. Reportar los defectos. Si hay defectos se deben reportar en una herramienta para el seguimiento de los mismos. En la siguiente sección se encuentra una explicación más detallada de este tema.
4. Revisar los defectos corregidos. Una vez el equipo de desarrollo corrige los defectos encontrados, se procede a volver a ejecutar los test cases que contienen esos defectos o en ocasiones sólo es necesario ir directamente al lugar donde se encontró el defecto. Se valida dicho defecto y se cierra si está corregido o se reabre si el defecto todavía está sucediendo.

5. Ejecutar las pruebas de regresión. Normalmente se designa un conjunto de test cases que abarca la mayor parte de la funcionalidad de la aplicación y estos son los que se ejecutan en las pruebas de regresión. Se recomienda adicionalmente la ejecución de test cases que contengan las funciones críticas de la aplicación y la creación de puntos de control claves donde se pueda verificar si hay fallos o no.
6. Registrar los resultados y reportar los defectos de las pruebas de regresión.
7. Volver a probar los test cases que cubren los defectos que fueron arreglados de las pruebas de regresión.
8. Verificar la etapa del ciclo de pruebas en la que se está y comenzar la nueva etapa si aplica. Es decir, si se está en la etapa uno del ciclo de pruebas, se continúa la ejecución y si se está en la etapa final, se informa que el ciclo de pruebas ha terminado y se comienza la etapa de finalización de pruebas.
9. Revisar los resultados de las pruebas realizando un informe de incidentes con el fin de poder brindar un entendimiento de lo evidenciado durante las distintas fases y las actividades de validación; es importante en este punto la recopilación y clasificación de los distintos incidentes. Además, se debe hacer un análisis de causas y la creación de acciones correctivas, preventivas o de mejora, ya que una vez clasificado lo evidenciado durante las pruebas es importante poder brindar información clara sobre lo que se puede hacer para actuar al respecto y minimizar el impacto en próximas tareas de prueba. Esto puede incluir la modificación de la estrategia de prueba para la siguiente iteración, entre otras acciones.
10. Entregar los resultados de las pruebas a las partes interesadas. En este paso normalmente el analista de pruebas entrega una carta de certificación con el resumen de las pruebas, una vez aprobada por el líder de pruebas, se entrega a las partes interesadas.

A continuación se presenta el detalle de la gestión de los defectos.

9.7.1. GESTIÓN DE LOS DEFECTOS

La gestión de los defectos permite realizar un seguimiento detallado a la calidad del producto en cuanto a los incidentes detectados a lo largo del ciclo de vida del proyecto. Para que el seguimiento sea efectivo es indispensable que todas las actividades relacionadas con los defectos sean registradas en una herramienta centralizada.

Cada defecto, error o sugerencia cambia de estado con el paso del tiempo y es importante llevar registro de estos cambios para controlar el avance del proyecto.

Generalmente los defectos son detectados por el equipo de pruebas, pero en ocasiones los usuarios, analistas, arquitectos, desarrolladores, etc. también reportan defectos sobre la aplicación; sin embargo, el líder de pruebas es quien lleva control sobre los mismos.

La gestión de defectos incluye la realización de reportes con las estadísticas de los defectos; por ejemplo, cuántos defectos fueron reportados, arreglados, verificados, cerrados; cuánto tiempo dura el ciclo de vida del defecto desde que se abre hasta cuando es cerrado; cuántos defectos se reabrieron, etc.

En relación con los defectos, se debe garantizar que en todo momento:

- Se vele por la solución de los incidentes severos que impactan la aceptación de los objetos de prueba.
- Se cuide la calidad definida en el plan de pruebas.
- Se identifiquen las regresiones necesarias entre los distintos ciclos de pruebas.

9.7.2. DEFECTO

Un defecto de software es el término usado para describir un error, defecto o fallo en un programa o sistema que produce un resultado incorrecto o inesperado, o causa un comportamiento no deseado. La mayoría de los defectos surgen de errores o equivocaciones cometidas por las personas, ya sea en el código fuente de un programa o en su diseño, y algunas son causadas por los compiladores que pueden producir código incorrecto.

Un defecto puede ser escrito contra el código, la base de datos o la documentación. Los siguientes podrían no considerarse como defectos:

- Los problemas con el ambiente que no están relacionados con el código.
- Temas que son el resultado de una instalación incorrecta o incompleta o de configuración de la aplicación.
- Problemas conocidos que pueden ser causados por falta de datos.

Para asegurar que un defecto quede bien escrito se deben seguir los siguientes pasos:

- Estructurar: El reporte de un defecto debe comenzar con el resultado esperado y con el resultado observado y en cómo difieren, para esto se deben estructurar las pruebas cuidadosamente.
- Reproducir: Ante la aparición de un comportamiento extraño en la aplicación, se debe repetir la prueba. Siempre se debe asegurar que el defecto sea reproducible y se deben escribir los pasos detallados para lograrlo.
- Aislar: Se deben realizar diferentes pruebas alrededor del mismo problema para tratar de identificar las condiciones específicas en las que sucede.
- Generalizar: Se deben buscar fallas relacionadas con el defecto que se va a reportar para saber si ocurre en otros módulos.

- Comparar: Se deben revisar los resultados obtenidos en pruebas similares, realizadas en versiones anteriores o variando las condiciones.
- Resumir: Cada defecto debe tener una línea que lo describa brevemente, pero que ayude a identificar el origen y la gravedad del defecto.
- Condensar: Se debe eliminar toda la información innecesaria, como palabras extrañas o pasos sobrantes.
- Eliminar la ambigüedad: Eliminar, parafrasear o aclarar las palabras o frases que no sean claras.
- Neutralizar: Se deben evitar los ataques personales, la crítica destructiva, el uso de sarcasmo. Los defectos deben contener solamente hechos y evidencias.
- Revisar: Es una buena práctica hacer que otro tester revise los defectos reportados para sugerir mejoras, hacer preguntas de temas que necesiten aclaración o cuestionar el defecto, en caso de ser necesario.

Todo defecto debe tener la siguiente estructura cuando se reporta:

1. ID: Cada defecto debe estar identificado.
2. Tester: Nombre del tester que reporta el defecto.
3. Fecha: Fecha en que se reporta el defecto.
4. Resumen: Este campo debe contener una corta descripción del defecto, pero que sea lo suficientemente informativa.
5. Descripción: En este campo se encuentra la mayor cantidad de información del defecto. Incluye una descripción de cómo se está comportando el sistema actualmente y cómo debería estar trabajando. Además, se incluye toda la información relevante, como mensajes de error, si hay rutas alternativas para ejecutar el flujo que tiene el error, si se usaron datos especiales en las pruebas o alguna configuración específica y además el impacto del defecto.
6. Severidad: Acá se selecciona la severidad según el tipo de defecto.
7. Probabilidad: Acá se indica si es reproducible o no y con qué frecuencia.

8. Tipo: Se indica qué tipo de defecto es, es decir, si es de base de datos, funcional, interfaz gráfica, etc.
9. Estado: El estado inicial es Abierto (Nuevo).
10. Prioridad: De acuerdo con la severidad y probabilidad, se indica la prioridad.
11. Lugar: Dónde fue encontrado el defecto. Este campo podría referirse al caso de uso, módulo, aplicativo, pantalla, etc.
12. Versión: Versión del producto donde fue encontrado el defecto.
13. Responsable: La persona encargada de asignar o corregir el defecto según el flujo de trabajo definido.
14. Pasos para reproducir: Este campo contiene documentación detallada de los pasos necesarios para reproducir el defecto. Esto permite al desarrollador reproducir el defecto fácilmente.
15. Archivos adjuntos: Pueden ser capturas de pantalla donde se vea el defecto, o videos con la secuencia de pasos, logs de ejecución, o archivos usados en las pruebas.
16. Notas: Campo opcional donde se puede escribir alguna otra información relevante.

9.7.3. CICLO DE VIDA DE UN DEFECTO

Cada defecto tiene un ciclo, desde que es reportado hasta que finalmente es cerrado. La figura 6 ilustra las distintas etapas por las que pasa un defecto:

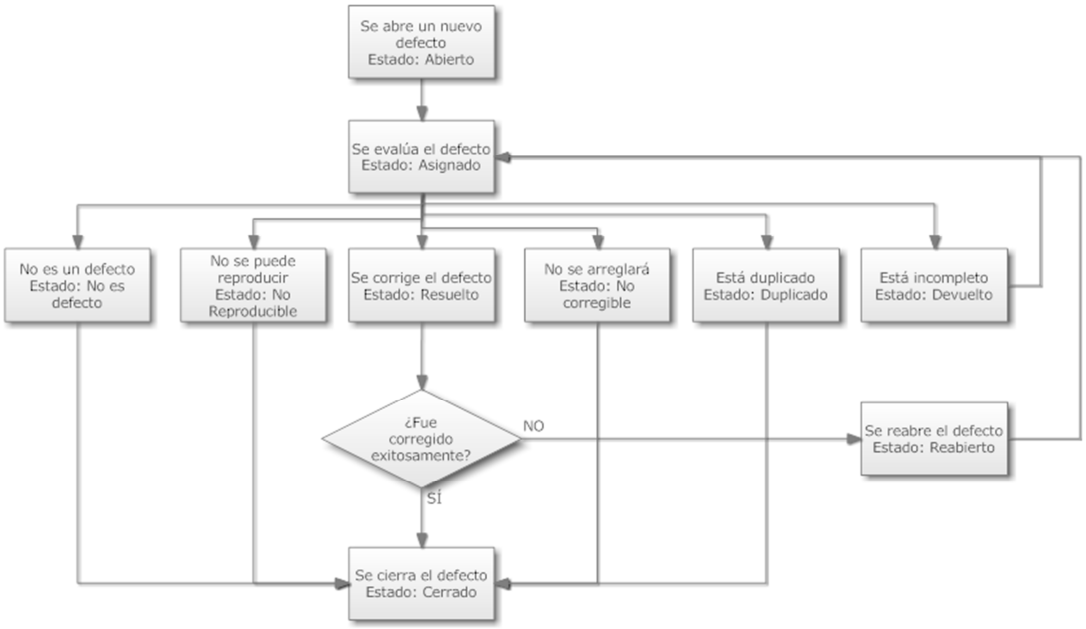


Figura 6. Ciclo de vida de un defecto.

(Adaptación propia)

9.7.4. CLASIFICACIÓN DE LOS DEFECTOS

Los defectos encontrados en las pruebas deben ser clasificados según su nivel de severidad, probabilidad y prioridad; es decir, según su impacto al negocio y a la aplicación y según los riesgos que conllevan.

Cada negocio puede tener su propia clasificación según sus necesidades específicas; por ejemplo, en algunas aplicaciones un defecto de interfaz gráfica podría tener una mayor severidad, dependiendo de la prioridad que le dé el cliente a estos errores.

Para la **severidad** se pueden usar las siguientes categorías:

- Severidad 1 - Grave: El defecto impacta áreas críticas del negocio y puede detener las pruebas. Puede ser un bloqueo, una salida involuntaria del sistema o una operación grave en las reglas del negocio.
- Severidad 2 - Mayor: El defecto afecta seriamente alguna parte funcional de la aplicación. Normalmente son defectos que causan una salida crítica del sistema que no permite que el usuario complete una tarea, puede afectar el cumplimiento de estándares, políticas o regulaciones. Incluye la corrupción de datos y la falta de integridad de estos.
- Severidad 3 - Normal: El defecto está en una pantalla o funcionalidad específica y tiene un flujo alternativo que permite realizar la tarea.
- Severidad 4 - Menor: Son todos aquellos defectos que implican una afinación en la interfaz gráfica, errores en el texto.

La **probabilidad** de ocurrencia de un defecto se puede clasificar de la siguiente manera:

- Frecuente: El defecto siempre se presenta.
- Probable: Es posible que el defecto suceda, y su ocurrencia es generalizada.
- Ocasional: En ocasiones sucede, pero no se tiene identificada una tendencia.
- Aleatorio: En este caso son errores fantasmas, que se presentaron una vez y no volvieron a suceder o que no se han podido reproducir.

La **prioridad** se clasifica, en:

- Alta: El defecto debe arreglarse inmediatamente. Las pruebas podrían estar detenidas hasta que sea arreglado.
- Media: El defecto se debe arreglar lo más pronto posible. Las pruebas de una sección específica del sistema se pueden detener, sin embargo el resto del sistema puede continuar siendo revisado.

- Baja: Su arreglo puede postergarse un poco ya que no afecta las pruebas.

Adicionalmente hay otros aspectos que influyen en la asignación de prioridad de un defecto, tales como:

- Qué tanto afecta al cliente
- Complejidad del arreglo
- Costos del arreglo
- Cómo se afecta el resto de la aplicación

9.7.5. TIPOS DE DEFECTOS

Los defectos se pueden clasificar por su origen según las siguientes categorías:

- Base de Datos: Errores que se generan por problemas relacionados con elementos de la base de datos. Falta una tabla, un campo, un *trigger*, tamaño de campos, tipos, índices o claves.
- Codificación: Errores que se presentan cuando se hace revisión de código.
- Componente Externo: Errores en dll's, componentes, controles externos. Errores de productos con los que se integra el sistema que se está probando.
- Configuración: Errores que se generan por problemas de configuración en archivos .ini, .config, tablas de parámetros, variables de ambiente, etc.
- Despliegue: Errores que se presentan cuando se está probando la instalación. Falta de componentes, dll´s, que no son de configuración.
- Diseño: Errores en los diagramas de diseño (clases, secuencia, estados, colaboración etc.) que afectan la funcionalidad.
- Documentación: Error en un manual del sistema, manual de usuario, manual técnico, manual de instalación o configuración.
- Especificación: Errores en la especificaciones, como ambigüedad, completitud, comprensión, trazabilidad, que sea verificable, etc.

- Excepción: Error no controlado del sistema, errores técnicos, errores de la herramienta de desarrollo, errores de la base de datos.
- Experiencia de usuario: Errores de usabilidad, funcionalidades no intuitivas, funcionalidades difíciles de operar, etc.
- Funcional: No cumple con lo especificado, reglas del caso de uso, flujos alternos, condiciones, validaciones etc.
- Interfaz gráfica: No se cumplen con lineamientos gráficos, ortografía, gramática, colores, tamaños de letra, alineación de campos, mensajes de configuración, alerta, interrogación, etc.
- Sugerencia: Mejoras al sistema que no están especificadas.

9.8. FINALIZACIÓN DE LAS PRUEBAS

El objetivo de la etapa de finalización de las pruebas es preparar el registro histórico del esfuerzo hecho en pruebas para el proyecto. Esta tarea es realizada principalmente por el líder de pruebas quien se encargará de generar los reportes requeridos.

Se debe comenzar con un análisis de la cobertura de requisitos en las pruebas, es decir, si se probaron todos los requisitos que se planearon inicialmente. El líder del equipo de pruebas debe encargarse de generar un reporte de cobertura de las pruebas donde indique la cantidad de requisitos probados y los motivos por los cuales no se cubrieron todos los requisitos, si ese es el caso.

A continuación, el líder del equipo de pruebas debe hacer un análisis de la ejecución de las pruebas, qué problemas se tuvieron, cómo se resolvieron, qué hallazgos importantes tuvieron, etc. con el fin de poder brindar un entendimiento de lo evidenciado durante las distintas fases y las actividades de validación. Una vez clasificado lo evidenciado durante las pruebas es importante poder brindar información clara sobre lo que se puede hacer para actuar al respecto y minimizar

el impacto en próximas proyectos de pruebas mediante la creación de acciones correctivas, preventivas o de mejora.

Posteriormente se hace un análisis de los defectos encontrados durante las pruebas. La estrategia en este punto es realizar una actividad de análisis de causas con el equipo que participó en las pruebas para identificar los puntos clave a tener en cuenta.

Una vez se tenga la información recopilada, se procede a la generación de gráficas y estadísticas de los resultados de las pruebas, se prepara el reporte final y se entrega al director de pruebas para que sea revisado y archivado.

9.9. LECCIONES APRENDIDAS

Se podrían considerar lecciones aprendidas las conclusiones que se obtienen una vez que se ha realizado una tarea concreta. Esto es, cuando se evalúa la labor realizada comparándola con los resultados obtenidos según los objetivos marcados y el tiempo, el esfuerzo y, en muchas ocasiones, el dinero invertido. Las conclusiones resultantes tras este tipo de evaluaciones se podrían considerar lecciones aprendidas. El fin detrás de estos conceptos es evitar la repetición de errores y tratar de no tener que “reinventar la rueda” cada que nos encontramos ante una situación o un problema ante el que, con toda probabilidad, otros se han encontrado anteriormente.

El propósito de esta actividad en testing es determinar los puntos donde se pueden realizar mejoras en el proyecto. Lo ideal es que al final de cada proyecto se tenga programada una actividad donde se discutan estos puntos, para analizar los problemas que se presentaron y para resolver cualquier duda que pudiera haber quedado del ciclo de pruebas o del proceso de pruebas.

El equipo de pruebas identifica y analiza los baches que se presentaron en el proyecto, desde la perspectiva de pruebas, creando un plan de contingencia para evitar que se vuelvan a repetir en futuros proyectos.

En esta actividad se pueden seguir los siguientes pasos para identificar las lecciones aprendidas:

1. Con la información que se recopile en la actividad de Finalización de las Pruebas, el siguiente paso es revisar si se siguieron todos los protocolos de pruebas establecidos para el proyecto, listando aquellos que no se siguieron.
2. Luego se procede a enumerar aquellas actividades que se realizaron bien, es decir, donde se cumplieron los objetivos a cabalidad.
3. A continuación se detallan las áreas o elementos que presentaron falencias y que necesitan una mejora y se desarrolla un plan para evitar que en un próximo proyecto vuelvan a suceder.
4. Se presentan las lecciones aprendidas al equipo de testing.
5. Se guarda el documento de lecciones aprendidas en el repositorio de documentos del proyecto.

9.10. MÉTRICAS

Hay varias razones por las cuales se recolectan métricas en un proyecto, pero la principal es informar al responsable del proyecto acerca del estado de las pruebas. Adicionalmente, las métricas en cualquier proyecto pueden ayudar a definir el éxito del proceso, ya que brindan información sobre el estado actual, controlando el proyecto contra lo planificado, y apoyan el proceso de toma de decisiones preventivas y correctivas.

Lo más importante de las métricas es usar aquellas que se ajusten al negocio, al tipo de proyecto y que sean útiles para el mismo.

A continuación se presenta una lista de algunas métricas para proyectos de testing:

NOMBRE	MEDICIÓN	FRECUENCIA	INTERESADOS
Estado de los defectos	Defectos cuyo estado es Nuevo, Cerrado, Corregido o Devuelto, etc., según la clasificación que se tenga.	Por versión entregable del producto.	- Director de pruebas. - Gerente del proyecto.
Estado de la ejecución de los test cases	Número de test cases ejecutados por estado del test case (Ejecutado, No Ejecutado, Bloqueado, Pasó, Falló).	Por versión entregable del producto.	- Director de pruebas. - Gerente del proyecto.
Resumen de la ejecución	Ejecuciones diarias por módulo, aplicación, versión, o según se necesite.	Diariamente.	- Líder de pruebas. - Director de pruebas.
Resumen de la ejecución por tester	Ejecuciones diarias por tester.	Diariamente.	- Líder de pruebas. - Director de pruebas.
Cantidad de pruebas que han tenido que repetirse	Número de test cases que se han tenido que ejecutar más de 2 veces por versión del producto.	Por versión entregable del producto.	- Líder de pruebas.

Efectividad del tester	# Defectos por tester/# Defectos encontrados en pruebas de aceptación * (# Defectos por tester * 100%)	Por versión entregable del producto.	- Líder de pruebas. - Director de pruebas.
Densidad de defectos	# Defectos encontrados/Líneas de código	Por versión entregable del producto.	- Director de pruebas. - Gerente del proyecto.
Cobertura de las pruebas	Cantidad de test cases ejecutados/Cantidad de test cases planeados	Por entrega al cliente.	- Director de pruebas. - Gerente del proyecto. - Líder de pruebas.
Causa de los defectos	Cantidad de defectos funcionales, de datos, de código, de documentación, de requerimientos, etc.	Por versión entregable del producto.	- Líder de pruebas. - Director de pruebas.

9.11. HERRAMIENTAS

Las herramientas utilizadas en el proceso de pruebas dependen del tamaño y presupuesto que tenga la organización, así como de sus necesidades específicas.

A continuación se presenta una lista de algunas herramientas que pueden resultar útiles:

9.11.1. HERRAMIENTAS PARA PROBAR CÓDIGO FUENTE

Producto	Proveedor	Descripción
BoundsChecker	Compuware	Detección y depuración de errores Run-time para desarrolladores de C++. Soporta C/C++,.net,ASP,ASP.net.
Bullseye Coverage	Bullseye Testing Technology	Cubrimiento de código para C/C++.
CodeCheck	Abraxas Software	Mide mantenibilidad, portabilidad, complejidad y cumplimiento de estándares de código en C and C++.
devAdvantage	Anticipating Minds	Crea estándares de codificación en C#. Detecta y corrige para cumplir con las mejores prácticas de .NET.
LDRA Testbed	LDRA	Herramienta de análisis estático de código para C/C++, C#, Ada83, Ada95, Java, Visual Basic, Cobol, Coral66, Fortran, Pascal, PL/Mx86, PL/1, Algol e Intel, Motorola, Texas Instruments y ensambladores PowerPC. Incluye visualización de código, cumplimiento de estándares de programación y métricas.
Logiscope	Telelogic	Herramienta de análisis de código fuente.
McCabe TQ	McCabe Software	Ayuda a identificar, medir y reportar la complejidad del código en la aplicación y a nivel de empresa.
Predictive Lite	ism	Analizador de código que predice defectos en C, C++, Java o C#, basándose en estándares de industria.

9.11.2. HERRAMIENTAS PARA PRUEBAS FUNCIONALES

Producto	Proveedor	Descripción
Automated Test Designer	AtYourSide Consulting	Herramienta para crear test cases basándose en los requerimientos funcionales, la cual puede ser usada en un ambiente colaborativo por todos los miembros del proyecto. Ayuda a gestionar cambios y requerimientos de cualquier tipo de complejidad. Usa un algoritmo de una red neural optimizada para generar el mínimo número de test cases para certificar el 100% de los requerimientos.
AutoTester One	AutoTester	Software para realizar pruebas funcionales, de regresión y de integración en Windows, Cliente – Servidor y Web. También ofrece una versión mejorada para SAP, con un motor para realizar pruebas de carga y de estrés.
GUITAR	University of Maryland	Herramienta libre que genera nuevos test cases basándose en una tecnología que usa gráficos de flujo de eventos. Las pre y post-condiciones se usan para generar la respuesta esperada.
Holodeck	Security Innovations	Simula el mundo real y los errores de una aplicación. Permite a los testers y desarrolladores trabajar en un ambiente controlado y repetible para analizar y depurar código en ambientes complejos.
MITS.GUI		Tiene una máquina de estados que toma decisiones en tiempo real para navegar a través de una porción de GUI de una aplicación sin scripts. Los testers simplemente entran los datos en una hoja de cálculo, y estos se usan para poblar los objetos que aparezcan en un escenario particular definido.

QACenter	Compuware	Herramienta para el desarrollo y ejecución de pruebas de GUI y basadas en arquitectura cliente – servidor.
SAP Software Quality Assurance Testing Tools	Sucid	El proveedor de herramientas y productos para aseguramiento de la calidad del software.
ScriptTech	TMX	Genera scripts ejecutables para WinRunner, TestPartner, SilkTest y otras herramientas sin necesidad de entender el lenguaje de programación.
Smalltalk Test Mentor	SilverMark, Inc.	Herramienta de pruebas GUI y de objetos para IBM.
Squish	froglogic	Framework para aplicaciones en C++. Permite hacer pruebas a las aplicaciones usando scripts grabados o creados manualmente en diferentes lenguajes.
TestArchitect	LogiGear	Framework para automatización que soporta la mayoría de las plataformas existentes.
TestWorks	Software Research	Suite que integra Pruebas de Regresión Automatizadas y Análisis de Cobertura para UNIX, Windows/2000/NT/XP con múltiples reportes y scripts reutilizables en C, C++ y JAVA.
Unified Test Pro	Software Development Technologies	Una solución de "3ra generación" para pruebas de automatización, que usa un enfoque basado en roles para el diseño, construcción y ejecución de test cases.
WinRunner®	Mercury	Graba y reproduce pruebas de GUI para aplicaciones Windows.
X-Unity	MIIK Ltd.	Ambiente de pruebas unitarias para el framework de Microsoft .NET.
Astra QuickTestTM	Mercury	*Web site* para testing funcional.
AutoTester One	AutoTester	Sirve para realizar pruebas funcionales, de regresión e integración de aplicaciones Windows, Cliente – Servidor o Web.
PesterCat	PesterCat	Herramienta de pruebas Web diseñada para realizar pruebas funcionales sobre aplicaciones Web. Permite grabar scripts usando cualquier navegador. Funciona en Linux, Mac OSX y Windows.
Ranorex	Ranorex Software	Framework de automatización para pruebas en Windows GUI y C++, Python y los lenguajes .NET.
Rational Robot	Rational Software	Permite realizar pruebas funcionales, de regresión y smoke test en aplicaciones Web.
Selenium	ThoughtWorks	Herramienta de pruebas para aplicaciones Web. Se ejecuta directamente en un navegador, tal y como lo harían los usuarios reales. Corre en Internet Explorer, Mozilla y Firefox, y en Windows, Linux y Macintosh.
TestWeb	Original Software	Soluciones de automatización para IBM iSeries, Microsoft y Oracle.
QTP	Mercury - HP	El Quick Test Professional es utilizado popularmente como herramienta de automatización de pruebas de regresión.
Yawet	InforMatrix	Herramienta Java para crear, ejecutar y depurar pruebas sobre Web. Sirve para verificar HTML, PDF y XML.

9.11.3. HERRAMIENTAS PARA PRUEBAS DE DESEMPEÑO

Producto	Proveedor	Descripción
BugTimer	BugStomper Software	Aplicación que graba, muestra, guarda, ordena e imprime los resultados de las pruebas de desempeño.
DB Stress	DTM	Aplicación para hacer pruebas de estrés sobre aplicaciones y DBMSs.
LoadRunner®	Mercury	Herramienta para hacer pruebas de carga en clientes, servidores y Web.
IxLoad	Ixia	Aplicación para ejecutar pruebas de desempeño que puede emular millones de usuarios. Emula protocolos como HTTP, SSL, FTP, SMTP, POP3, Streaming/VoIP, IPSEC, PPPoX, entre otros.
SilkPerformer	Segue	Aplicación para hacer pruebas de carga.
ANTS	Red Gate	Para realizar pruebas de carga y escalabilidad en servicios Web de .NET y en aplicaciones.
LoadTracer	Trace Technologies	Herramienta para pruebas de carga, desempeño, estrés y escalabilidad de aplicaciones Web. Simula múltiples instancias de un cliente Web entrando a un servidor Web y es compatible con Internet Explorer y Netscape.
Microsoft Application Center Test	Microsoft	Diseñada para realizar pruebas de estrés en servidores Web y analizar el desempeño de la aplicación, incluyendo páginas en ASP y sus componentes. Soporta diferentes esquemas de autenticación y el protocolo SSL, muy útil en pruebas de sitios seguros.
NeoLoad	Neotys	Simula cientos de usuarios virtuales en un sitio Web obteniendo estadísticas de desempeño y revelando los errores que surgen de las pruebas.
OpenSTA	OpenSTA	Software *open source* para generar cargar realmente pesadas para similar la actividad de miles de usuarios virtuales.
Proxy Sniffer	Ing. Fischer	Herramienta de pruebas de carga y estrés que permite analizar las características de desempeño y la estabilidad de una aplicación bajo diferentes condiciones de carga.
Siege	Joe Dog	Herramienta *open source* para pruebas en sistemas UNIX.
TestMaker	PushToTest	Framework *open source* para construir agentes de pruebas inteligentes para probar servicios Web (HTTP, HTTPS, SSL, Servlet, JSP, EJB, ActiveX, SOAP, .NET). Creado en Java, se puede ejecutar en cualquier sistema que permita Java, incluyendo Windows, Linux, Solaris, Macintosh y más.
Web Application Stress Tool	Microsoft	Herramienta que simula múltiples pedidos de diferentes páginas de diferentes browsers
Webload	Radview Software	Herramienta para hacer pruebas de desempeño en aplicaciones en Internet e Intranet.
Web Performance Trainer	Web Performance	Para realizar pruebas de desempeño Web en Windows y UNIX.
Web Server Stress Tool	Paessler	Aplicación para pruebas de carga en servidores Web.
Load Gold	ApTest	Combina herramientas open source con consultoría por parte de expertos y transferencia de tecnología al equipo.
SiteStress	WebMetrics	Un completo servicio para generar transacciones que soporten http y que crecen de cientos a miles.

WebStress	Moniforce	Servicio para realizar pruebas de desempeño Web y pruebas de estrés.

9.11.4. HERRAMIENTAS PARA PRUEBAS EN BASES DE DATOS

Producto	Proveedor	Descripción
AETG	Telcordia	Sus algoritmos usan técnicas de diseño combinatorio para crear sets de prueba mínimos que cubren todas las interacciones entre los datos de entrada.
Data Generator	DTM	Sirve para generar datos, tablas (vistas, procedimientos, etc.) para pruebas en bases de datos.
Datatect	Banner Software	Genera datos de pruebas a archives ASCII o directamente en el RDBMS (sirve para Oracle, Sybase, SQL Server e Informix).
Jenny		Herramienta gratuita similar a AETG y a ALLPAIRS. JENNY indica la cantidad de test cases que se deben generar para probar todas las características de la aplicación.
Jumpstart	Talking Frog	Herramienta de generación de datos para SQL Server.
TestIt!	TdgTeam	Genera gran cantidad de datos de prueba aleatorios, como nombres, empresas, direcciones, ciudades, IDs, etc.
TurboData	Canam Software	Generador gratuito de datos de prueba bastante reales con las *foreing key* resueltas. También genera los Select, Update y Delete en SQL.
utPLSQL		Herramienta *open source*, para pruebas unitarias en Oracle PL/SQL.
GenerateData.com	Benjamin Keen	Generador *on-line* gratuito de hasta 200 datos. Por ejemplo Nombres, Números Telefónicos, Direcciones de Correo Electrónico, Ciudades, Estados, Provincias, Países, Fechas, Direcciones, Rangos de Números, Cadenas Alfanuméricas, textos Lorem Ipsum y más, guardando los datos en formato HTML, XML, Excel o SQL.

9.11.5. HERRAMIENTAS PARA PRUEBAS DE LINKS Y HTML

Producto	Proveedor	Descripción
AccVerify/AccRepair	HiSoftware	Verifica, corrige, monitorea y gestiona el sitio Web y aplicaciones basadas en Web para cumplir con la W3C y la Sección 508.
CSE HTML Validator	AI Internet Solutions	Aplicación para chequear links, escritura y accesibilidad en HTML, XHTML y CSS.
Cyber Spyder Link Test	Aman Software	Programa para la gestión de contenidos Web, usado para verificar las URLs del sitio y para análisis del contenido.
HTML Candy	Anetto Software	Software para la preparación de páginas HTML. Es capaz de arreglar gran cantidad de los problemas con la sintaxis HTML, incluyendo etiquetas, estilos, atributos, valores y elementos obsoletos.

HTML PowerTools	Talicom	Suite de Windows para revisión de HTML.
LinkSleuth	Xenu	Herramienta gratuita para detectar links rotos.

9.11.6. HERRAMIENTAS PARA PRUEBAS DE SERVICIOS HTML

Producto	Proveedor	Descripción
CSSCheck	Web Design Group	Herramienta gratuita *on-line* para revisar hojas de estilos.
HTML Validator	WDG	Servicio *on-line* gratuito que sirve para validar documentos HTML.
HTML Validation Service	W3C	Servicio *on-line* que revisa documentos HTML para asegurar que cumplen con las normas W3C HTML y XHTML.
Link Alarm	LinkAlarm	Servicio *on-line* para detectar links rotos.
NetMechanic	Monte Sano Software	Servicio que ayuda a encontrar links rotos, etiquetas incorrectas y tiempos de respuesta del servidor.
Web Page Purifier	Delorie Software	Utilidad *on-line* que mapea una página a los estándares HTML 2.0, HTML 3.2, HTML 4.0, o WebTV 1.1.
XML Validation	Scholarly Technology Group	Aplicación que valida documentos XML.

9.11.7. HERRAMIENTAS PARA PRUEBAS DE SEGURIDAD

Producto	Proveedor	Descripción
QA Inspect	SPI Dynamics	Incorpora pruebas de seguridad Web automatizadas en el proceso de administración de pruebas en general. Los usuarios de Mercury pueden gestionar tanto las pruebas funcionales como las pruebas de seguridad desde una única plataforma.

9.11.8. HERRAMIENTAS PARA CONTROL DE DEFECTOS

Producto	Proveedor	Descripción
AceProject	Websystems	Software para el control de defectos, diseñado para gerentes de proyectos y desarrolladores.
ADT Web	Borderwave	Herramienta de control de defectos diseñada para compañías de software pequeñas, medianas y grandes. Permite hacer seguimiento a los defectos y sugerencias por versión, por cliente, etc.
BugAware.com	Bugaware	Solución para control de defectos que incluye notificaciones por email, base de conocimientos, reportes dinámicos, gestión del equipo, hilos de discusión entre los usuarios, etc.
BugHost	Active-X.COM	BugHost es un sistema de control de defectos ideal para

		compañías pequeñas y medianas que quieren usar un sistema seguro y basado en Web.
BugRoster	Codeca Corp.	Sistema multiusuario para el control de defectos y reportes.
Bug Tracker	CrimsonLink	Herramienta Web para el control de defectos, ayudando al proceso de gestión de los mismos.
Bugvisor	softwarequality, Inc.	Aplicación parametrizable para capturar, gestionar y comunicar peticiones de cambios, reportar defectos y gestionar todo el flujo de trabajo del proyecto.
Bugzilla	Bugzilla.org	Sistema *open source* para el control de defectos desarrollado por Mozilla.
Census BugTrack	MetaQuest	Herramienta parametrizable para el control y monitoreo de defectos. Incluye integración con VSS, notificaciones, flujo de trabajo, reportes e historial de cambios.
JIRA	Atlassian	Herramienta para el control de defectos y la gestión del proyecto basada en J2EEE.
Jitterbug	Samba	Herramienta libre para el control de defectos.
Mantis		Sistema *open source* liviano y simple, fácilmente modificable, parametrizable y actualizable.

9.11.9. HERRAMIENTAS PARA LA GESTIÓN DE LAS PRUEBAS

Producto	Proveedor	Descripción
ApTest Manager	ApTest	Repositorio basado en Web, permite realizar la ejecución, la generación de reportes, pruebas de regresión, control de acceso, cronograma, reportes técnicos y para la gerencia, asignación de pruebas a los testers y gestión de defectos.
QADirector	Compuware	Gestión del proceso de pruebas.
SilkPlan Pro	Segue Software	Sistema para gestión de la calidad que permite la gestión de las pruebas y un framework de colaboración diseñado para aplicaciones de gran tamaño.
T-Plan Professional	T-Plan, Limited	Herramienta para gestionar el proceso de pruebas.
TestDirector™	Mercury	Sistema para la gestión de las pruebas.
HP Quality Center	HP	Gestiona y controla los procesos de calidad y automatiza las pruebas de software en el entorno de las aplicaciones y TI.

9.11.10. HERRAMIENTAS PARA PRUEBAS DE COMUNICACIONES

Producto	Proveedor	Descripción
Chariot	netIQ	Herramienta para hacer pruebas en redes.
Cheetah	Tollgrade	Esta herramienta permite probar y monitorear el desempeño de aplicaciones sobre VoIP y VoD.
Emulation Engine XT	Communication Machinery Corporation	Ejecuta pruebas sofisticadas de desempeño, capacidad, escalabilidad y estrés de productos 802.11 y arquitecturas Wireless LAN.
Fault Factory	ExtraData	Permite reproducir errores relacionados con sockets o con

		fallas en HTTP/SOAP.
iSoftTechTAS	iSoftTech	Ayuda a automatizar pruebas en redes.
LANTraffic	Omnicor	Permite generar automáticamente tráfico TCP y UDP, parametrizando la carga.
NuStreams 2000	Omnicor	Permite la realización de pruebas de LAN, WAN, en redes, switches, routers e IPs. Prueba, mide y certifica las redes y dispositivos.
Silvercreek SNMP Test Suite	Interworking Labs, Inc.	Verifica el cumplimiento de estándares, manejo de errores y excepciones, condiciones límite y desempeño con cargas pesadas en la red.
WAN Emulator	Itheon	Permite recrear redes con ambientes segmentados, por ejemplo, WAN, WLAN, etc., para probar aplicaciones, incluyendo VoIP, en fase de desarrollo, de pruebas o antes del despliegue.

9.11.11. HERRAMIENTAS PARA GESTIÓN DE REQUISITOS

Producto	Proveedor	Descripción
Analyst Pro	Goda Software	Potente herramienta para gestión de requisitos, que incluye la posibilidad de crear y modificar especificaciones y diagramas.
Caliber	Borland	Ayuda a los equipos a hacer seguimiento de cada etapa del proyecto con bastante precisión. Tiene una interfaz intuitiva y soporta el proceso de toma de decisiones.
RequisitePro	IBM	Herramienta de gestión de requisitos y casos de uso para equipos que quieran mejorar la comunicación de las metas del proyecto, mejorar el desarrollo colaborativo, reducir los riesgos del proyecto e incrementar la calidad de las aplicaciones antes de ser puestas en producción.
SteelTrace	SteelTrace	Herramienta de captura de requisitos que permite a todos los involucrados en el negocio trabajar colaborativamente para entender, definir, comunicar y gestionar los requisitos del software.

9.11.12. OTRAS HERRAMIENTAS

Producto	Proveedor	Descripción
Aprobe	OC Systems	Herramienta para depurar que recolecta y analiza datos instrumentando el archivo ejecutable.
Bug Shot	BugStomper Software	Programa de captura de pantalla diseñado especialmente para testers de software. Facilita el proceso de captura de imágenes de la pantalla como parte del proceso de reporte de errores porque sirve para capturar, guardar, editar, comentar e imprimir capturas de pantalla.
Exchange Simulator	Aegis Software	Simula pruebas entre diferentes interfaces de diferentes protocolos.
InSpec	ISM	Sirve para gestionar el proceso de revisión y *walkthrough* en código fuente.
KaNest	KaSYS	Software para hacer pruebas transaccionales y de comunicaciones

		entre dos sistemas a nivel de aplicación.
LogStomper	BugStomper Software	Ayuda a automatizar el proceso de revisión de logs.
TestBench400	Original Software	Automatización de pruebas en AS/400.

9.12. FORMATOS

A continuación se encuentran plantillas que apoyan el proceso de testing. Se recomienda usar y adaptar cada una según las necesidades de la organización.

9.12.1. FORMATO PARA EL PLAN DE PRUEBAS

	NOMBRE EMPRESA	***NOMBRE DEL PROYECTO***
		PLAN DE PRUEBAS

1. PROPÓSITO DEL DOCUMENTO

El propósito de este documento es definir la estrategia, cronograma y alcance que será cubierto durante la etapa de validación por parte de <*Nombre Empresa*> al <*Aplicativo*> del <*Cliente*>. Este plan incluye tipos de pruebas a realizar, documentación, requerimientos a ser probados según el alcance definido, guía para el reporte de incidentes, metodología, herramientas y criterios de aceptación.

2. GLOSARIO

A continuación se presentan algunos términos utilizados en este documento:

TÉRMINO	DEFINICIÓN

3. ALCANCE Y TIPOS DE PRUEBAS A EJECUTAR

Los siguientes son los tipos de pruebas que hacen parte del alcance de este plan de pruebas. Para cada tipo de prueba se define la funcionalidad o entregable que será validado y los artefactos que se toman como insumo para verificar el cumplimiento de los requerimientos, lineamientos y criterios de aceptación definidos y aprobados por el cliente.

Tipo de prueba:	*Funcional, de Usabilidad, de Regresión, de Desempeño, etc.*
Objetivo:	*Qué se espera de la ejecución de las pruebas, es decir, el objetivo es encontrar tantos defectos como sea posible, encontrar los problemas importantes rápidamente, identificar, percibir y atacar los riegos de calidad, certificar un estándar o verificar una especificación.*
Alcance:	*Listar funcionalidades, casos de uso, requisitos, etc. que serán objeto de las pruebas.*
Insumos:	*Listar los ítems que servirán de entrada al proceso de pruebas.*
Estrategia:	*Cómo se van a abordar las pruebas, qué elementos se van a probar primero, etc.*
Datos de prueba:	*Definir si serán generados por el equipo de pruebas, si los va a proporcionar el cliente, etc.*
Herramientas:	*Herramientas que serán utilizadas en las pruebas.*

4. ÍTEMS FUERA DEL ALCANCE DE LAS PRUEBAS

Los siguientes elementos no están considerados dentro del alcance de este plan de pruebas, ni de las pruebas a ejecutar:

(Listar los ítems y justificar)

5. ENTREGABLES DEL PROYECTO

Los artefactos que evidencian la ejecución de las pruebas y los resultados obtenidos son:

- *Plan de pruebas.*
- *Casos de pruebas que incluyen los resultados de la ejecución.*
- *Errores reportados y gestionados en la herramienta de seguimiento de defectos definida para el proyecto.*
- *Listas de chequeo de pruebas.*
- *Scripts generados.*
- *Resultados de las pruebas.*
- *Otros (Especifique los demás entregables que considere)*

6. ESTÁNDARES Y PROCEDIMIENTOS

Para la ejecución de las pruebas, normalmente se utiliza la metodología definida en el proceso de la organización y el reporte, clasificación y gestión de incidentes se sigue de acuerdo con lo definido en el proceso de gestión de defectos.

Si la metodología a seguir es la definida por el cliente o una acordada en conjunto, debe referenciarse el documento donde quedó dicha definición.

Cualquier desviación de estos estándares y procedimientos debe quedar explícita en esta sección.

7. HERRAMIENTAS A UTILIZAR

Listar las herramientas de uso general:

- *Microsoft Office 2010*
- *Algún repositorio de información*
- *Base de datos*
- *Herramienta para la gestión de los defectos*

8. AMBIENTE DE PRUEBAS

Para la realización de las pruebas se tiene definido que el equipo de *<pruebas -*

desarrollo> realizará el despliegue de la aplicación, tanto del sistema como de los componentes de la base de datos.

Las descripciones de cada uno de los componentes de la arquitectura definida para el sistema, son:

- *Definir de acuerdo con la arquitectura los requisitos de hardware, software y configuración requerida para cada uno de los componentes de la arquitectura.*

9. ORDEN DE EJECUCIÓN

Las actividades de pruebas se listan en esta sección para definir apropiadamente el orden en las cuales se ejecutarán, por ejemplo:

1. *Pruebas de humo por parte de desarrollo antes de entregar la aplicación.*
2. *Pruebas de aceptación por parte del equipo de testing para poder aceptar el producto.*
3. *Validación de los defectos que ya fueron corregidos.*
4. *Ejecución de las pruebas.*
5. *Ejecución de las pruebas de regresión.*

10. CRITERIO DE SUSPENSIÓN Y REACTIVACIÓN DE LAS PRUEBAS

Las pruebas pueden ser suspendidas bajo las siguientes circunstancias:

- *El ambiente de pruebas se encuentra inestable. Esto puede deberse a que no se entregó la versión correcta, por problemas de instalación de la aplicación o de infraestructura.*
- *Cuando no se cuenta con el conjunto de datos apto para las pruebas.*
- *Las pruebas unitarias en la versión entregada no se ejecutaron exitosamente.*

Las pruebas se reanudarán una vez la situación que impide el inicio de las pruebas sea subsanada.

Defina bajo cuáles otras circunstancias las pruebas pueden suspenderse o restablecerse.

11. CRITERIO DE SUSPENSIÓN Y REACTIVACIÓN DE LAS PRUEBAS PARA UN CASO DE USO O REQUISITO

Las pruebas de un caso de uso o requisito definido dentro del alcance de este plan de pruebas pueden ser suspendidas bajo las siguientes circunstancias:

- *Cuando lo definido en desarrollo no concuerda con lo validado en la aplicación.*
- *Cuando la ruta básica del caso de uso o requisito no pasa la validación.*
- *Por baja calidad del caso de uso o requisito entregado: el analista de pruebas*

con base en la cantidad de incidentes, severidad y momento de ejecución de las pruebas define y sustenta que no es viable continuar con la ejecución debido a la baja calidad.

Además de reportar la incidencia el analista de pruebas informa de la situación al gerente del proyecto.

Para reactivar las pruebas del caso de uso o requisito el analista recibe una nueva versión de la aplicación y por medio de una ejecución exitosa de la ruta básica y de confirmar que lo definido en desarrollo concuerda con la aplicación, decide que se pueden iniciar nuevamente las pruebas.

Defina bajo cuáles circunstancias las pruebas pueden suspenderse o restablecerse.

12. CRITERIOS DE FINALIZACIÓN DE LAS PRUEBAS

- *Todos los casos de prueba fueron ejecutados al menos una vez.*
- *Todos los errores de las siguientes severidades se encuentran cerrados: Grave, Mayor, Normal.*

Defina bajo cuáles otros criterios pueden considerarse que las pruebas se encuentran completas.

13. SUPUESTOS Y RESTRICCIONES

Liste los supuestos y restricciones bajo las cuales se lleva a cabo la ejecución del plan de pruebas.

14. CRONOGRAMA

En esa sección se listan las fases, las fechas y los entregables de cada fase.

15. EQUIPO

Para cada rol que se vaya a tener en el equipo se debe listar el responsable de ejecutarlo. También es importante mencionar en esta sección si se requiere de algún tipo de entrenamiento.

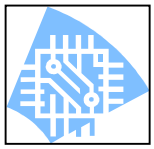
16. RIESGOS

Listar los riesgos del proyecto que puedan afectar el desarrollo de las pruebas, y adicionalmente los riesgos que aplican sólo al proceso de pruebas. Para cada riesgo se debe especificar el respectivo plan de contingencia.

17. MÉTRICAS

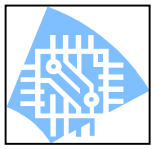
Listar las métricas que se utilizarán en las pruebas, explicando cómo se utilizarán. Debe incluirse el detalle de la frecuencia de la métrica, las personas a las que se les informará y la metodología para revisarla y hacer la respectiva realimentación al equipo.

9.12.2. FORMATO PARA EL DISEÑO DE TEST CASES

	***NOMBRE EMPRESA***	***NOMBRE DEL PROYECTO***
		DISEÑO DE TEST CASES

ID TEST CASE	*Identificación para el test case*
NOMBRE	*Nombre del test case*
AUTOR	*Autor del test case*
FECHA CREACIÓN	*Fecha en que fue creado*
CANDIDATO PARA AUTOMATIZACIÓN	*Sí o No*
MÓDULO/ APLICACIÓN	*Módulo, Aplicación, Caso de Uso, Requerimiento que cubre el test case.*
PRIORIDAD DEL TEST CASE	- *Crítica* - *Alta* - *Estándar*
PRERREQUISITOS	*Qué condiciones se deben cumplir antes de comenzar la ejecución del test case.*
PASOS	- *Nombre del paso* - *Descripción* - *Resultados esperados*
ARCHIVOS ADJUNTOS	*Si es necesario adjuntar alguna información.*

9.12.3. FORMATO PARA LA EJECUCIÓN DE LAS PRUEBAS

	***NOMBRE EMPRESA***	***NOMBRE DEL PROYECTO***
		EJECUCIÓN DE TEST CASES

ID TEST CASE	*ID del test case que se ejecutó.*	
EJECUCIÓN CICLO 1	*FECHA*	*Fecha en que se ejecutó el test case.*
	RESULTADO	*Pasó/Falló*
	DEFECTO	*Número del defecto si se encontró alguno.*
	COMENTARIOS	*Información adicional de la ejecución.*
EJECUCIÓN CICLO 2	*FECHA*	*…*
	RESULTADO	*…*
	DEFECTO	*…*
	COMENTARIOS	*…*
EJECUCIÓN CICLO 3	*FECHA*	*…*
	RESULTADO	*…*
	DEFECTO	*…*
	COMENTARIOS	*…*

9.12.4. FORMATO PARA EL REPORTE DE DEFECTOS

			NOMBRE EMPRESA				***NOMBRE DEL PROYECTO***								
							LISTADO DE DEFECTOS								
ID	**TESTER**	**FECHA**	**RESUMEN**	**DESCRIPCIÓN**	**SEVERIDAD**	**PROBABILIDAD**	**TIPO**	**ESTADO**	**PRIORIDAD**	**LUGAR**	**VERSIÓN**	**RESPONSABLE**	**PASOS**	**ADJUNTOS**	**NOTAS**

9.12.5. FORMATO PARA LECCIONES APRENDIDAS

	NOMBRE EMPRESA		***NOMBRE DEL PROYECTO***	
			LECCIONES APRENDIDAS	
CLIENTE	**PROYECTO**	**TIPO DE LECCIÓN**	**INCIDENTE**	**DESCRIPCIÓN DE LA LECCIÓN APRENDIDA**
Nombre del Cliente	*Nombre del Proyecto*	*- Qué se hizo bien para repetirlo.* *- Qué se hizo mal para no volverlo a hacer.* *- Qué no se hizo y se debe hacer.*	*Dar una breve descripción de la lección.*	*Describir detalladamente la lección aprendida.*

9.12.6. FORMATO PARA INFORME DE AVANCE

	NOMBRE EMPRESA	***NOMBRE DEL PROYECTO***
		INFORME DE AVANCE

Asunto: Informe de Avance de Pruebas
Proyecto: *Nombre del Proyecto*
Ciclo de Prueba: *Ciclo de prueba actual*

PERIODO CORRESPONDIENTE A ESTE INFORME *(Se hace referencia al periodo que abarca el informe de avance)*

Fecha Inicio:	<dd/mm/yyyy>	**Fecha Fin:**	<dd/mm/yyyy>
Fecha Creación:	<dd/mm/yyyy>	**Fecha Envío:**	<dd/mm/yyyy>
Responsable:	*Nombre del analista de pruebas*		

ACTIVIDADES REALIZADAS

Actividad	**Fecha**		**Avance (%)**	
	Inicio	**Fin**	**Respecto al total**	**Respecto a lo programado**

COMENTARIOS GENERALES

-

REGISTRO DE DIFICULTADES

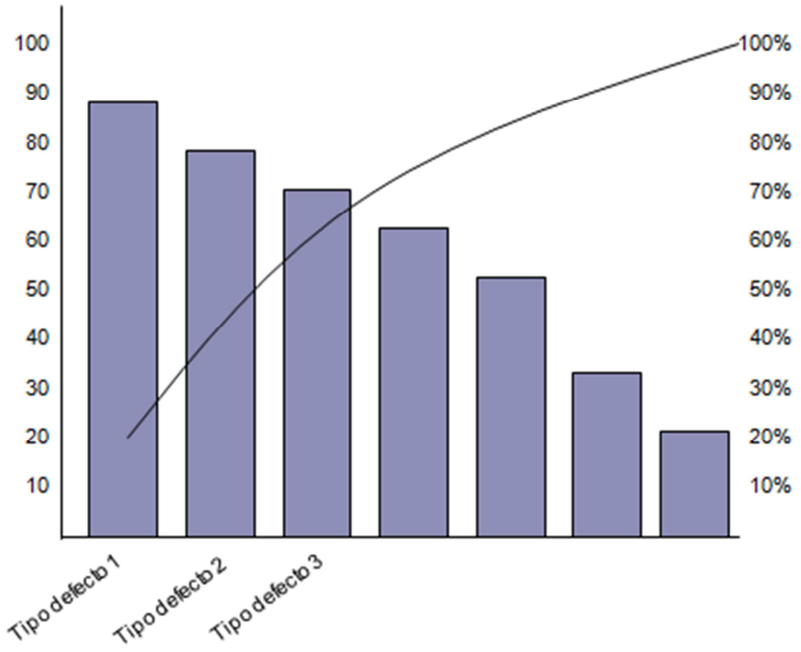
Fecha	Analista de Pruebas	Detalle de la Dificultad	Prioridad	Atendido
<dd/mm/yyyy>	*Nombre del analista de pruebas.*	*Detalle de la dificultad.*	*Alta/ Media/ Baja*	*Sí/No*

9.12.7. FORMATO PARA ANÁLISIS DE CAUSAS

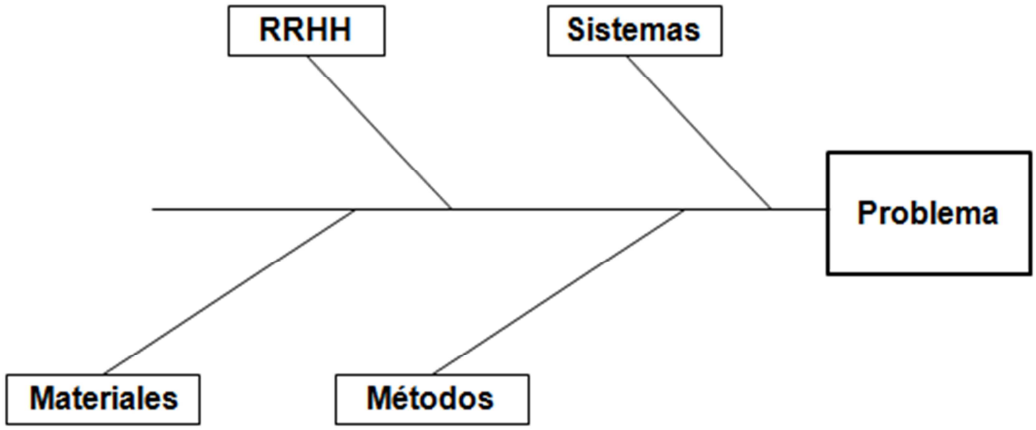
	NOMBRE EMPRESA	*NOMBRE DEL PROYECTO*
		ANÁLISIS DE CAUSAS

DIAGRAMA DE PARETO

TIPO DE DEFECTO	CANTIDAD	PARTICIPACIÓN	% ACUMULADO
Tipo defecto 1			
Tipo defecto 2			
…			
TOTAL	**∑ CANTIDAD**		**100%**



ESPINA DE PESCADO



5 POR QUÉ

PROBLEMA	1 POR QUÉ	2 POR QUÉ	3 POR QUÉ	4 POR QUÉ	5 POR QUÉ

PLAN DE ACCIÓN

CAUSA	QUÉ	CÓMO	QUIÉN	CUÁNDO
Descripción del problema.	*Qué hacer para solucionar la causa.*	*Cómo hacer para solucionar el problema. Pasos a seguir.*	*Responsable de implementar el plan.*	*Fecha planeada.*

9.12.8. FORMATO PARA CARTA DE CERTIFICACIÓN

	NOMBRE EMPRESA	***NOMBRE DEL PROYECTO***
		CARTA DE CERTIFICACIÓN DE PRUEBAS

NOMBRE CLIENTE

NOMBRE PROYECTO

Fecha finalización de la prueba:	*<dd/mm/yyyy>*
Fecha entrega documentación pruebas:	*<dd/mm/yyyy>*
Analista de Pruebas Líder:	*Nombre del analista de pruebas.*
Líder de Pruebas:	*Nombre del líder del equipo de pruebas.*
Alcance: *Incluye iteración, módulo, caso de uso, etc. definido en los entregables del proyecto o en su defecto lo que el gerente de proyecto acordó con el cliente para la entrega.*	
Ambiente de pruebas: *Incluye la descripción del ambiente en el que se realizaron las pruebas.*	
Datos de prueba: *Incluye los tipos de datos de prueba, usuarios, accesos, permisos etc. usados durante las pruebas.*	

Nombre Empresa certifica que al *"Nombre Proyecto/Producto"* se le realizó un proceso de pruebas formales de acuerdo con la información recibida por medio de reuniones con los analistas y desarrolladores, más la documentación elaborada para soportar el producto.

Se entrega la aplicación con los siguientes defectos identificados, los cuales están siendo atendidos por el equipo de desarrollo. *Se describen todas las incidencias identificadas que aún no se han solucionado. Por cada incidente se agregan los acuerdos con el equipo y las acciones posteriores. Ejemplo:*

- *Defecto #####: Descripción corta del error. Plan de acción.*
-

Se certifican los siguientes Casos de Uso/Módulos:

- *Se listan los casos de uso, requerimientos, controles de cambio y demás artefactos verificados.*

-

Líder de Pruebas	Analistas de Pruebas

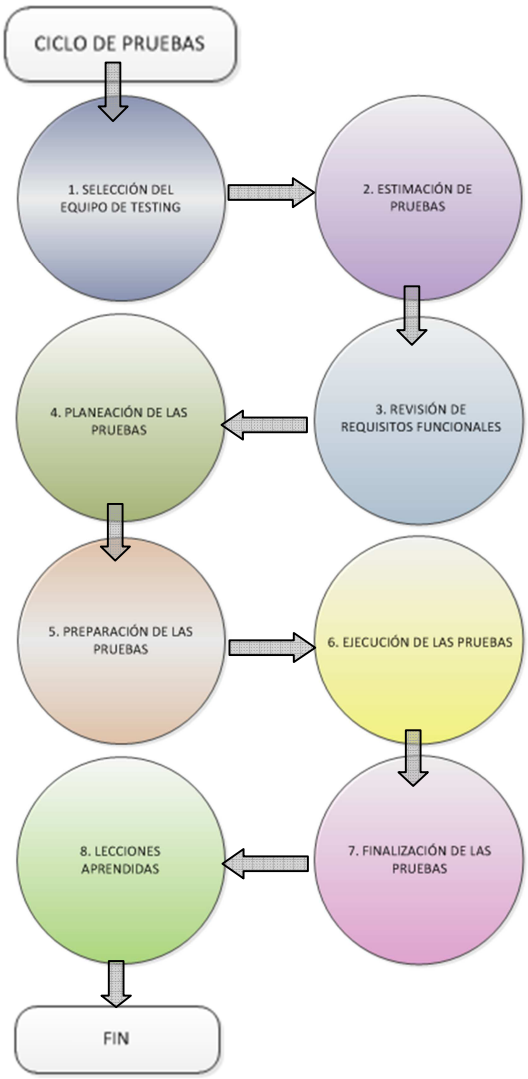
9.12.9. FORMATO PARA MATRIZ DE TRAZABILIDAD

	NOMBRE EMPRESA	*NOMBRE DEL PROYECTO*
		MATRIZ DE TRAZABILIDAD

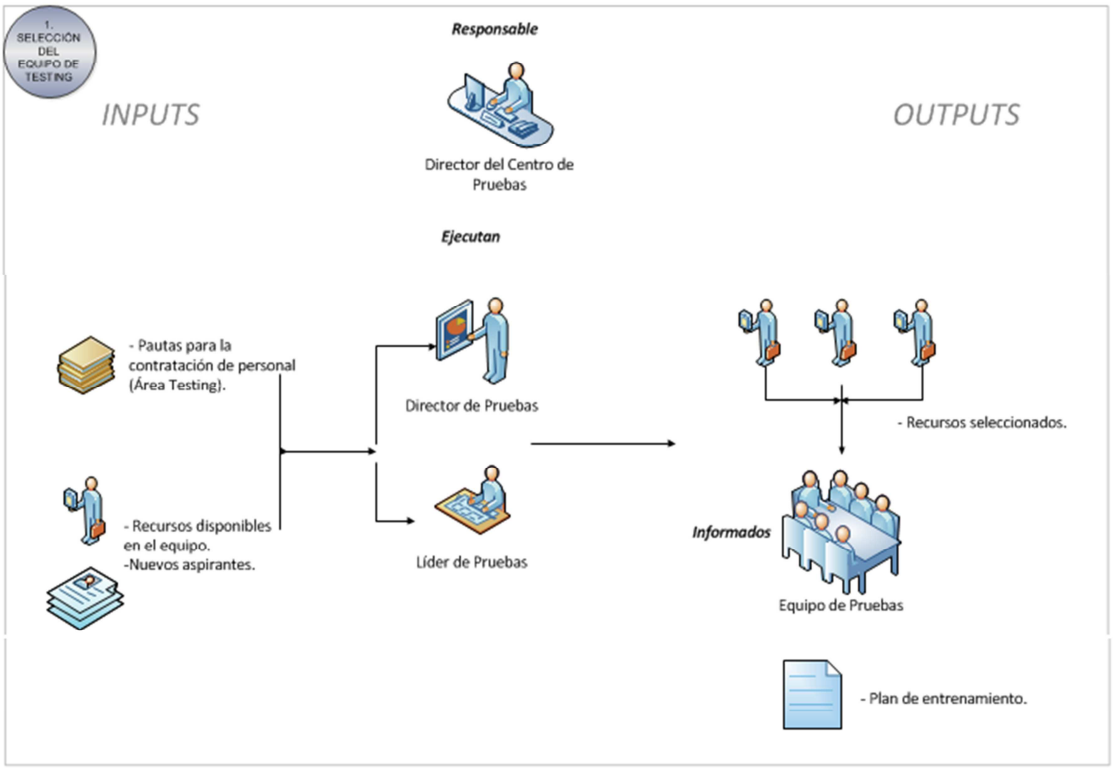
ESCENARIO DE NEGOCIO		**ESCENARIO DE PRUEBA**			
ID REQUISITO	**NOMBRE REQUISITO**	**NOMBRE ESCENARIO DE PRUEBAS**	**ID TEST CASE**	**NOMBRE TEST CASE**	**COMENTARIOS**

10. MAPA SENSITIVO

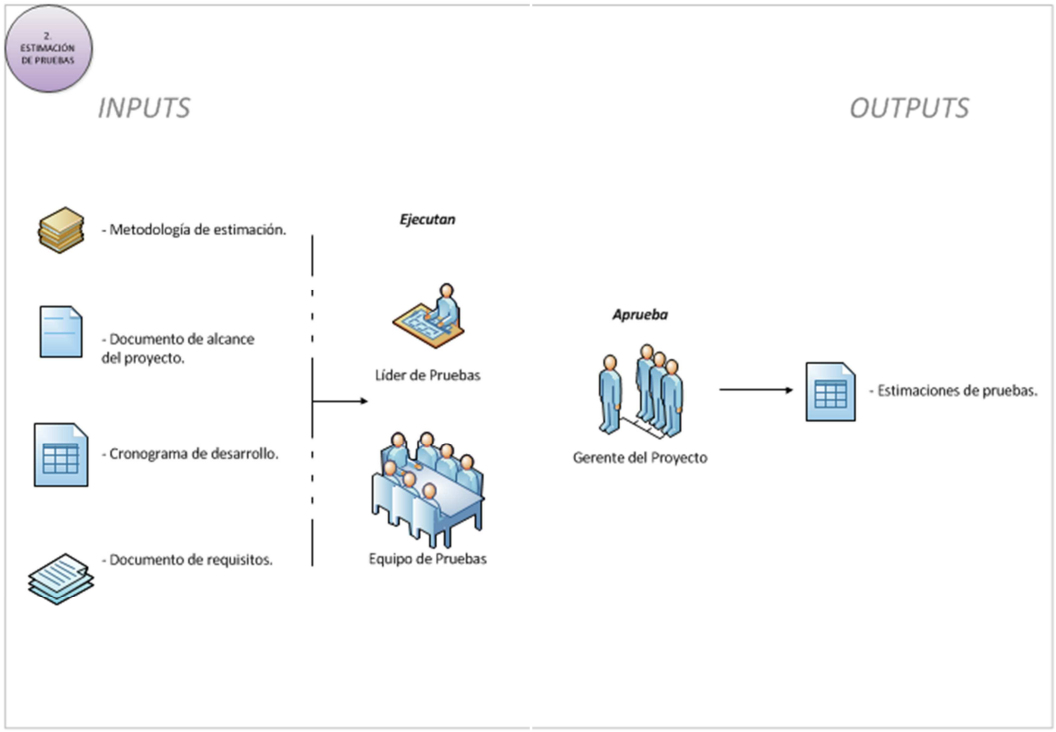
El mapa sensitivo que se encuentra a continuación, complementa la información del framework del proceso de testing de software que engloba la teoría, permitiendo visualizar las actividades, los roles, las entradas y las salidas del proceso. Para ver el detalle de una sección siga el link correspondiente.



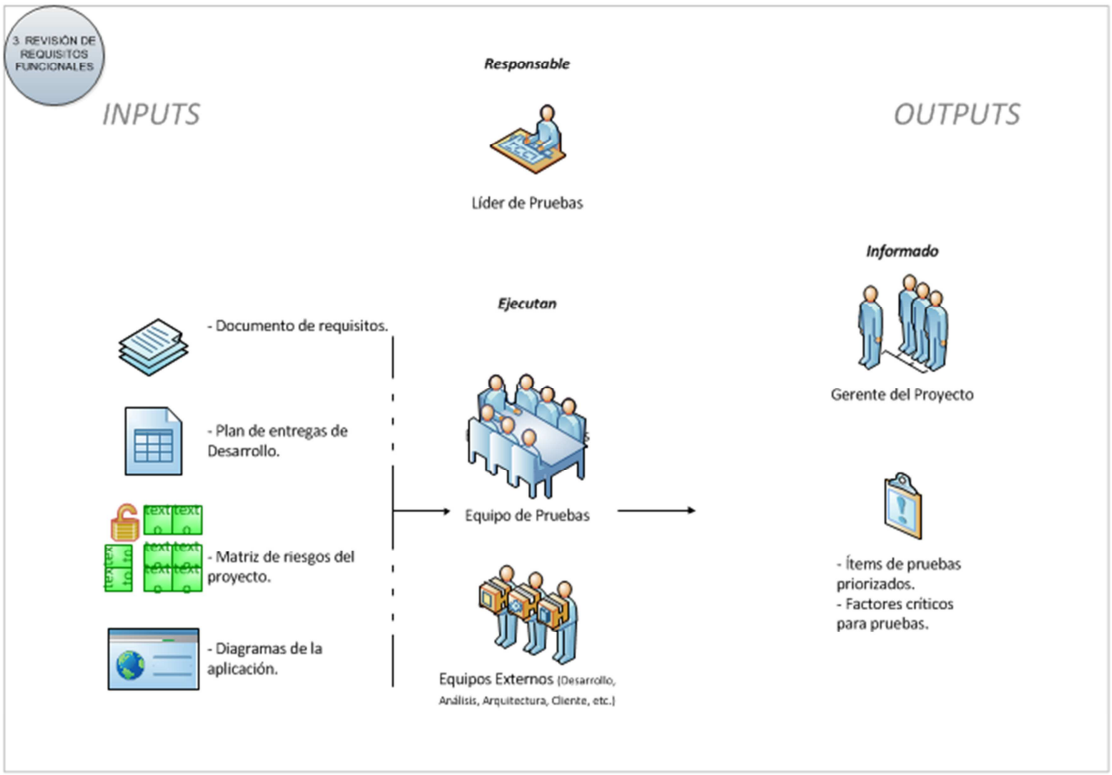
10.1. SELECCIÓN DEL EQUIPO DE TESTING



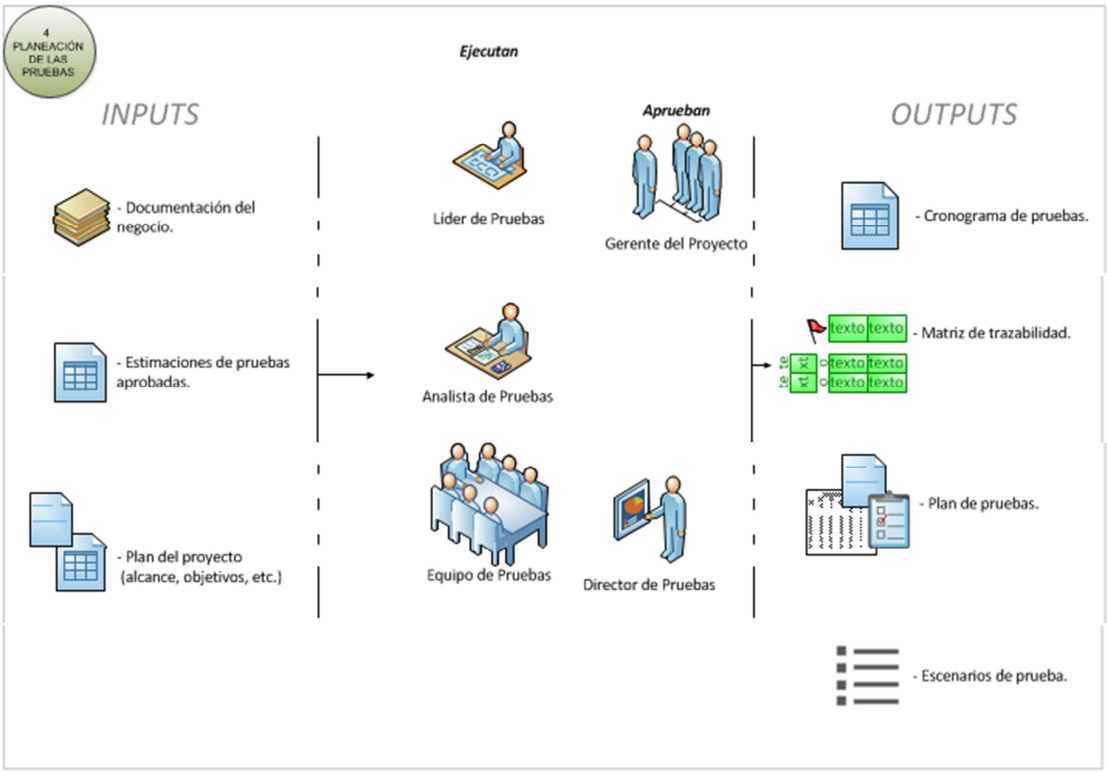
10.2. ESTIMACIÓN DE PRUEBAS



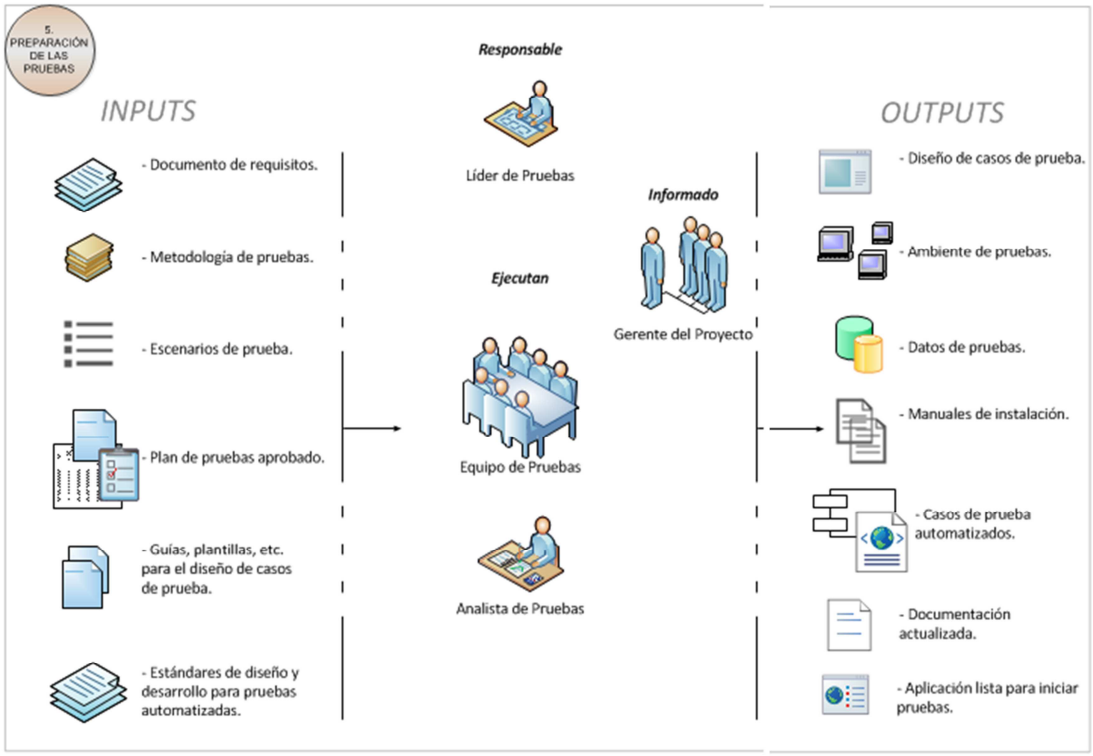
10.3. REVISIÓN DE REQUISITOS FUNCIONALES



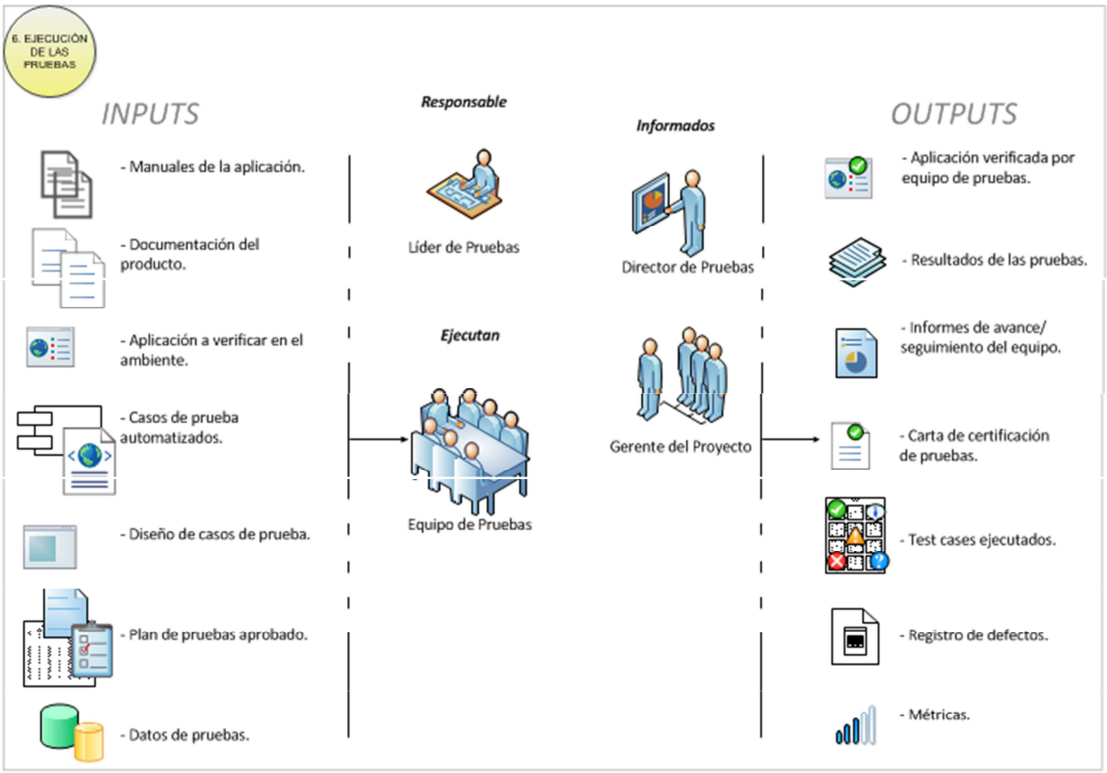
10.4. PLANEACIÓN DE LAS PRUEBAS



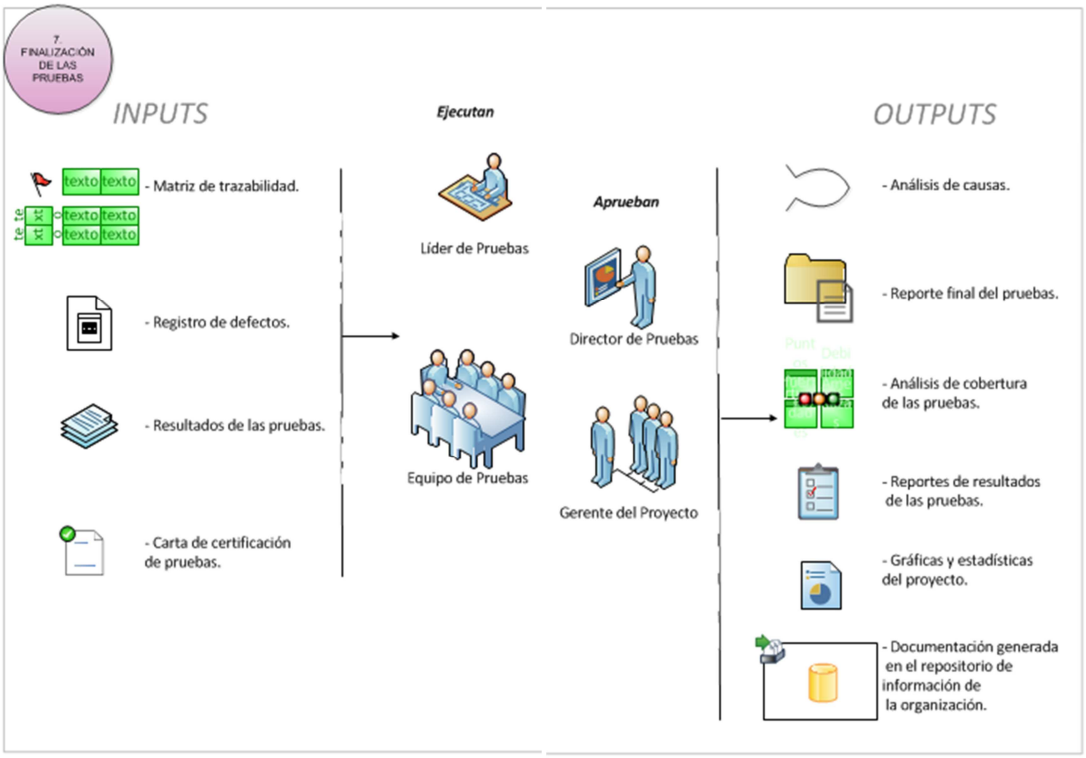
10.5. PREPARACIÓN DE LAS PRUEBAS



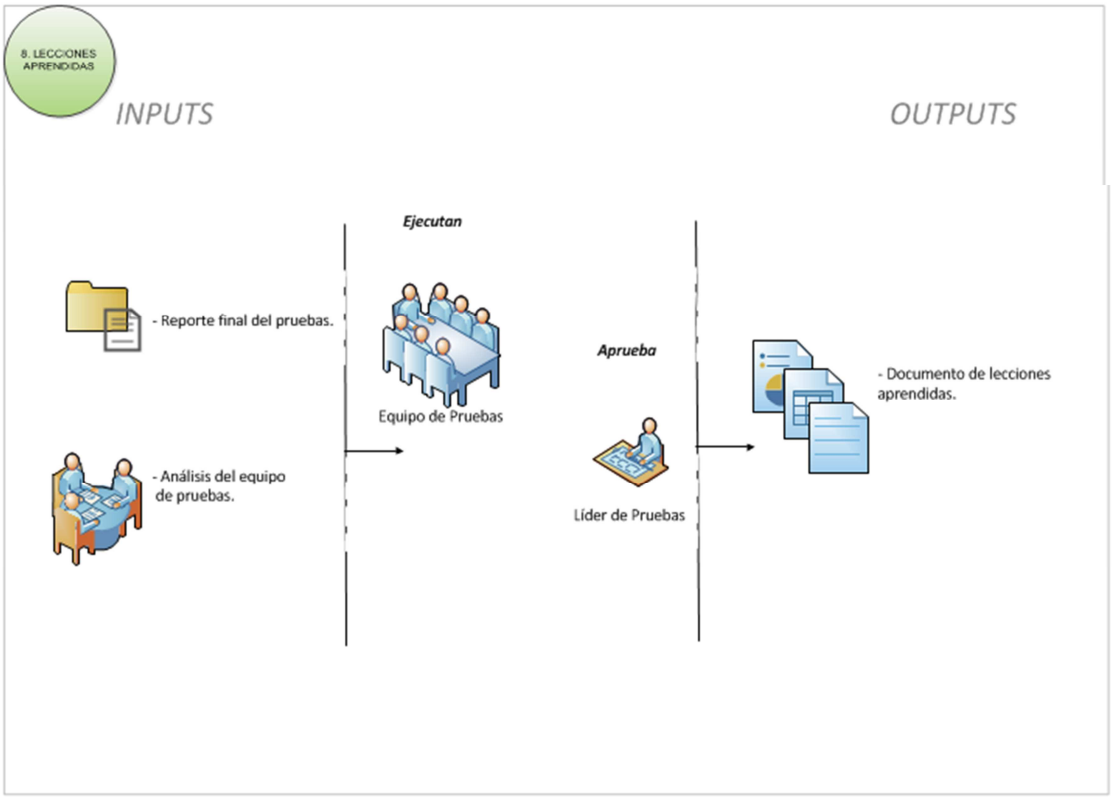
10.6. EJECUCIÓN DE LAS PRUEBAS



10.7. FINALIZACIÓN DE LAS PRUEBAS



10.8. LECCIONES APRENDIDAS



11. CONCLUSIONES

El concepto de calidad ha tomado tal importancia en estas últimas décadas que su aplicación está más allá de algún tipo de organización en particular o de algún producto específico; el concepto de calidad es una filosofía aplicable en todos los tipos de organizaciones y alcanza a tocar a cada persona perteneciente a la organización; así pues, que la calidad tiene que ver con todo y con todos, y es el factor que permite a las organizaciones asegurar su permanencia en el mercado, especialmente en el desafiante mercado del software.

El testing debe estar integrado en todo el proceso de desarrollo para poder aportar a la calidad del producto. La formalidad con la cual sea realizado es fundamental para entregar a los clientes productos con excelente calidad, así, el proceso de desarrollo y el de pruebas deben estar soportados por una metodología formal para ejecutar el proyecto.

Actualmente el testing de software puede apoyarse en una creciente cantidad de metodologías y aplicaciones diseñadas exclusivamente para la etapa de pruebas, incluyendo la gestión del proceso de software testing, la administración y seguimiento de defectos, la administración de los casos de prueba, la automatización de pruebas, etc.

Las personas y los proyectos se gestionan por objetivos. Las personas tienden a alinear sus planes con los objetivos establecidos por la gerencia y por los *stakeholders* involucrados, un ejemplo de esto sería encontrar defectos o confirmar que el software funciona, con lo cual cambia completamente el enfoque de las pruebas y la actitud de equipo. Por lo tanto, es importante expresar claramente el objetivo de las pruebas. Identificar fallas durante las pruebas puede ser percibido como una crítica dirigida contra el producto o contra el autor. Por esto mismo, las pruebas son percibidas como una actividad destructiva, aunque

para la calidad del producto es una actividad constructiva. Si los errores, defectos o fallas son comunicados de manera constructiva, los roces entre los testers y los analistas, diseñadores y desarrolladores pueden ser evitados.

Un punto importante para el éxito de las pruebas de un producto software, es permitirle al equipo de pruebas involucrarse desde el inicio del proyecto, para que entiendan el negocio del cliente, comprendan la importancia y criticidad del producto que van a verificar y finalmente, durante la ejecución, puedan enfocarse en aquellos aspectos más relevantes para el usuario final, desde el punto de vista de la funcionalidad, la usabilidad, el desempeño, etc.

Cada tipo de prueba realizado a un software tiene enfoques distintos, diferentes tipos de personas ejecutando, diferentes estrategias, varias herramientas, diversos equipos involucrados y en especial, distintas bases de conocimiento, por lo tanto, la gestión de un equipo de pruebas debe ser coherente, estructurada y organizada.

Los temas presentados a lo largo de este documento, permiten identificar los elementos críticos del testing. Esto facilita el proceso de gestión del mismo ya que permite diferenciar los aspectos relevantes y da pautas de cómo poner en práctica los conceptos vistos.

El framework propuesto es un modelo que permite a quien esté interesado en el testing incorporar las actividades en su proceso como un primer acercamiento a esta práctica. Además, el contenido de todo el documento, así como las referencias mencionadas permiten profundizar en los conocimientos sobre testing de software según sea necesario. Igualmente se recomienda usar los conceptos expuestos con criterio y teniendo en cuenta el entorno y tipo de organización donde se van a utilizar.

El entendimiento a cabalidad de los temas expuestos requiere de un estudio más profundo; sin embargo, ante la gran abundancia de técnicas, herramientas y metodologías, el framework permite ver todo el panorama sin ser tan confuso como lo podría ser el material que se pueda encontrar en las distintas fuentes.

BIBLIOGRAFÍA

1. Everett, Gerald D., D. & McLeod, Raymond Jr.; (2007) Software Testing: Testing Across the Entire Software Development Life Cycle. Wiley-IEEE Press - Released online.

2. Farrell-Vinay, Peter. (2008) Manage Software Testing. Auerbach Publications, Boca Raton, FL, USA, 527 pp.

3. Kaner, Cem & Falk, Jack & Nguyen, Hung Q. (1999) Testing Computer Software, 2nd Edition. Wiley, New York, NY, USA, 480pp.

4. Myers, Glenford J. (2004) The Art of Software Testing, 2nd Edition. Wiley, New York, NY, USA, 255pp.

5. Pressman, Roger S. (2002) Ingeniería del software, un enfoque práctico. McGrawHill, 5ta. Edición, 640pp.

6. Rodríguez, Juan David & López Paula Andrea; (2004) Metodología para la realización de Pruebas de Software. Proyecto de Grado. Ingeniería de Sistemas. Departamento de Informática y Sistemas, Escuela de Ingeniería, Universidad EAFIT, 125pp.

Páginas Web:

1. http://www.astqb.org

2. www.testingfaqs.org

3. http://www.siliconindia.com/

4. www.wilsonmar.com

5. http://www.computerworld.com/

6. http://www.utest.com

7. https://softwaretechnews.thedacs.com

8. http://v-modell.iabg.de/v-modell-xt-html-english/index.html

9. http://www.quality-testing.com/category/roles

10. http://en.wikipedia.org/wiki/Software_bug

11. http://www.rbcs-us.com/

12. http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node67.html

13. http://v-modell.iabg.de/v-modell-xt-html-english/index.html

14. http://www.tkomarek.com/category/embedded-software/

15. http://www.sqatester.com/qateam/qualitiesofagoodtester.htm

16. http://buildtesting.blogspot.com/2008/01/history-software-testing.html

17. http://www.siliconindia.com/magazine_articles/Software_Testing_The_Next_Big_Employment_Wave-QSUG917969199.html

18. http://www.sistedes.es/TJISBD/Vol-1/No-4/articles/pris-07-perez-gpf.pdf

19. http://www.softwaretestinghelp.com

20. http://www.perftestplus.com/terminology.htm

21. http://www.sistedes.es/TJISBD/Vol-3/No-4/articles/pris-09-yague-agiles.pdf

22. http://www.testingeducation.org/a/nature.pdf