PRUEBAS DE SISTEMAS Y PRUEBAS DE ACEPTACION

Año de la inversión para el desarrollo rural y la seguridad alimentaria

FACULTAD DE INGENIERIA DE SISTEMAS
CICLO: IX

Investigadores: -Valdez Huaraca Abner Gerardo
-Mendoza Valdez Jorge Luis
-Torres Alarcón Stewart Junior
-Pachas Laura Carlos Orlando
-Matías Sebastián Jonathan
-Alfaro Medina Jazmín

FIS-UNICA, 2013

DEDICATORIA

A nuestros padres, a su
comprensión y esfuerzo
Por hacer de nosotros día a día
Mejores personas para
La sociedad.

INTRODUCCION

La aplicación de pruebas de software es actualmente una herramienta importante para llevar el control de la realización de un proyecto de software de calidad y cada vez es más rigurosa estas pruebas por el hecho de la gran competitividad creciente existente en el mercado.

Las pruebas de sistemas conforman uno de los niveles de pruebas de software conjuntamente con diversas pruebas como pruebas recuperación, seguridad, resistencia, desempeño, comunicación, volumen, tensión, disponibilidad de datos, facilidad de uso, operación, entorno, almacenamiento, configuración, instalación y documentación. Cada cual tiene diversa finalidad y con un objetivo en común que demuestre una visión sistemática para proyecto.

Las pruebas de aceptación es otro tipo de nivel que se complementa en los niveles de pruebas de software, estas pruebas son muy fundamentales ya que son las que nos van a permitir que el producto cumpla con los estándares necesarios y que a la vez satisfaga a los usuarios de acuerdo a los requerimientos que plantearon desde un inicio.

Contenido

Índice de Figuras

Índice de tablas

1.- PRUEBA DE SISTEMAS

Las Pruebas de sistemas buscan discrepancias entre el programa y el objetivo o requerimiento, enfocándose en los errores hechos durante la transición del proceso al diseñar la especificación funcional.Esto hace a las pruebas de sistema un proceso vital de pruebas, ya que en términos del producto, número de errores hechos, y severidad de esos errores, es un paso en el ciclo de desarrollo generalmente propenso a la mayoría de los errores.

Figura1: Prueba de sistemas de iterativo

Las Pruebas de sistemas no son procesos para probar las funciones del sistema o del programa completo, porque esta seria redundante con el proceso de las pruebas funcionales. Las pruebas del sistema tienen un propósito particular: para 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, no resuelve sus objetivos o requerimientos.

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

Las pruebas del sistema son una fase de pruebas de investigación, en la que se asegura que cada componente unitario o módulo (identificado en las pruebas unitarias) interactúe con otros componentes o módulos, tal como fue diseñado.

Las pruebas de sistema tienen como objetivo ejercitar profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.

Un problema clásico de la prueba de sistema es “señalar con el dedo”. Esto ocurre cuando se descubre un error y el desarrollador de cada elemento del sistema culpa a los demás. En lugar de caer en este absurdo, el ingeniero del software debe de anticiparse a posibles problemas de la interfaz:

- ✓ diseñar ruta de manejo de errores que prueben toda la información proveniente de otros elementos del sistema.

- ✓ Aplicar una serie de pruebas que simulen datos incorrectos u otros posibles errores en la interfaz de software.
- ✓ Registrar los resultados de la prueba como evidencia en el caso de que se culpe.
- ✓ Participar en la planeación y el diseño de las pruebas del sistemas para asegurar que le software se ha probado adecuadamente.

En realidad, la prueba de sistema abarca una serie de pruebas diferentes cuyo propósito principal es ejecutar profundamente el sistema en cómputo. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se hayan integrado adecuadamente todos los elementos del sistema y que realicen las funciones apropiadas.

1.2 Visión sistémica de la prueba

Cuando se deben realizar pruebas, debe mantenerse un enfoque sistémico, es decir integral, que está detrás de todo desarrollo de software. Al hablar de enfoque sistémico se indica que:

a) Todo sistema tiene una serie de objetivos que le dan sentido. Esos objetivos están asociados con indicadores de éxito que permiten determinar si los objetivos se cumplen y en qué medida.

b) Todo sistema tiene una serie de elementos que lo forman y la interacción de tales elementos se orienta a satisfacer los objetivos.

c) Todo sistema tiene una frontera que lo separa de un medio ambiente. Los elementos de ese medio ambiente influyen sobre el sistema proporcionándoles una serie de entradas y obteniendo del mismo un conjunto de salidas.

d) Ningún sistema existe en aislamiento; siempre interaccionan con otros sistemas constituyendo un sistema mayor.

Al aplicar esos conceptos a la prueba de software, se obtienen una serie de principios que servirán de base para la prueba:

1. Debe asegurarse de conocer con precisión los objetivos del software a probar, así como sus indicadores de éxito. Estos elementos se localizan en los documentos obtenidos en la etapa de recolección de requerimientos, así como en las especificaciones del software. Esta información será indispensable para preparar el plan de pruebas y será la base para iniciar el desarrollo de los casos de prueba.

2. Deben determinarse las entradas y salidas del sistema aprobar. Éste aspecto es necesario en lapreparación de los casos de prueba ytambién en el establecimiento deprocedimientos de

prueba, orientados especialmente a los casos de prueba que muestran el cumplimiento de los objetivos.

3. Considerar el sistema mayor donde opera el software a probar.Generalmente es un ambiente organizacional, formado por elementos de hardware, de software y personas (usuarios). Todos estos elementos influyen mucho sobre el sistema y ayudan especialmente en la preparación de casos de prueba de situaciones no deseadas, relacionadas con datos inadecuados, ausencia de elementos necesarios y ocurrencia de excepciones.

1.3 Vista general de la prueba de sistemas

El proceso de prueba de un sistema tiene dos etapas que pueden estar muy separadas en el tiempo: la preparación de las pruebas y la aplicación de las mismas. La primera está muy ligada a la obtención de requerimientos, por lo que ocurre en las primeras etapas del proyecto, mientras que la segunda requiere del sistema completo o al menos una ***integración***, como se denomina a un producto parcial, aún no liberado, para poder aplicar las pruebas, por lo que ocurre en etapas avanzadas del proyecto. La situación exacta de estas partes depende del modelo de ciclo de vida que se haya elegido.

La etapa de preparación de pruebas incluye al menos tres actividades:

a) preparar un plan de pruebas,

b) preparar una lista de verificaciones de los requerimientos

c) preparar casos de prueba.

Para ejecutar la segunda y la tercera actividades se requiere contar con el documento de requerimientos.

La primera etapa de pruebas provee retroalimentación para el análisis de requerimientos, identificando huecos, ambigüedades y otros problemas. También provee valiosas sugerencias para el diseño y la implementación del sistema, si apenas está desarrollando.

La etapa de aplicación de pruebas requiere del plan de pruebas y de una versión del sistema que sea ejecutable (una integración). Sobre ésta se aplicarán los casos de prueba que se prepararon, se analizarán los resultados y se identificarán posibles defectos.

Esta segunda etapa provee retroalimentación a la implementación y al diseño, mostrando posibles defectos que deben ser corregidos. También provee información que será de utilidad en la liberación del sistema, su aceptación, la estimación de su confiabilidad y para su mantenimiento.

En la Figura 1 se muestra el proceso de prueba de sistema y su relación con otros procesos.

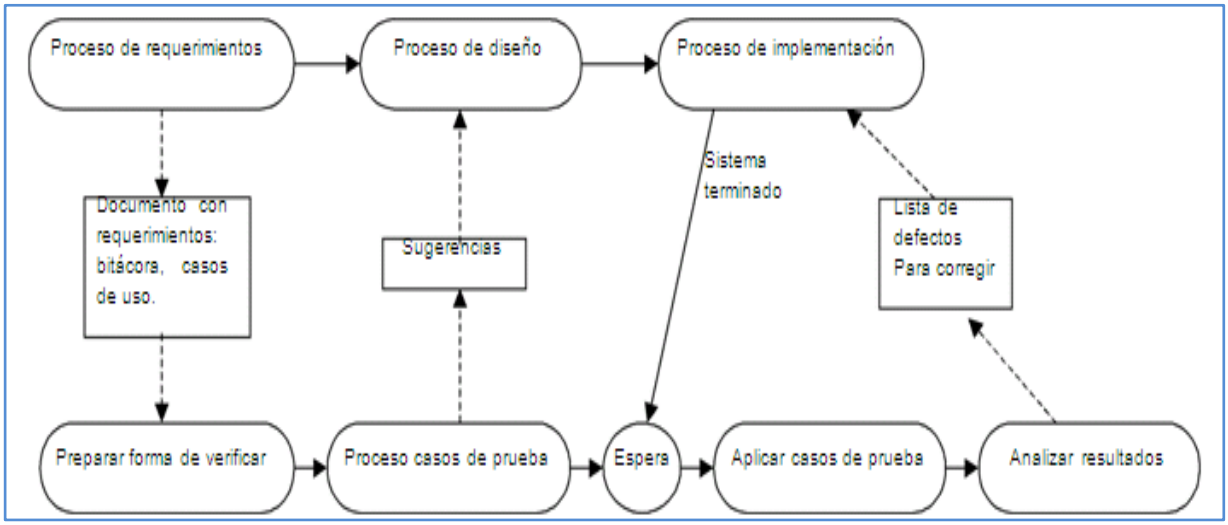

Figura 2.- Proceso de prueba de sistema

La prueba de sistemas tiene varias suposiciones importantes:

a) Cada unidad que forma el sistema ha sido probada por separado y se han eliminado sus defectos.

b) Las interfaces humano-computadoras (de texto o gráficas) han sido probadas también por separado.

c) Se han realizado pruebas de integración para analizar la interacción entre partes del sistema y se han eliminado los defectos identificados.

El segundo punto es importante, ya que algunas veces se confunde la prueba de sistema con la prueba de la interfaz. La primera verifica la interacción de todas las partes, mientras que la segunda únicamente analiza los elementos de la interfaz y posiblemente el manejo de eventos asociados. Sin embargo, las herramientas que ayudan a la prueba de interfaz pueden utilizarse para iniciar las pruebas de sistema.

1.4Plan de prueba

En la sección anterior se dio una idea general de lo que es un caso de prueba y cómo se utilizan en la práctica informalmente. Surgen varias preguntas: ¿cuántos casos serán suficientes? ¿Cómo generar los menos posibles? ¿Qué valores son adecuados?

1.4.1 Plan de pruebas

El plan de pruebas es un documento muy importante dentro del proceso de prueba del software. En él se explican los propósitos y enfoques de las pruebas, el plan de trabajo, los procedimientos operacionales, las herramientas necesarias y las responsabilidades. La extensión y detalle del plan debe adecuarse al proyecto y a las necesidades de la empresa, pudiendo usarse como guía el estándar IEEE 829[4]. A continuación se muestra una propuesta mínima de contenido, para proyectos pequeños y medianos.

a) Identificación del plan de pruebas y el sistema al que se aplica

b) Elementos a probar: qué módulos, clases, casos de uso se van a probar;

Cuando se emplea desarrollo iterativo, deben especificarse las prestaciones (funcionalidades) a probar y cuáles no se probarán (ya sea que se probaron antes o que se implementarán después).

c) Enfoque: vista general de la estrategia de prueba.

d) Criterio de aceptación o rechazo de un caso de prueba: criterio para dar por bueno o malo un caso de prueba al ser ejecutado.

e) Criterio de suspensión: ya sea por tiempo o por cobertura.

f) Productos a entregar: desde el propio plan, los casos y procedimientos de prueba, los resultados.

g) Tareas a realizar para satisfacer el proceso.

h) Necesidades ambientales: hardware, software y espacio de trabajo necesarios.

i) Responsabilidades: quién es responsable de cada cosa.

j) Personal necesario y si requieren entrenamiento.

k) Calendario: tiempos e hitos en el proceso.

l) Riesgos y contingencias que pueden ocurrir en el proceso de prueba.

1.4.2 Lista de verificaciones

Para comenzar el proceso de pruebas del sistema se parte del documento de requerimientos. En caso que no exista y se deba evaluar un sistema para aceptarlo o seleccionarlo de entre un conjunto, habrá que preparar dicho documento, aun cuando sea extemporáneo.

[5]FComputer Society. Computer, 2004

Un documento de requerimientos debe contener la lista de las funciones que se desea realice el software,describiéndolas y priorizándolas; también debe incluir los requerimientos no funcionales, que pueden incluir aspectos organizacionales, de rendimiento y otros.

Un documento de requerimientos bien preparado debe proveer, para cada requerimiento, una forma de verificar que se satisface. En el caso de las funciones, será una descripción y en caso de requerimientos no funcionales pueden ser especificaciones muy precisas, como puede ser el tiempo de respuesta.

Por el momento concentraremos la atención en los requerimientos funcionales, dejando los otros para una sección posterior.

La actividad de preparar lista de verificación incluye los pasos siguientes:

a) asegurarse que para cada requerimiento exista una descripción de la manera en que se verificará; si no existe, debe desarrollarse. Una buena descripción debe contener al menos el funcionamiento típico de la función a que corresponde y los principales comportamientos alternos: variaciones menos frecuentes, respuestaante datos incompletos y fallas del ambiente.

b) revisión de las descripciones: cada descripción debe revisarse para asegurarse que se entiende claramente, que efectivamente es realizable.

1.5 ¿Por qué son importantes las pruebas de sistemas?

Las pruebas de sistemas son importantes por lo siguiente:

- Es donde se prueba el sistema como un todo dentro del ciclo de vida del desarrollo del sistema.
- El sistema es probado para verificar si cumple con sus requisitos funcionales y técnicos.
- El sistema es probado en un entorno lo más parecido al entorno de producción.
- Las pruebas de sistema permiten probar, verificar y validar, tanto los requisitos de negocios como la arquitectura de la aplicación.

1.6Tipo de Pruebas de Sistemas

Pruebas Funcionales

- **Prueba de Integración.-**En las que el equipo de prueba tiene acceso al código fuente del sistema. Cuando se descubre un problema, el equipo de integración intenta encontrar la

fuente del problema e identificar los componentes que tienen ser depurados. Las pruebas de integración se ocupan principalmente de encontrar defectos en el sistema.

- **Prueba de entrega.-** Aquí se prueba una versión del sistema que podría ser entregada al usuario. Aquí el equipo de pruebas se ocupa de validar que el sistema satisface sus requerimientos y con asegurar que el sistema es confiable. Las pruebas de entrega son normalmente prueba de caja negra en las que el equipo de prueba se ocupa simplemente de demostrar si el sistema funciona o no correctamente.

Pruebas No Funcionales

- Pruebas de comunicaciones.
- Pruebas de recuperación
- Pruebas de volumen.
- Pruebas de sobrecarga.
- Pruebas de tensión.
- Pruebas de disponibilidad de datos.
- Pruebas de facilidad de uso.
- Pruebas de operación.
- Pruebas de entorno.
- Pruebas de seguridad.
- Pruebas de usabilidad.
- Prueba de Rendimiento
- Pruebas de Implantación
- Pruebas de instalación
- Pruebas de Desempeño
- Pruebas de Resistencia
- Pruebas de almacenamiento.
- Pruebas de configuración.
- Pruebas de instalación.
- Pruebas de la documentación.

1.5.1 Pruebas Funcionales

A) Pruebas de Integración:

El proceso de integración del sistema implica construir este a partir de sus componentes y probar el sistema resultante para encontrar problemas que pueden surgir debido a la integración de los componentes. Los componentes que se integran pueden ser componentes comerciales,

componentes reutilizables que han sido adaptados a un sistema particular, o componentes nuevos desarrollados. Para muchos sistemas grandes, es probable que usen los tres tipos de componentes. La prueba de integración comprueba que estos componentes funcionen realmente juntos, son llamados correctamente y transfieren los datos correctos en el tiempo preciso a través de sus interfaces.

La integración del sistema implica identificar grupos de componentes que proporcionan alguna funcionalidad del sistema e integrar estos añadiendo códigos para hacer que funcione conjuntamente. Algunas veces, primero se desarrolla el esqueleto del sistema en su totalidad, y se le añaden los componentes. Esto se denomina integración descendente. De forma alternativa, pueden integrarse primero los componentes de infraestructura que proporcionan servicios comunes, tales como el acceso a bases de datos y redes, y a continuación pueden añadirse los componentes funcionales. Esta es la integración ascendente. En la práctica, para muchos sistemas, la estrategia de integración es una mezcla para ambas, añadiendo en incrementos componentes de infraestructura y componentes funcionales. En ambas aproximaciones de integración, normalmente tiene que desarrollarse código adicional para simular otros componentes y permitir que el sistema se ejecute.

La principal dificultad que surge durante las pruebas de integración es la localización de los errores. Existen interacciones complejas entre los componentes del sistema, y cuando se descubra una salida anómala, puede resultar difícil identificar donde ha ocurrido el error. Para hacer más fácil la localización de errores, siempre debería utilizarse una aproximación incremental para la integración y pruebas del sistema.

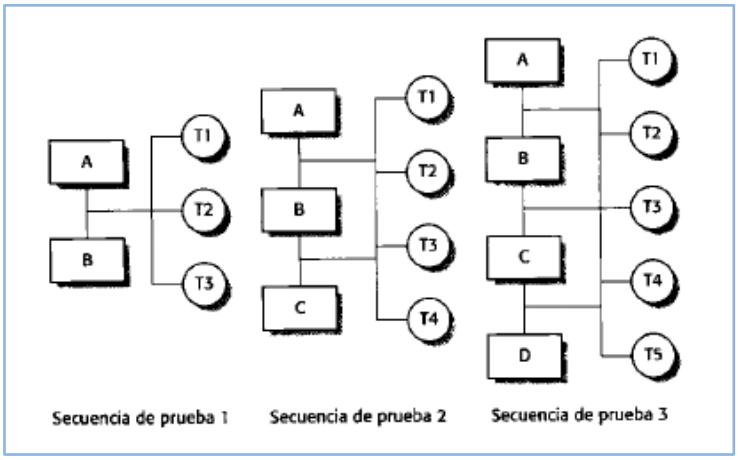

Figura 3.- Secuencia de Prueba

B) Pruebas de Entrega

Las Pruebas de Entrega son el proceso de probar una entrega del sistema que será distribuido a los clientes. El principal objetivo de este proceso es incrementar la confianza del suministrador en que el sistema satisface sus requerimientos. Si es así este puede entregarse como un producto o ser entregado al cliente. Para demostrar que el sistema satisface sus requerimientos, tiene que

mostrarse que éste entregue la funcionalidad especificada, rendimiento y confiabilidad, y que no falla durante su uso normal.

Las pruebas de entregas son normalmente un proceso de pruebas de caja negra en las que las pruebas se derivan a partir de la especificación del sistema. El sistema se trata como una caja negra cuyo comportamiento solo debe ser determinado estudiando su entradas y salidas relacionadas. Otro nombre para esto es Pruebas funcionales, debido a que el probador solo le interesa la funcionalidad y no la implementación del software.

En la siguiente figura veremos la ilustración del modelo de un sistema que admite en la prueba de caja negra. El probador presenta las entradas al componente o al sistema y examina las correspondientes.

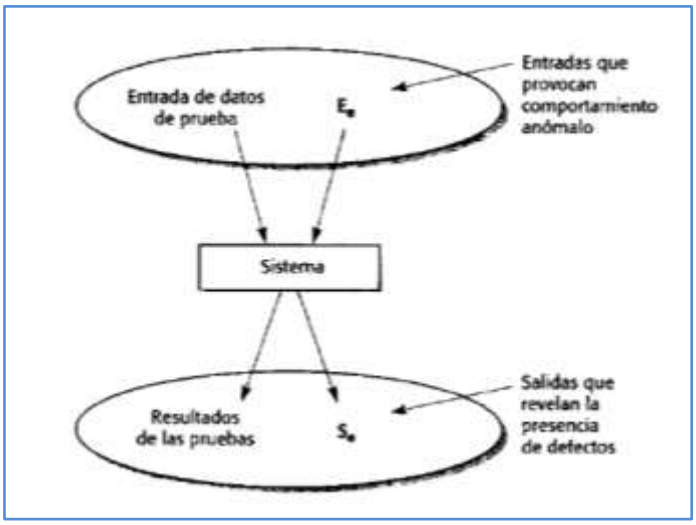

Figura del Pruebas de Entrega

1.5.2 Pruebas No Funcionales

A) Pruebas de Comunicación:

Determinan que las interfaces entre los componentes del sistema funcionan adecuadamente, tanto a través de dispositivos remotos, como locales. Así mismo, se han de probar las interfaces hombre/máquina.

B) Pruebas de Recuperación:

Muchos sistemas de cómputo deben de recuperarse de fallas y reanudar el procesamiento en el tiempo determinado. En algunos casos, un sistema debe ser tolerante con las fallas; es decir, las fallas de procesamiento no deben llevar a la caída del sistema, en general. En otros casos una

falta del sistema debe de corregir dentro de un periodo específico o se sufrirá un fuerte daño económico.

La prueba de recuperación es una prueba de sistema que obliga al software a fallar de varias maneras y a verificar que la recuperación se realice apropiadamente. Si la recuperación es automática (la realiza el propio sistema) debe de evaluarse que sean correctos la re-inicialización, los mecanismos de respaldo del sistema, la recuperación de datos y el nuevo arranque. Si la recuperación requiere de intervención humana, se debe evaluar el tiempo medio de recuperación (TMR) para determinar si se encuentra dentro de los límites aceptables.

Figura 4.- Una Muestra se Recuperación de sistemas

C) Pruebas de Volumen:

Consisten en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos, simulando las cargas de trabajo esperadas.

D) Pruebas de sobrecarga:

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.

E) Pruebas de tensión:

La prueba de tensión es poner al programa a grandes cargas o tensiones. Esto no se debe confundir con la prueba de volumen; una gran tensión es volumen máximo de datos, o de actividad, en un tiempo corto. Una analogía sería evaluar a un mecanógrafo. Una prueba de volumen se determinaría si el mecanógrafo hiciera frente a un bosquejo de un informe grande; una prueba de tensión se determinaría si el mecanógrafo puede mecanografiar a un índice de 50 palabras por minuto.

F) Pruebas de disponibilidad de datos:

Consisten en demostrar que el sistema puede recuperarse ante fallos, tanto de equipo físico como lógico, sin comprometer la integridad de los datos.

G) Pruebas de facilidad de uso:

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

H) Pruebas de operación:

Consisten en comprobar la correcta implementación de los procedimientos de operación, incluyendo la planificación y control de trabajos, arranque y re arranque del sistema.

I) Pruebas de entorno:

Consisten en verificar las interacciones del sistema con otros sistemas dentro del mismo entorno.

J) Pruebas de seguridad:

Cualquier sistema de cómputo que maneje información confidencial o que desencadenen acciones que dañen o beneficien inapropiadamente a los individuos es un blanco para irrupciones impropias o ilegales. La irrupción abarca un amplio rango de actividades:

- ✓ Hacker que tratan de entrar en los sistemas por juego.
- ✓ Empleados disgustados que tratan de irrumpir como forma de venganza.
- ✓ Individuos deshonestos que buscan ganancias personales ilícitas.

La prueba de seguridad comprueba que los mecanismos de protección integrados en el sistema realmente lo protejan de irrupciones inapropiadas.

“los sistemas se deben de probarse para la seguridad del sistema para asegurar que es invulnerable a los ataques frontales, pero también a los perpetrados por los flancos o las retaguardia”.

Durante la prueba de seguridad, quien la aplica desempeña el papel del individuo que desea entrar en el sistema. ¡Todo vale! Debe de tratar de obtener contraseñas por cualquier medio externo; podría atacar el sistema con software personalizados, diseñados para burlar cualquier defensa que se haya construido; podría saturar el sistema, negando así el servicio a otros; podría producir errores intencionales en el sistema para tratar de tener acceso durante la recuperación: podría revisar datos sin protección, con la idea de encontrar la clave de acceso al sistema.

Si se dan el tiempo y los recursos suficientes, una buena prueba de seguridad terminara por irrumpir en el sistema. El papel del diseñador del sistema es que el costo de la irrupción sea mayor que el valor de la información que habrá de obtenerse.

K) Pruebas de usabilidad:

Otra categoría importante de casos de prueba de sistema es la tentativa de encontrar problemas de factores humanos, o usabilidad. Sin embargo, un análisis de factores humanos sigue siendo una cuestión altamente subjetiva.

L) Pruebas de Almacenamiento:

Los programas tienen de vez en cuando objetivos de almacenamiento que indiquen, 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 almacenaje no han sido encontrados.

M) Pruebas de Configuración:

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 de este debe ser probada.

N) Prueba de Instalación:

Algunos tipos de sistemas de software tienen complicados procedimientos de instalación. Las pruebas de los procedimientos de instalación es una parte importante del proceso de prueba del sistema. Esto es particular de un sistema automatizado de instalación que sea parte del paquete del programa. Al funcionar incorrectamente el programa de instalación podría evitar que el usuario tenga una experiencia acertada con el sistema. La primera experiencia de un usuario es cuando él o ella instalan la aplicación. Si esta fase se realiza mal, entonces el usuario/el cliente puede buscar otro producto o tener poca confianza en la validez de la aplicación.

O) Pruebas de Documentación:

Las pruebas de sistema también se refieren a la exactitud de la documentación del usuario. Una manera de lograr esto es utilizar la documentación para determinar la representación de los casos anteriores de prueba del sistema. Esto es, una vez se desea idear el caso de sobrecarga, se utilizaría la documentación como guía para escribir el caso de prueba real. También, la documentación del usuario debe ser el tema de una inspección, comprobándola para saber si hay exactitud y claridad. Cualesquiera de los ejemplos ilustrados en la documentación se deben probar y hacer parte de los casos y alimentarlos al programa.

P) Pruebas de Implantación:

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 en base al cumplimiento de los requisitos no funcionales especificados.

Q) Pruebas de Resistencia:

Los pasos de prueba analizados antes en este trabajo, llevan a una evaluación completa de las funciones y el desempeño normalmente del programa. Las pruebas de resistencia están diseñadas para confrontar los programas en situaciones anormales. En esencia, la persona que realiza la prueba de resistencia se preguntara.

¿Hasta dónde llevar esto antes de que falle?

La prueba de resistencia ejecuta un sistema de tal manera que requiera una cantidad, una frecuencia o volumen anormal de recursos. Por ejemplo:

- ✓ Se diseña pruebas especiales que generen 10 interrupciones por segundo cuando la taza de promedio es de una o dos.
- ✓ Se aumenta la frecuencia de entrada de datos una magnitud que permita como responderá las funciones de entrada.
- ✓ Se ejecutan casos de pruebas que requieran el máximo de memoria u otros recursos.
- ✓ Se diseña casos de pruebas que causen problemas de administración de memoria.

Se crean casos de pruebas que produzcan búsquedas excesivas de datos en el disco.

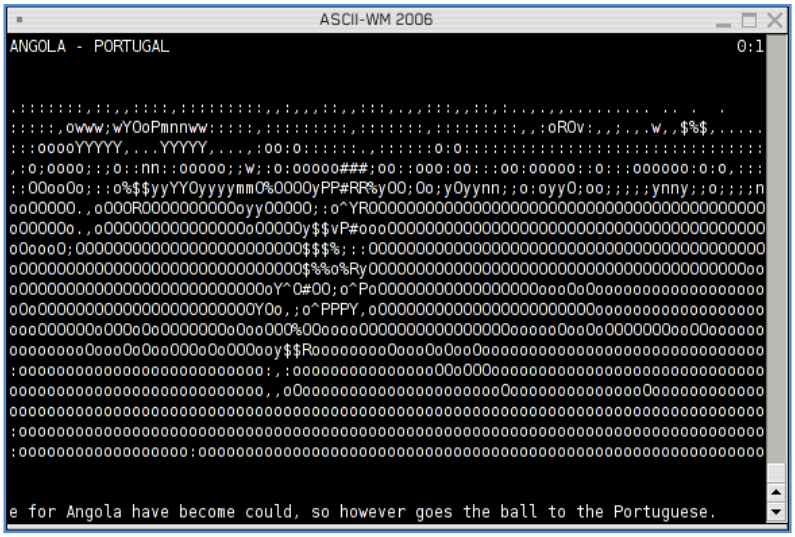

Figura 5.- Muestra de Prueba de Resistencia

R) Pruebas de Desempeño:

La prueba de desempeño está diseñada para probar el desempeño del software en tiempos de ejecución dentro del contexto de un sistema integrado. La prueba de desempeño se aplica en todos los pasos del proceso de la prueba. Incluso al nivel de la unidad. El desempeño de un módulo individual debe de evaluarse mientras se realizan las pruebas. Sin embargo, no es sino hasta que se encuentre totalmente integrado todos los elementos del sistema que es posible asegurar el verdadero desempeño del sistema.

Con frecuencia las pruebas de desempeño se vinculan con pruebas de resistencia y suelen requerir instrucción de software y hardware. Es decir, a menudo resulta necesario medir con exactitud la utilización de recursos (por ejemplo: los ciclos de procesador). Mediante instrumentación externa pueden vigilarse de manera regular los intervalos de ejecución, los eventos que se registran (como las interrupciones) y los estados de muestra del equipo. Si se instrumenta un sistema, la persona que aplica la prueba descubrirá situaciones que lleven a la degradación y posibles fallas del sistema.

2.- PRUEBAS DE ACEPTACIÓN.

Estas pruebas se realizan para que el cliente certifique que el sistema es válido para él. La planificación detallada de estas pruebas debe haberse realizado en etapas tempranas del desarrollo, con el objetivo de utilizar los resultados como indicador de su validez: si se ejecutan las pruebas documentadas a satisfacción del cliente, el producto se considera correcto y, por tanto, adecuado para su puesta en producción.(Isabel Ramos Román, José Javier Dolado Cosín, 2007)[6]

Las pruebas de aceptación(INTECO, 2009)[2], son básicamente pruebas funcionales sobre el sistema completo, y buscan comprobar que se satisfacen los requisitos establecidos. Su ejecución es facultativa del cliente, y en el caso de que no se realicen explícitamente, se dan por incluidas dentro de las pruebas de sistema. Es decir, las pruebas de aceptación son, a menudo, responsabilidad del usuario o del cliente, aunque cualquier persona involucrada en el negocio puede realizarlas. La ejecución de las pruebas de aceptación requiere un entorno de pruebas que represente el entorno de producción.

Esta fase o nivel toma como punto de partida la línea base de aceptación del producto ya instalado en el entorno de certificación. A partir de dicha línea base se acepta el producto, tomando como referencia la especificación de requisitos y comprobando que el sistema cubre satisfactoriamente los requisitos del cliente.(INTECO, 2009)[2]

[2]Instituto Nacional de Tecnologías de Comunicación, 2009
[6]Isabel Ramos Román, José Javier Dolado Cosín, 2007

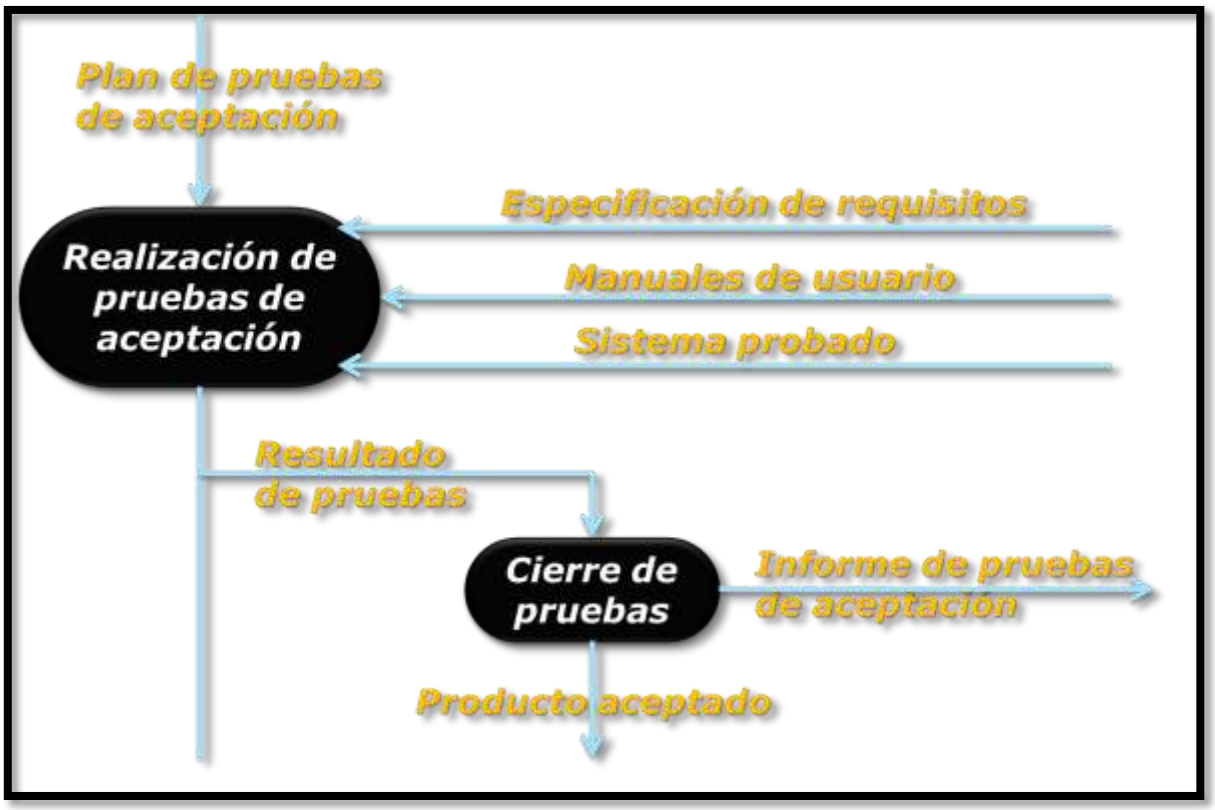

Figura 6.- Flujo de control de pruebas de aceptación.

2.1 Estudio de la situación actual en Pruebas de Aceptación

Dentro del tipo de pruebas que tenemos para validar el software, una de las más importantes son las de aceptación (user's tests). Son aquellas pruebas que son diseñadas por el propio equipo de desarrollo en base a los requisitos funcionales especificados en la fase de análisis para cubrir todo ese espectro, y ejecutadas por el propio usuario final, no por todos evidentemente, pero sí por una cantidad de usuarios finales significativo que den validez y conformidad al producto que se les está entregado en base a lo que se acordó inicialmente.

Dependiendo de la complejidad del sistema a probar, si está o no dividido por módulos, etc. la realización de dichas pruebas se ejecuta de forma diferente. Si estuviera una aplicación dividida en módulos, se tratarían esto como subsistemas y estudiando si tienen o no la suficiente complejidad como para tratarlos de forma diferente, habría que realizar sesiones de prueba de aceptación diferentes.

2.2 Objetivo de la pruebas de aceptación

Las pruebas de aceptación tienen como objetivo obtener la aceptación final del cliente antes de la entrega del producto para su paso a producción. Cuando la organización ha realizado las pruebas

de sistema y ha corregido la mayoría de sus defectos, el sistema será entregado al usuario o al cliente para que dé su aprobación.(INTECO, 2009)

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

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

La validación del sistema se consigue mediante la realización de pruebas de caja negra que demuestran la conformidad con los requisitos y que se recogen en el plan de pruebas, el cual define las verificaciones a realizar y los casos de prueba asociados. Dicho plan está diseñado para asegurar que se satisfacen todos los requisitos funcionales especificados por el usuario teniendo en cuenta también los requisitos no funcionales relacionados con el rendimiento, seguridad de acceso al sistema, a los datos y procesos, así como a los distintos recursos del sistema.

2.3 Generación de las pruebas de aceptación.

El sistema ha de ser aceptado por el usuario. Por tal motivo, a partir de las especificaciones estructuradas del sistema, el analista produce un conjunto de casos de prueba que tendrá que pasar satisfactoriamente. Como las pruebas de aceptación se pueden desarrollar en paralelo con las actividades de diseño e implementación, es normal que esta actividad la inicie el analista nada más finalizar la actividad de "Análisis Estructurado"[3].

2.4 Estrategias de pruebas de aceptación

Si el sistema ha sido desarrollado para el mercado masivo entonces no será práctico probarlo para usuarios o clientes individuales, en algunos casos sería imposible. En estos casos, antes de que el producto se ponga a la venta, es necesaria una retroalimentación[5].

A menudo este tipo de sistemas tiene dos etapas de pruebas de aceptación.

Pruebas alfa y beta

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.

[3] F.Alonso amo, Loic Martínez Normand
[5] Asociación de técnicos de informática, 2010

A) Pruebas alfa (α).

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.

Las pruebas α-alfa consisten en invitar al cliente a que pruebe el sistema en el entorno de desarrollo. Se trabaja en un entorno controlado y el cliente siempre tiene un experto a mano para ayudarle a usar el sistema. El desarrollador va registrando los errores detectados y los problemas de uso.

B) Pruebas beta (β).

Las pruebas β-beta se realizan con posterioridad a las prueba α-alfa, y se desarrollan en el entorno del cliente. En este caso, el cliente se queda a solas con el producto y trata de encontrarle fallos de los que informa al desarrollador**.**

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.

2.5 Entradas, salidas, tareas y roles de pruebas de aceptación.

Entradas	• Especificación de Requisitos. • Manuales de Usuario. • Sistema probado. • Plan de Pruebas.
Tareas	• Preparación del entorno de pruebas. Se recomienda la existencia de un entorno de pruebas específico para la realización de este tipo de pruebas. • Instalación en el entorno de pruebas.

	• Identificación de las pruebas a realizar. • Planificación de las pruebas. Se establecerán las posibles dependencias que hubiera entre pruebas y se establecerá el orden o secuencia de ejecución de las pruebas en base a dichas dependencias. • Ejecución de las pruebas. • Obtención y registro de resultados. • Corrección de fallos y errores detectados. • Reiteración de la tarea hasta superar todas las pruebas. • Elaboración de un Informe de Pruebas de aceptación. • Revisión de la correcta ejecución y resultados de todas las pruebas planteadas. • Creación de la línea base de producción. • Cierre formal de la actividad.
Salidas	• Resultados de pruebas. • Producto aceptado • Informe de Pruebas de aceptación.
Roles	• Ingeniero de Pruebas. • Jefe de Pruebas. • Jefe de Proyecto (Cierre formal de la actividad).

2.6 Tabla de E/S, tareas y roles[2].

El hecho de centrarnos en las pruebas de aceptación, surge de intentar reforzar la visión de que el usuario integrado en dicha fase, de forma temprana, ayudaría a mejorar dicho proceso desde la fase de planificación y diseño de las pruebas, con las consiguientes mejoras de muchos aspectos cuantificables y deseables como pueden ser:

- Aumento en la calidad del software integrado
- Minimización de costes
- Aumento de la fiabilidad en los resultados del proyecto
- Se produce un incremento de la satisfacción del cliente al utilizar un software con una cantidad de errores inferior.
- Se incrementa la eficiencia del proceso de desarrollo.

2.7 Criterios de pruebas de aceptación.

La aceptación del software se logra mediante una serie de pruebas que demuestren que cumple con los requerimientos. Un plan de prueba delinea la clase de prueba que se aplicara y un procedimiento de prueba define los casos de prueba especificas tanto el plan como el procedimiento se diseñan para asegurar que satisfacen todos los requerimientos funcionales, que se alcanzan todos las características de comportamiento, que se cumplen con todos los requisitos de desempeño, que la documentación es correcta y se cumple también con todos los requisitos de facilidades uso y otros requisitos especificados (portabilidad, compatibilidad, recuperación de errores, facilidad de mantenimiento).

2.8 Herramientas para pruebas de aceptación.

Herramienta	Descripción
1. CANOO	**Canoo** es una herramienta de Software Libre para automatizar las pruebas de aplicaciones web de una forma muy efectiva. http://webtest.canoo.com/webtest/manual/WebTestHome.html
2. CONCORDION	**Concordion** es un framework Java de Software Libre que permite convertir especificaciones en texto común sobre requerimientos

	en pruebas automatizadas http://www.dosideas.com/wiki/Concordion
3. FITNESSE	**FitNesse** es una herramienta colaborativa para el desarrollo de software. Permite a los clientes, testers, y programadores aprender lo que su software debería hacer, y automáticamente compara lo que realmente hace. Compara las expectativas de los clientes a los resultados reales http://www.dosideas.com/wiki/FitNesse
4. JBEHAVE	JBehave es un framework Java para mejorar la colaboración entre desarrolladores, QA, analistas de negocio, Dueño Del Producto y todos los miembros del equipo a través de escenarios automatizados y legibles. http://www.dosideas.com/wiki/JBehave
5. JMETER	JMeter es un proyecto de Apache Jakarta que puede ser usado para realizar una Prueba De Carga, para así analizar y mediar la Performance De Aplicaciones de distinto tipo, aunque con especial foco en Aplicaciones Web http://jakarta.apache.org/jmeter/ http://www.dosideas.com/wiki/JMeter
6. SAHI	**Sahi** es una herramienta de automatización de pruebas para aplicaciones Web, con la facilidad de grabar y reproducir scripts. Está desarrollada en Java y JavaScript, la herramienta usa simplemente JavaScript para ejecutar los eventos en el browser. http://www.dosideas.com/wiki/Sahi
7. SELENIUM	**Selenium** es una herramienta de Software Libre para pruebas de aplicaciones Web. Las pruebas de **Selenium** se ejecutan directamente en un navegador y facilitan las pruebas de compatibilidad en navegadores, también como pruebas

	funcionales de aceptación de aplicaciones Web. http://www.dosideas.com/wiki/Selenium
8. SOAPUI	**SoapUI** es una herramienta de Software Libre gráfica, está basada en Java y sirve para el testeo de Web Service y generación de Cliente De Web Service http://www.dosideas.com/wiki/SoapUI
9. WATIR	**Watir** es una librería simple de código abierto para la automatización web en los navegadores. Permite escribir pruebas que sean fáciles de leer y fáciles de mantener. Se ha optimizado para la simplicidad y flexibilidad. http://watir.com/

2.9 Garantía de calidad o control de calidad con respecto a pruebas de aceptación.

Se conoce esta actividad como prueba final o prueba de aceptación. Requiere como entrada los datos de las pruebas de aceptación y el sistema integrado producido en la actividad. La prueba la realizara algún miembro o departamento del usuario, o incluso un departamento independiente de control de calidad. Interesa señalar que es importante realizar actividades de control de calidad en cada una de las actividades anteriores de análisis, diseño e implementación para asegurar que se han realizado con un nivel apropiado de calidad. Así se asegura que el analista está desarrollando especificaciones de calidad, que el diseñador está produciendo diseños de calidad y que el programador está codificando programas de calidad. La actividad de control de calidad es simplemente la prueba final de la calidad del sistema.(F. Alonso Amo, Loic Martínez Normand)

Las pruebas de aceptación comprueba el comportamiento del sistema frente a las necesidades del cliente, sin embargo estos pueden haber sido expresado, los clientes se comprometen, o especificar, las tareas típicas para comprobar que sus exigencias se han cumplido o que la organización ha identificado estos para el mercado objetivo para el software.

3.- IMPLEMENTACION DE PRUEBAS DE SISTEMAS E ACEPTACION

3.1 Implementación de Pruebas de Sistemas

Una implementación o implantación es la realización de una aplicación, o la ejecución de un plan, idea, modelo científico, diseño, especificación, estándar, algoritmo o política.

En ciencias de la computación, una implementación es la realización de una especificación técnica o algoritmos como un programa, componente software, u otro sistema de cómputo. Muchas implementaciones son dadas según a una especificación o un estándar. Por ejemplo, un navegador web respeta (o debe respetar) en su implementación, las especificaciones recomendadas según el World Wide Web Consortium, y las herramientas de desarrollo del software contienen implementaciones de lenguajes de programación.(ALEGSA, 2012)

> En este caso se presenta un ejemplo, basado en el perfil de pruebas de UML, para la implementación en código ejecutable de objetivos de prueba definidos mediante escenarios y variables operacionales.

3.1.1 Generación de objetivos de prueba a partir de casos de uso.

Se resumen trabajos anteriores de los autores para obtener objetivos de prueba, los cuáles son el punto de partida para desarrollar pruebas automáticas.

En el contexto de pruebas del sistema a partir de los casos de uso, un objetivo de prueba puede expresarse como un escenario del caso de uso. Dicho escenario estará compuesto de una secuencia de pasos, sin alternativa posible, y de un conjunto de valores de prueba, así como las pre-condiciones y post-condiciones relevantes para dicho escenario.

Para la generación de los escenarios de prueba, en primer lugar, se construye un diagrama de actividades a partir de la secuencia principal y secuencias erróneas y alternativas del caso de uso. En el diagrama de actividades, se estereotipa las acciones realizadas por el sistema y las acciones realizadas por los actores. Después, se realiza un análisis de caminos y, cada camino del diagrama de actividades, será un escenario del caso de uso y, por tanto, un potencial objetivo de prueba.

3.1.2 Implementación de pruebas del sistema.

A. Una arquitectura de prueba del sistema.

La arquitectura para la ejecución y comprobación automática de pruebas del sistema se muestran en la figura 7

Esta arquitectura es similar a la arquitectura necesaria para la automatización de otros tipos de pruebas, como las pruebas unitarias. La principal diferencia estriba en que, en una prueba unitaria, la propia prueba invoca al código en ejecución, mientras que una prueba funcional del sistema

necesita un mediador (el elemento UserEmulator) que sepa cómo manipular su interfaz externa. (ingsoft , 2007)

UserInterface.- La clase UserInterface representa la interfaz externa del sistema bajo prueba.

UserEmulator.- La clase UserEmulator, define el elemento que podrá interactuar con el sistema utilizando las mismas interfaces que una persona real. Si, por ejemplo, el sistema a prueba es una aplicación web (como en el caso práctico) la clase UserEmulator será capaz de interactuar con el navegador web para indicarle la URL qué tiene que visitar, rellenar formularios, pulsar enlaces, etc. A partir de la interacción de dicha clase con el sistema, se obtendrán uno o varios resultados (clase TestResult), por ejemplo, en el caso del sistema web, se obtendrá código HTML. El perfil de pruebas de UML no define ningún elemento para representar los resultados obtenidos del sistema a prueba, por lo que se ha modelado con una clase sin estereotipar.

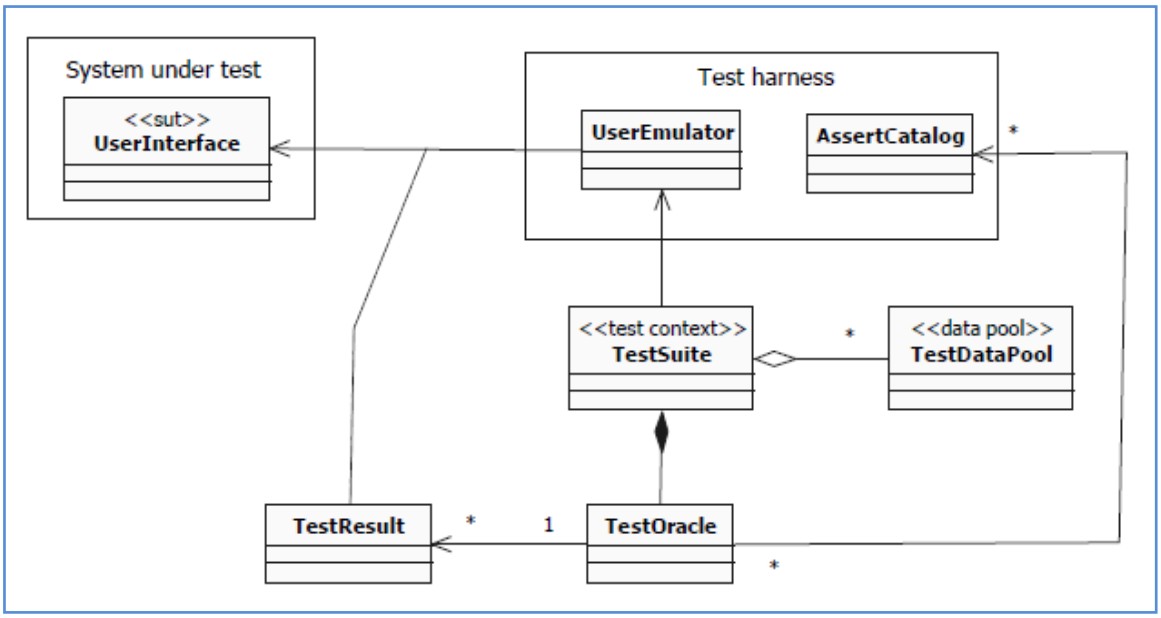

Figura 7 - Arquitectura de prueba de sistema

AssertCatalog.- La clase AssertCatalog define la colección de asertos a disposición de los casos de prueba para determinar si el resultado obtenido del sistema a prueba (clase TestResult) es correcto o no.

Tanto el UserEmulator como el AssertCatalog se han agrupado en un clasificador que representa el test harness, dado que estos dos elementos, suelen ser comunes para todas las prueba de diversos sistemas y existe gran variedad de ofertas en el mercado tanto de pago como libres y gratuitas.

TestSuite.- La clase TestSuite, estereotipada como un Test Context del perfil de pruebas de UML, representa un conjunto de casos de prueba (Test Case en el perfil de UML). En el perfil de pruebas, todo Test Context tiene un elemento Arbiter y un elemento Scheduler, sin embargo, ambos elementos no se van a utilizar en esta propuesta, por lo que han sido omitidos de la figura J1.

TestOracle.- Esta clase sirve de contenedor de todas las acciones de validación (expresadas mediante la ejecución de asertos sobre el resultado obtenido) que determinarán el veredicto de los casos de prueba del contexto de prueba.

TestDataPool.- La clase TestDataPool (estereotipada como Data Pool según el perfil de UML) contendrá un conjunto de métodos Data Selector para seleccionar los distintos valores de prueba según las distintas particiones identificadas en las variables de los casos de uso.

B. Implementación de los casos de prueba

Un caso de prueba es una implementación de un objetivo de prueba. Según el perfil de pruebas de UML, un caso de prueba no es un elemento arquitectónico sino la definición de un comportamiento dentro de un elemento estereotipado como Test Context.

El comportamiento genérico para un caso de prueba se lista en la tabla 1.

1. Invocación del *set up* del caso de pruebas.
2. Invocación del método de prueba
2.1. Ejecución de una acción sobre el sistema.
2.2. Comprobación del resultado de la acción.
3. Invocación del *tear down* del caso de pruebas.

Tabla 1. Comportamiento genérico de un caso de prueba.

Cada caso de uso tendrá asociado un test suite .Dicha suite contendrá las pruebas de todos los escenarios de dicho caso de uso.

Como se ha visto en los objetivos de prueba, en cada uno de los pasos del escenario debe indicarse si es realizado por un actor o por el sistema a prueba. Esta información es muy relevante a la hora de la codificación de los métodos de prueba de la suite.

Todos los pasos realizados por un actor se traducirán en el código del caso de prueba a una interacción entre el caso de prueba y el sistema. El test oracle de un caso de prueba será el conjunto de ValidationActions, o acciones de validación, obtenidas principalmente, a partir de los pasos realizados por el sistema.

Se va a aplicar la técnica de variables operacionales y de categoría-partición para la definición de los valores de prueba necesarios. Se han identificado tres tipos distintos de variables operacionales. Cada uno de los tipos se implementará de manera distinta en los casos de prueba.

El primer tipo lo componen aquellas variables operacionales que indican un suministro de información al sistema por parte de un actor externo.

Para cada variable de este tipo se definirá una nueva clase cuyos objetos contendrán los distintos valores de prueba para dicha variable. Para cada partición del dominio identificada, el elemento TestDataPool tendrá, al menos, una operación que devuelva un valor de prueba perteneciente a dicha partición. Un ejemplo de una variable operacional de este tipo se muestra en el caso práctico.

El segundo tipo lo componen aquellas variables operacionales que indican una selección entre varias opciones que un actor externo tiene disponible. En este caso, no tiene sentido implementar estas variables como métodos del TestDataPool. En su lugar, dicha selección se implementará directamente como parte del código que implementa la interacción entre el actor y el sistema. Un ejemplo de una variable operacional de este tipo se muestra en el caso práctico.

El tercer tipo lo componen aquellas variables operacionales que indican un estado del sistema. Para implementar el método de set up del caso de prueba, se debe escribir el código necesario para establecer adecuadamente el valor de las variables operacionales que describen los estados del sistema, o bien comprobar que dichos valores son los adecuados. De manera análoga, el método tear up debe restaurar dichos valores a sus estados originales. Además, el método tear down debe eliminar, si es procedente, la información introducida por el caso de prueba en el sistema durante la ejecución del caso de prueba. Varios ejemplos de variables operacionales de este tipo se muestran en el caso práctico.

Un caso práctico.

En este caso práctico (Departamento de Lenguajes y Sistemas Informáticos), en primer lugar se aplicará lo visto, para obtener un conjunto de objetivos de prueba a partir de un caso de uso. Después, se definen las características del Test harness utilizado (sección B). Finalmente, se aplica lo visto en las secciones anteriores para implementar un caso de prueba a partir de un objetivo de prueba (sección C). Los artefactos del sistema bajo prueba se han definido en inglés, ya que el español no está soportado por las herramientas utilizadas.

A) Objetivos de prueba.

El caso de uso de la tabla 2, describe la introducción de un nuevo enlace en el sistema. Como complemento, se muestra también el requisito de almacenamiento de información que describe la información manejada por cada enlace (tabla 3). Los patrones usados se describen en la metodología de elicitación NDT. A partir del caso de uso, y de manera automática, se han generado un conjunto de escenarios los cuáles serán los objetivos de prueba de dicho caso de uso. Dado que el caso de uso presenta bucles no acotados, con un número infinito de potenciales repeticiones, criterio de cobertura elegido para obtener los caminos es el criterio 01, el cual

consiste en obtener todos los caminos posibles para una repetición de ninguna o una vez de cada uno de los bucles.

Todos los escenarios obtenidos con este criterio y traducidos al español se listan en la tabla 4. Para este caso práctico, seleccionamos el escenario 09, el cual se describe en detalle en la tabla 5 (dado que el caso de uso se ha redactado en inglés, su escenario principal también se muestra en inglés), para su implementación.

Name	UC-01. Add new link
Precondition	No
Main sequence	1 The user selects the option for introduce a new link. 2 The system recovers all the stored categories and it asks for the information of a link. 3 The user introduces the information of the new link. 4 The system stores the new link.
Alternatives	3.1 At any time, the user can cancel this operation, then this use case ends.
Errors	2.1 If there was an error recovering the categories, then the system shows an error message and this use case ends. 2.2 If there were not categories found, then the system shows and error message and this use case ends. 3.2 If the link name, category or link URL is empty, then the system shows an error message with the result of repeat step 2. 4.1 If there is an error storing the link, then the system shows an error message and this use case ends.
Post condition	A new link is stored.
Notes	The categories that a user may choose, are all the categories registered by an administrator into the system.

Tabla 2. Caso de uso bajo prueba.

Name	SR-01. Link.
Specific data	*Name* / *Domain* Identifier / Integer Name / String Category / Integer URL / String Description / String Approved / Boolean Date / Date and time
Restrictions	The identifier must be unique. Name, category, URL and approved are mandatory Default value for approved is false (0) and for date is the actual date.

Tabla 3. Requisito de información de los enlaces.

También ha sido posible aplicar el método de categoría-partición. Todas las variables operacionales encontradas automáticamente por la herramienta ValueGen se enumeran en la tabla 6. Las particiones para cada una de dichas variables (también encontradas por la herramienta ValueGen) se enumeran en la tabla 7.

En este caso, no se ha continuado refinando el conjunto de particiones aunque para algunas variables, como V04, sí podrían identificarse particiones adicionales.

Para la implementación del escenario de éxito, todas las variables operacionales deben tener un valor perteneciente a las particiones C02.

Escenario	Descripción
01	Aparece un error recuperando las categorías.
02	El usuario cancela la operación.
03	El usuario introduce un enlace incorrecto y, después, aparece un error recuperando las categorías.
04	El usuario introduce un enlace incorrecto y, después, el usuario cancela la operación.
05	El usuario introduce un enlace incorrecto y, después, aparece un error al almacenar el enlace.
06	El usuario introduce un enlace incorrecto y, después, el usuario introduce un enlace correcto.
07	El usuario introduce un enlace incorrecto y, después, el sistema no encuentra ninguna categoría.
08	Aparece un error al almacenar el enlace.
09	El usuario introduce un enlace correcto (*camino principal*).
10	El sistema no encuentra ninguna categoría.

Tabla 4. Escenarios del caso de uso.

Paso	Descripción
01	The user selects the option for introduce a new link.
02	The system recovers all the stored categories and it asks for the information of a link.
03	Not(there was an error recovering the categories) AND Not(there were not categories found)
04	Not(cancel this operation)
05	The user introduces the information of the new link.
06	Not(the link name, category or URL are empty)
07	The system stores the new link.
08	Not(there is an error storing the link)

Tabla 5. Escenario principal.

Variable	Descripción
V01	Error al recuperar categorías.
V02	Categorías encontradas.
V03	Opción del usuario
V04	Datos del enlace
V05	Error al almacenar el enlace,

Tabla 6. Variables identificadas para el caso de uso.

Variable	Particiones
V01	C01: Ocurre un error. C02: No ocurre un error.
V02	C01: No se encontraron categorías. C02: Sí se encontraron categorías.
V03	C01: Cancela la operación. C02: No cancela la operación.
V04	C01: El nombre, categoría o URL están vacías. C02: El enlace es correcto.
V05	C01: Error almacenando el enlace. C02: No ocurre un error.

Tabla 7. Categorías para las variables identificadas.

B) Test harness.

Tal y cómo se describió en la figura 7, el test harness tiene la misión de simular el comportamiento del usuario y ofrecer un conjunto de asertos para evaluar el resultado obtenido.

En este caso práctico, al ser el sistema bajo prueba una aplicación web, es necesario que el test harness sea capaz de comunicarse con el navegador web y sea capaz también de realizar comprobaciones en el código HTML recibido como respuesta. Por ellos, hemos elegido la herramienta de código abierto Selenium (www.openqa.org/selenium) la cuál cumple estas características.

Como se puede ver en la figura 8, Selenium ofrece un interfaz que permite abrir un navegador web e interactuar con él de la misma manera que un actor humano. Respecto al catálogo de asertos, al estar basado en la popular herramienta JUnit, Selenium incorpora el mismo conjunto de asertos que JUnit y, además, funciones para acceder a los resultados visualizados en el navegador web.

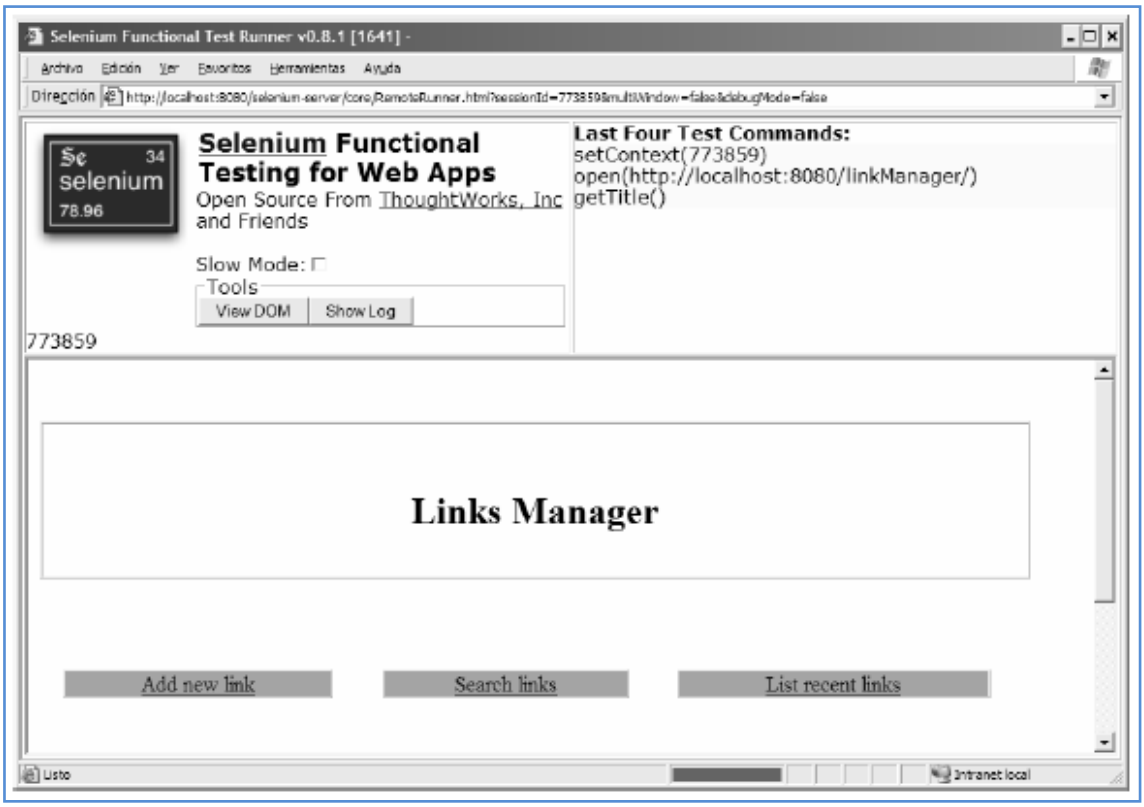

Figura 8. Ejecución del caso de prueba del escenario principal con la herramienta Selenium.

C) Implementación de un caso de prueba

En primer lugar se ha implementado el data pool y los valores de prueba tal y como se muestra en la figura 9.

A continuación, se toman todos los pasos ejecutados por el actor humano y se traducen a código Java para la herramienta Selenium. Dicha traducción se realiza actualmente a mano y se muestra en la tabla 8.

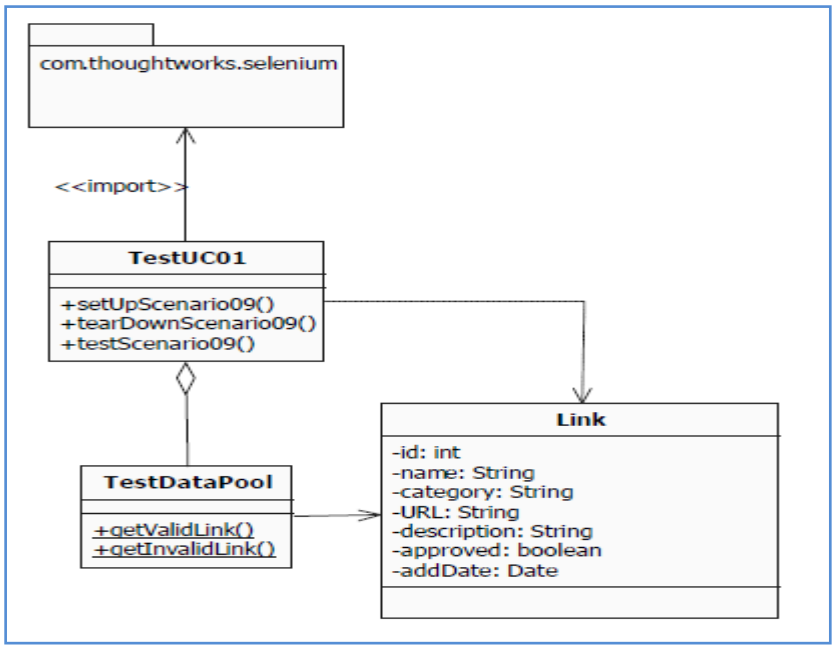

Figura 9. Implementación del caso de prueba.

Paso:	Código:
01: The user selects the option for introduce a new link.	`s.click("AddNewLink");` `s.waitForPageToLoad("5000");`

05: The user introduces the information of the new link.	`Link l = TestDataPool.getValidLink();`
	`s.type("name", l.getName());`
	`s.type("URL", l.getURL());`
	`s.type("description", l.getDescription());`
	`s.click("addEvent_0");`

Tabla 8. Traducción a código ejecutable de los pasos realizados por el usuario en el escenario principal.

A continuación, en la tabla 9, se escribir los asertos a partir de los pasos que realiza el sistema.

Paso:	Código:

02: The system recovers all the stored categories and it asks for the information of a link.	assertEquals("Add new link form", sel.getTitle());
	assertTrue(s.isTextPresent("Name*:"));
	assertTrue(s.isElementPresent("addEvent_name"));
	assertTrue(s.isTextPresent("Category*:"));
	assertTrue(s.isElementPresent("addEvent_category"));
	assertTrue(s.isTextPresent("URL*:"));
	assertTrue(s.isElementPresent("addEvent_URL"));
	assertTrue(s.isTextPresent("Description:"));
	assertTrue(s.isElementPresent("addEvent_description"));
	assertTrue(s.isTextPresent("Date:"));
	assertTrue(s.isElementPresent("addEvent_date"));
	assertTrue(s.isElementPresent("addEvent_0"));

07: The system stores the new link.	assertEquals("Links manager", s.getTitle());
	assertFalse(s.isTextPresent("Error storing new link"));
	assertTrue(isLinkStored(TestDataPool.getValidLink()));

Tabla 9. Traducción a código ejecutable del test Oracle para el escenario principal.

La implementación del método de set-up consistió en comprobar que todas las variables operacionales de la tabla 2 tuvieran un valor de la partición C02. Es decir, comprobar que hay categorías y que no hay ninguna circunstancia que ocasione un error al recuperar las categorías o insertar el nuevo enlace. La implementación del método de tear down, consistió en la restauración del conjunto original de enlaces almacenado en el sistema.

3.2 Implementación de pruebas de aceptación.

Las pruebas de aceptación sólo funcionan con el apoyo de los clientes, o por lo menos un proxy para el cliente, para ayudar a definir los criterios. Sin el conductor de los criterios de aceptación, se hace difícil verificar si usted está construyendo el software correcto. El cliente, junto con todos los miembros del equipo de desarrollo debe reunirse para definir el sistema en términos de una serie de "escenarios" que describen lo que el sistema debe hacer y cómo debe hacerlo.

Mediante la creación de pruebas con requisitos claros y criterios de aprobación, el software se encuentra una mejor oportunidad de cumplir con las expectativas del cliente. Sin embargo, esto implica que alguien manualmente verifique que se cumplan los requisitos y la aplicación funcione como se esperaba. Aquí es donde las pruebas de aceptación automáticas entran en lugar de los requisitos en un documento obsoleto, los requisitos se definen como ejemplos y escenarios, se protegen en control de origen con los artefactos de implementación y se pueden ejecutar en cualquier momento para comprobar si se implementa cualquier requisito y funciona correctamente. Puede tomar el mismo enfoque para escribir las pruebas, pero en lugar de escribirlos en software de administración de casos de pruebas o en una hoja de cálculo, escribirlos directamente en el código.

3.2.1 Pruebas de aceptación automática.

Una de las prácticas más valiosas para salir del movimiento de Software Ágil es un sistema automatizado, previamente probado estilo de desarrollo, a menudo denominado Test-Driven Development, o TDD. Un principio fundamental de TDD es que la creación de pruebas se trata tanto de diseño y el desarrollo de la orientación, ya que se trata de la verificación y la regresión. Es también acerca del uso de la prueba para especificar una unidad de la funcionalidad requerida, y el uso de esa prueba a escribir a continuación, sólo el código necesario para ofrecer esta funcionalidad. Por lo tanto, el primer paso en la implementación de cualquier nueva funcionalidad es para describir sus expectativas con una prueba.

Muchos desarrolladores y los equipos han tenido gran éxito con TDD. Los demás no tienen, y encontrar que luchan con la gestión del proceso a través del tiempo, sobre todo porque el volumen de pruebas empieza a crecer y la flexibilidad de las pruebas se empieza a degradar. Algunos no son seguro de cómo empezar con TDD, mientras que otros encuentran TDD fáciles de iniciar, sólo para ver lo abandonaron como los plazos cercanos a los retrasos y el telar grande. Por último, muchos desarrolladores interesados cumplir con la resistencia a la práctica dentro de sus organizaciones, ya sea porque la palabra "prueba" implica una función que pertenece a otro equipo o por la falsa percepción de que los resultados de TDD en código extra demasiado y ralentiza los proyectos.(Satrom, 2010)

a) Enfoque Test-Driven Development (TDD).

El enfoque Test-Driven Development (TDD) se basa en que las pruebas deben dirigir el desarrollo del producto software. El TDD se ha aplicado esencialmente en el ámbito de laimplementación, particularmente siguiendo e planteamiento “no escribir código hasta disponer de las pruebas que debe satisfacer dicho código”. El TDD ha tenido un fuerte impulso con las metodologías ágiles, las cuales lo incorporan entre sus prácticas esenciales.

En productos software industriales, cuando se utilizan prácticas de ingeniería de requisitos, éstas mayoritariamente están soportadas por lenguaje natural, lo cual conlleva los ya conocidos inconvenientes relativos a ambigüedad. Sin embargo, la necesidad de validación puede más que las ventajas que puede ofrecer una especificación más formal y rigurosa de los requisitos. El cliente debe poder leer y comprender los requisitos para así poder dar su conformidad respecto de ellos. Las técnicas más populares para especificación de requisitos se resumen en Casos de Uso (utilizados en metodologías tradicionales, como RUP) e Historias de Usuario (fichas de XP o representaciones similares en otras metodologías ágiles como Scrum). Si bien los elementos Actor (o tipo de usuario) y Requisitos (Caso de Uso o Historia de Usuario) pueden visualizarse y manipularse gráficamente o en fichas, la descripción de cada requisito se elabora esencialmente de forma textual en lenguaje natural. Para definir los requisitos es clave involucrar al cliente. El acuerdo/contrato con el cliente y la planificación del desarrollo del producto deberían estar basados en los requisitos que se pretenden incorporar en el producto. Los requisitos son el objetivo a conseguir, es decir, es lo que espera el cliente que el producto software satisfaga.

Nuestra propuesta se denomina Test-Driven Requirements Engineering (TDRE) por referirnos a TDD pero en el ámbito de los requisitos y la planificación. TDRE integra los artefactos, actividades y

roles, asociados a la especificación y validación de los Requisitos y de las Pruebas de Aceptación. El concepto de Requisitos se convierte en un contenedor de Pruebas de Aceptación, y son éstas las que adquieren protagonismo como especificación de cada requisito. Una de las ocho "buenas características" que recomienda la IEEE 830 para una especificación de requisitos se refiere a que cada requisito debe ser "Verificable". En nuestra propuesta los requisitos son verificables pues están especificados como Pruebas de Aceptación.

b) TDRE.

Para comprender cómo los requisitos pueden ser especificados mediante Pruebas de Aceptación comenzaremos con un ejemplo. Consideremos el requisito "Retirar dinero" en el contexto de un cajero automático. Una típica especificación narrativa podría ser la siguiente:

"El cliente debe poder retirar dinero del cajero en cantidades seleccionables. Siempre recibe un comprobante, a menos que el cajero se quede sin papel. Cuando se trata de un cliente preferencial puede retirar más dinero del que tiene en su cuenta, pero se le debe advertir que se le cobrarán intereses. El cliente debería poder cancelar en cualquier momento antes de confirmar el retiro de dinero. Las cantidades deberían poder servirse con los billetes que en ese momento tenga el cajero y no se deberían aceptar otros montos. Sería conveniente avisar cuando el cajero esté realizando operaciones internas mostrando un mensaje: "El dinero retirado de la cuenta debe poder comprobarse en los registros de movimientos de la cuenta…".

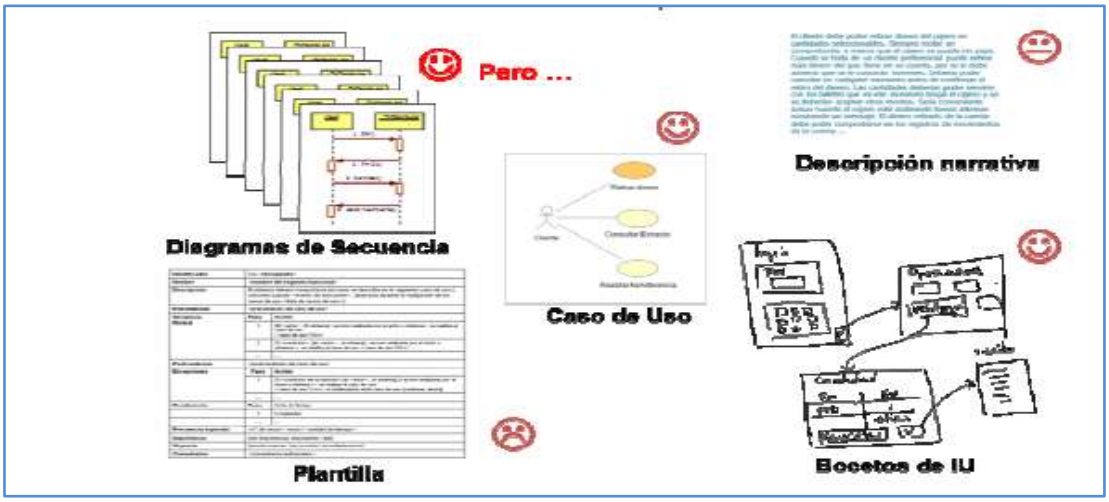

Figura 10.- Alternativas de Especificación

La Figura 10 ilustra algunas alternativas de especificación para este requisito (incluyendo la anterior descripción narrativa como opción por defecto). Los iconos reflejan la conveniencia de cada alternativa de especificación. Elaborar un Diagrama de Secuencia para definir cada escenario de ejecución del requisito puede parecer interesante, sin embargo, en general no resulta apropiado por la gran cantidad de diagramas generados. Resulta más interesante la identificación de los escenarios que la ilustración de cada uno de ellos en un diagrama.

La descripción narrativa no es descartable, al menos para dar una breve definición del requisito centrándose en definir los conceptos involucrados (con la idea de un glosario o de un sencillo Modelo de Dominio).

Un Modelo de Casos de Uso no es mala idea, especialmente para organizar y visualizar los requisitos de un sistema nuevo. Sin embargo, un Modelo de Casos de Uso no es apropiado para ilustrar la estructura de requisitos detallada de un producto software en situaciones de mantenimiento a más largo plazo, ya que un producto software de tamaño medio puede tener miles de requisitos. La visualización y gestión de gran cantidad de requisitos necesita de mecanismos más apropiados.

Los bocetos (visualizaciones muy preliminares) de la Interfaz de Usuario (IU) son siempre bienvenidos pues son una herramienta efectiva de comunicación y validación con el cliente, el cual puede hacerse una idea del producto. En este contexto de requisitos no debe pretenderse realizar el diseño final de las IUs sino más bien paneles con cierto ámbito de datos, sin profundizar en tipos de controles de interfaz o cuestiones de estética de formularios/páginas. Las plantillas son una de las alternativas de especificación más usadas para Casos de Uso. Las plantillas son elegantes y proporcionan una sensación de orden en la especificación. Sin embargo, en general resultan contraproducentes ya que tienden a dar un tratamiento uniforme en cuanto al nivel de detalle para todos los requisitos.

En aquellos muy simples se tiende a incluir cosas obvias o irrelevantes sólo para poder cubrir todos los apartados de la plantilla. Cuando un requisito incluye varios (o muchos) escenarios, el intento por sintetizar todos los escenarios en una plantilla (que sólo ofrece pasos y excepciones) lleva normalmente a especificaciones enrevesadas.

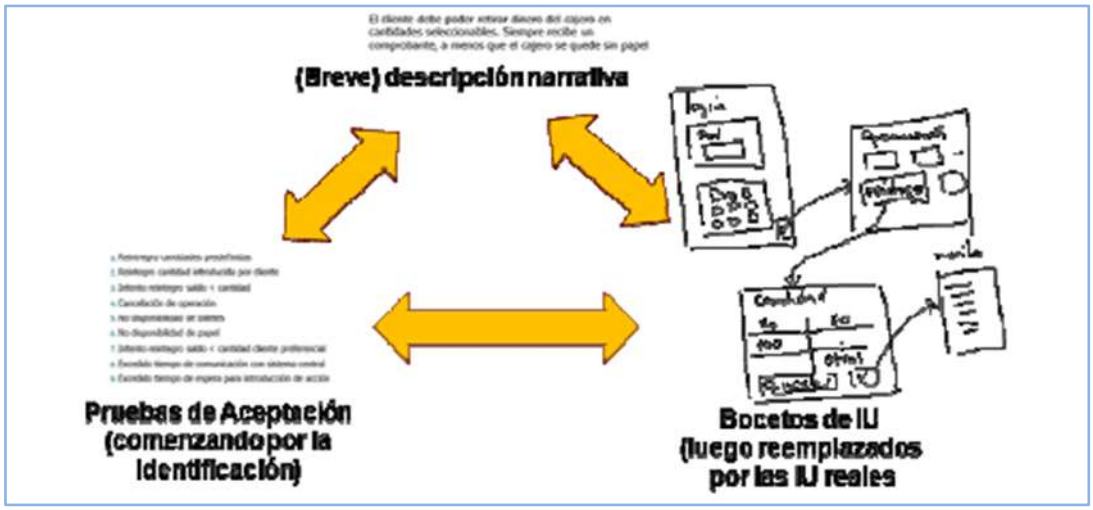

Figura 11.- Descripción narrativa

Nuestro enfoque TDRE apuesta por especificar los requisitos usando los elementos que se muestran en *la Figura 11*: una breve descripción narrativa que establece los conceptos y motivación del requisito, bocetos de la IU (si procede) y una lista de Pruebas de Aceptación (PAs). Estas PAs se refinarán desde simples frases que dan nombre a los escenarios, hasta pruebas diseñadas, listas para ser aplicadas o para automatizarse y aplicarse en regresión. Así, los requisitos actúan como contenedores para la Pas. Dependiendo del requisito, podría ser útil utilizar otras formas de especificación con carácter complementario. Por ejemplo un Diagrama de Actividad si el comportamiento asociado al requisito es de carácter algorítmico o un Diagrama de Estados si el comportamiento incluye habilitación o deshabilitación de acciones de acuerdo con el

estado del sistema. La premisa esencial es pragmatismo respecto de la especificación, con lo cual no se descarta el uso combinado de alternativas de especificación, pero el criterio primordial debe ser el rentabilizar el esfuerzo en especificación y facilitar el mantenimiento de dicha especificación. Por otra parte, desde el punto de vista de esfuerzo de mantenimiento, especialmente en cuanto a consistencia, es importante no abusar de solapes o duplicaciones de especificaciones en diferentes medios de representación.

Las miles de PAs que fácilmente puede llegar a tener un producto software deben organizarse adecuadamente para poder ser gestionadas de forma eficiente. Un grafo dirigido es una representación adecuada para realizar un refinamiento por niveles. Dicho grafo permite tanto la visualización de relaciones de descomposición como de dependencia entre requisitos. Así, cada nodo es un requisito funcional o no funcional. Los arcos entre nodos establecen relaciones padres-hijos (mediante las cuales se hace una descomposición de los requisitos) o relaciones de dependencia del tipo “nodos que afectan nodos” (estas relaciones son clave para realizar análisis de impacto).

Las PAs tendrán diferentes estados según el refinamiento de su especificación, de menor a mayor detalle dichos estados son: identificación (un nombre para la PA), definición (establecimiento de condiciones, pasos de ejecución y resultado esperado), diseño (instanciación con datos específicos) y ejecución en el producto (incluyendo el caso de ejecución en regresión, sea ésta automatizada o manual). En el resto del artículo nos centraremos en la identificación y definición de la PAs, lo cual está más asociado al marco de trabajo del analista. El diseño y ejecución de la PAs es responsabilidad del tester.

Así, en el ejemplo anterior, el requisito “Retirar dinero” podría ser un nodo de la estructura de requisitos. Los nombres que identifican sus PAs podrían ser:

- Reintegro usando cantidades predefinidas habilitadas.
- Reintegro con cantidad introducida por cliente.
- Reintegro saldo < cantidad.
- Cancelación de operación.
- No disponibilidad de billetes.
- No disponibilidad de papel para recibo.
- Reintegro saldo < cantidad con cliente preferencial.
- Excedido tiempo de comunicación con sistema central.
- Aviso de operaciones internas del cajero.
- Excedido tiempo de espera para introducción de acción.

Una PA tiene como propósito demostrar al cliente el cumplimiento parcial o total de un requisito del software. Las características de una PA son:

• Una PA describe un escenario (secuencia de pasos) de ejecución o un uso del sistema desde la perspectiva del cliente. Las PAs cubren desde escenarios típicos/frecuentes hasta los más excepcionales.

• Puede estar asociada a un requisito funcional o requisito no funcional. Un requisito tiene una o más PAs asociadas.

• Una PA puede tener infinitas instanciaciones (ejecuciones con valores concretos).

La definición de una PA se separa en cuatro apartados: Condición, Pasos, Resultado Esperado y Observaciones.

• **Condición.** Es opcional, y se utiliza para establecer condiciones previas antes de aplicar los pasos de la prueba. Normalmente se refieren al estado de la BD antes de ejecutar la prueba y/o la navegación necesaria en la IU para localizarse en el punto adecuado para realizar la prueba (cuando no sea obvio)

• **Pasos**. Son las acciones de interacción del actor con el sistema. Cuando son varias acciones, éstas pueden ponerse en una lista numerada. Deben ser simples, del estilo "seleccionar…", "introducir...", evitando hacer referencia explícita a controles de interfaz o cualquier tecnicismo que dificulte su validación con el usuario.

• **Resultado esperado.** Es el efecto de las acciones de interacción del actor. Cada acción puede provocar uno o más resultados. Es importante que cuando se trate de mensajes al usuario se incluya el texto como parte del resultado esperado, así el programador dispone de dicha información ya validada con el cliente.

• **Observaciones.** Es un apartado opcional, son recomendaciones que el analista estima conveniente hacerle al tester y/o programador.

Las condiciones y/o resultados esperados pueden establecerse en el ámbito del requisito contenedor de la PA o de otros requisitos. Esto, como indicaremos más adelante, permitirá establecer relaciones de dependencia entre requisitos.

A continuación se presenta como ejemplo la definición de la PA "Reintegro saldo < cantidad".

1. ***Condición***

   Debe existir un cliente normal (esta característica se establece en el apartado datos básicos del cliente).

2. ***Pasos***

   ✓ Intentar reintegro con cliente normal y con cantidad solicitada mayor al saldo.
   ✓ Resultado esperado.

- ✓ Se muestra el mensaje “La cantidad solicitada supera su saldo actual, vuelva a introducir la cantidad” y se retorna a la ventana para introducir la cantidad.

Pruebas de aceptación ayudará a validar que se está construyendo la aplicación que desea el cliente, mientras que la automatización de estos escenarios le permite verificar constantemente la aplicación en todo el proceso de desarrollo y los utilizan como parte de su suite de pruebas de regresión para asegurar que los futuros cambios no violan los requerimientos actuales.

Sin embargo, tener un cliente implicado con escritura de pruebas, especialmente pruebas automáticas, presenta un número de posibles problemas. Los clientes, en general, no son técnicos y tienden a alejarse del propio desarrollo de software. El cliente puede proporcionar los datos y ejemplos, mientras que los probadores o los desarrolladores rápidamente pueden codificar los escenarios y especificación ejecutable.

Las pruebas de aceptación de la interfaz de usuario

En los ejemplos, las pruebas de aceptación se centró en la lógica de negocio y los objetos de dominio con el fin de comprobar si la lógica trabajado con éxito. Pero, ¿qué acerca de cómo el usuario interactúa con la aplicación? Estas pruebas de aceptación deben estar allí para verificar la lógica es correcta el punto de vista del usuario-y el punto de vista del usuario es la interfaz de usuario.

Personalmente, creo que las pruebas de aceptación deben centrarse en la lógica de la aplicación y las secciones importantes del código que decidir qué hacer. Si la aplicación tiene un buen desacoplamiento, y una buena separación de la lógica del código de la interfaz de usuario, esto debería hacer las pruebas más fáciles de implementar. Si está probando en este nivel, las pruebas no sufrirán cambios en la interfaz de usuario.

Mientras que las pruebas deben centrarse exclusivamente en la lógica, esto no significa que usted no debería tener ninguna prueba de aceptación en todo el interfaz de usuario. Me gusta tener un conjunto de pruebas de humo que se dirigen a la base "camino feliz" de la interfaz de usuario. Se centran en las partes de la aplicación que los usuarios son más propensos a utilizar con el fin de obtener el máximo valor de la menor cantidad de pruebas. Si se intenta cubrir todos los posibles caminos y usos de la interfaz de usuario, y si los cambios de interfaz de usuario (por ejemplo, como opciones de pasar a un diálogo a otro), entonces usted tendría que cambiar todas las pruebas.

Por ejemplo, si se prueba la interfaz de usuario para un sitio de comercio electrónico, el camino sería feliz para seleccionar un artículo, incorporarlo a su carrito de compras, echa un vistazo, y ver la confirmación de compra. Si este escenario falla, usted realmente quieren saber lo más pronto posible.

Para ciertas aplicaciones, dependiendo de la complejidad y el tiempo de vida, es posible que desee tener más pruebas de aceptación que se ejecutan en la interfaz de usuario para asegurarse de que tienen más confianza en la capa de interfaz de usuario. Con éxito pruebas de la interfaz de usuario es un tema complejo, sin embargo, y no lo hago tiene el espacio para cubrirla aquí.(Francisco Suárez;Patricio Letelier)

3.2.3Implementación de pruebas de aceptación.

Una vez que tenga la historia y los escenarios en un formato claro y comprensible, el siguiente paso es automatizar la historia y los escenarios. Esto les permite ser ejecutado durante el desarrollo para seguir los progresos y detectar los errores de regresión.

Conclusiones y recomendaciones.

Las pruebas de aceptación son iterativas donde se pueden aplicar un conjunto de metodología como el programación extrema (XP), el scrum y RUP(casos de usos)están definidas como pruebas de Caja negra por la que el más importante de los actores que estas pruebas sean exitosas es el cliente es importante determinar las historias y escenarios en donde se van a especificar determinar los requisitos para que el cliente pueda validar si está conforme a lo especificado.

Por otra parte tenemos las pruebas de sistemas estas son las encargadas de evaluar el funcionamiento a lo largo del proceso para detectar los errores que se puedan producir para ello es necesario hacer una estrategia con los testing y separar el desarrollo del código con el desarrollo de la interfaz así se hace mejor una implementación de prueba de sistemas .

La implementación de estas dos pruebas de software se deben hacer con exámenes riguroso y que cumplan con determinados estándares y con la coordinación de los stakeholder que estén implicados en la elaboración del sistema .

Bibliografía

1.-Isabel Ramos Román, José Javier Dolado Cosín. Técnicas Cuantitativas para la Gestión en la Ingeniería del Software. Primera Edición ed. Tuya , Ramos Román I, Dolado Cosín JJ, editors. España: Netbiblo, 2007; 2007.

2.-INTECO. Instituto Nacional de Tecnologias de Comunicacion. [Online].; 2009 [cited 2012 [Plan Avanza 2 - Ministerio de Industria, Turismo y Comercio de España]. Available from: www.inteco.es/file/XaXZyrAaEYfaXKiMJlkT_g.

3.-F. Alonso Amo, Loic Martínez Normand. Introducción a la Ingeniería del software. Primera edicion ed. Barbero Ruiz J, editor. España: Delta Publicaciones, 2005.

4.-IEEE Computer Society. Computer. [Online].; 2004 [cited 2012. Disponible en: http://www.computer.org/portal/web/swebok/html/contentsch5#ch5.

5.-Asociacion de tecnicos de informatica. Pruebas de Aceptación en Sistemas Navegables. REICIS Revista Española de Innovación, Calidad. 2010 Noviembre; vol. 6(núm. 3): p. pp. 47-55.

6.-ALEGSA. ALEGSA - DICCIONARIO DE INFORMÁTICA. [Online]. 2012. Disponible en: http://www.alegsa.com.ar/Dic/implementacion.php.

7.- ingsoft. ingsoft-ETAPA DE PRUEBAS. [Online].2007. Disponible en: http://ingpau.blogspot.com/2007/09/etapa-de-pruebas.html.

8.-Departamento de Lenguajes y Sistemas Informáticos. ITESCAM. [Online]. Disponible en:
http://www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r43876.PDF.

9.-Satrom B. MSDN Magazine. [Online].; 2010. Disponible en:
http://msdn.microsoft.com/es-es/magazine/gg490346.aspx

10.-Francisco Suárez;Patricio Letelier. TUNE-UP. [Online]. Disponible en:
http://tuneup.dsic.upv.es/LinkClick.aspx?fileticket=Dshzxx-2qDY%3D&tabid=40.
11.-Universidad Distrital Francisco José de Caldas.ARQUISOFT.[Online]. Disponible en:
http://200.69.103.48/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/
12.- Instituto Tecnológico de Mérida.Desarrollo de proyecto de Software.[Online].Disponible en:
http://es.slideshare.net/juandiaz89/estrategias-de-aplicacin-de-pruebas-del-sistema-14842463
13.Ana Sosa. Magazine.Prueba general de Software.ITPUEBLA.[Online]. Disponible en:
http://www.itpuebla.edu.mx/Alumnos/Cursos_Tutoriales/Ana_Sosa_Pintle/ANALISIS_DISENO/ANALISIS%202%20PRUEBA%20GENERAL%20DEL%20SISTEMA.htm
14.-TestHouse.Pruebas de Sistemas. [Online]. Disponible en:
http://www.es.testhouse.net/pruebas-de-sistema/
15.- Departamento de Tecnologías y Sistemas de Información. Universidad Castilla-México. [Online].Disponible en:
http://www.inf-cr.uclm.es/www/mpolo/asig/0708/phd/apuntesDoctorado.pdf
16.-Jesus Barranco de Areba. Metodología de análisis estructurado de sistemas. Universidad Pontifica-España.[Online]. Disponible en:
http://books.google.com.pe/books?id=PUqxsNVaQC8C&pg=PA467&lpg=PA467&dq=pruebas+de+comunicacion+de+sistemas&source=bl&ots=bJjHFtBrBL&sig=EhPEBom8YGdbpcUvAGzYuJMF0lw&hl=es&sa=X&ei=LhmSUf3fLvfK4APlroDABw&ved=0CFwQ6AEwCTgK#v=onepage&q=pruebas%20de%20comunicacion%20de%20sistemas&f=false