ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y
SISTEMAS DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE MADRID

Departamento de Teoría de la Señal y Comunicaciones

Pruebas de Software. Fundamentos y Técnicas.

José Manuel Sánchez Peño

Junio 2015

GRACIAS.

A mi familia, mis padres y mi hermano, que siempre estuvieron en todos los malos y buenos momentos. En especial a mi madre, porque sin ti no sería nada en este mundo. Ojalá pueda llegar a ser la mitad de buena persona que eres tú.

A Arancha, por ser una de las personas mas maravillosas que he conocido y que seguramente conoceré.

A mis amigos, por creer en mi y apoyarme siempre. Algunos de vosotros sois parte de mi familia.

A todas las personas que nunca dejaron de creer que sus sueños podían hacerse realidad.

RESUMEN DEL PROYECTO:

Este proyecto estudia los fundamentos y las técnicas de las pruebas de software.

Veremos lo importante que pueden llegar a ser las pruebas, mostrando diferentes desastres causados por fallos en el software.

También estudiaremos las diferentes herramientas que se utilizan para llevar a cabo la gestión, administración y ejecución de estas pruebas.

Finalmente aplicaremos los conceptos estudiados mediante un caso práctico. Crearemos los casos de prueba funcionales basándonos en las especificaciones del protocolo MDB/ICP e instalaremos y aprenderemos cómo crear estos casos con una de las herramientas estudiadas en la parte teórica.

ABSTRACT:

This project studies the fundamentals and techniques of software testing.

We will see how important the evidence showing different disasters caused by bugs in the software can become.

We will also study the different tools used to carry out the management, administration and execution of these tests.

Finally, we apply the concepts studied by a case study. We create test cases based on functional specifications MDB/ICP protocol We will install and learn how to create such cases by one of the tools studied in the theoretical part.

ÍNDICE

1. INTRODUCCIÓN 11

2. PRUEBAS DE SOFTWARE 15

2.1 FUNDAMENTOS BÁSICOS DE LAS PRUEBAS 15

2.1.1 ¿Qué son las pruebas? 15

2.1.2 ¿Por qué son importantes las pruebas? 18

2.1.3 ¿Cuál es el objetivo de las pruebas? 20

2.1.4 ¿Cómo llevamos a cabo las pruebas? 21

2.2 TIPOS DE PRUEBA 25

2.2.1 Pruebas funcionales 25

2.2.2 Pruebas no funcionales 26

2.2.3 Pruebas estructurales 27

2.3 TÉCNICAS DE PRUEBA 28

2.3.1. Técnicas estáticas 28

2.3.1.1. Análisis estático 28

2.3.1.2. Revisiones 29

2.3.1.2.1. Revisión informal 29

2.3.1.2.2. Revisión guiada 30

2.3.1.2.3. Revisión técnica 30

2.3.1.2.4. Inspección 30

2.3.2. Técnicas dinámicas 31

2.3.2.1. Técnica de caja blanca 31

2.3.2.1.1. Pruebas de ruta básica 32

2.3.2.1.2. Pruebas de ciclos o bucles 36

2.3.2.1.3. Pruebas de condición y condición múltiple 39

2.3.2.2. Técnica de caja negra 39

2.3.2.2.1. Partición de equivalencia 40

2.3.2.2.2. Análisis del valor limite 41

2.3.2.2.3. Pruebas de tabla de decisión 41

2.3.2.2.4. Pruebas de transición de estado 42

2.3.2.2.5. Pruebas de caso de uso 42

2.3.3. Técnicas basadas en la experiencia 43

2.3.3.1. Predicción de error 43

2.3.3.2. Pruebas exploratorias 43

2.4 ESTRATEGIA DE PRUEBA 45

2.4.1. Modelo en V 45

2.4.2. Niveles de prueba 46

2.4.2.1. Verificación y validación 46

2.4.2.2. Pruebas unitarias o de componente 47

2.4.2.3. Pruebas de integración 47

2.4.2.3.1. Integración descendente 48

2.4.2.3.2. Integración ascendente 48

2.4.2.3.3. Integración ad-hoc 48

2.4.2.3.4. Integración del esqueleto 49

2.4.2.4. Pruebas de sistema 49

2.4.2.5. Pruebas de validación o aceptación 49

2.4.2.5.1. Pruebas de aceptación del contrato 50

2.4.2.5.2. Pruebas de aceptación del usuario 50

2.4.2.5.3. Pruebas operativas 50

2.4.2.5.4. Pruebas alfa y beta 51

2.5 OTROS TIPOS Y CONCEPTOS DE PRUEBAS. 52

2.5.1. Pruebas de aplicaciones WEB 52

2.5.2. Pruebas de regresión, humo y usabilidad 53

2.5.2.1. Pruebas de regresión 53

2.5.2.2. Pruebas de humo 54

2.5.2.3. Pruebas de usabilidad 54

2.5.3. Pruebas de automatización 55

2.5.4. ¿Qué es un "plan de pruebas"? 56

2.5.5. Normas y certificaciones *57*
2.5.5.1. Normas fundamentales de pruebas 57
2.5.5.2. Certificaciones 58

3. HERRAMIENTAS DE PRUEBAS 61
3.1 HERRAMIENTAS PARA PRUEBAS ESTÁTICAS 61
3.1.1. Herramientas de revisión *61*
3.1.2. Análisis estático *61*
3.1.3. Herramientas de modelado *61*
3.1.4. Ejemplos de herramientas *61*
3.2 HERRAMIENTAS PARA PLANIFICACIÓN Y GESTIÓN 63
3.2.1. Ejemplos de herramientas *63*
3.3 HERRAMIENTAS PARA PRUEBAS DE AUTOMATIZACIÓN 65
3.3.1. Ejemplos de herramientas *65*
3.4 HERRAMIENTAS DE PRUEBAS DE CARGA Y RENDIMIENTO 66
3.4.1. Ejemplos de herramientas *67*
3.5 DRIVERS Y STUBS 68
3.6 HERRAMIENTAS DE SEGURIDAD Y MONITORIZACIÓN 69
3.6.1. Ejemplos de herramientas *69*

4. PRUEBAS FUNCIONALES DE APLICACIONES QUE SE COMUNICAN CON EL PROTOCOLO MDB/ICP 73
4.1 ¿QUÉ ES EL PROTOCOLO MDB Y CÓMO FUNCIONA? 73
4.2 SISTEMA PARA TERMINALES DE PAGO CON TARJETA - TPV 77
4.3 CASOS DE PRUEBA FUNCIONALES DEL PROTOCOLO MDB/ICP 82

5. TESTLINK Y LA GESTIÓN DE LAS PRUEBAS 109
5.1 ¿QUÉ ES TESTLINK? 109
5.2 INSTALACIÓN DE TESTLINK 109
5.3 TESTLINK PARA LA CREACIÓN DE CASOS DE PRUEBA 115

6. CONCLUSIONES Y TRABAJOS FUTUROS 119

7. GLOSARIO DE TÉRMINOS 121

8. BIBLIOGRAFÍA 123

ÍNDICE DE ILUSTRACIONES

ILUSTRACIÓN 1. CARACTERÍSTICAS CALIDAD ISO 25010 [WEB11] 21
ILUSTRACIÓN 2. CAJA BLANCA 31
ILUSTRACIÓN 3. NOTACIÓN DE GRAFO DE FLUJO 32
ILUSTRACIÓN 4. A)DIAGRAMA DE FLUJO B)GRAFO DE FLUJO.[PRE10] 33
ILUSTRACIÓN 5. GRAFO DE FLUJO CON CONDICIONES COMPUESTAS [PRE10]. 34
ILUSTRACIÓN 6. BUCLE SIMPLE 36
ILUSTRACIÓN 7. BUCLE CONCATENADO 37
ILUSTRACIÓN 8. BUCLES ANIDADOS 38
ILUSTRACIÓN 9. BUCLE NO ESTRUCTURADO 38
ILUSTRACIÓN 10. CAJA NEGRA 39
ILUSTRACIÓN 11. MODELO EN V [SCH14] 46
ILUSTRACIÓN 12. DRIVER 68
ILUSTRACIÓN 13. STUBS 68
ILUSTRACIÓN 14. MÁQUINA DE ESTADOS DEL PERIFÉRICO TPV 77
ILUSTRACIÓN 15. PANEL DE CONTROL XAMPP 110
ILUSTRACIÓN 16. PÁGINA DE INICIO DEL SERVIDOR 110
ILUSTRACIÓN 17. CONFIGURACIÓN TESTLINK 1 111
ILUSTRACIÓN 18. CONFIGURACIÓN TESTLINK 2 111
ILUSTRACIÓN 19. CONFIGURACIÓN TESTLINK 3 112
ILUSTRACIÓN 20. CONFIGURACIÓN TESTLINK 4 113
ILUSTRACIÓN 21. CONFIGURACIÓN TESTLINK 5 114
ILUSTRACIÓN 22. CÓMO USAR TESTLINK 1 115
ILUSTRACIÓN 23. CÓMO USAR TESTLINK 2 115
ILUSTRACIÓN 24. CÓMO USAR TESTLINK 3 116
ILUSTRACIÓN 25. CÓMO USAR TESTLINK 4 116
ILUSTRACIÓN 26. CÓMO USAR TESTLINK 5 117
ILUSTRACIÓN 27. CÓMO USAR TESTLINK 6 117

1. INTRODUCCIÓN

En el año 2012 se celebró el año de Turing coincidiendo con el centenario de su nacimiento. A raíz de esto el periódico 'El País' publicó una noticia [WEB01] cuyo titular decía: ¿Es posible construir software que no falle?. El artículo continuaba explicando que hoy en día estamos acostumbrados a que el software falle.

Al construir software habitualmente se comenten errores. En la industria, la técnica para solucionar los problemas derivados de dichos errores, serán las pruebas de software ('testing'), que consistirán en una serie de pasos realizados antes y después de la construcción de este software.

Históricamente, por la ausencia o por la incorrecta realización de las pruebas sobre el software, se han producido varios desastres que han llegado, no solo a tener consecuencias económicas nefastas, sino a producir la perdida de vidas humanas.

Las pruebas son parte fundamental de cualquier proyecto, ya que nos ayudarán a tener mejores resultados, ofreceremos una calidad mayor de nuestro producto y en consecuencia los clientes estarán más satisfechos.

Según Ilene Burnstein [BUR03], el proceso de prueba de software tiene tres procesos principales: el desarrollo de los casos de prueba, la ejecución de estos casos de prueba y el análisis de los resultados de la ejecución. En este proyecto nos centraremos en el proceso del desarrollo de los casos de prueba y para ello contaremos con un estudio teórico y otro práctico.

La parte teórica consistirá en ver los tipos de pruebas y las técnicas que se aplican. Además estudiaremos las herramientas que existen para realizar dichas pruebas sobre los diferentes tipos de software.

Nos haremos varias preguntas: ¿qué son las pruebas?, ¿por qué tenemos que realizar pruebas?, ¿cómo realizo esas pruebas?, etc. Todas y cada una de estas preguntas, y alguna más que veremos, tienen como objetivo introducirnos en el mundo de las pruebas.

En la parte práctica del proyecto se realizará un caso práctico con los puntos estudiados en la parte teórica.

Por un lado veremos la creación de los casos de prueba mediante la especificación de un protocolo que tomaremos como ejemplo para la creación de los casos de prueba. En este caso, el protocolo a estudio será el protocolo MDB/ICP de la NAMA. Más concretamente, se realizarán los casos de prueba de un dispositivo de pago con tarjeta con una máquina expendedora.

Para la creación de los casos de prueba primero se realizará un estudio del protocolo para posteriormente realizar las pruebas funcionales entre los sistemas indicados anteriormente. En esta parte veremos qué es el protocolo MDB, cómo funciona y las características de la comunicación con el terminal de tarjetas electrónicas.

Al realizar el estudio del protocolo, estaremos iniciando la búsqueda de la estrategia que utilizaremos para realizar los casos de prueba.

Por otro lado, iniciaremos la construcción de los casos de prueba y para ello vamos a utilizar la herramienta TestLink, herramienta de gestión de casos de prueba, para documentar éstos.

Se verá también como se instala esta herramienta y su funcionamiento básico para poder realizar los casos de prueba.

PARTE TEÓRICA

2. PRUEBAS DE SOFTWARE

2.1 Fundamentos básicos de las pruebas

2.1.1 ¿Qué son las pruebas?

Antes de ver una definición de prueba, veremos la percepción que tienen los desarrolladores acerca de las pruebas. Según [MYE11], los desarrolladores siguen las siguientes definiciones que llevan a una percepción falsa:

- "*Las pruebas son el proceso de demostrar que no hay errores presentes*".
- "*El propósito de las pruebas es demostrar que un programa realiza las funciones indicadas correctamente*".
- "*Las pruebas son el proceso de establecer confianza en que un programa hace lo que se supone que debe hacer*" .

Jhon Myers indica que estas definiciones están mal planteadas. Cuando probamos un programa se quiere aportar un valor añadido a lo que estamos probando, elevar la calidad y fiabilidad y esto nos lleva a tener que encontrar y eliminar los errores en el programa.

Esto quiere decir que no tenemos que probar un programa para demostrar que funciona, sino que tenemos que partir de la suposición de que el programa va a contener errores. La definición de prueba que aporta Myers es:

- **"La prueba es el proceso de ejecución de un programa con la intención de encontrar errores".**

Pero, ¿por qué se toma esta definición como válida y las anteriores no?.

Si tomamos la primera definición mencionada - "El testing es el proceso de demostrar que no hay errores presentes", psicológicamente estaremos dirigidos hacia esa meta y tenderemos a seleccionar los casos de prueba con una baja probabilidad de causar que el programa falle. Si por el contrario tomamos como objetivo demostrar que un programa tiene fallos, tendremos una mayor probabilidad de encontrar errores.

Si tomáramos la definición de "Testing es el proceso de establecer confianza en que un programa hace lo que se supone que debe hacer" estaríamos dando por sentado que un programa que hace lo que se supone debe hacer, no tiene errores y eso es un error ya que, aunque el programa realice su función, puede contener errores.

El ISTQB (International Software Testing Qualifications Board), una organización sin ánimo de lucro creada en el año 2002 por empresas, instituciones, organizaciones y personas especializadas en el campo de las pruebas y la industria del software, define las pruebas como [ISTQB]:

- *"El proceso que consiste en todas las actividades del ciclo de vida, tanto estáticas como dinámicas relacionadas con la planificación, preparación y evaluación de productos de software y productos relacionados con el trabajo para determinar que cumplen los requisitos especificados, para demostrar que son aptos para el propósito y para detectar defectos".*

Cem Kaner, profesor de Ingeniería de software en el instituto tecnológico de Florida, es uno de los defensores y principales gurús de las pruebas de software, las define como [WEB02]:

- *"Las pruebas de software son la investigación empírica y técnica realizada para facilitar a los interesados información sobre la calidad del producto o servicio bajo pruebas".*

Kaner introduce la figura del técnico que mediante la investigación aportará datos sobre la calidad del producto y no se centrará únicamente en la detección del fallo.

Edsger W. Dijstra, científico de la computación entre cuyas contribuciones a la ciencia esta 'la solución del problema del camino más corto', también conocido como el algoritmo de Dijstra, define el proceso de pruebas como [WEB03]:

- *"Las pruebas de software pueden ser una manera muy eficaz de mostrar la presencia de errores, pero son totalmente inadecuadas para mostrar su ausencia."*

Todas y cada una de estas definiciones tienen algo en común, todas se centran en mayor o menor manera en la detección de errores.

Para terminar de entender las pruebas debemos diferenciar los términos error, fallo y defecto. Estos conceptos están relacionados entre si, pero tienen significados diferentes. Para comprender estas palabras y por ende las pruebas, vamos a ver como las define el ISTQB:

- *"Una persona puede cometer un **error** que a su vez puede producir un **defecto** en el código de programa o en un documento. Si se ejecuta un defecto en el código, el sistema puede no hacer lo que debiera (o hacer algo que no debiera), lo que provocaría un **fallo**. Algunos defectos de software pueden dar lugar a fallos, pero no todos los defectos lo hacen."*

Así pues tenemos:

- **Error**: está provocado por la acción humana, por ejemplo el error lo provocará el desarrollador que realizará una incorrecta interpretación de un método del programa que producirá un resultado no esperado.
- **Defecto**: provocado por un error de implementación, por ejemplo el defecto lo provocará el haber utilizado el operador "x + y **>** z" en vez de "x + y **=>** z"
- **Fallo**: al ejecutar el programa con un defecto obtendremos resultados no deseados, por ejemplo, cuando el resultado de la suma de los dos componentes fuese igual, no obtendríamos los mismos resultados al compararlos con las sentencias indicadas anteriormente. En sistemas muy complejos, como pueden ser una lanzadera espacial o una central eléctrica, pueden llegar a producir efectos catastróficos.

2.1.2 ¿Por qué son importantes las pruebas?

Hoy en día, forman parte de nuestras vidas multitud de sistemas que contienen software, como por ejemplo los coches, smartphones, sistemas de producción de energía, programas bancarios, etc.

Para responder a la pregunta anterior vamos a ver una serie de sucesos ocurridos durante el siglo XX, que ejemplifican lo importante que puede llegar a ser probar el software antes de ponerlo en producción:

- El lanzamiento de la lanzadera Ariane-5 vuelo 501 (1996) fue considerado uno de los fallos de programación más caros de la historia hasta ese momento, sólo la carga que llevaba tenía un valor de 500 millones de dólares. Fue un cohete lanzado por la ESA (European Space Agency's o agencia espacial europea) destruido aproximadamente a los 40 segundos de ser lanzado. Según el informe de la ESA [WEB04], el fallo de la Ariane 501 fue causado por la completa pérdida de guía e información de orientación treinta y siete segundos después del comienzo de la secuencia de ignición del motor principal. Esta pérdida de información se debió a errores de especificación y diseño en el software del sistema de referencia inercial. Las extensas revisiones y test llevados a cabo durante el programa de desarrollo del Ariane-5 no incluyeron el adecuado análisis y prueba del sistema de referencia inercial o del sistema de control de vuelo completo, lo cual podría haber detectado los fallos potenciales [WEB05].

- El lanzamiento de la sonda Mariner 1 de la NASA (1962), tuvo que ser abortado por un fallo de software que afectaba a la trayectoria del cohete. El cohete fue destruido antes de que soltara la sonda que transportaba, ya que corría peligro de estrellarse en las rutas marítimas del atlántico norte. El coste aproximado del proyecto de la sonda Mariner 1 fue de 554 millones de dólares [WEB06].

- Therac 25, una máquina de radioterapia producida por la Atomic Energy of Canada Limited (AECL) en 1985, fue la causante de la muerte de tres personas directamente y otras tres sufrieron graves daños por la

administración de sobredosis masivas de radiación. Una de las razones de que se produjera esta sobredosis fue un mal diseño del software, el código fuente no había sido revisado de forma independiente [WEB07].

- Thomas Nicely, de la Universidad de Lynchburg, descubrió un error en la unidad de coma flotante del Pentium en 1994. El divisor en la unidad de coma flotante contaba con una tabla de división incorrecta, donde faltaban cinco entradas sobre mil, que provocaba errores de redondeo. Aunque el error afectaba a pocos usuarios, este hecho hizo mucho daño a la imagen de Pentium y el costo total fue de 475 millones de dólares. Finalmente reemplazó los chips de todos los usuarios que lo solicitaron [PAT05].

- En otoño de 1994, la compañía Disney lanzó su primer juego multimedia en formato CD-ROM. Las ventas fueron enormes ya que fue uno de los juegos más comprados la navidad de ese año. El 26 de diciembre, el departamento de atención al cliente de Disney se vio desbordado por las llamadas de un gran número de usuarios descontentos que habían comprado el juego. Resulta que Disney no realizó pruebas en los diferentes modelos de PC disponibles en el mercado. Solo se realizaron pruebas sobre los PC que utilizaban los desarrolladores [PAT05].

Como se puede apreciar en los ejemplos anteriores, no probar adecuadamente el software, antes de ponerlo en producción, puede producir no sólo pérdidas económicas, sino también daños personales, llegando incluso a producir la muerte en algunos casos.

A día de hoy el funcionamiento de casi todas las empresas depende en gran medida del software, ya sea por el sistema de finanzas de dicha empresa o por la maquinaria que lleva a cabo la fabricación de los productos, por lo que las empresas dependen del funcionamiento del software y de que éste pueda llegar a causar grandes fallos como los mencionados anteriormente que llevan a la pérdida de miles de millones de euros. No a todas las empresas les afectan de la misma

manera los fallos producidos en el software, por lo que tenemos que evaluar los riesgos de éste, ya que pueden llegar a producir pérdidas irreparables.

2.1.3 ¿Cuál es el objetivo de las pruebas?

El objetivo principal de las pruebas es aportar ***calidad*** al producto que se está desarrollando.

Pero, ¿qué es calidad?

Este termino es definido por muchas organizaciones e ilustres en el mundo de la calidad de formas diferentes:

- **ISO 9000** es un conjunto de normas sobre calidad y gestión de calidad, la define como *"la calidad es el grado en el que un conjunto de características inherentes cumple con los requisitos"* [WEB08].
- **ISO 25000** es un conjunto de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software, dice: *" la calidad es el grado en que el producto de software satisface las necesidades expresadas o implícitas, cuando es usado bajo condiciones determinadas"* [WEB08].
- **Philip Bayard Crosby,** que contribuyó a las prácticas de la gestión de la calidad, la define como *"Conformidad con los requisitos"* [WEB09].
- **Armand V. Feigenbaum**, que diseñó el concepto del control total de la calidad, la define como *"satisfacción de las expectativas del cliente"* [WEB09].
- **Genichi Taguchi**, que desarrolló una metodología para la aplicación de estadísticas para mejorar la calidad de los productos manufacturados, dice: *"calidad es la pérdida (monetaria) que el producto o servicio ocasiona a la sociedad desde que es expedido"* [WEB10].

Podríamos seguir indicando definiciones de calidad de numerosos ilustres en el tema, en donde cada uno nos diría su definición y su manera de llegar a esta

calidad. Hay dos puntos principales que tienen casi todas las definiciones de calidad: la satisfacción del cliente y el cumplimiento de los requisitos del producto.

La ISO 25010 [WEB11], norma donde se trata la calidad del producto software, nos indica qué características tiene que tener un software para tener la calidad deseada:

Ilustración 1. Características CALIDAD ISO 25010 [WEB11]

2.1.4 ¿Cómo llevamos a cabo las pruebas?

Para llevar a cabo las pruebas verificaremos el comportamiento del programa sobre un conjunto de casos de prueba. Estos casos de prueba se generarán mediante técnicas y estrategias específicas de pruebas que nos ayudarán a conseguir la búsqueda de los errores de un programa.

Pero antes de continuar, y teniendo en cuenta las definiciones anteriores, vamos a plantearnos una pregunta muy importante que tendremos que tener en cuenta cuando veamos las estrategias y técnicas de prueba.

¿Es posible encontrar todos los errores de un programa?

En casi toda la documentación estudiada para este proyecto, [ISTQB], [MYE11], [PRE05], etc., nos encontramos con un apartado donde vemos que la respuesta a esta pregunta es negativa, no suele ser práctico, en lo que a tiempo y coste empleado en un proyecto se refiere, y la gran mayoría de las veces es imposible encontrar todos los errores de un programa.

Para encontrar los errores, dos de las técnicas más utilizadas en las pruebas son las técnicas de 'caja blanca' y 'caja negra'.

La técnica de pruebas de caja negra, consiste en ver el programa que queremos probar como una caja negra despreocupándonos del comportamiento interno y concentrando el esfuerzo en encontrar el comportamiento incorrecto, de acuerdo a las especificaciones de dicho programa, teniendo sólo en cuenta las entradas y salidas de dicho programa.

La técnica de pruebas de caja blanca, al contrario de las pruebas de caja negra, consiste en verificar la estructura interna de un programa.

Si utilizáramos el método de pruebas de caja negra para encontrar todos los errores del programa, como respuesta a la pregunta anterior, generando un caso de prueba para cada entrada, el número de casos de prueba que tenemos que utilizar sería muy elevado y no sería productivo. Para ver esta afirmación se va a proponer un ejemplo. Un programa que clasifica tres entradas numéricas de carácter entero representando cada entrada la longitud de un lado de un triángulo. Dicho programa procesará los datos y mostrará al usuario de qué tipo de triángulo se trata, escaleno, isósceles o equilátero. Ahora bien, tomando como estrategia las pruebas de caja negra, para probar el programa a fondo, tendríamos que crear, como dijimos anteriormente, un caso de prueba para cada entrada del programa. Generando un caso de prueba para encontrar todos los tipos de triángulo, tendríamos un número de casos de prueba muy elevado, es decir, si tenemos la entrada en el programa 3-3-3, tendríamos un triángulo equilátero, pero también tendríamos un triángulo equilátero para las entradas 5-5-5 y 300-300-300. Para probar todos los casos y encontrar todos los errores no sólo probaríamos los casos válidos sino que también tendríamos que probar todos los casos incorrectos, es decir, todo tipo de entradas válidas y no válidas. Esto nos llevaría a un número de casos de prueba infinito, que sería imposible de probar. Si nos parece complicado encontrar todos los errores de un programa con tres entradas, entonces, un programa mucho más complejo, como puede ser un programa que realice la gestión de las cuentas de una empresa con miles de entradas y salidas, sería más complicado todavía.

Si, por lo contrario, eligiéramos las pruebas de caja blanca para contestar a la pregunta, no solo tendríamos que probar todas las líneas de un programa sino que tendríamos que realizar todos los posibles caminos lógicos que se pueden

producir, ya que este tipo de método, como veremos más adelante, se fundamenta en las sentencias de tipo 'if', 'while', 'case', 'for', etc. Por lo tanto, al igual que en las pruebas de caja negra, tendríamos un número de casos de prueba astronómicamente grande, que sería inviable para cualquier tipo de proyecto.

Como vemos, no podemos probar todas las combinaciones y no podemos realizar un test para demostrar que el programa está libre de errores por ello tenemos optimizar las pruebas. Para ello vamos a establecer prioridades que nos marcarán las limitaciones y los objetivos de un proyecto de software.

En el libro [EVE07] Gerald D. Everett y Raymond McLeod, Jr., establecen cuatro prioridades:

- Identificar la magnitud y las fuentes de riesgo del desarrollo reducible por las pruebas .
- Realizar pruebas para reducir los riesgos de negocio identificados.
- Saber cuándo se ha completado la prueba.
- Administrar las pruebas como un proyecto más dentro del desarrollo del proyecto.

Everett y McLeod explican con estos puntos que la principal motivación para probar es reducir los riesgos de un proyecto, para reducir el riesgo de gastos de desarrollo no planificados, o peor aún, el riesgo de fracaso del proyecto por varios motivos. Lo que nos lleva a realizar un análisis exhaustivo de los riesgos de negocio, antes de probar, conociendo con éste, el tamaño del riesgo y la probabilidad de que se produzca.

Por lo tanto, a la hora de probar tenemos que establecer prioridades. Una de las prioridades más importantes que hay que tener en cuenta son los recursos de los que se va a disponer en el proyecto. Al realizar un análisis de los riesgos del negocio queremos conocer cuales son los principales riesgos para asegurar que se dispone de recursos suficientes para poder llevar a cabo las pruebas. Estos recursos irán desde el personal, hasta las herramientas que se vayan a utilizar.

Uno de los principios que se suelen aplicar a la hora de realizar las pruebas es el principio de Pareto, "Regla del 80/20". Este principio dice [WEB12]:

- *"El 80 % de los fallos de un software es generado por un 20 % del código de dicho software, mientras que el otro 80 % genera tan solo un 20 % de los fallos".*

Teniendo en cuenta estas prioridades conseguiremos el objetivo de optimizar las pruebas.

2.2 Tipos de prueba

Hay diferentes tipos de prueba de software. Las que buscan probar una funcionalidad del software, las que buscan probar una característica no funcional, como puede ser la fiabilidad, y las que buscan probar la estructura del software. Teniendo en cuenta esto, vamos a diferenciar los tipos de prueba en tres puntos principales:

- Pruebas funcionales.
- Pruebas no funcionales.
- Pruebas estructurales.

2.2.1 Pruebas funcionales

Este tipo de pruebas se basa en las funcionalidades de un sistema que se describen en la especificación de requisitos, es decir, lo que hace el sistema. También pueden no estar documentadas pero se requiere un nivel de experiencia elevado para interpretar estas pruebas.

Las características de funcionalidad según las establece la ISO 25010 son idoneidad, exactitud, interoperabilidad y seguridad [SCH14]. En la ISO 25010 [WEB11] indican que "*la funcionalidad representa la capacidad del producto de software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones especificas*". La funcionalidad a su vez, la dividen en las siguientes características:

- Completitud funcional: el grado en el que las funcionalidades cubren todas las tareas y objetivos del usuario especificados.
- Corrección funcional: capacidad del producto o sistema para proveer resultados correctos con el nivel de precisión requerido.
- Pertenencia funcional: capacidad del producto de software para proporcionar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados.

Estas pruebas pueden llevarse a cabo en todos los niveles de prueba, como por ejemplo pruebas de componente basadas en una especificación.

Las pruebas funcionales suelen estar asociadas a las técnicas de diseño de pruebas de caja negra, ya que tienen en cuenta el comportamiento externo del software.

2.2.2 Pruebas no funcionales

Este tipo de pruebas tienen en cuenta el comportamiento externo del software, es decir cómo funciona el sistema, y se suelen utilizar técnicas de diseño de caja negra.

Al igual que las características funcionales, las características no funcionales tienen que estar definidas en las especificaciones del producto.

La ISO 25010 también define las características que han de tener estas pruebas que son fiabilidad , facilidad de uso, eficiencia, compatibilidad y seguridad [WEB11].

Según [MYE11], se han de tener en cuenta las siguientes características no funcionales en las pruebas:

- Pruebas de carga: consisten en la medición del comportamiento del sistema para aumentar la carga del mismo, ya sea mediante el número de peticiones que se realizan a una WEB al mismo tiempo, el número de usuarios que trabajan simultáneamente, etc.
- Pruebas de rendimiento: en estas pruebas se medirán la velocidad de procesamiento y el tiempo de respuesta del sistema.
- Pruebas de volumen: se mide la capacidad del sistema para procesar gran cantidad de datos, como procesar archivos con tamaños muy grandes.
- Pruebas de esfuerzo: se realizan pruebas donde se sobrecarga el sistema y se analiza la capacidad de recuperación.

- Pruebas de seguridad: se realizan diferentes pruebas de accesos no autorizados, ataque de denegación de servicio, etc.
- Pruebas de estabilidad, eficiencia, robustez: se realiza una medición de la respuesta del sistema a los errores de funcionamiento.
- Pruebas de compatibilidad: son pruebas del funcionamiento del sistema con los diferentes sistemas operativos, plataformas de hardware, etc., con los que puede interactuar el programa.
- Pruebas de usabilidad: se mide la facilidad de uso, efectividad y satisfacción, siempre dentro de un grupo específico de usuarios.

2.2.3 Pruebas estructurales

Las pruebas estructurales permiten medir la totalidad de las pruebas mediante la evaluación de tipo estructura. En estas pruebas se aplican las técnicas de diseño de caja blanca y el ISTQB utiliza el término 'prueba estructural' para las pruebas de caja blanca.

2.3 Técnicas de prueba

Para conseguir el objetivo de que el producto tenga la calidad deseada vamos a ver diferentes técnicas de prueba que se pueden aplicar a la hora de realizar las pruebas. Estas técnicas tienen el objetivo de identificar condiciones de la prueba, casos de prueba y datos de la prueba.

Estudiaremos tres tipos de técnicas de prueba:

- *Técnicas estáticas.*
- *Técnicas dinámicas.*
- *Técnicas basadas en la experiencia.*

Como se verá más adelante, las pruebas dinámicas detectan los fallos, mientras que las pruebas estáticas detectan sus causas, los defectos.

2.3.1. Técnicas estáticas

Este tipo de técnicas son aquellas que no ejecutan la aplicación. Se llevan a cabo a nivel de especificaciones. No ejecutan código, pero si realizarán un análisis estático del código. Se realizarán revisiones de todos los documentos del proyecto como pueden ser las especificaciones de diseño, de requisitos, los casos de prueba, etc.

2.3.1.1. Análisis estático

El análisis estático tiene como objetivo detectar defectos en el código fuente del software y en los modelos de software, y se realizará sin ejecutar dicho software.

Para llevar a cabo estas pruebas se utilizan herramientas que analizan el código del programa y las salidas generadas.

Estas pruebas ayudan a la detección temprana de defectos, ya sean aspectos sospechosos del código o del diseño. Permiten identificar defectos que no se encuentran fácilmente mediante las técnicas dinámicas.

Según el ISTQB los defectos típicos que se pueden encontrar en las revisiones son:

- Referencias a una variable con un valor definido.
- Interfaces inconsistentes entre módulos.
- Variables que no se utilizan o que se declaran de forma incorrecta.
- Código inaccesible y ausencia de lógica.
- Infracciones de los estándares así como de la sintaxis de código y modelos de software.

2.3.1.2. Revisiones

Las revisiones permiten la detección y corrección temprana de posibles defectos así como la reducción de tiempo y dinero invertido en el desarrollo y pruebas de software.

Según el ISTQB los defectos típicos que se pueden encontrar en las revisiones son:

- Defectos de requisitos.
- Desviaciones de los estándares.
- Defectos de diseño.
- Especificaciones de interfaz incorrectas.

Hay varios tipos de revisiones dependiendo del grado de formalidad de las mismas.

2.3.1.2.1. Revisión informal

En las revisiones informales los revisores carecen de instrucciones escritas, los resultados de las mismas no tienen porque estar documentados y el objetivo principal es obtener defectos de forma barata. Normalmente este tipo de revisiones se llevan a cabo por parte del líder técnico de los diseños y el código.

2.3.1.2.2. Revisión guiada

Estas revisiones pueden variar desde muy formales a bastante informales. Son llevadas acabo por el autor de un documento del proyecto y el objetivo principal es encontrar defectos y establecer un entendimiento común.

2.3.1.2.3. Revisión técnica

Al igual que las revisiones guiadas, estas revisiones pueden variar desde muy formal hasta muy informal. Los objetivos de estas revisiones son debatir, tomar decisiones, evaluar alternativas, resolver problemas técnicos, comprobar la conformidad con las especificaciones, estándares y normativas y se centrarán en alcanzar un consenso. Es un proceso documentado donde se realizará un informe de revisión.

2.3.1.2.4. Inspección

Es la técnica de revisión más formal. Se basa en el examen visual de documentos para detectar defectos como puede ser el no cumplimiento de estándares de desarrollo. Es un proceso documentado e incluye recopilación de métricas y un informe con una lista de conclusiones.

Todas las revisiones tendrán objetivos claros y contarán con las personas adecuadas para cada una de ellas. Existen tres tipos diferentes de personas involucradas en las revisiones:

- Revisor: Persona involucrada en la revisión, identificando las posibles anomalías del producto o proceso bajo revisión.
- Escriba: Persona que registrará en un acta cada defecto y sugerencia para la mejora durante revisión.
- Moderador: Principal persona responsable de una inspección u otro proceso de revisión que mediará entre los distintos punto de vista.

2.3.2. Técnicas dinámicas

Este tipo de técnicas son las realizadas ejecutando la aplicación y son las utilizadas para el diseño de los casos de prueba.

La mayoría del software puede probarse de dos maneras diferentes. Conociendo el funcionamiento interno, podemos probar que todos los módulos encajan unos con otros, es decir, desde una visión interna. Estas pruebas son las pruebas de caja blanca.

Al conocer las funciones específicas del producto se pueden llevar a cabo pruebas que demuestren que estas funciones son operativas y la búsqueda de errores en dichas funciones. Estas pruebas se realizan desde una visión externa, mediante las pruebas de caja negra.

Estas dos técnicas nos ayudarán a definir los casos de prueba para tener la mayor probabilidad de encontrar errores ahorrando esfuerzo y tiempo.

2.3.2.1. Técnica de caja blanca

La técnica de caja blanca , a veces definida como prueba de “caja de cristal” o “caja transparente”, es una técnica de diseño de casos de prueba que usa la estructura de control para obtener los casos de prueba.

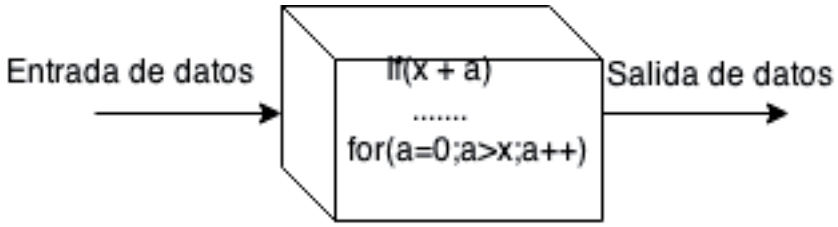

Ilustración 2. Caja blanca

Dentro de esta estructura de control podemos encontrar la estructura de un componente de software como puede ser sentencias de decisiones, caminos distintos del código, la estructura de una pagina web, etc.

Los métodos de prueba de caja blanca aportan los siguientes puntos:

- Garantizan que todas las rutas del código se revisan al menos una vez.
- Revisan las condiciones lógicas.
- Revisan estructuras de datos.

2.3.2.1.1. Pruebas de ruta básica

La prueba de ruta básica es un método de prueba de caja blanca, que inicialmente propuso Tom McCabe.

Este tipo de pruebas se basa en diseñar un caso de prueba por cada camino independiente del programa. Con esta técnica se intenta garantizar que se prueben todos los caminos de ejecución del programa, al menos una vez.

Para aplicar las pruebas de ruta básica hay que conocer la representación que se conoce como *grafo de flujo* y las *rutas de programa independiente* [PRE10].

Notación de grafo de flujo

Esta representación se realiza a partir del PDL (Program Design Language) o pseudocódigo y es una representación de los caminos que puede tomar un programa durante su ejecución.

La notación que se utilizará se muestra en la ilustración 3.

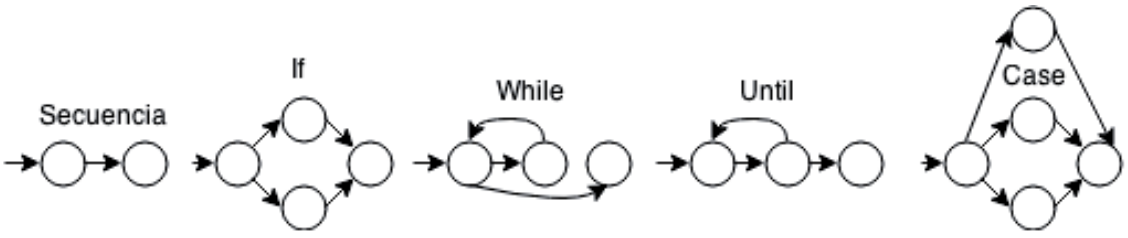

Ilustración 3. Notación de grafo de flujo

Cada estructura de control tiene su correspondiente símbolo en la notación de grafo de flujo. Estas estructuras tienen un punto de entrada y un punto de salida.

En la figura 4-a se muestra un diagrama de flujo. Y en la figura 4-b se muestra la representación del diagrama de flujo con notación de grafo de flujo.

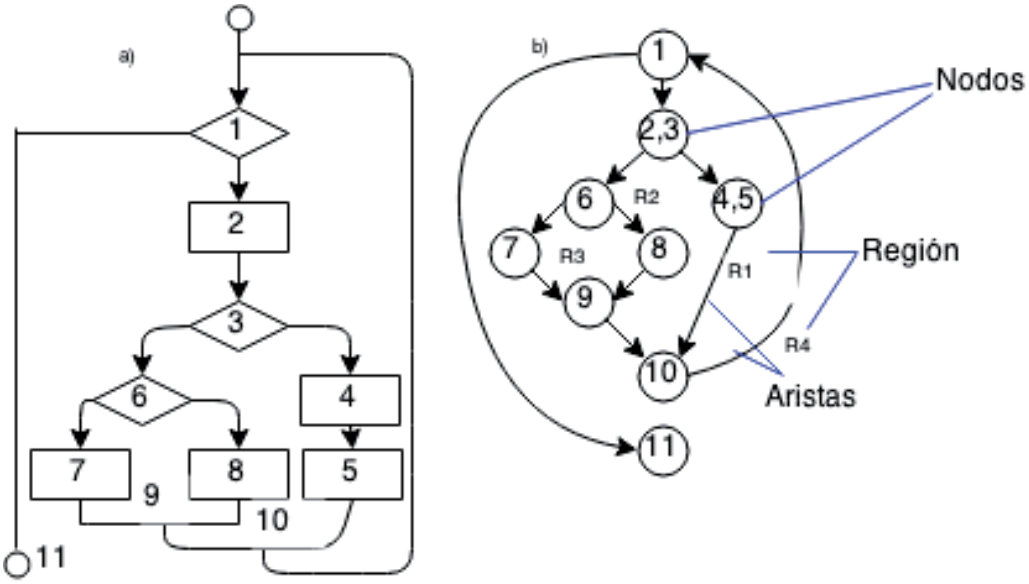

Ilustración 4. a)Diagrama de flujo b)Grafo de flujo.[PRE10]

El diagrama de flujo mostrado no contiene condiciones compuestas en los diamantes de decisión. Dentro del este grafo de flujo tenemos que diferenciar tres partes fundamentales que son los nodos, aristas y regiones.

Los nodos representan una o más instrucciones en donde una secuencia de cajas de proceso y los diamantes de decisión pueden representar un solo nodo.

Las aristas, flechas en el grafo de flujo, representan el flujo de control. Una arista siempre debe terminar en un nodo.

Las regiones son las áreas acotadas por los nodos y aristas. En el momento de contar las regiones, el área fuera del grafo se incluye como región.

Cuando en las estructuras de control existen condiciones compuestas del tipo, AND, OR, NAND, NOR, el grafo de flujo se vuelve más complicado. En estos casos se crearán nodos separados para cada una de las condiciones anteriores. Cada nodo creado se llama nodo predicado y saldrán dos aristas de él. Del ejemplo puesto en la ilustración 4 podemos obtener el siguiente grafo de flujo para condiciones compuestas. Ver ejemplo ilustración 5.

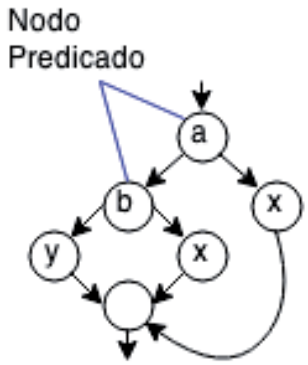

Ilustración 5. Grafo de flujo con condiciones compuestas [PRE10].

Rutas de programa independientes y complejidad ciclomática

La ruta de programa independiente es cualquiera que introduce una nueva condición en el programa o una nueva línea de instrucciones. Si nos fijamos en la ilustración 4-b, veremos que una línea independiente será aquella que recorra una arista por la que no hemos pasado anteriormente. Así pues, para la ilustración indicada anteriormente, podemos decir que tenemos las siguientes rutas:

- Ruta 1: 1-11
- Ruta 2: 1-2-3-4-5-10-1-11
- Ruta 3: 1-2-3-6-8-9-10-1-11
- Ruta 4: 1-2-3-6-7-9-10-1-11

Estas rutas forman lo que se llama un conjunto básico. Un conjunto básico garantiza que toda instrucción del programa será ejecutada al menos una vez.

Pero, ¿cómo sabemos el número de rutas existentes en un programa?

Para conocer el número de rutas independientes de un programa vamos a utilizar 'la complejidad ciclomática' (cyclomatic complexity).

La complejidad ciclomática [WEB13] es una métrica o medición de software que proporciona una evaluación cuantitativa de la complejidad lógica de un programa.

La complejidad ciclomática aplicada a las pruebas de ruta básica permite definir el número de caminos independientes del conjunto básico de un programa y nos permitirá saber el número de pruebas a realizar para asegurarnos de que se ejecuta al menos una vez todas las líneas del programa.

En el [PRE05] muestran que la complejidad ciclomática se puede calcular de tres formas diferentes:

- *La complejidad ciclomática V(G) para un grafo de flujo G se define como V(G) = E – N + 2 donde E es el número de aristas del grafo de flujo y N el número de nodos del grafo de flujo.*
- *También se define como V(G) = P + 1 donde P es el número de nodos predicado contenidos en el grafo de flujo.*
- *La complejidad ciclomática también se corresponde con el número de regiones del grafo de flujo.*

Tomando como ejemplo la ilustración 5 para aplicar las fórmulas anteriores tenemos que:

- Para el primer punto tenemos que el grafo de flujo tiene cuatro regiones.
- En el segundo punto la calculamos como:
  - V(G) = 3 nodos predicado + 1 = 4
- En el tercer puntos tenemos:
  - V(G) = 11 aristas – 9 nodos = 4

2.3.2.1.2. Pruebas de ciclos o bucles

Este tipo de prueba se centra exclusivamente en la validación de las construcciones de bucles.

Según Boris Beizer en su libro 'Software Testing Techniques' [BEI90] se pueden definir cuatro tipos diferentes de bucles:

- Simples: las pruebas pueden aplicarse como bucles simples donde 'n' es el número máximo de iteraciones permitidas por el bucle.

    1. Saltar por completo el ciclo.
    2. Pasar una sola vez a través del ciclo.
    3. Dos pasadas a través del ciclo.
    4. Realizar 'm' pasadas a través del ciclo con m<n.
    5. Hacer n-1, n y n+1 pasos por el bucle.

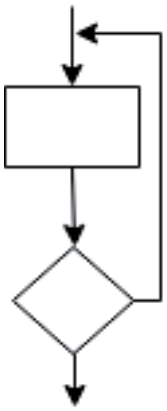

Ilustración 6. Bucle simple

- Concatenados: los ciclos concatenados se pueden enfocar como bucles simples, siempre que cada ciclo sea independiente de los otros, pero si dos o más ciclos que se concatenan no son independientes, es decir, se usa el valor del ciclo 1 como el valor inicial del ciclo 2, entonces se recomienda usar el enfoque aplicado a bucles anidados.

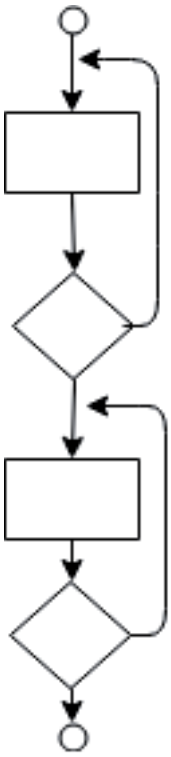

Ilustración 7. Bucle concatenado

- Anidados: el número de pruebas con este tipo de bucles crece exponencialmente conforme el nivel de bucle anidado aumenta. Para reducir el número de pruebas propone seguir los siguientes pasos:

    1. Comenzar con el bucle más interno y establecer todos los otros bucles a valores mínimos.
    2. Realizar pruebas de ciclo simple para el ciclo más interno mientras se mantienen los ciclos exteriores en sus valores mínimos de parámetro de iteración. Agregar otras pruebas para valores fuera de rango.
    3. Trabajar hacia fuera y realizar pruebas para el siguiente ciclo pero mantener los otros ciclos exteriores en valores mínimos y los otros ciclos anidados en valores típicos.
    4. Continuar hasta que todos los ciclos se hayan probado.

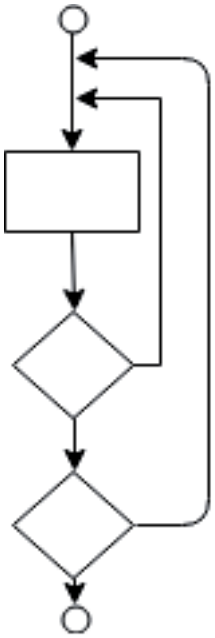

Ilustración 8. Bucles anidados

- No estructurados: estos ciclos Beizer también los denomina como 'horribles' y, siempre que se pueda, esta clase de bucles deben ser rediseñados en base a una programación estructurada.

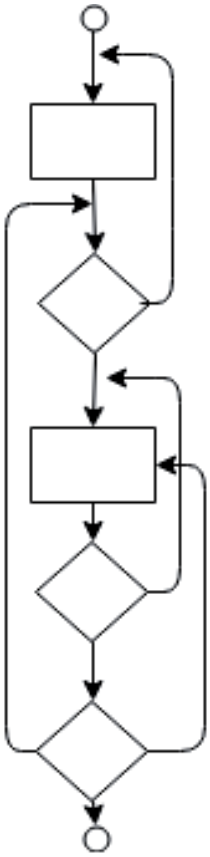

Ilustración 9. Bucle No estructurado

2.3.2.1.3. Pruebas de condición y condición múltiple

Con este tipo de prueba vamos a diseñar los casos de prueba revisando las condiciones lógicas contenidas en un módulo de programa.

Una condición simple es una variable booleana o una expresión relacional. Como ejemplo, una expresión relacional puede tomar la siguiente forma:

- *Expresión 1* <operador relacional> *Expresión 2*.

Las expresiones 1 y 2 son expresiones aritméticas y el operador relacional toma una de las siguientes expresiones:

- <,>,=,≥,≠,≤

Cuando nos referimos a condición múltiple nos estamos refiriendo a dos o más condiciones simples que incluyan OR, AND y NOT.

Una condición sin expresiones relacionales es una expresión booleana. Si una condición es incorrecta, entonces al menos un componente de la condición es incorrecto. Este método se enfoca en la prueba de cada condición del programa para asegurar que no contiene errores.

2.3.2.2. Técnica de caja negra

Las técnicas de diseño de caja negra, también llamadas pruebas de comportamiento, son las que utilizan el análisis de la especificación, tanto funcional como no funcional, sin tener en cuenta la estructura interna del programa para diseñar los casos de prueba y, a diferencia de las pruebas de caja blanca, estas pruebas se suelen realizar durante las últimas etapas de la prueba.

Ilustración 10. Caja negra

Con los métodos de caja negra se intenta encontrar los errores:

- Funciones incorrectas o faltantes.
- Errores de inicialización y terminación.
- Errores de interfaz.
- Errores en las estructuras.

Hay varios métodos que se pueden aplicar a la hora de escoger la técnica de caja negra como modelo para las pruebas.

2.3.2.2.1. Partición de equivalencia

En las pruebas de partición de equivalencia, los valores de entrada del programa o del sistema se dividen en grupos que vayan a tener un comportamiento similar, de manera que puedan ser procesados de la misma forma. Las particiones de equivalencia o clases son aplicables a datos válidos y datos no válidos. También pueden aplicarse a los valores de salida, valores internos, valores relativos al tiempo o a los parámetros de interfaz [ISTQB].

Las clases de equivalencia se definen de acuerdo a las siguientes directrices [PRE05]:

- Si una condición de entrada especifica un rango, se definen una clase de equivalencia válida y dos inválidas.
- Si una condición de entrada requiere un valor específico, se definen una clase de equivalencia válida y dos inválidas.
- Si una condición de entrada especifica un miembro de un conjunto, se definen una clase de equivalencia válida y otra inválida.
- Si una condición de entrada es booleana, se definen una clase de equivalencia válida y otra inválida.

Al aplicar estas directrices se desarrollarán y ejecutarán los casos de prueba para cada objeto de los datos del dominio de entrada.

2.3.2.2.2. Análisis del valor limite

Es mayor el número de errores que se presenta en los límites del dominio de entrada que en el centro [PRE05].

Esta técnica de diseño de casos de prueba complementa la técnica de partición de equivalencia u otras pruebas de caja negra.

Los valores máximos y mínimos de una partición son sus valores límites [ISTQB]. Las pruebas pueden diseñarse para cubrir tanto los valores límites válidos, como no válidos.

Las directrices para el análisis del valor límite son similares a las de partición de equivalencia [PRE05]:

- Si una condición de entrada especifica un rango limitado por los valores a y b, los casos de prueba deben diseñarse con esos valores, además de los que se encuentran por encima y por debajo de ellos.
- Si una condición de entrada especifica diversos valores, deben desarrollarse casos de prueba que ejerciten los números máximo y mínimo. También se prueban los valores ubicados por encima y por debajo de estos máximos y mínimos.
- Aplicar las directrices 1 y 2 a las condiciones de salida.
- Si la estructura interna de datos del programa tiene límites prescritos debe diseñarse un caso de prueba para ejercitar los límites de la estructura de datos.

2.3.2.2.3. Pruebas de tabla de decisión

Estas pruebas se realizan cuando la lógica a probar esta basada en decisiones o, dicho de otra manera, donde la lógica predominante es del tipo if-then else [WEB14].

Las tablas de decisión constituyen una buena manera de reflejar los requisitos del sistema que contienen condiciones lógicas y de documentar el diseño del sistema interno. Incluyen las condiciones de activación, a menudo

combinaciones de verdadero y falso, para todas las condiciones de entrada y las acciones resultantes para cada combinación [ISTQB].

La ventaja de la tabla de decisión es que crea combinaciones que de otro modo no se habrían encontrado.

2.3.2.2.4. Pruebas de transición de estado

Un sistema puede mostrar respuestas diferentes en función de sus condiciones actuales o su estado. Estos aspectos del sistema pueden mostrarse mediante el diagrama de transición de estados [ISTQB].

El objeto de estas pruebas es partir de un estado inicial y recorrer los diferentes estados mediante eventos que desencadenan cambios de estado.

Una tabla de estado muestra la relación entre los estados y las entradas, y eventualmente puede poner de manifiesto posibles transiciones no válidas [ISTQB].

Las pruebas de transición de estados deben ejecutar todas las funciones de un estado al menos una vez. Pueden diseñarse pruebas para cubrir una secuencia típica de estados, cubrir todos los estados, ejercitar secuencias específicas de transiciones o probar transiciones inválidas [ISTQB].

2.3.2.2.5. Pruebas de caso de uso.

Estas pruebas son las que se derivan de los casos de uso. Un caso de uso expresa todas las maneras de utilizar un sistema para alcanzar una meta particular para un usuario. En conjunto, los casos de uso le proporcionan todos los caminos útiles de usar el sistema e ilustran el valor que este provee [WEB15].

Los casos de uso dejan claro lo que hará un sistema y, por omisión intencional, lo que no hará. Estos posibilitan una visión efectiva, el manejo del alcance y el desarrollo incremental de sistemas de cualquier tipo y cualquier tamaño [WEB15].

Los casos de prueba derivados de los casos de uso resultan muy útiles a la hora de descubrir defectos en todos los caminos útiles durante el uso real del sistema. [ISTQB].

Los casos de uso son de gran utilidad para diseñar las pruebas de aceptación con la participación del cliente/usuario.

2.3.3. Técnicas basadas en la experiencia

El ISTQB define también las técnicas basadas en la experiencia y las define como, *"las pruebas basadas en la experiencia son aquellas en las que las pruebas se derivan de la habilidad e intuición del probador y de su experiencia con aplicaciones y tecnologías similares"*.

Estas pruebas pueden tener poca o mucha efectividad dependiendo de la experiencia del probador y suelen aplicarse en proyectos donde la documentación es escasa o inadecuada.

Dos de las técnicas basadas en la experiencia son:

- Predicción de error.
- Pruebas exploratorias.

2.3.3.1. Predicción de error

En este tipo de pruebas, se diseñarán los casos de prueba en base a la experiencia del probador. Se anticiparán los errores en base a la experiencia de quien prueba el sistema o componente y se diseñarán pruebas específicas en base a esa experiencia para encontrar los errores.

2.3.3.2. Pruebas exploratorias

Este tipo de pruebas fue propuesto por Cem Kaner en 1983, el cual define las pruebas como, *"las pruebas exploratorias son un estilo de pruebas que hacen hincapié en la libertad personal y la responsabilidad del probador para optimizar continuamente el valor de su trabajo mediante el aprendizaje de las pruebas, diseño*

de pruebas, ejecución de las pruebas y la interpretación de los resultados de las pruebas como actividades que se apoyan mutuamente y que se ejecutan paralelamente a lo largo del proyecto" [WEB16].

Estas pruebas basan su principal característica en que el aprendizaje, el diseño y la ejecución de las pruebas, se realiza de forma simultanea. Durante la realización de las pruebas se irán realizando más pruebas basadas en la experiencia de la ejecución anterior.

En el articulo [WEB17] James Bach expone: "*En términos operativos, las pruebas exploratorias son un proceso interactivo de la exploración del producto, la prueba diseño y ejecución de la prueba. El resultado de una sesión de prueba exploratoria es un conjunto de notas sobre el producto, los fracasos que encontró , y un registro de cómo se probó el producto. Cuando se practica por probadores experimentados, produce siempre resultados valiosos y auditables*".

2.4 Estrategia de prueba

En este capítulo vamos a ver los tipos de prueba según se realizan dentro del ciclo de vida del software.

Existen diferentes modelos de desarrollo de software. Entre ellos tenemos métodos clásicos como pueden ser el modelo en cascada, el modelo general en V o métodos más populares hoy en día como pueden ser metodologías ágiles, programación extrema, etc. Cada uno de estos métodos tiene una vía de trabajo diferente, indicando cómo se tiene que trabajar durante la vida del proyecto.

Las pruebas están dentro de estos modelos de maneras muy diferentes. Por ejemplo, dentro del modelo en cascada las pruebas se ejecutan al final del proyecto y dentro del modelo en V se realizan en los diferentes niveles de desarrollo.

2.4.1. Modelo en V

Se va a utilizar el modelo en V como referencia para ver cómo se integran las pruebas dentro del ciclo de vida del software. En este modelo las tareas de desarrollo y pruebas tienen la misma importancia.

En la ilustración 11 vemos que contiene dos ramas, una representa las actividades de desarrollo (izquierda) y la otra representa las actividades de integración y pruebas (derecha).

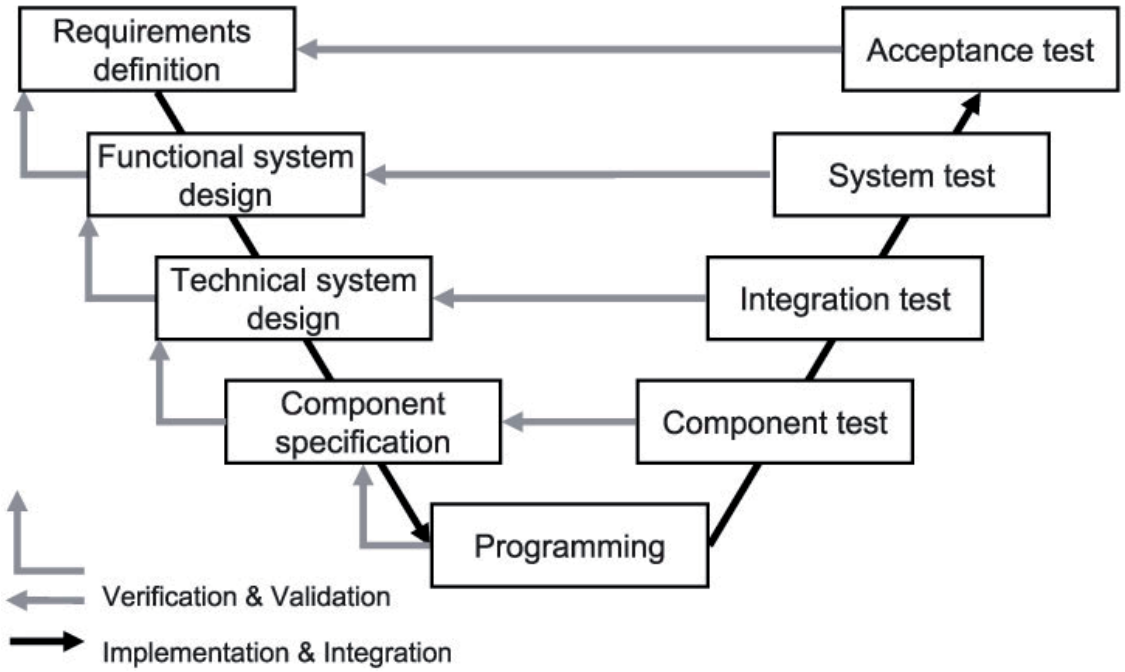

Ilustración 11. Modelo en V [SCH14]

2.4.2. Niveles de prueba

Para cada nivel de desarrollo se define un nivel de prueba. Dentro de cada nivel de prueba el probador debe asegurarse de que los resultados cumplen con la verificación y la validación del software.

2.4.2.1. Verificación y validación.

Según [PRE05], la prueba de software es un elemento más de lo que se denomina 'Verificación' y 'Validación' (VyV). "Verificación es el proceso de asegurar que el software implementa correctamente una función específica" y "validación es un conjunto de actividades que asegura que el software desarrollado respeta los requisitos del cliente que se encuentran en las especificaciones de requisitos".

Bohem [WEB18], tiene una visión diferente sobre estos términos. Sobre el término verificación se pregunta, *"¿estamos construyendo el producto correctamente?"* y sobre validación, *"¿estamos construyendo el producto correcto?"*.

Las pruebas son una parte fundamental de la VyV pero también se necesitan actividades como revisiones técnicas, monitorización de desempeño, simulación,

revisión de la documentación, pruebas de desarrollo, etc., actividades para asegurar la calidad del software.

Dentro de cada nivel de prueba, el probador tiene que asegurar la verificación y validación.

2.4.2.2. Pruebas unitarias o de componentes.

El primer nivel de las pruebas es el de la prueba unitaria o de componente que consiste en la verificación de unidades del software de forma aislada, es decir, probar el correcto funcionamiento de una unidad de código.

En el [PRE05] se indica que una unidad de código es considerada una unidad de programa, como una función o método de una clase que es invocada desde fuera de la unidad y que puede invocar otras unidades. Es por ello que hay que probar que cada unidad funcione separada de las demás unidades de código.

Estas pruebas suelen ser realizadas por los desarrolladores, ya que es muy recomendable conocer el código fuente del programa y generalmente se realizarán pruebas de caja blanca o se analizará el código para comprobar que cumple con las especificaciones del componente.

No obstante, no solo se realizarán pruebas de la estructura del código sino que también se generarán casos de prueba funcionales para comprobar el funcionamiento del componente.

2.4.2.3. Pruebas de integración.

El segundo nivel de las pruebas dentro del modelo en V es la prueba de integración. En casi toda la documentación consultada, se hacen siempre la siguiente pregunta: *¿por qué las pruebas de integración son necesarias si ya hemos probado que los componentes individualmente funcionan?.* Aunque los componentes funcionen bien individualmente, puede darse el caso de que al juntar los diferentes componentes del sistema ocurran errores que no teníamos contemplados en un primer momento.

El ISTQB nos dice que estas pruebas se ocupan de probar las interfaces entre los componentes, las interacciones con distintas partes de un mismo sistema, como el sistema operativo, el sistema de archivos, el hardware y las interfaces entre varios sistemas.

¿En qué orden llevamos a cabo la integración de los componentes?

Para llevar a cabo estas pruebas en el [SCH14] nos muestran las siguientes estrategias para la prueba de integración:

- Integración descendente.
- Integración ascendente.
- Integración Ad hoc.
- Integración del esqueleto.

2.4.2.3.1. Integración descendente

La prueba de integración descendente es un enfoque incremental para la construcción de la arquitectura del software. La prueba se iniciará con el componente de más alto nivel del sistema que llama a otros componentes del sistema pero no a sí mismo. La integración avanza con componentes de nivel inferior.

2.4.2.3.2. Integración ascendente

La prueba comienza con los componentes elementales del sistema que no requieren componentes adicionales. Los subsistemas más grandes se ensamblan a partir de los componentes probados.

2.4.2.3.3. Integración Ad-hoc

Los componentes se han integrado en el orden en que están terminados. Cuando un componente ha pasado la prueba de componentes, se inicia la prueba de integración para ver si encaja con otro componente ya probado y se inicia la prueba de integración.

2.4.2.3.4. Integración del esqueleto

Cuando un esqueleto o columna vertebral del sistema se ha terminado, los componentes se integran gradualmente en él.

2.4.2.4. Pruebas de sistema

Una vez que se han probado los componentes y la integración de los mismos, entramos en el tercer nivel de prueba, donde se va a comprobar si el producto cumple con los requisitos especificados.

El ISTQB nos dice que las pruebas de sistemas pueden incluir pruebas basadas en riesgos y/o especificaciones de requisitos, procesos de negocio, casos de uso u otras descripciones de texto de alto nivel o modelos de comportamiento de sistema, interacciones con el sistema operativo y recursos del sistema.

Las pruebas de sistema deben de estudiar los requisitos funcionales y no funcionales del sistema y las características de calidad. Para ello se aplicarán técnicas de prueba de caja negra.

2.4.2.5. Pruebas de validación o aceptación.

Las pruebas indicadas anteriormente se encuentran bajo la responsabilidad de quien está produciendo el programa. Las pruebas de aceptación son responsabilidad del cliente y pueden ser la única parte de las pruebas en donde estén involucrados.

Estas pruebas se llevan a cabo antes de que el programa se ponga en funcionamiento en real y tienen que satisfacer las expectativas del cliente.

Hay cuatro formas típicas de las pruebas de aceptación [ISTQB]:

- Pruebas de aceptación del contrato.
- Pruebas de aceptación del usuario.
- Pruebas operativas.
- Pruebas alfa y beta.

2.4.2.5.1. Pruebas de aceptación del contrato

Las pruebas de aceptación del contrato toman como base los criterios de aceptación previstos en el contrato realizado al principio del proyecto.

Estas pruebas son muy importantes, ya que la empresa encargada de desarrollar el software ha podido malinterpretar los criterios de aceptación y no tener en cuenta aspectos que sí tiene en cuenta el cliente. Por esto, el software se entrega al cliente para que realice sus pruebas. Estas pruebas suelen ser las mismas que se realizaron en las pruebas de sistema, con la salvedad de que se realizan en el entorno del cliente, característica muy importante, ya que pueden aparecer errores que antes no aparecieron.

2.4.2.5.2. Pruebas de aceptación del usuario

Hay casos en los que el cliente y el usuario final son diferentes y lo que le parece válido a un usuario final, puede ocurrir que no le parezca válido a otro. Por este motivo, es fundamental realizar pruebas con los usuarios finales.

2.4.2.5.3. Pruebas operativas

Estas pruebas son llevadas a cabo por los administradores del sistema que se va a poner en producción. En estas pruebas se incluyen las siguientes tareas:

- Copia de seguridad/restauración.
- Recuperación de desastres.
- Gestión de usuarios.
- Comprobación de vulnerabilidades de seguridad.
- Carga de datos.
- Tareas de mantenimiento.

2.4.2.5.4. Pruebas alfa y beta

Suele ocurrir que cuando se realiza la entrega al cliente de un programa, este puede ser usado por diferentes usuarios finales y que la cantidad de usuarios finales sea muy amplia. Cuando esto ocurre no es práctico realizar pruebas de aceptación con cada uno de estos clientes, para lo cual se utilizan las pruebas de alfa y beta que van a descubrir errores que sólo el usuario final va a encontrar.

Las pruebas alfa son las que se ejecutan en las oficinas del desarrollador del producto por un grupo de personas que representa al cliente final. En estas pruebas, el desarrollador estará junto a estos usuarios registrando errores y problemas de uso.

Las pruebas beta se realizan en las oficinas de un cliente o en varias situaciones específicas, ya que el usuario final puede ser varios clientes a los que se les entrega el producto. En estas pruebas el desarrollador no está presente. Estos clientes registran todos los problemas derivados del uso de este producto que más tarde se harán llegar al desarrollador.

2.5 Otros tipos y conceptos de pruebas.

2.5.1. Pruebas de aplicaciones WEB.

Los objetivos que existen a la hora de probar aplicaciones WEB no difieren mucho de los que tenemos al probar el software en general.

Lo primero que vamos a tener en cuenta, cómo ya dijimos en la definición de prueba, es que se van a realizar pruebas para encontrar errores.

Las aplicaciones web interactúan con diferentes sistemas operativos, navegadores, hardware, protocolos de comunicación, por lo que la búsqueda de errores puede ser más complicada.

Para entender los objetivos de realizar pruebas en una aplicación WEB se tienen que tener en cuenta varias dimensiones de calidad. En el libro [PRE05] se indica que se tienen que tener en cuenta las siguientes dimensiones:

- *El contenido*: se evalúa a nivel sintáctico y semántico.
- *La función*: descubrir errores en función de los requerimientos del cliente.
- *La estructura*: se entrega correctamente el contenido, es extensible y permite agregar nuevas funcionalidades.
- *La usabilidad*: se asegura que la interfaz soporta a cada categoría de usuario.
- *La navegabilidad*: descubrir errores de navegación, como links rotos, inadecuados y erróneos.
- *La performance*: se prueba bajo funcionamientos extremos para asegurar que no sufre degradación.
- *La compatibilidad*: se realizan pruebas para garantizar varias configuraciones de *host*, tanto en el cliente como en el servidor.
- *La interoperabilidad*: se prueba para garantizar que interactúa correctamente con otras aplicaciones.

- *La seguridad*: se realiza la evaluación de las potenciales vulnerabilidades.

Para llevar a cabo las pruebas se utilizarán las mismas estrategias y métodos vistos para el software general.

2.5.2. Pruebas de regresión, humo y usabilidad

2.5.2.1. Pruebas de regresión

Según [BLA07] la regresión la podemos encontrar de tres maneras básicas.

La primera es una "*regresión local"* donde el cambio o corrección de errores crea un nuevo error. La segunda es una "*regresión al descubierto"* que ocurre cuando la modificación o corrección de errores revela un error existente. Y la tercera es una "*regresión a distancia"* donde una solución de un error en un componente produce otro error en otro componente del sistema.

Tenemos que las pruebas de regresión son aquellas que se realizan cuando el software ha sufrido un cambio, ya que este cambio se puede tratar de un nuevo módulo dentro del programa o de cambios que se han hecho en el software al solucionar errores, ya que tanto el cambio como la solución pueden producir nuevos errores.

Estas pruebas son muy importantes porque lo que en principio ya funcionaba puede haber dejado de funcionar y no se puede caer en la trampa de pensar "*esto ya funcionaba antes, tiene que funcionar*".

En [BLA07] se proponen varias estrategias a seguir para realizar las pruebas de regresión de las cuales vamos a realizar mención de algunas de ellas.

Una de las estrategias que podemos seguir a la hora de realizar las pruebas de regresión es la de 'volver a realizar todas las pruebas' o como indican en el libro, 'estrategia de la fuerza bruta'. Esta estrategia minimizará los riesgos de calidad pero aumentará el costo del proyecto ya que requiere de más tiempo y recursos para realizar las pruebas que las demás estrategias.

Otra estrategia será la de repetir sólo algunas de las pruebas. Para ello tenemos que conocer muy bien el sistema que vamos a probar y los riesgos de

negocio, ya que serán ejecutadas sólo alguna parte de las pruebas que ya se realizaron.

Antes de comenzar la regresión tenemos que tener muy en cuenta el tiempo y los recursos de los que se va a disponer para llevar a cabo las pruebas.

Otra estrategia, que según opinión del autor del proyecto, es la manera más eficiente de realizar pruebas de regresión siempre que el proyecto lo permita, es la automatización de las pruebas que nos permitirá realizar la regresión utilizando menos tiempo y recursos. Pero tenemos que tener en cuenta que no todas las pruebas y todos los sistemas son automatizables.

2.5.2.2. Pruebas de humo

Las pruebas de humo son aquellas en las que se realiza una revisión rápida del programa comprobando que funciona como tiene que funcionar y que no se interrumpen los procesos básicos.

Estas pruebas pueden ejecutarse durante todo el proceso del desarrollo por los desarrolladores antes de la entrega de versiones al equipo de pruebas, o por el equipo de pruebas antes de entregar una versión al cliente final.

2.5.2.3. Pruebas de usabilidad

La usabilidad es una cualidad que todos los productos desean tener pero que no todos consiguen, por esto, las pruebas de usabilidad son cada vez más aceptadas por las empresas que someten sus productos a unas pruebas de usabilidad cada vez mayores.

Este tipo de pruebas consiste en la prueba del producto por parte de muchos usuarios para comprobar que el producto funciona correctamente cumpliendo el propósito para el cual fue diseñado.

Se medirán variables como:

- La eficiencia: si el programa funciona con una rapidez suficiente para que el usuario lleve a cabo sus tareas eficientemente.

- La efectividad: los usuarios comprobarán que el producto funciona como esperaban.
- La facilidad de manejo: donde se comprobará si el usuario es capaz de operar con el producto con una cierta facilidad teniendo un periodo de formación.
- La satisfacción: satisfacción de los usuarios y la percepción y opiniones que tienen sobre el producto en general.

2.5.3. Pruebas de automatización

Las pruebas que se aplican en un proyecto, aunque este sea un proyecto pequeño, pueden llegar a ser muy numerosas. Por lo que para realizar pruebas como son la regresión, que exigen realizar las mismas pruebas que ya se han realizado, conllevaría el gasto de una gran cantidad de recursos y tiempo, normalmente más del que un proyecto se puede permitir para realizar las pruebas.

En el libro [DUS99] mencionan dos afirmaciones muy usadas en los proyectos que son, "*Necesitamos la nueva aplicación de software tan pronto como se pueda*" y "*Necesito estas nuevas características en la aplicación ya*".

Las pruebas automáticas aportan al proyecto velocidad, eficiencia, precisión y resistencia, ya que con la automatización se incrementa la velocidad al ejecutar las pruebas y, al no ejecutarlas manualmente por un probador, éste se puede dedicar a realizar otras tareas del proyecto. Las pruebas manuales pueden presentar defectos al ser ejecutadas de nuevo por los probadores ya que al probar de nuevo el sistema se tiende a relajar el nivel de búsqueda de errores y la herramienta utilizada para automatizar nunca se cansa ni se distrae, pudiendo ser ejecutada en horas fuera del proyecto.

Las pruebas se realizan mediante la generación de scripts, que ejecutarán el software de manera automática. Para la generación de estos scripts se encuentran en el mercado varias herramientas, de pago y libres. Éstas funcionan con diferentes lenguajes de programación, lo que permite una gran diversidad a la hora de elegir una de ellas para realizar la automatización de las pruebas del proyecto.

2.5.4. ¿Qué es un "plan de pruebas"?

La planificación de las pruebas es uno de los puntos más importantes a la hora de probar un proyecto, tenemos que tener controlados todos los aspectos, desde la manera de la que vamos a probar, cómo lo vamos a probar, los recursos de los que vamos a disponer, las pruebas de regresión , la documentación del proyecto, etc.

Hay varias maneras de realizar un plan de pruebas. Una de ellas es aplicar la norma "*ISO/IEC 29119 Software testing*" que sustituye a la norma "*IEEE 829*" y que define en el punto 3 como tiene que ser la estructura del plan de prueba. En esta norma se indican los puntos que tiene que tener un plan de prueba [WEB19]:

- Identificador del plan de pruebas.
- Introducción.
- Elementos de la prueba.
- Características a probar y no probar.
- Método/enfoque de prueba (implementación de la estrategia de prueba).
- Criterios de paso/fallo de la prueba.
- Criterios de entrada, salida, suspensión y reanudación de las pruebas.
- Entregables de las pruebas.
- Tareas/hitos clave de las pruebas.
- Cronograma.
- Necesidades del entorno.
- Responsabilidades.
- Dotación de personal y necesidades de capacitación.
- Riesgos.
- Aprobaciones.

La planificación de las pruebas es uno de los principales puntos a la hora de probar el software con éxito. En ausencia de un buen test plan es muy poco probable que las pruebas se ejecuten en un plazo estipulado e incluso se incrementara el costo del proyecto al consumir más tiempo y recursos. Esta planificación tiene que estar incluida dentro de la planificación general de un proyecto de software.

2.5.5. Normas y certificaciones

Hay varios órganos reguladores internacionales que documentan en sus normas varios estándares para las pruebas de software, como son:

- IEEE (Institute of Electrical and Electronics Engineers) [WEB20].
- ISO (International Organization for Standardization) [WEB21].
- AENOR (Asociación Española de Normalización y Certificación) [WEB22].

También existen varias instituciones internacionales que se dedican a formar en el ámbito de las pruebas y difundir los beneficios de éstas:

- ISTQB (International Software Testing Qualification Board) [ISTQB].
- ISST (International Society for Software Testing) [WEB23].
- TMMI (Testing Maturity Model Integration) [WEB24].

2.5.5.1. Normas fundamentales de pruebas

Las Normas más importantes para las pruebas de software son:

- ISO/ IEC 29119 Pruebas de software:
  - Parte 1: Definiciones y vocabulario.
  - Parte 2: Proceso de prueba.
  - Parte 3: Documentación de prueba.
  - Parte 4: Técnicas de ensayo.
  - ISO/IEC 33063 – Modelo de proceso de evaluación para los procesos de pruebas de software.

- Con esta norma se reemplaza a las normas:
  - IEEE 829 – Documentación de prueba.
  - IEEE 1008 – Unidad de pruebas.
  - BS 7925-1 – Vocabulario de términos en pruebas de software.
  - BS 7925-2 – Componente de pruebas de software estándar.

- ISO 25000 – División para gestión de calidad.
  - ISO/IEC 2501n – División para el modelo de calidad.
  - ISO/IEC 2502n – División para la medición de calidad.
  - ISO/IEC 2503n – División para los requisitos de calidad.
  - ISO/IEC 2504n – División para la evaluación de calidad.
  - Esta norma esta basada en:
    - ISO/IEC 9126.
    - ISO/IEC 14598.

- ISO 15504 – Norma para evaluar procesos:
  - Modelo Pathfinder.
  - Modelo AENOR.
  - ISO 12207.

2.5.5.2. Certificaciones

El aumento de la demanda de las empresas a probar sus productos ha hecho que se incremente la demanda y la competencia en el mercado a la hora de encontrar a personal formado en el área de pruebas. Se han creado instituciones internacionales que se dedican a formar mediante certificaciones.

Tres de las principales certificaciones como hemos visto antes, son ISST, ISTQB y el TMMI.

Cada una de estas certificaciones tienen diferentes niveles que van desde nivel principiante, en el que se imparten los conocimientos fundamentales de las

pruebas, hasta niveles avanzados que para poder ser cursados, el alumno debe aportar varios años de experiencia laboral en el mundo de las pruebas.

3. HERRAMIENTAS DE PRUEBAS

Todos los proyectos, por muy pequeños que sean, pueden llegar a tener una cantidad de casos de pruebas muy elevado, sin contar que las pruebas se repetirán varias veces debido a las pruebas de regresión. Estos proyectos, necesitan de una administración, planificación y ejecución, así como de herramientas que permitan realizar pruebas automáticas.

Para llevar a cabo estas tareas existen diferentes tipos de herramientas que ayudarán en todo lo posible a que el proyecto se maneje más eficientemente y que ayudarán a conseguir la calidad deseada. Existen herramientas que se utilizan para diseñar casos de prueba, gestionar y administrar pruebas y monitorizar sistemas en pruebas. Una simple hoja de datos puede ser considerada una herramienta de pruebas. Al inicio del proyecto, el desarrollador y el probador son los encargados de estudiar y plantear el tipo de herramientas necesarias que se van a usar durante el proyecto.

Se van a clasificar en varias categorías dependiendo del uso que se pueda hacer de ellas:

- Herramientas para pruebas estáticas.
- Herramientas para planificación y gestión de pruebas.
- Herramientas para pruebas de automatización.
- Drivers y Stubs.
- Herramientas para pruebas carga y rendimiento.
- Herramientas de monitorización y seguridad.

En los ejemplos que se mencionarán de las diferentes herramientas se diferenciará entre herramientas de código abierto que serán gratuitas (libres) y herramientas comerciales (pago por licencia), ya que todas ellas tienen sus ventajas e inconvenientes.

3.1. Herramientas para pruebas estáticas

Este tipo de herramientas ayudan a encontrar defectos en las etapas tempranas del proyecto.

Existen diferentes tipos de herramientas de pruebas estáticas. Se van a diferenciar entre herramientas de revisión, análisis estático y herramientas de modelado.

3.1.1. Herramientas de revisión

Son útiles en los procesos de revisión, listas de comprobación y directrices de revisión, y se utilizan para almacenar y comunicar comentarios de revisión, informes sobre defectos y esfuerzos [ISTQB].

3.1.2. Análisis estático

Estas herramientas sirven para localizar defectos en el código antes de realizar las pruebas dinámicas proporcionando soporte para aplicar las normas de codificación, análisis de estructuras y las dependencias [ISTQB].

3.1.3. Herramientas de modelado

Herramientas que sirven para validar modelos de software, como por ejemplo modelos de datos físicos (PDM) de una base de datos relacional, enumerando inconsistencia y localizando defectos [ISTQB].

3.1.4. Ejemplos de herramientas

Hay una gran cantidad de herramientas presentes para las pruebas estáticas. A continuación se muestran algunas de estas herramientas:

- **PMD** (libre): es un analizador de código fuente. Encuentra defectos comunes de programación como las variables utilizadas, bloques catch vacíos, la creación de objetos innecesarios, y así sucesivamente [WEB25].

- **ChekStyle** (libre): es una herramienta de desarrollo para ayudar a los programadores a chequear que el código Java cumple un estándar de codificación [WEB26].
- **SONAR** (libre): es una herramienta que permite analizar, recopilar y visualizar métricas del código fuente. Podríamos decir que es una mezcla entre las dos anteriores PMD y ChekStyle [WEB27].
- **Simian** (libre): herramienta que detecta software duplicado. Entre los lenguajes de programación que acepta están, Java, C#, C++, C COBOL, Ruby, ASP, JSP, HTML, XML y Visual Basic [WEB28].
- **FindBugs** (libre): es una herramienta que detecta errores en el código. Se utiliza en java y es capaz de encontrar errores con el manejo de hilos, el mal uso de los métodos del API, etc. [WEB29].
- **MacCabeTest** (pago por licencia): implementa una gran variedad de técnicas de prueba derivadas de una evaluación de complejidad ciclomática y de otras mediciones [WEB30].

3.2. Herramientas para planificación y gestión

Las herramientas de gestión ayudan al probador a documentar y evaluar los casos de pruebas. Tienen como objetivo proporcionar mecanismos que permitan realizar de una manera controlada la documentación, el mantenimiento de las pruebas y la gestión de resultados. Dentro de las herramientas de gestión no sólo tenemos las que gestionan las pruebas, sino que también tenemos las que gestionan incidencias. Estas herramientas se utilizan para documentar, analizar, distribuir y gestionar las incidencias dentro de un proyecto

Las herramientas de planificación permiten al probador escoger una estrategia de prueba así como tener una visión general de los casos de prueba del proyecto.

3.2.1. Ejemplos de herramientas

- **Teslink** (libre): es una herramienta que permite crear y gestionar casos de prueba y organizarlos dentro de planes de prueba. Se puede ejecutar los casos de prueba a partir de los planes creados en la misma herramienta. También permite la generación de informes, así como priorizar y asignar tareas [WEB31].
- **Redmine** (libre): es una aplicación para gestión y planificación de proyectos con interfaz web. Esta diseñada para facilitar el control y seguimiento de proyectos [WEB32].
- **Trello** (libre): es una herramienta con interfaz web que permite organizar proyectos y tareas. Esta basada en el método ágil Kanban. [WEB33].
- **Mantis** (libre): es una herramienta para gestionar incidencias. Permite llevar un control y mantener un historial de las incidencias así como especificar un número indeterminado de opciones de la incidencias, como los estados de éstas (abierta, cerrada, resuelta, reabierta, etc.), la severidad (baja, media, alta), etc. Es una herramienta con interfaz web [WEB34].

- **HP Quality Center** (pago por licencia): esta herramienta es un todo en uno propiedad de la empresa HP. El objetivo de esta herramienta es el control de la calidad de software y tiene varios módulos que nos permitirán gestionar los requisitos de un proyecto, la gestión, el diseño y la creación de pruebas y la gestión de incidencias. También tiene un módulo para pruebas automáticas que se verá más adelante [WEB35].
- **IBM Rational Quality Manager** (pago por licencia): tiene las mismas funciones y características que la herramientas de HP pero en este caso la empresa propietaria de esta herramienta es IBM [WEB36].
- **Testopia** (libre): Es un administrador de casos de prueba. Está diseñado para realizar el seguimiento de casos de prueba desde el diseño de las pruebas hasta la gestión de resultados. Es una herramienta con interfaz Web que maneja extensiones para interactuar con Bugzilla [WEB37].
- **BugZilla** (libre): Herramienta de gestión de incidencias al estilo de Mantis desarrollada por Mozilla [WEB38].

3.3. Herramientas para pruebas de automatización

El objetivo de estas herramientas es la creación de scripts en diferentes lenguajes de programación, dependiendo de la herramienta que vayamos a utilizar, que permitan ejecutar las pruebas funcionales automáticamente.

3.3.1. Ejemplos de herramientas

- **Selenium** (libre): es un conjunto de herramientas para automatizar los navegadores web a través de muchas plataformas que nos permitirá crear conjuntos de prueba sobre aplicaciones web. Permite la creación de scripts mediante una gran variedad de lenguajes de programación como Java, Ruby, Phyton, etc. [WEB44].
- **QTP - Quick Test Professional** (pago por licencia): es el módulo de automatización de la empresa HP. Permite la automatización de casos de prueba y los scripts son programados en Visual Basic Script [WEB36].
- **SoapUI** (libre/pago por licencia): permite probar, simular y generar código de servicios web partiendo del contrato de los mismos en formato WSDL y con vinculo SOAP sobre http [WEB40].
- **Cucumber** (libre): permite la automatización de la prueba de aceptación para aplicaciones web. Esta basada en BDD (Behaviour Driven development) y el lenguaje de generación de scripts es Ruby [WEB41].
- **IBM Rational Automation Framework** (pago por licencia): es el modulo para la generación de scripts y automatización de pruebas de la empresa IBM que se puede integrar con los módulos de gestión. Permite la generación de scripts con varios lenguajes de programación [WEB36].

3.4. Herramientas de pruebas de carga y rendimiento

El objetivo de estas herramientas es simular situaciones límite en los sistemas y estudiar la respuesta de los mismos. Muchos sistemas que trabajan en tiempo real requieren este tipo de pruebas, donde se probará hasta dónde es capaz de llegar el sistema antes de sobrecargarse. Estos sistemas en tiempo real como pueden ser, sistemas cliente/servidor o aplicaciones web, tienen que cumplir un tiempo de respuesta.

3.4.1. Ejemplos de herramientas

- **Jmeter** (libre): es una herramienta de pruebas de carga para analizar y medir el desempeño de una variedad de servicios, con énfasis en aplicaciones web. Es una de las herramientas más utilizadas para las pruebas de rendimiento [WEB42].
- **LoadUI** (libre): es una herramienta de pruebas de carga y rendimiento que se integra con SoapUI. Es la mejor alternativa a Jmeter [WEB43].
- **FunkLoad** (libre): es una herramienta que permite realizar pruebas de carga de aplicaciones web, monitorizar rendimiento de servidores o realizar pruebas de estrés para comprobar la capacidad de recuperación de las aplicaciones [WEB44].
- **HP Load runner** (pago por licencia): es el módulo de pruebas de carga que se integra dentro del paquete de aplicaciones de HP [WEB35].

3.5. Drivers y Stubs

Los drivers y stubs son herramientas que reemplazan los módulos del programa que estamos desarrollando. Podríamos decir que son pequeños programas creados para realizar pruebas en la integración de módulos del programa. También se pueden utilizar en las pruebas unitarias.

Los drivers simulan módulos de programa de nivel alto, es decir, realizan la función de ofrecer datos a los módulos que tiene por debajo.

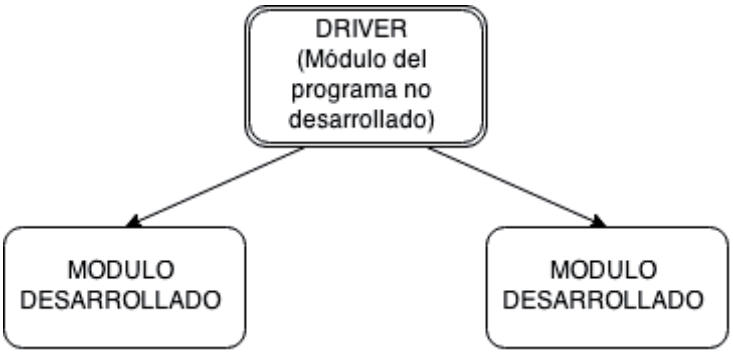

Ilustración 12. Driver

Los stubs son lo contrario de los drivers, simulan módulos de programa de nivel bajo, son programas que realizan la simulación de otro modulo recibiendo los datos de los módulos que tiene por encima.

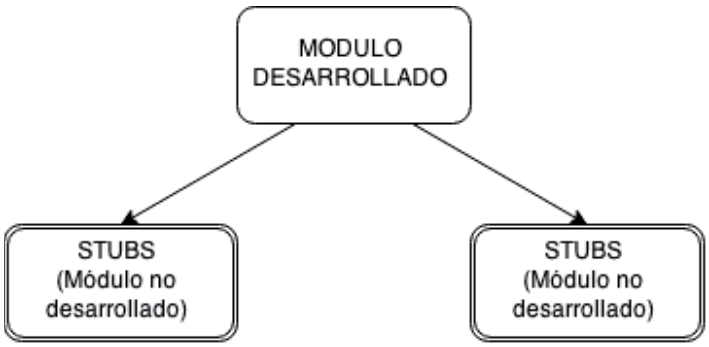

Ilustración 13. Stubs

3.6. Herramientas de seguridad y monitorización

Las herramientas de seguridad permiten detectar las vulnerabilidades de seguridad y la posible explotación de éstas por parte de personas no autorizadas.

Los monitores son herramientas que permiten ver detalles en el funcionamiento de un software en ejecución que normalmente no se podrían ver. Hay varios tipos de monitores, entre los que destacan los analizadores de protocolos y los depuradores que se integran en la mayoría de los compiladores de código fuente que permiten visualizar el contenido de las variables y el flujo del código.

3.6.1. Ejemplos herramientas de seguridad y monitores

- **Wireshark** (libre): permite al desarrollador/probador realizar análisis y solucionar problemas en las redes de comunicaciones. Es un analizador de protocolos de redes. Se utiliza en auditorías de seguridad y pruebas de intrusión [WEB45].
- **NMAP** (libre): esta herramienta permite la exploración de las redes. Utiliza paquetes IP para determinar qué equipos están disponibles en la red, que servicios ofrecen, que sistemas operativos están usando, que tipos de filtros/cortafuegos están en uso, etc. [WEB46].

PARTE PRÁCTICA

4. PRUEBAS FUNCIONALES DE APLICACIONES QUE SE COMUNICAN CON EL PROTOCOLO MDB/ICP

En este apartado veremos cómo aplicar algunas de las técnicas aprendidas en la parte teórica. Para ello se ha seleccionado como ejemplo el protocolo MDB/ICP, que veremos más adelante, para realizar las pruebas funcionales. Más concretamente realizaremos las pruebas funcionales del funcionamiento de este protocolo con un terminal de pago con tarjeta (TPV – Terminal de punto de venta) y una máquina expendedora.

4.1. ¿Qué es el protocolo MDB y cómo funciona?

El protocolo estándar MDB/ICP, es un protocolo abierto de la NAMA (National Automatic Merchandising Association) [WEB47]. Es utilizado por el controlador electrónico de las máquinas expendedoras, VMC (Vending Machine Controller), para comunicarse con los diferentes periféricos instalados en la máquina expendedora.

Este protocolo fue creado en Estados Unidos por la empresa Coinco y era muy utilizado por las máquinas de la empresa de bebidas Coca-Cola. Esta empresa de bebidas impuso a Coinco en 1992 a que liberase este protocolo para aumentar la competencia. La NAMA lanzó la primera versión de este protocolo en 1995. Hoy en día es el protocolo más extendido en todo tipo de máquinas expendedoras.

El documento de la NAMA define una interfaz serial, 'MDB(Multi-Drop Bus)'. Esta interfaz se basa en la comunicación Maestro – Esclavo, donde el controlador VMC es el maestro y los periféricos son los esclavos.

El maestro tendrá la capacidad de comunicarse con hasta 32 periféricos. A cada periférico se le asignará una única dirección y un conjunto de comandos únicos.

El funcionamiento básico consiste en: el VMC enviará a los periféricos los comandos a través del bus, con la intención de detectar si los dispositivos están activos/disponibles. Estos periféricos deben responder en un tiempo definido, en

caso de que no respondan, el controlador asume que el periférico no está conectado al bus. Los periféricos responderán con un 'Acknowledge', 'Negative Acknowledge' o el comando específico dependiendo de su estado actual.

Cada periférico responde únicamente cuando es preguntado por el VMC, evitando así la interferencia de la información dentro del bus.

Todos los periféricos deben reconocer los comandos enviados por el Maestro y poder ser habilitados y deshabilitados cuando éste, estime oportuno.

La última versión del estándar MDB contiene nuevas especificaciones como distintos niveles y opciones de funcionamiento. Estos cambios tienen la finalidad de aumentar la compatibilidad para futuros requerimientos e implementaciones de nuevas funciones. Existen tres niveles diferentes. Si el sistema se ha desarrollado para el nivel 3 podrá actuar con los niveles inferiores pero si tiene desarrollado el nivel 1 no podrá actuar con los niveles superiores.

Este protocolo tiene las siguientes características:

- Velocidad de transmisión 9600 baudios.
- Formato de comunicación:
    - Byte de formato:
        - 1 bit de inicio.
        - 8 bits de datos.
        - 1 bit de modo.
        - 1 bit de parada.

LSB MSB

Inicio	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7	Modo	Parada

Bit de modo:

Puede darse en dos sentidos diferentes:

- **Maestro – Esclavo:**

Permite diferenciar entre un byte de dirección (uno lógico) o un byte de datos (cero lógico). Los bytes de dirección deben ser leídos por todos los periféricos mientras que los bytes de datos solo pueden ser leídos por el periférico que ha sido direccionado.

- **Esclavo – Maestro**

El bit de modo se coloca en uno lógico cuando se envía el ultimo byte de dato.

Formato del bloque de datos:

- **Maestro esclavo:**

El bloque de comunicación enviado en sentido maestro a esclavo se define como, 1 byte de dirección, byte de datos (opcional) y un byte de chequeo (CHK).

El bloque estará limitado a un máximo de 36 bytes. Los 5 bits mas significativos (MSB) del byte de dirección corresponden al direccionamiento mientras que los 3 bits menos significativos (LSB) contienen el comando especifico para el periférico.

El VMC responderá a los datos que se envíen desde el periférico con un Acknowledge (ACK), Negative Acknowledge (NACK) o un byte de retransmitir datos (RET).

Si el VMC no recibe ningún byte proveniente del periférico, se retransmitirán los datos hasta que se haya cumplido el tiempo máximo definido.

- **Esclavo – Maestro:**

En este sentido de la comunicación tendremos un bloque de datos y un byte CHK, un byte de Acknowledge o un byte Negative Acknowledge.

El periférico tendrá que indicar en el bit de modo del último byte a enviar a 1, para indicar el fin de la transacción.

Un bloque de datos consiste de uno o más bytes de datos seguidos de un cheksum (CHK).

En caso de que el Maestro no responda a los datos enviados por el periférico, se deberá reenviar de nuevo los datos o enviarlo en la siguiente comunicación. El CHK es el último byte que se envía en cada paquete de datos y es la suma del byte de direccionamiento y todos los bytes de datos a enviar. El bit de carga es ignorado.

4.2. Sistema para terminales de pago con tarjeta - TPV

El TPV (Terminal de Punto de Venta) es uno de los periféricos que pueden utilizarse para realizar ventas en una máquina expendedora. Este periférico tiene que verse como una máquina de estados, en donde, cada estado puede realizar unas acciones u otras dependiendo del estado en el que se encuentre y del comando que reciba del VMC.

En las especificaciones del protocolo MDB se indica que se pueden soportar hasta dos TPV diferentes en una misma máquina expendedora. A un terminal se le asignará la dirección de recepción de comandos (10H) y al otro la dirección (60H).

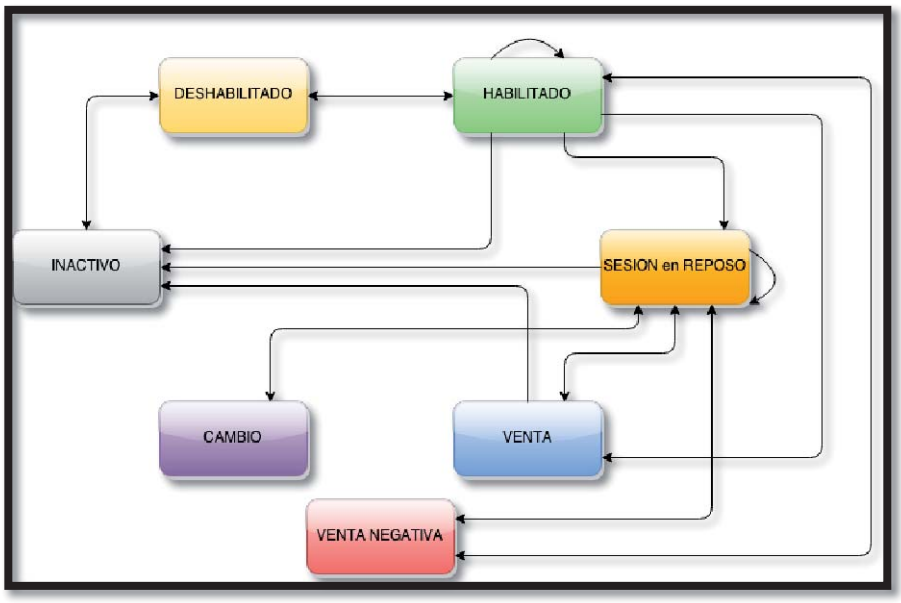

Ilustración 14. Máquina de estados del periférico TPV

Existen diferentes comandos que permiten recorrer los estados anteriores y realizar diferentes funciones. Cada comando puede tener subcomandos que permiten realizar diferentes funciones y se pueden utilizar o no dependiendo del nivel(1, 2 o 3) que se tenga desarrollado:

- **RESET**: Mediante este comando se reseteará el TPV pasando al estado inactivo desde cualquier estado en el que se encuentre. Para realizar el reset, el TPV tiene que recibir desde el VMC un '10H'. El TPV responderá con un '00H' JUST RESET.
- **SETUP**: Mediante este comando el VMC, configurará el TPV. Tiene dos subcomandos para configurar las diferentes características del

TPV como por ejemplo el mínimo y el máximo precio admitido por él para realizar ventas. Para realizar el SETUP, el TPV tiene que recibir desde el VMC un '11H' más los datos de configuración. El TPV responderá con el comando de respuesta asociado al subcomando de configuración.

- **POLL**: Este comando es usado por el VMC para obtener información del TPV. Esta información puede incluir acciones de los usuarios como por ejemplo iniciar o cancelar una venta. Este comando está continuamente enviándose desde el VMC hacia el TPV. El VMC enviará un '12H' y el TPV responderá con un comando de respuesta dependiendo de su estado o de la información que tenga pendiente de enviar al VMC.

- **VEND**: Mediante este comando y los subcomandos asociados a él, se realizarán las diferentes acciones para llevar a cabo las ventas o ventas negativas. Para ello el TPV tiene que recibir desde el VMC un '13H' más los datos del tipo de venta, es decir, el subcomando asociado a la acción que se quiera realizar. El TPV responderá con el comando de respuesta asociado al tipo de acción que se está realizando.

- **READER**: Mediante este comando el VMC puede controlar el TPV, como por ejemplo deshabilitándole o habilitándole. Para ello el TPV tiene que recibir desde el VMC un '14H' más los datos del tipo de acción que se quiera realizar. El TPV responderá con el comando de respuesta asociado al tipo de acción que se está realizando.

- **REVALUE**: Mediante este comando se iniciará un tipo de acción especial que permite al usuario de la máquina expendedora realizar una carga de dinero en la tarjeta electrónica. Para ello el TPV tiene que recibir desde el VMC un '15H' más los datos del tipo de acción que se quiera realizar. El TPV responderá con el comando de respuesta asociado al tipo de acción que se está realizando.

- **EXPANSION**: Mediante este comando se configurarán opciones especiales que puede tener el TPV. Para ello el TPV tiene que recibir desde el VMC un '17H' más los datos del tipo de configuración que se quiera realizar. El TPV responderá con el comando de respuesta asociado al tipo de acción que se está realizando.

En los comandos anteriores sólo se ha hecho mención al formato de mensaje que se utiliza para uno de los dos TPV que se pueden instalar en una máquina expendedora (10H). Para el otro periférico será igual, pero los comandos empezarán por 6.

Los estados que puede tomar el TPV tienen las siguiente características:

- **Inactivo (Inactive)**

Éste es el estado del TPV en el encendido o después de un reinicio. Mientras esté en el estado inactivo, no se aceptarán peticiones de la máquina expendedora. El dispositivo no podrá salir del estado inactivo hasta que no reciba toda la información de configuración desde el VMC, mediante un comando SETUP.

- **Deshabilitado (Disabled)**

El dispositivo de venta entrará automáticamente en este estado, desde el estado *inactivo*, cuando reciba la configuración especificada, mediante un comando SETUP. También entrará en el estado *desactivado* desde el estado *habilitado* cuando reciba el comando READER DISABLE. Mientras que se encuentre en el estado *deshabilitado* no se aceptarán ventas. El dispositivo permanecerá en este estado hasta que: reciba un comando READER ENABLE (entrará en el estado *habilitado*) o reciba un comando RESET (entrará en el estado *inactivo*).

- **Habilitado (Enabled)**

En este estado se permite realizar transacciones MDB y se permanecerá en él hasta que: se lea un medio de pago válido (entrará en el estado *sesión en espera*),

reciba un READER DISABLE (volverá al estado *deshabilitado*) o reciba un RESET (entrará en el estado *inactivo*).

Cuando el dispositivo está habilitado para operar en el modo *siempre en espera (always idle),* con un comando VEND pasará directamente al estado *venta* y con un comando NEGATIVE VEND entrará directamente en el estado *venta negativa*.

Durante el modo *siempre en espera* el TPV puede realizar sesiones normales comenzando con un comando BEGIN SESSION. Para ello el VMC necesitará aceptar ambas situaciones y al detectar un BEGIN SESSION, deshabilitará temporalmente el modo *siempre en espera.*

- **Sesión en reposo (Session Idle)**

Cuando se procesa un medio de pago válido en el estado *habilitado*, el dispositivo emitirá un comando BEGIN SESSION como respuesta al comando POLL enviado por el VMC y entrará en el estado *sesión en reposo*. Esto indica que el TPV está disponible para realizar ventas con la máquina expendedora.

En este estado pueden ocurrir tres tipos de venta mediante los comandos VEND, NEGATIVE VEND y REVALUE que harán salir del estado *sesión en reposo* al TPV y entrará en el estado del comando que haya recibido.

La única manera de salir del estado *sesión en reposo* al estado *habilitado,* cuando volvemos de una operación de venta es a través del comando SESSION COMPLETE. Al recibir este comando, el TPV responde con un END SESSION y pasará al estado *habilitado*.

También se saldrá de este estado si se recibe un comando RESET. El TPV pasará directamente al estado *inactivo*.

- **Venta (Vend)**

Se entrará a este estado desde el estado *sesión en reposo* tras recibir un comando VEND desde el VMC. El TPV retornará al estado *sesión en reposo* tras haber completado la secuencia de comandos, es decir al enviar un comando VEND

el terminal tiene que responder aprobando o denegando la venta y posteriormente el VMC enviará el comando correspondiente.

- **Cambio (Revalue)**

Este estado consiste en el ingreso de una cantidad de dinero en la tarjeta electrónica del usuario, por medio de la máquina expendedora y el TPV. Se introducirá la cantidad deseada en la máquina expendedora y por medio del comando REVENUE se cargará la cantidad deseada en la tarjeta electrónica que se introduzca.

- **Venta Negativa (Vend Denied)**

Este comando se utiliza en máquinas expendedoras que permiten recoger productos como botellas, por las que el terminal pagará una cantidad de dinero al usuario. Se entrará en este estado tras introducir el producto en la máquina e insertar la tarjeta electrónica en el TPV, lo que producirá un intercambio de mensajes mediante el comando VEND DENIED.

4.3. Casos de Prueba funcionales del protocolo MDB/ICP

Hemos realizado un estudio del funcionamiento del protocolo MDB y del uso de este protocolo con un periférico del tipo TPV. Se dividirá la creación de casos de prueba en distintas fases.

Los casos de prueba se realizarán de acuerdo a la estructura que tienen los casos de prueba en la herramienta TestLink.

- **FASE 1:**

El sistema de terminales de pago electrónico se trata como una máquina de estados, por lo que para realizar los casos de pruebas de esta parte del protocolo, utilizaremos la técnica de caja negra, 'Pruebas de transición de estado'.

El objetivo principal de esta técnica de diseño de pruebas, es partir de un estado inicial y recorrer los diferentes estados mediante los eventos que desencadenan los cambios de estado.

- **FASE 2:**

Una vez recorridos todos los estados, veremos varios casos de prueba donde a través de diferentes situaciones, mediante ventas y operaciones especiales, comprobaremos que el periférico actúa como indican las especificaciones y reconoce los diferentes comandos existentes.

En la fase 2 aplicaremos la técnica de creación de casos de prueba basada en la experiencia, en la que el conocimiento del probador de sistemas similares, ayudará a la creación de estos casos de prueba, escogiendo los casos de prueba que más se adecuan al sistema a probar.

Número de Prueba		MDB-1 (FASE 1)	
Título del caso de prueba		Cambio de estado *inactivo* a *deshabilitado.*	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello se realizarán las acciones oportunas para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *inactivo* y queremos pasar a estado *deshabilitado*.	
Precondiciones		El TPV tiene que estar en estado *inactivo*.	
Pasos		**Resultados**	
1	El VMC envía el comando 'SETUP' subcomando 'Config Data' al TPV.	**1**	El TPV tiene que haber entrado en estado *deshabilitado*.

Número de Prueba		MDB-2 (FASE 1)	
Título del caso de prueba		Cambio de estado *deshabilitado* a *inactivo.*	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *deshabilitado* y queremos pasar a estado *inactivo*.	
Precondiciones		El TPV tiene que estar en estado *deshabilitado.*	
Pasos		**Resultados**	
1	El VMC envía el comando 'RESET' al TPV.	**1**	El TPV tiene que haber entrado en estado *inhabilitado.*

Número de Prueba		MDB-3 (FASE 1)	
Título del caso de prueba		Cambio de estado *deshabilitado* a *habilitado.*	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realizará el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *deshabilitado* y queremos pasar a estado *habilitado*.	
Precondiciones		El TPV tiene que estar en estado *deshabilitado.*	
Pasos		**Resultados**	
1	El VMC envía el comando 'READER' subcomando 'Reader Enable' al TPV.	**1**	El TPV tiene que haber entrado en estado *habilitado.*

Número de Prueba		MDB-4 (FASE 1)	
Título del caso de prueba		Cambio de estado *habilitado* a *inactivo.*	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *habilitado* y queremos pasar a estado *inactivo*.	
Precondiciones		El TPV tiene que estar en estado *inhabilitado.*	
Pasos		**Resultados**	
1	El VMC envía el comando 'RESET' al TPV.	**1**	El TPV tiene que haber entrado en estado *inactivo*.

Número de Prueba		MDB-5 (FASE 1)	
Título del caso de prueba		Cambio de estado *habilitado* a *deshabilitado*	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *habilitado* y queremos pasar a estado *deshabilitado*.	
Precondiciones		El TPV tiene que estar en estado *habilitado.*	
Pasos		**Resultados**	
1	El VMC envía el comando 'READER' subcomando 'Reader Disable' al TPV.	**1**	El TPV tiene que haber entrado en estado *deshabilitado.*

Número de Prueba		MDB-6 (FASE 1)	
Título del caso de prueba		Cambio de estado *habilitado* a *sesión en reposo.*	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *habilitado* y queremos pasar a estado *sesión* en *reposo*.	
Precondiciones		El TPV tiene que estar en estado *habilitado* y pendiente de enviar un comando 'BEGIN SESSION' en respuesta a un comando POLL recibido del VMC.	
Pasos		**Resultados**	
1	El VMC envía el comando 'POLL' al TPV.	**1**	El TPV responde con el comando 'BEGIN SESSION' y pasará al estado *sesión en reposo.*

Número de Prueba		MDB-7 (FASE 1)	
Título del caso de prueba		Cambio de estado *sesión en reposo* a *habilitado*.	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *sesión en reposo* y queremos pasar a estado *habilitado*.	
Precondiciones		El TPV tiene que estar en el estado *sesión en reposo* y pendiente de enviar un comando 'END SESSION' en respuesta a un 'POLL' recibido del VMC.	
Pasos		**Resultados**	
1	El VMC envía el comando 'POLL' al TPV.	**1**	El TPV responde con el comando 'END SESSION' y pasará al estado *habilitado*.

Número de Prueba		MDB-8 (FASE 1)	
Título del caso de prueba		Cambio de estado *sesión en reposo* a *venta*.	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *sesión en reposo* y queremos pasar a estado *venta*.	
Precondiciones		El TPV tiene que estar en estado *sesión en reposo*.	
Pasos		**Resultados**	
1	El VMC envía el comando 'VEND' subcomando 'Vend Request' al TPV.	**1**	El TPV tiene que haber entrado en el estado *venta*.

Número de Prueba		MDB-9 (FASE 1)	
Título del caso de prueba		Cambio de estado *sesión en reposo* a *inactivo*.	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *sesión en reposo* y queremos pasar a estado *inactivo*.	
Precondiciones		El TPV tiene que estar en estado *sesión en reposo*.	
Pasos		**Resultados**	
1	El VMC envía el comando 'RESET' al TPV.	**1**	El TPV tiene que haber entrado en el estado *inactivo*.

Número de Prueba		MDB-10 (FASE 1)	
Título del caso de prueba		Cambio de estado *venta* a *sesión en reposo*.	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *venta* y queremos pasar a estado *sesión en reposo*.	
Precondiciones		El TPV tiene que estar en estado *venta*.	
Pasos		**Resultados**	
1	El VMC envía el comando 'VEND' subcomando 'Vend Cancel' al TPV.	**1**	El TPV tiene que haber entrado en el estado *sesión en reposo*.

Número de Prueba		MDB-11 (FASE 1)	
Título del caso de prueba		Cambio de estado *venta* a *inactivo*.	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *venta* y queremos pasar a estado *inactivo*.	
Precondiciones		El TPV tiene que estar en estado *venta*.	
Pasos		**Resultados**	
1	El VMC envía el comando 'RESET' al TPV.	**1**	El TPV tiene que haber entrado en el estado *inactivo*.

Número de Prueba		MDB-12 (FASE 1)	
Título del caso de prueba		Cambio de estado *sesión en reposo* a *venta negativa*.	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *sesión en reposo* y queremos pasar a estado *venta negativa*.	
Precondiciones		El TPV tiene que estar en estado *sesión en reposo*.	
Pasos		**Resultados**	
1	El VMC envía el comando 'VEND' subcomando 'Negative Vend Request' al TPV.	**1**	El TPV tiene que haber entrado en el estado *venta negativa*.

Número de Prueba		MDB-13 (FASE 1)	
Título del caso de prueba		Cambio de estado *sesión* en *reposo a cambio*.	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *sesión en reposo* y queremos pasar a estado *cambio*.	
Precondiciones		El TPV tiene que estar en estado *sesión en reposo*.	
Pasos		**Resultados**	
1	El VMC envía el comando 'REVALUE' al TPV.	**1**	El TPV tiene que haber pasado al estado *cambio*.

Número de Prueba		MDB-14 (FASE 1)	
Título del caso de prueba		Pasar de estado *habilitado* a *venta negativa.*	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *habilitado* y queremos pasar a estado *venta negativa*.	
Precondiciones		El TPV tiene que estar en estado *habilitado* y tiene que tener configurado el modo '*siempre en reposo'*.	
Pasos		**Resultados**	
1	El VMC envía el comando 'VEND' subcomando 'Negative Vend Request' al TPV.	**1**	El TPV pasará al estado *venta negativa*.

Número de Prueba		MDB-15 (FASE 1)	
Título del caso de prueba		Pasar de estado *habilitado* a *venta*.	
Resumen de la prueba		El objetivo de la prueba es comprobar que se realiza el cambio de estado correctamente y para ello, se realizarán los pasos oportunos para comprobar el correcto funcionamiento del TPV cuando se encuentra en estado *habilitado* y queremos pasar a estado *venta*.	
Precondiciones		El TPV tiene que estar en estado *habilitado* y tiene que tener configurado el modo '*Siempre en reposo'*.	
Pasos		**Resultados**	
1	El VMC envía el comando 'VEND' subcomando 'Vend Request' al TPV.	**1**	El TPV tiene que haber pasado al estado *venta*.

Número de Prueba		MDB-16 (FASE 2)	
Título del caso de prueba		Realizar una venta satisfactoria – 'VEND SUCCESS'.	
Resumen de la prueba		El objetivo de la prueba es comprobar el comportamiento del TPV con la máquina expendedora, cuando se inicia una venta que será aprobada mediante un 'VEND SUCCESS'. La máquina expendedora dispensará el producto seleccionado correctamente.	
Precondiciones		El TPV tiene que estar en estado *habilitado* y responder a un 'POLL' enviado por el VMC con un 'BEGIN SESSION' para pasar al estado *sesión en reposo.*	
Pasos		**Resultados**	
1	El VMC envía el comando 'VEND' subcomando 'Vend Request' al TPV.	**1**	El TPV pasa al estado *venta* y cuando la transacción es aprobada por la pasarela de pago el TPV envía un comando 'VEND APPROVED'.
2	El VMC envía el comando 'VEND' subcomando 'Vend Success' al TPV.	**2**	El TPV responde con un 'ACK' y vuelve al estado *sesión en reposo*.
3	El VMC envía el comando 'VEND' subcomando 'Session Complete' al TPV.	**3**	El TPV responde con un comando 'END SESSION' y vuelve al estado *habilitado.*

Número de Prueba		MDB-17 (FASE 2)	
Título del caso de prueba		Realizar una venta fallida – 'VEND FAILURE'.	
Resumen de la prueba		El objetivo de la prueba es comprobar el comportamiento del TPV con la máquina expendedora, cuando se inicia una venta que será denegada mediante un 'VEND FAILURE' debido a que la máquina expendedora no ha podido entregar el producto seleccionado.	
Precondiciones		El TPV tiene que estar en estado *habilitado* y responder a un 'POLL' enviado por el VMC con un 'BEGIN SESSION' para pasar al estado *sesión en reposo*.	
Pasos		**Resultados**	
1	El VMC envía un comando 'VEND' subcomando 'Vend Request' al TPV.	**1**	El TPV pasa al estado *venta* y cuando la transacción es aprobada por la pasarela de pago se envía un 'VEND APPROVED'.
2	El VMC envía un comando 'VEND' subcomando 'Vend Failure' al TPV.	**2**	El TPV responde con un 'ACK' y vuelve al estado *sesión en reposo*.
3	El VMC envía un comando 'VEND' subcomando 'Session Complete' al TPV.	**3**	El TPV responde con un comando 'END SESSION' y vuelve al estado *habilitado.*

Número de Prueba		MDB-18 (FASE 2)	
Título del caso de prueba		Venta cancelada por la máquina expendedora – 'VEND CANCEL'.	
Resumen de la prueba		El objetivo de la prueba es comprobar el comportamiento del TPV con la máquina expendedora, cuando se inicia una venta que será denegada mediante un 'VEND CANCEL'.	
Precondiciones		El TPV tiene que estar en estado *habilitado* y responder a un 'POLL' enviado por el VMC con un 'BEGIN SESSION' para pasar al estado *sesión en reposo*.	
Pasos		**Resultados**	
1	El VMC envía un comando 'VEND REQUEST' al TPV.	**1**	El TPV tiene que haber pasado al estado *venta*.
2	Antes de que la pasarela de pago apruebe la transacción el VMC envía un comando 'VEND' subcomando Vend Cancel' al TPV.	**2**	El TPV responde con un comando 'VEND DENIED' y vuelve al estado *sesión en reposo*.
3	El VMC envía un comando 'VEND' subcomando 'Session Complete' al TPV.	**3**	El TPV responde con un comando 'END SESSION' y vuelve al estado *habilitado.*

Número de Prueba		MDB-19 (FASE 2)	
Título del caso de prueba		Venta múltiple.	
Resumen de la prueba		El objetivo de la prueba es realizar una venta múltiple que consiste en la venta de varios productos sin pasar por el estado *habilitado*.	
Precondiciones		El TPV tiene que estar en estado *habilitado* y responder a un 'POLL' enviado por el VMC con un 'BEGIN SESSION' para pasar al estado *sesión en reposo*.	
Pasos		**Resultados**	
1	El VMC envía un comando 'VEND' subcomando 'Vend Request' al TPV.	**1**	El TPV pasa al estado *venta* y cuando la transacción es aprobada por la pasarela de pago se envía un 'VEND APPROVED'.
2	El VMC envía un comando 'VEND' subcomnado 'Vend Success' al TPV.	**2**	El TPV responde con un 'ACK' y vuelve al estado *sesión en reposo*.
3	El VMC envía un comando 'VEND' subcomando 'Vend Request' al TPV.	**3**	El TPV pasa al estado *venta* y cuando la transacción es aprobada por la pasarela de pago se envía un 'VEND APPROVED'.
4	El VMC envía un comando 'VEND' subcomnado 'Vend Success' al TPV.	**5**	El TPV responde con un 'ACK' y vuelve al estado *sesión en reposo*.
3	El VMC envía un comando 'VEND' subcomando 'Session Complete' al TPV.	**3**	El TPV responde con un comando 'END SESSION' y vuelve al estado *habilitado.*

Número de Prueba		MDB-20 (FASE 2)	
Título del caso de prueba		Realizar un 'RESET' después de un 'VEND APPROVED'.	
Resumen de la prueba		El objetivo de la prueba es realizar un 'RESET' después de que el TPV haya enviado el comando 'VEND APPROVED' y comprobar que se dispensa el producto correctamente pero el TPV realiza un ciclo de reset.	
Precondiciones		Tenemos que haber iniciado una venta y que el TPV haya contestado con un 'VEND APPROVED'.	
Pasos		**Resultados**	
1	El VMC envía un comando 'RESET' al TPV.	**1**	El TPV tiene que iniciar un ciclo de reset (ver prueba MDB-21) pero la máquina expendedora tiene que haber dispensado el producto satisfactoriamente.

Número de Prueba		MDB - 21(FASE 2)	
Título del caso de prueba		Ciclo de 'RESET' del TPV.	
Resumen de la prueba		Cuando se resetea el TPV, ya sea por que se ha realizado una modificación de la configuración de datos o por otro motivo en que el VMC lo considere necesario, se realizará una secuencia de envío/recepción de varios comandos.	
Precondiciones		El TPV puede estar en cualquier estado.	
Pasos		**Resultados**	
1	El VMC envía el comando 'RESET' al TPV.	**1**	El TPV responde con un 'ACK'.
2	El VMC envía el comando 'POLL' al TPV	**2**	El TPV responde con un 'JUST RESET' y pasa a estado *inactivo.*
3	El VMC envía el comando 'SETUP' subcomando 'Config Data' al TPV.	**3**	El TPV responde con un 'READER CONFIG DATA' y pasa al estado *deshabilitado*
4	El VMC envía el comando 'SETUP' subcomando 'Max/Min Prices' al TPV.	**4**	El TPV responde con un 'ACK'.
5	El VMC envía el comando 'EXPANSION' subcomando 'Request ID'.	**5**	El TPV responde con un 'PERIPHERAL ID'.
6	*Sólo para nivel 3* El VMC envía el comando 'EXPANSION' subcomando 'Optional Featured Enabled'.	**6**	El TPV responde con un 'ACK'.

7	*Sólo para el nivel 3* El VMC envía el comando 'SETUP' subcomando 'Max/Min Prices (32)' al TPV.	**8**	El TPV responde con un 'ACK'.
8	El VMC envía el comando 'READER' subcomando 'Reader Enable' al TPV.	**8**	El TPV pasa al estado *habilitado.*

Número de Prueba		MDB-22 (FASE 2)	
Título del caso de prueba		Realizar una venta - 'VEND NEGATIVE'.	
Resumen de la prueba		El objetivo de la prueba es comprobar el comportamiento del TPV con la máquina expendedora, cuando se inicia una venta negativa. La máquina expendedora recogerá el producto entregado y realizará el ingreso del importe en la tarjeta de pago.	
Precondiciones		El TPV tiene que estar en estado *habilitado* y responder a un 'POLL' enviado por el VMC con un 'BEGIN SESSION' para pasar al estado *sesión en reposo*. Una vez en este estado el usuario introduce en la máquina expendedora el producto y en el TPV la tarjeta.	
Pasos		**Resultados**	
1	El VMC envía un comando 'VEND' subcomando 'Vend Negative' al TPV.	**1**	El TPV pasa al estado *venta negativa* (el TPV preparado para ingresar el crédito del producto.) El TPV envía un comando 'VEND APPROVED'.
2	El VMC envía un comando 'VEND' subcomando 'Vend Success' al TPV.	**2**	El TPV responde con un 'ACK', se ingresa el crédito en la tarjeta y pasa al estado *sesión en reposo*.
3	El VMC envía un comando 'VEND' subcomando 'Session Complete' al TPV.	**3**	El TPV responde con un comando 'END SESSION' y vuelve al estado *habilitado.*

Número de Prueba		MDB-23 (FASE 2)	
Título del caso de prueba		Venta denegada desde el TPV.	
Resumen de la prueba		El objetivo de la prueba es comprobar el funcionamiento TPV con la máquina expendedora cuando se deniega desde el propio TPV la venta.	
Precondiciones		El TPV tiene que estar en estado *habilitado* y responder a un POLL enviado por el VMC con un 'BEGIN SESSION' para pasar al estado *sesión en reposo*. La pasarela de pago tiene que denegar la operación.	
Pasos		**Resultados**	
1	El VMC envía un comando 'VEND' subcomando 'Vend Negative' al TPV.	**1**	El TPV pasa al estado *venta*, se deniega la transacción por la pasarela de pago y envía un comando 'VEND DENIED'.
2	El VMC envia un comando 'VEND' subcomando 'Vend Failure' al TPV.	**2**	El TPV responde con un 'ACK' y vuelve al estado *sesión en reposo*.
3	El VMC envia un comando 'VEND' subcomando 'Session Complete' al TPV.	**3**	El TPV responde con un comando' END SESSION' y vuelve al estado *habilitado.*

Número de Prueba		MDB-24 (FASE 2)	
Título del caso de prueba		Venta desde estado *habilitado* – 'Always Idle State' –*Nivel 3.*	
Resumen de la prueba		El objetivo de la prueba es comprobar el comportamiento del TPV con la máquina expendedora, cuando se inicia una venta desde el estado *habilitado*. Para poder realizar la venta desde este estado, el TPV tiene que haber sido configurado mediante el comando OPTIONAL FEATURED ENABLE.	
Precondiciones		El TPV tiene que estar en estado *habilitado*.	
Pasos		**Resultados**	
1	El VMC envía un comando 'VEND' subcomando 'Vend Request' al TPV.	**1**	El TPV pasa al estado *venta* y cuando la transacción es aprobada por la pasarela de pago se envía un comando 'VEND APPROVED'.
2	El VMC envía un comando 'VEND' subcomando 'Vend Success' al TPV.	**2**	El TPV responde con un 'ACK' y vuelve al estado *sesión en reposo*.
3	El VMC envía un comando 'VEND' subcomando 'Session Complete' al TPV.	**3**	El TPV responde con un comando 'END SESSION' y vuelve al estado *habilitado.*

Número de Prueba		MDB-25 (FASE 2)	
Título del caso de prueba		Realizar un REVALUE	
Resumen de la prueba		El objetivo de la prueba es comprobar el comportamiento del TPV con la máquina expendedora, cuando se realiza un REVALUE.	
Precondiciones		El TPV tiene que estar en estado *habilitado* y responder a un 'POLL' enviado por el VMC con un 'BEGIN SESSION' para pasar al estado *sesión en reposo*. Se tiene que haber introducido crédito en la máquina expendedora y se introduce tarjeta de pago en el TPV.	
Pasos		**Resultados**	
1	El VMC envía el comando 'REVALUE' subcomando 'Revalue Request' al TPV.	**1**	Se realiza el ingreso del crédito en la tarjeta y el TPV responde con un comando 'REVALUE APPROVED' y vuelve al estado *sesión en reposo*.
2	El VMC envía un comando 'VEND' subcomando 'Session Complete' al TPV.	**2**	El TPV responde con un comando 'END SESSION' y vuelve al estado *habilitado.*

5. TESTLINK Y LA GESTION DE LAS PRUEBAS

5.1. ¿Qué es TesLink?

TestLink es una herramienta libre que permite gestionar y crear casos de prueba, organizados en planes de prueba [WEB36]. Esta herramienta se utiliza a través de una interfaz WEB.

5.2. Instalación de TesLink

Como TestLink funciona mediante interfaz web, para instalar la herramienta tenemos primero que instalar un servidor WEB y un servicio SQL de base de datos. Para ello vamos a instalar **XAMPP** [WEB56], que es una distribución de Apache gratuita que contiene MySQL, PHP y Perl.

Para iniciar la instalación tenemos que descargarnos la aplicación de:

- https://www.apachefriends.org/es/download.html

Una vez descargada la aplicación, la instalamos en el equipo que vaya a ser utilizado para el proyecto como servidor WEB.

La instalación se realiza ejecutando el instalable que nos hemos descargado.

Cuando la aplicación se encuentre instalada, la ejecutamos para activar los servicios de Apache y MySQL. En la ilustración 15 podemos ver el panel de control una vez se instala la aplicación y se pone en funcionamiento.

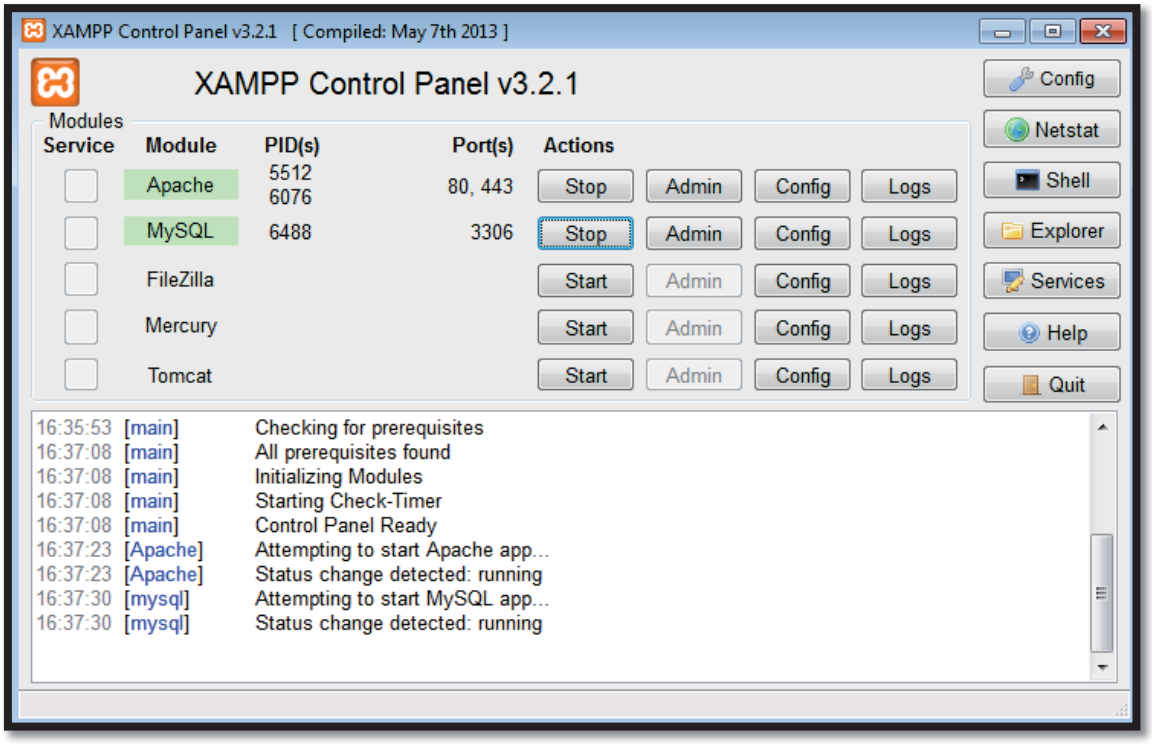

Ilustración 15. Panel de control XAMPP

Entramos en la página de inicio del servidor (localhost/xampp/index.php) para asegurarnos de que los servicios están correctamente ejecutados en el equipo.

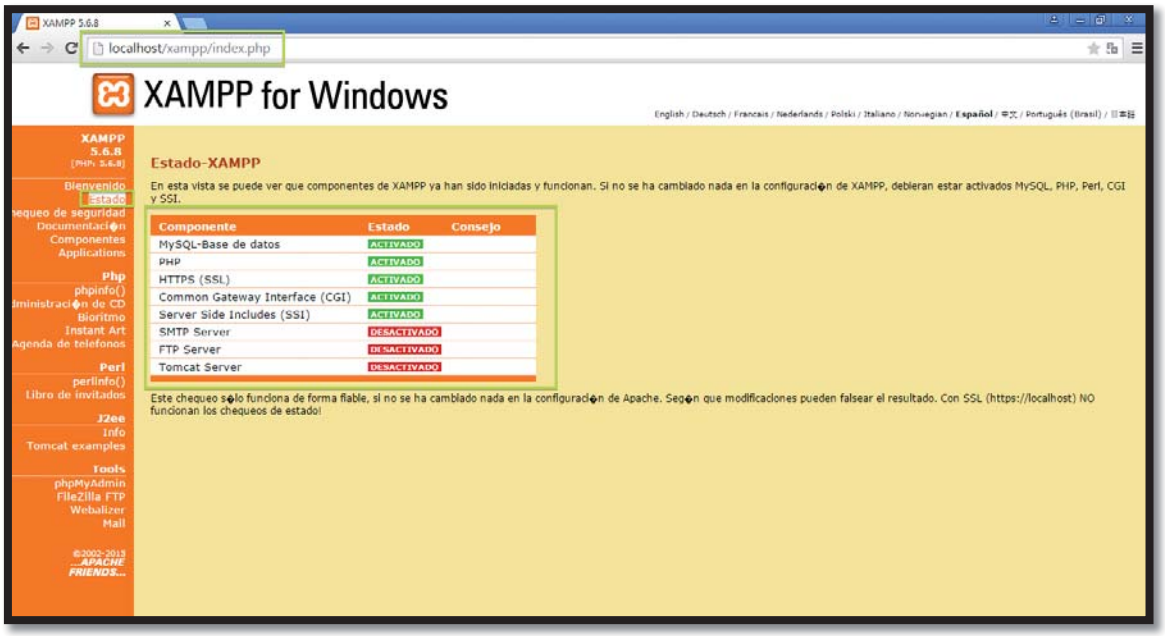

Ilustración 16. Página de inicio del servidor

Una vez tengamos instalado el servidor WEB, iniciamos la instalación de TestLink. Descargamos la aplicación desde la página:

- http://sourceforge.net/projects/testlink/files/TestLink%201.9/TestLink%201.9.5/

Una vez descargado, descomprimimos el paquete en la ruta XAMPP\htdocs en la carpeta TestLink. Abrimos un navegador y ponemos localhost/testlink/install/index.html y se abrirá la página que se muestra a continuación (ilustración 17):

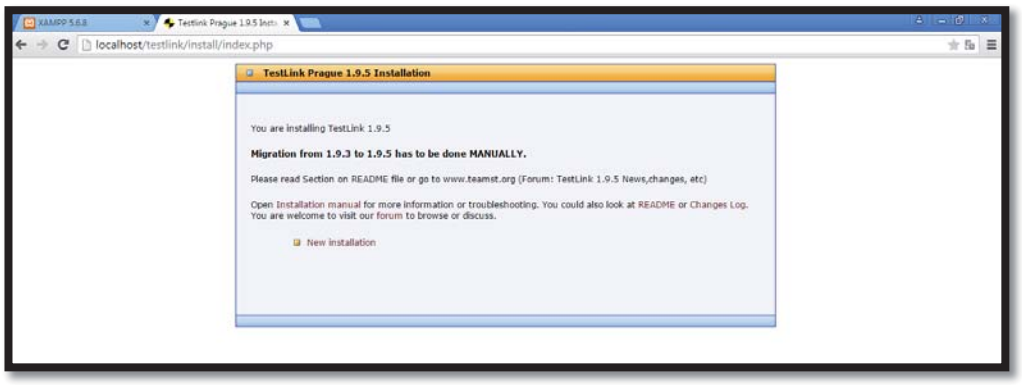

Ilustración 17. Configuración TestLink 1

Pulsamos en 'New installation' y comenzamos a instalar TestLink. Se abrirá la siguiente página (ilustración 18).

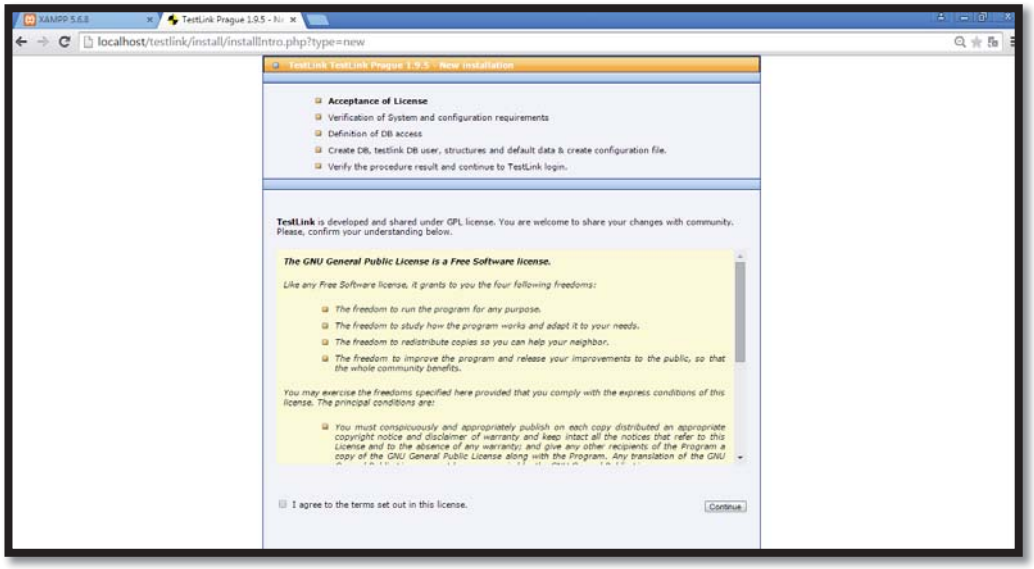

Ilustración 18. Configuración TestLink 2

Aceptamos los términos después de leerlos si estamos de acuerdo y pulsamos en continuar. Al pulsar en continuar se realiza una verificación del sistema donde se está instalando la herramienta (ilustración 19).

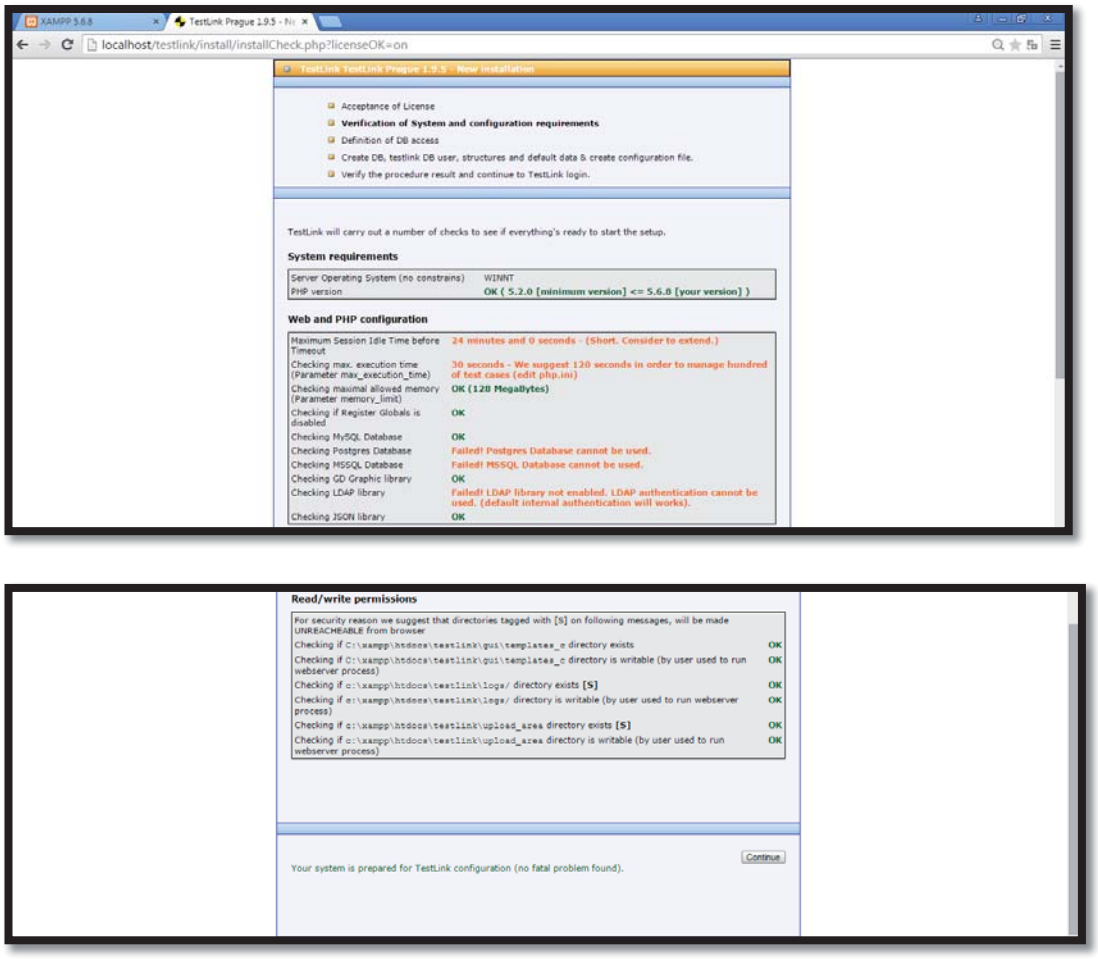

Ilustración 19. Configuración TestLink 3

Una vez realizado el paso anterior, entraremos en la configuración de la base de datos e introduciremos un usuario para la administración de la base de datos (ilustración 20).

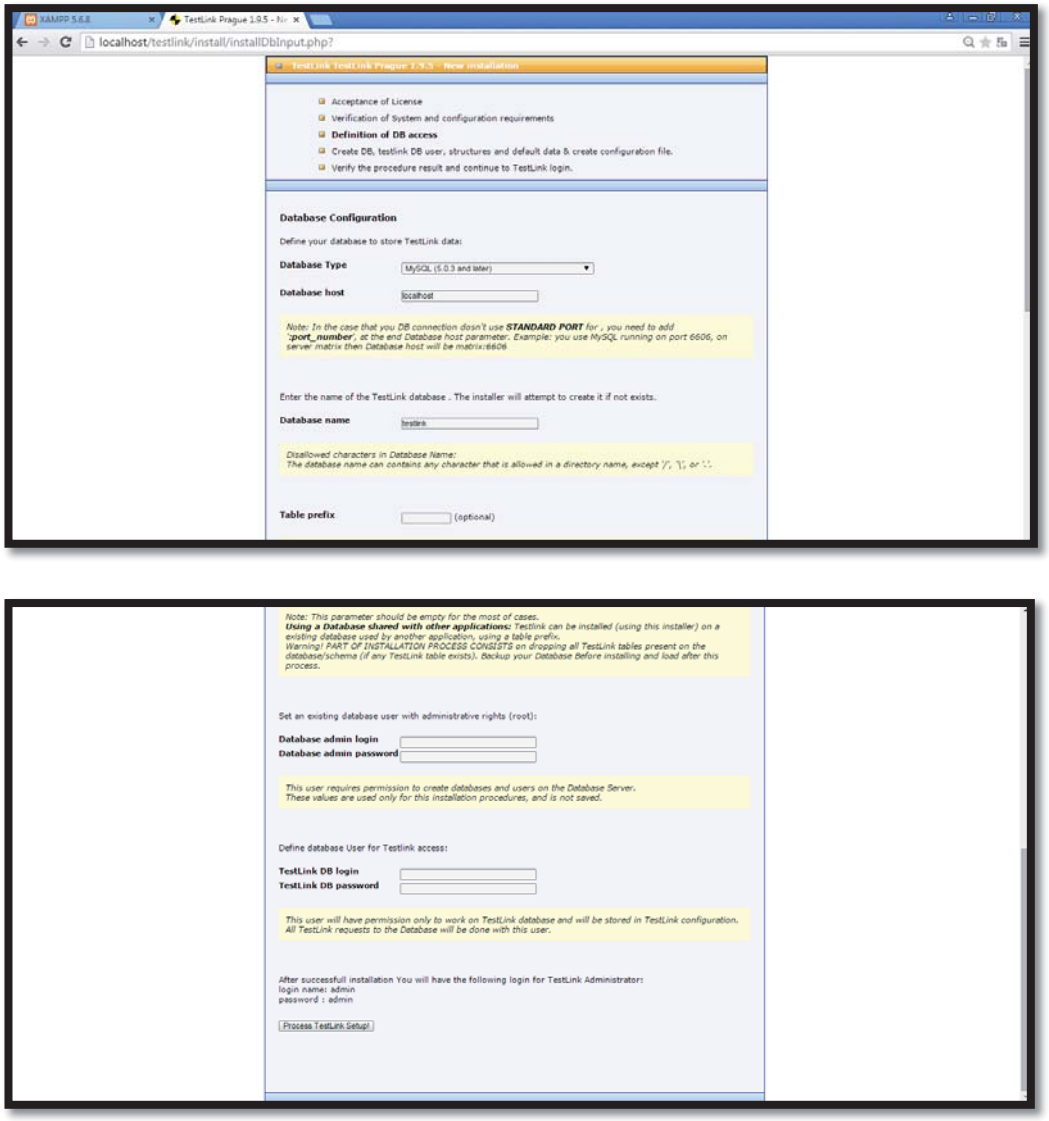

Ilustración 20. Configuración TestLink 4

Pulsamos en Process TestLink Setup cuando hayamos introducido los datos y nos mostrará la siguiente página. A partir de este paso ya estará configurado TestLink para usarse en nuestro servidor web (ilustración 21).

Ilustración 21. Configuración TestLink 5

5.3. TestLink para la creación de casos de prueba

Una vez completada la instalación, entramos en http://Localhost/testlink/login.php e introducimos usuario - **admin** y contraseña - **admin**. En nuestro caso entraremos con los datos del administrador (ilustración 22).

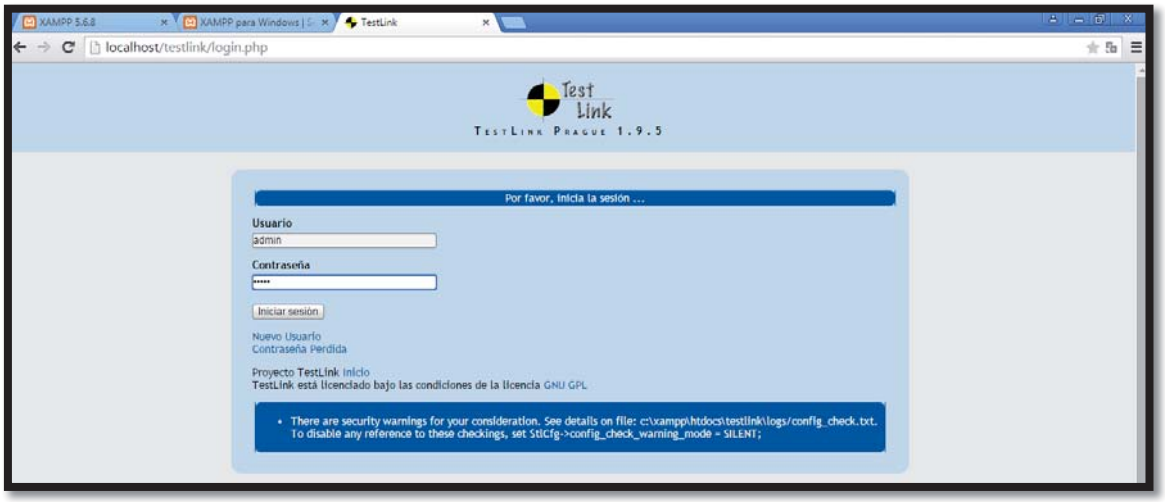

Ilustración 22. Cómo usar TestLink 1

La primera página que se abre es la de creación de un nuevo proyecto (ilustración 23).

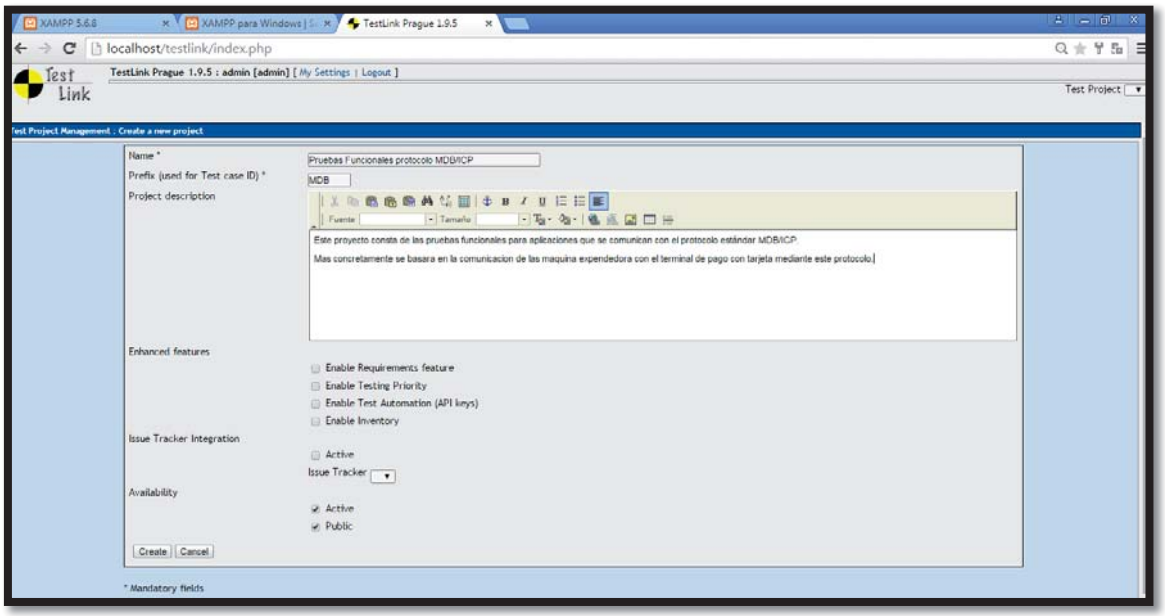

Ilustración 23. Cómo usar TestLink 2

Rellenamos los siguientes campos y pulsamos en 'create':

- **'Name':** nombre del proyecto. En nuestro caso será 'Pruebas funcionales del protocolo MDB/ICP'.
- '**Prefix**': prefijo que servirá para identificar los casos de prueba del proyecto.
- '**Project Description**': una descripción del proyecto que nos servirá para identificar las pruebas que vamos a realizar.

Al pulsar en aceptar nos aparecerá una pantalla con el proyecto creado (ilustración 24).

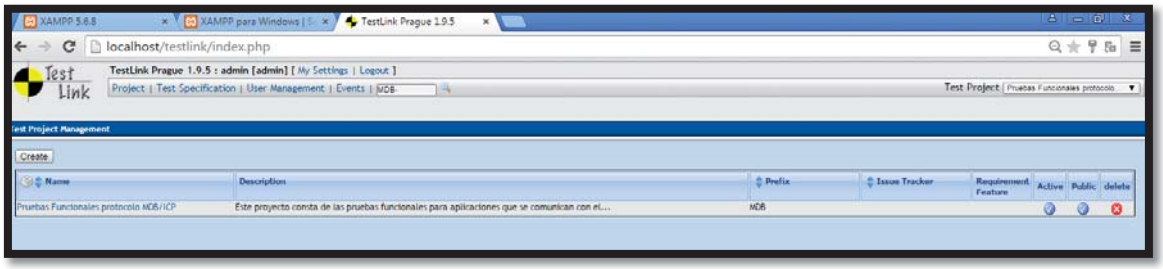

Ilustración 24. Cómo usar TestLink 3

Una vez creado el proyecto ya podemos empezar a crear los casos de prueba en el proyecto que hemos creado. Para ello pulsaremos sobre 'Test Specification' en la parte superior izquierda de la pantalla y se nos abrirá la siguiente página (ilustración 25).

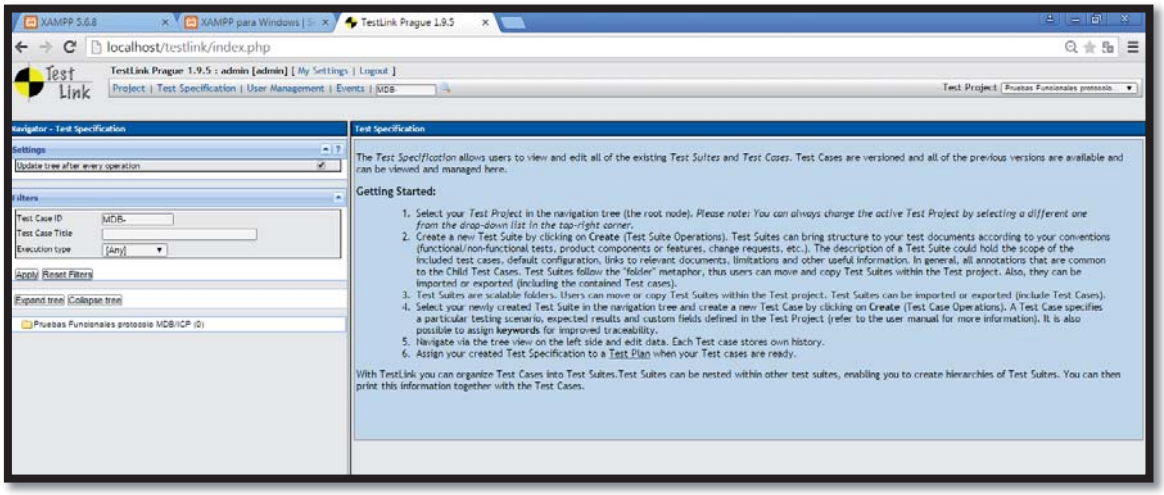

Ilustración 25. Cómo usar TestLink 4

Pulsamos sobre la carpeta con el nombre de nuestro proyecto y nos aparecerá la posibilidad de crear un 'test suite' que será donde se almacenarán los casos de prueba.

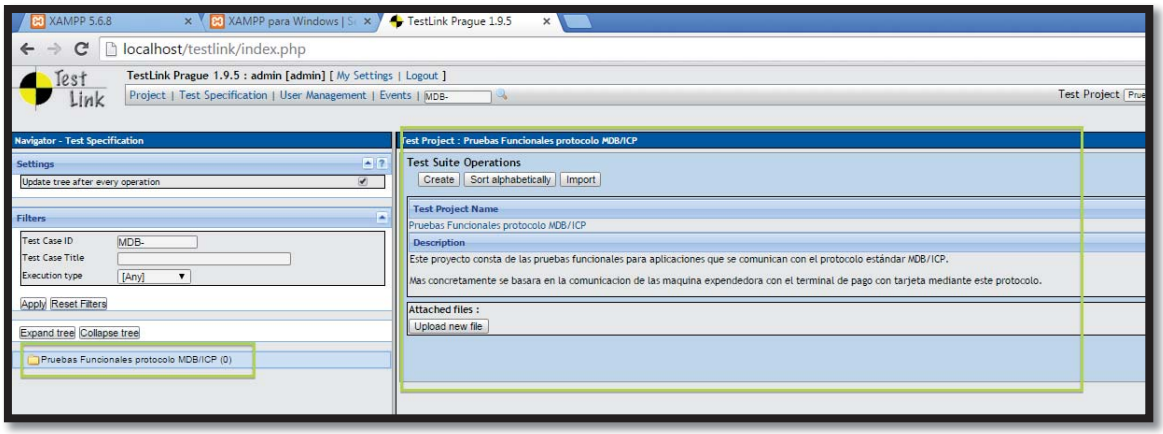

Ilustración 26. Cómo usar TestLink 5

Rellenamos el nombre del test suite y en los detalles en que consistirán las pruebas:

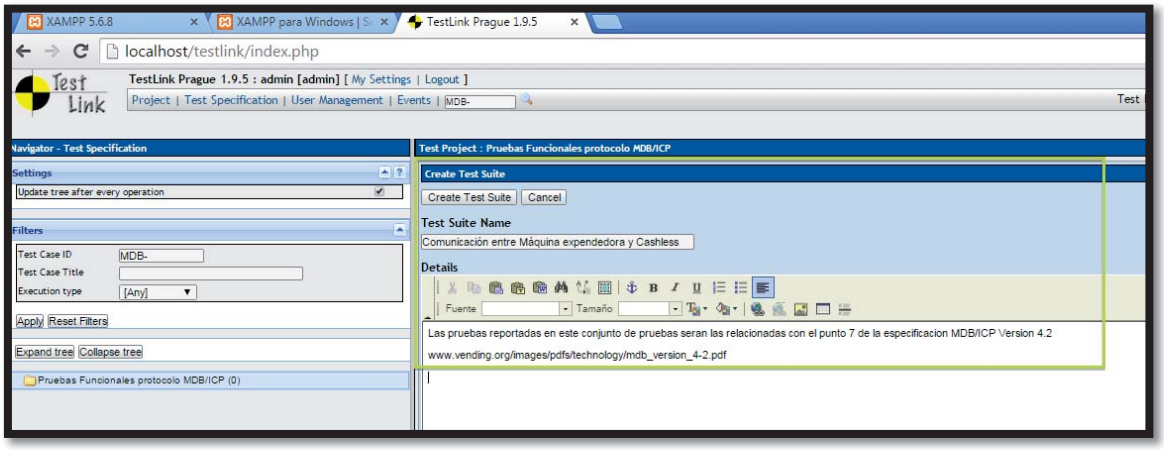

Ilustración 27. Cómo usar TestLink 6

Una vez creada la test suite podremos crear los casos de prueba.

Los campos que tendríamos que rellenar para crear los casos de prueba serían:

- **Test case title** (Título del caso de prueba): en este campo indicaremos un título que nos servirá para identificar el caso de prueba.
- **Summary** (Resumen): se indicará un resumen para indicar en qué consiste la prueba.

- **Preconditions** (Precondiciones): en algunos casos de prueba, se requiere que antes de realizarla se cumplan unas condiciones.
- **Create steps**:
    - **Step actions** (Pasos para realizar): acciones que hay que realizar para ejecutar el caso de prueba.
    - **Expected Results** (Resultados esperados): resultado esperado al ejecutar el paso indicado anteriormente.

6. CONCLUSIONES Y TRABAJOS FUTUROS

En este proyecto se han estudiado los tipos, métodos y estrategias de pruebas de software así como las diferentes herramientas que permiten llevar estos procesos de creación/gestión de pruebas más eficientemente.

Mediante varios ejemplos reales hemos comprendido la importancia que tienen las pruebas sobre los proyectos de software.

Hemos visto teórica y prácticamente como desarrollar casos de prueba, lo cual nos ha permitido comprender diferentes conceptos existentes en el mundo de las pruebas, así como hemos puesto en práctica alguna de las estrategias de prueba con un ejemplo.

También se ha aprendido a instalar y utilizar una de las herramientas libres más importante en el mundo de las pruebas como es TestLink.

La gran mayoría de la información del proyecto ha sido sacada de los libros escritos por profesionales del sector de las pruebas y de diferentes WEBs de referencia.

Como se indicó en la introducción, el proceso de prueba de software tiene tres procesos principales, el desarrollo de los casos de prueba, la ejecución de estos casos de prueba y el análisis de los resultados de la ejecución por lo que los puntos a explorar en uno o varios proyectos a futuro podría ser:

- Realizar la ejecución de los casos de prueba planteados, realizando primero una aplicación que se comunique con un simulador de VMC. En este punto se podría instalar una aplicación que gestionara incidencias, para tratar las posibles incidencias que nos surjan en el momento de la ejecución.
- Realizar un análisis de resultados de la ejecución en donde estudiar la manera de recoger e interpretar los resultados obtenidos. También se podría realizar un estudio de una herramienta que analice los resultados obtenidos.
- Realizar un estudio sobre los conceptos y la manera de llevar a cabo la automatización de los casos de prueba dentro de un proyecto,

incluyendo, el estudio de las diferentes herramientas de automatización existentes y el uso de una de estas herramientas.

7. GLOSARIO DE TÉRMINOS

- **AECL:** Atomic Energy of Canada Limited. Empresa estatal canadiense encargada de administrar el programa canadiense de energía nuclear.
- **AENOR:** Asociación Española de Normalización y Certificación. Entidad privada que contribuye al desarrollo de la normalización y la certificación en los sectores industriales y servicios.
- **CD-ROM:** Compact Disc Read-Only Memory. Soporte físico de almacenamiento de datos.
- **CHK:** Checksum. Suma de verificación que tiene como propósito detectar cambios en una secuencia de datos.
- **ESA:** European Space Agency. La agencia espacial europea es una organización internacional dedicada a la exploración espacial.
- **IEC:** International Electrotechnical Commision. La comisión electrotécnica internacional, es una organización de normalización en los campos relacionados con la electrónica.
- **IEEE:** Institute of Electrical and Electronics Engineers. El instituto de ingeniería eléctrica y electrónica, es una asociación mundial dedicada a la estandarización y desarrollo en áreas técnicas.
- **ISO:** International Organization for Standarization. La organización internacional de normalización, es el encargado de promover el desarrollo de normas internacionales.
- **ISST:** International Society for Software Testing. Sociedad internacional para las pruebas de software que promueve el papel que tienen los probadores en las pruebas.
- **ISTQB:** International Software Testing Qualifications Board. Organización formada por instituciones, empresas, organizaciones y personas que centra su interés en la industria del software y el campo de las pruebas.

- **LSB:** Least Significant Bit. Bit menos signigicativo.
- **MDB/ICP:** Multi-Drop Bus / Internal Communication Protocol. Protocolo estándar de máquinas expendedoras de la NAMA.
- **MSB:** Most Significant Bit. El bit que de acuerdo a su posición, tiene el mayor valor.
- **NAMA:** National Automatic Merchandising Association. Asociación americana para el desarrollo y la promoción de las máquinas expendedoras.
- **PC:** Personal Computer. Ordenador personal
- **PHP:** Hypertext Preprocessor. Lenguaje de código abierto usado para el desarrollo WEB.
- **SQL:** Structured Query Language. Lenguaje de acceso a bases de datos relacionales.
- **TMMI:** Testing Madurity Model Integration. Estandar internacional promovido por la fundación TMMi para servir como guía y referencia para la mejora de los procesos de prueba.
- **TPV:** Terminal de Punto de Venta. Es un dispositivo electrónico que permite realizar pago con tarjetas electrónicas.
- **VMC:** Vending Machine Controller. Es el controlador electrónico de la máquina expendedora.
- **WEB:** término que hace referencia a la World Wide Web(WWW) o Red informática mundial.

8. BIBLIOGRAFÍA

Referencias de libros:

- [PAT05] – Ron Patton. "Software Testing". Segunda Edición. Sams Publishing 2005.

- [PRE05] – Roger S. Pressman. "Ingeniería del Software: Un enfoque practico. Sexta edición". Mc Graw Hill 2005.

- [MYE11] – Glenford J. Myers. "The art of software testing". Segunda edición. John Wiley & Sons 2004.

- [FAR08] – Peter Farrel Vinay. "Manage software testing". Primera edición. Auerbach Publications 2008.

- [KSH08] – Kshirasagar Naik, Priyadarshi Tripathy. "Software testing and quality assurance: Theory and practice". Primera edición. John Wiley & Sons 2008.

- [EVE07] – Gerald D. Everett, Raymond MacLeod Jr. "Software testing: Testing across the entire software development life cycle". Primera edición. John Wiley & Sons 2007.

- [SCH14] – Hans Schaefer, Tilo Linz, Andreas Spillner. "Software testing foundations". Cuarta edición. Rocky Nook 2014.

- [RUB08] – Jeffrey Rubin, Dana Chisnell, Jared Spoll. "Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests". Segunda edición. John Wiley & Sons 2008.

- [DUS99] – Elfriede Dustin, Jeff Rashka, John Paul – "Automated Software Testing: Introduction, Management, and Performance". Primera edición. Addison Wesley Professional, 1999.

- [BEI90] – Boris Beizer. "Software Testing Techniques". Segunda edición. Dreamtech, 1990.

- [BLA07] – Rex Black. "Pragmatic Software Testing: Becoming an Effective and Efficient Test Professional". Primera edición. John Wiley & Sons 2007.

- [BUR03] –Ilene Burnstein. "Practical Software Testing". Primera edición. Springer, 2003.

Referencias en internet:

- [ISTQB] Manual - ISTQB Foundation – SSTQB (17 de mayo de 2015): http://www.sstqb.es/ficheros/sstqb_file95-a69acf.pdf

- [WEB01] - Enlace el país (17 de mayo de 2015): http://blogs.elpais.com/turing/2012/07/es-posible-construir-software-que-no-falle.html

- [WEB02] - Definición de testing CEM Kaner – KANER (17 de Mayo de 2015): http://www.kaner.com/pdfs/KanerSocialScienceSTEP.pdf

- [WEB03] – Definición de pruebas Disjtra – WIKIPEDIA (17 de Mayo de 2015): http://es.wikipedia.org/wiki/Edsger_Dijkstra

- [WEB04] - Informe fallo Ariane 5 – ESA (17 de Mayo de 2015): http://www.esa.int/esapub/bulletin/bullet89/dalma89.htm

- [WEB05] - Fallos lanzadera Ariane-5 – COMPUTERWORLD (17 de mayo de 2015): http://www.computerworld.com/article/2515483/enterprise-applications/epic-failures--11-infamous-software-bugs.html?page=3

- [WEB06] – Fallo sonda Mariner-1 – COMPUTERWORL (17 de mayo de 2015): http://www.computerworld.com/article/2515483/enterprise-applications/epic-failures--11-infamous-software-bugs.html?page=2

- [WEB07] - Therac 25 – WIKIPEDIA (17 de mayo de 2015): http://es.wikipedia.org/wiki/Therac-25

- [WEB08] – Definición de calidad ISO 9000 / ISO 25000 – WIKIPEDIA (17 de mayo de 2015): http://es.wikipedia.org/wiki/Calidad

- [WEB09] – Definición de calidad – TEGSOLTIONS (17 de mayo de 2015): http://www.tegsolutions.com/Que%20es%20la%20Calidad.htm

- [WEB10] – Definición de calidad – Genichi Taguchi – WIKIPEDIA (17 de mayo de 2015): http://es.wikipedia.org/wiki/Genichi_Taguchi

- [WEB11] - Norma ISO 25010 – ISO (17 de mayo de 2015): http://iso25000.com/index.php/normas-iso-25000/iso-25010

- [WEB12] - Principio de Pareto(Regla 80/20) – WIKIPEDIA (17 de mayo de 2015): http://es.wikipedia.org/wiki/Principio_de_Pareto

- [WEB13] - Complejidad ciclomática - Artículo original Tom MacCabe – LITERATEPROGRAMMING(17 de mayo de 2015): http://www.literateprogramming.com/mccabe.pdf

- [WEB14] - Pruebas de tabla de decisión – BLOG ABSTRACTA (17 de mayo de 2015): http://blog.abstracta.com.uy/2014/02/diseno-de-pruebas-con-tablas-de-decision.html

- [WEB15] - PDF Casos de uso 2.0 – IVARJACOBSON (17 de mayo de 2015): www.ivarjacobson.com/download.ashx?id=2018

- [WEB16] - Pruebas exploratorias: - KANER (17 de mayo de 2015): http://kaner.com/?p=46

- [WEB17] - Pruebas exploratorias – SATISFICE (17 de mayo de 2015): http://www.satisfice.com/tools/procedure.pdf

- [WEB18] – Bohem – Verificación y Validación BUENASTAREAS (17 de mayo de 2015): http://www.buenastareas.com/ensayos/Verificaci%C3%B3n-y-Validaci%C3%B3n-Del-Software/67734013.html

- [WEB19] - IEEE 29119 – SOFTWARETESTINGSTANDARD (17 de mayo de 2015): http://www.softwaretestingstandard.org/

- [WEB20] - IEEE – IEEESPAIN (17 de mayo de 2015): http://www.ieeespain.org/

- [WEB21] - ISO – (17 de mayo de 2015): http://www.iso.org/iso/home.html

- [WEB22] - AENOR – (17 de mayo de 2015): http://www.aenor.es/aenor/aenor/perfil/perfil.asp#.VWH742S8PGc

- [WEB23] – ISST – (17 de mayo de 2015): http://www.commonsensetesting.org/mission/

- [WEB24] TMMI – TMMI (17 de mayo de 2015): http://www.tmmi.org/

- [WEB25] – Herramienta PMD (17 de mayo de 2015): http://pmd.sourceforge.net/

- [WEB26] – Herramienta CHECK STYLE (17 de mayo de 2015): http://checkstyle.sourceforge.net/

- [WEB27] – Herramienta SONAR (17 de mayo de 2015): http://www.sonarqube.org/

- [WEB28] – Herramienta – Simian (17 de mayo de 2015): http://www.redhillconsulting.com.au/products/simian/

- [WEB29] – Herramienta – FindBugs (17 de mayo de 2015): http://findbugs.sourceforge.net/

- [WEB30] – Herramienta – MacCabe (17 de mayo de 2015): http://www.mccabe.com/

- [WEB31] – Herramienta – TESTLINK (17 de mayo de 2015): http://testlink.org/

- [WEB32] – Herramienta – Redmine (17 de mayo de 2015): http://www.redmine.org/

- [WEB33] – Herramienta – Trello (17 de mayo de 2015): https://trello.com/

- [WEB34] – Herramienta – Mantis (17 de mayo de 2015): http://www.mantisbt.org/

- [WEB35] – Herramienta – HP-Quality center (17 de mayo de 2015): http://www8.hp.com/es/es/software-solutions/quality-center-quality-management/

- [WEB36] – Herramienta – IBM Rational (17 de mayo de 2015): http://www-03.ibm.com/software/products/es/ratiqualmana

- [WEB37] – Herramienta – Testopia (17 de mayo de 2015): https://developer.mozilla.org/es/docs/Mozilla/Bugzilla/Testopia

- [WEB38] – Herramienta – Bugzilla (17 de mayo de 2015): https://www.bugzilla.org/about/

- [WEB39] – Herramienta – Selenium (17 de mayo de 2015): http://www.seleniumhq.org/

- [WEB40] – Herramienta – SoapUI (17 de mayo de 2015): http://www.soapui.org/about-soapui/what-is-soapui-.html

- [WEB41] – Herramienta – Cucumber (17 de mayo de 2015): https://cucumber.io/

- [WEB42] – Herramienta – Jmeter (17 de mayo de 2015): http://jmeter.apache.org/

- [WEB43] – Herramienta – LoadUI (17 de mayo de 2015): http://www.loadui.org/

- [WEB44] – Herramienta – FunkLoad (17 de mayo de 2015): http://funkload.nuxeo.org/intro.html

- [WEB45] – Herramienta – Wireshark (17 de mayo de 2015): https://www.wireshark.org/about.html

- [WEB46] – Herramienta – NMAP (17 de mayo de 2015): https://nmap.org/

- [WEB47] – Enlace de protocolo MDB (17 de mayo de 2015): http://www.vending.org/images/pdfs/technology/mdb_version_4-2.pdf

- [WEB48] – Definición de calidad – MAESTROSDELACALIDAD (17 de mayo de 2015): http://maestrosdelacalidadop100111.blogspot.com.es/2012/09/filosofia-phillip-crosby.html

- [WEB49] – Definición de pruebas – WIKIPEDIA (17 de mayo de 2015): http://en.wikipedia.org/wiki/Portal:Software_testing

- [WEB50] – Fallos de software – VARIABLENOTFOUND (17 de mayo de 2015): http://www.variablenotfound.com/2008/11/20-desastres-famosos-relacionados-con_23.html

- [WEB51] – Fallos de software – WIKIPEDIA (17 de mayo de 2015): http://en.wikipedia.org/wiki/List_of_software_bugs

- [WEB52] – Informe Fallo Ariane 5 – UNIVERSIDAD POLITECNICA VALENCIA (17 de mayo de 2015): http://www.upv.es/satelite/trabajos/pract_9/kike/paginas/accid.htm

- [WEB53] - Datos de la destrucción de la MARINER 1 – NASA (17 de mayo de 2015): http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=MARIN1

- [WEB54] - Pruebas estáticas – UNIVERSIDAD NACIONAL DE LANUS (17 de mayo de 2015): http://sistemas.unla.edu.ar/sistemas/sls/ls-4-optativa-algoritmos-y-lenguajes-prueba-del-software/pdf/Pruebas-de-Software-C04-Tipos-de-pruebas-estaticas.pdf

- [WEB55] - Grafo de flujo – WIKIPEDIA (17 de mayo de 2015): http://es.wikipedia.org/wiki/Grafo_de_control_de_flujo

- [WEB56] - Técnica de caja negra – UTEST (17 de mayo de 2015): https://university.utest.com/black-box-software-testing-bbst-bug-advocacy-effective-bug-investigation-and-reporting-lecture-1-basic-concepts/

- [WEB57] - Técnica de caja negra – TESTINGEDUCATION (17 de mayo de 2015): http://www.testingeducation.org/BBST/foundations/BBSTFoundationsNov2010.pdf

- [WEB59] - Pruebas exploratorias - UTEST (17 de mayo de 2015): https://university.utest.com/exploratory-testing-the-basics/

- [WEB60] - Calidad y pruebas – SOFTWAREQATEST (17 de mayo de 2015): http://softwareqatest.com/

- [WEB61] - Calidad y pruebas – APTEST (17 de mayo de 2015): http://aptest.com/resources.html

- [WEB62] - Pruebas de alpha y beta – TDLP (17 de mayo de 2015): http://www.tldp.org/HOWTO/Software-Proj-Mgmt-HOWTO/users.html#ALPHABETA

- [WEB63] - Test plan – WIKIPEDIA (17 de mayo de 2015): http://en.wikipedia.org/wiki/Test_plan:

- [WEB64] - Norma ISO 9000 (17 de mayo de 2015): http://www.iso.org/iso/iso_9000

- [WEB65] - Norma ISO 25000 – ISO (17 de mayo de 2015): http://iso25000.com/index.php/normas-iso-25000

- [WEB66] - TMMI – CALIDADYSOFTWARE(17 de mayo de 2015): http://www.calidadysoftware.com/testing/tmmi_nuevo_cmmi_enfocado_a_pruebas.php

- [WEB67] - IEEE 29119 – UNIOVI (17 de mayo de 2015): http://in2test.lsi.uniovi.es/gt26/?lang=es

- [WEB68] – Herramientas de pruebas – SOFTWAREQATEST (17 de mayo de 2015): http://softwareqatest.com/qatweb1.html

- [WEB69] – Herramientas de pruebas – TESTEANDOSOFTWARE (17 de mayo de 2015): http://testeandosoftware.com/las-mejores-herramientas-para-realizar-pruebas-de-software/

- [WEB70] – Herramientas de pruebas – JAVASOURCE (17 de mayo de 2015): http://java-source.net/open-source/code-analyzers

- [WEB71] – Herramientas de pruebas estáticas – BLOG JAVIERGARZAS (17 de mayo de 2015): http://www.javiergarzas.com/2012/03/herramientas-de-calidad-software.html

- [WEB72] – Herramientas de pruebas UNIVERSIDAD MURCIA (17 de mayo de 2015): http://www.um.es/docencia/barzana/IAGP/Enlaces/CASE_principales.html

- [WEB73] – Herramienta WIRESHARK (17 de mayo de 2015): https://www.wireshark.org/

- [WEB74] – Herramientas de seguridad TESTEANDOSOFTWARE (17 de mayo de 2015): http://testeandosoftware.com/las-mejores-herramientas-para-pruebas-de-seguridad/

- [WEB75] – Herramientas de seguridad HACKPLAYERS (17 de mayo de 2015): http://www.hackplayers.com/2013/12/top10-mejores-herramientas-2013-toolwatch.html

- [WEB76] XMAPP – Servidor Apache (17 de mayo de 2015): https://www.apachefriends.org/es/index.html