**Desarrollo de Software**
**Semestre 8**

**Programa de la unidad didáctica:**
**Pruebas y mantenimiento de sistemas de software**

**Unidad 2.**
**Pruebas de sistemas de software**

**Clave:**
**15144842**

**Ciudad de México, mayo del 2024**

**Universidad Abierta y a Distancia de México**

DCEIT

## Índice

Unidad 2. Pruebas de sistemas de software ..... 3
Presentación de la unidad ..... 3
Logros ..... 4
Competencia específica ..... 4
2.1. Administración de procesos de pruebas de software ..... 5
2.1.1. Plan de pruebas de software ..... 9
2.1.2. Seguimiento del plan de pruebas ..... 22
2.2. Técnicas de pruebas ..... 25
2.2.1. Pruebas de caja blanca ..... 27
2.2.2. Pruebas de caja negra ..... 38
2.3. Niveles de pruebas ..... 47
2.3.1. Pruebas unitarias ..... 48
2.3.2. Pruebas de integración ..... 50
2.3.3. Pruebas de implantación ..... 54
2.3.4. Pruebas de aceptación ..... 59
2.3.5. Pruebas de regresión ..... 68
Cierre de la unidad ..... 75
Para saber más ..... 77
Fuentes de consulta ..... 78

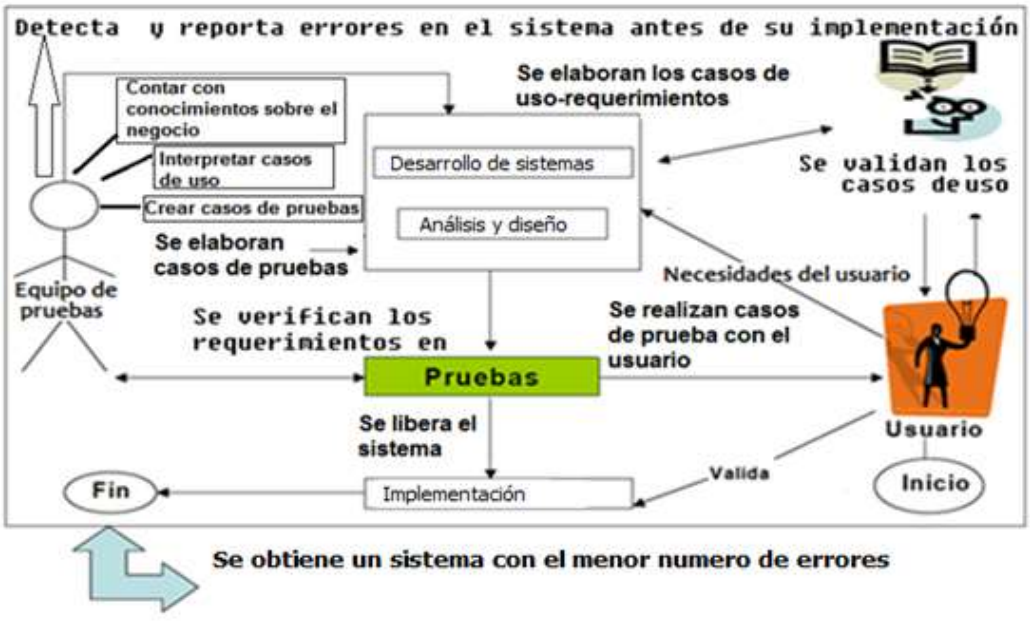
## Unidad 2. Pruebas de sistemas de software

### Presentación de la unidad

En la Unidad 1. Fundamentos del aseguramiento de la calidad del software, se expuso el marco para el aseguramiento de la calidad; sin embargo, es necesario comprobar a través de una batería de pruebas que este marco efectivamente funciona; y si no, hacer los arreglos pertinentes para garantizar la calidad del software.

En la fase de diseño de pruebas es importante considerar que cada sistema de software es diferente. Habrá algunos sencillos, otros complejos; de cualquier modo, el equipo de pruebas tiene la función de guiar a todos los involucrados por el camino correcto, para que el proyecto se lleve a buen término. Después de todo, las pruebas arrojan información sobre que se basan las decisiones determinantes sobre el producto de software.

En el siguiente esquema se observa cómo las pruebas están en todo el proceso de desarrollo del sistema de software. Aunque se ubique gráficamente después del análisis y diseño, es precisamente desde el análisis y diseño del software donde se comienzan a elaborar los casos de prueba, que se aplican durante el desarrollo y antes de la implementación del sistema de software.




**Las pruebas en el ciclo de vida de desarrollo del sistema de software**

En la presente unidad se sientan las bases para el diseño de pruebas de sistemas de software, y marca el precedente a la unidad 3. Mantenimiento de sistemas de software. Ya que un sistema puesto en marcha, después de un cierto tiempo, necesita mantenimiento para garantizar la calidad de su funcionabilidad.

En el tema 2.1. Administración de procesos de prueba de software, se expondrán los elementos necesarios para establecer una estrategia de administración adecuada, que permita diseñar un plan de pruebas concreto para los requerimientos del software, lo que da paso al tema 2.2. Técnicas de pruebas, donde se abordarán las distintas formas de realizar las pruebas de software. En el tema 2.3. Niveles de pruebas, identificarás el objeto de las pruebas a los cuales se aplican las técnicas, y de acuerdo con estos objetos, determinar el nivel de aplicación que va desde un módulo hasta el sistema completo.

Algunas marcas mencionadas en esta unidad son:
Bugzilla, software de código abierto de Netscape Communications
Test Manager, marca registrada de Microsoft Corporation.
WiNamp, marca registrada de Nullsoft Inc.
Internet Explorer y el logo Internet Explorer, marcas registradas de Microsoft Corporation.
Mozilla Firefox, Firefox y el logo Firefox, marcas registradas de Firefox Foundation.
Google y el logo Google Chrome, marcas registradas de Google Inc
EXCEL, marca registrada de Microsoft Corporation.

## Logros

Al término de esta unidad lograrás:

- Identificar en qué consiste la administración de procesos de pruebas de software.
- Identificar los tipos de técnicas de pruebas y sus aplicaciones.
- Relacionar los niveles de pruebas con su aplicación en el desarrollo de productos de software.
- Determinar el tipo de prueba y nivel requerido en un producto de software.

## Competencia específica

- Diseñar un plan de pruebas para efectuar la revisión de productos de software durante el proceso de desarrollo, mediante el análisis de los requerimientos del software y el tipo de prueba correspondiente

## 2.1. Administración de procesos de pruebas de software

La *administración de procesos de pruebas* es una acción mediante la cual se verifica que el sistema de software no tenga errores, de acuerdo con paradigma de programación utilizado, mediante un plan de pruebas que contiene a su vez casos de prueba, que se aplican bajo ciertos escenarios, según los requerimientos del sistema.

Según la IEEE Computer Society, el proceso de prueba del software es"la verificación dinámica del comportamiento del software a partir de un conjunto finito de casos de prueba" (Abran y Moore, 2004, p. 4-1).

La prueba se aplica una vez que se diseña un número particular basado en los casos de uso y, a medida que se aplica la prueba en cada caso, se analizan los resultados, se hacen cambios y se vuelve a probar para verificar la mejora del comportamiento, de ahí el dinamismo de la verificación.

Para Davis (2006), la administración del proceso de pruebas es la organización, el control del proceso y de los artefactos requeridos para la aplicación de pruebas.

La administración de procesos de pruebas se divide en las siguientes fases:

**Organización.** Consiste en tener claro quién, qué, y cuándo se va a evaluar. Quién va evaluar, qué técnica de prueba se va aplicar, y quién diseñará los casos de pruebas, son temas que la fase de organización se encarga de poner en perspectiva.

**Planificación.** Es un plan detallado de pruebas. Es el conjunto global de las tareas que aborda las cuestiones de por qué, qué, dónde y cuándo medir. El por qué se evalúa se debe a que generalmente es necesario evaluar un requisito específico. Para el qué, debe probarse si se seleccionan los casos de pruebas necesarios. El dónde probar, determina la documentación del componente de software y la configuración del hardware

necesarios. El cuándo hacer la prueba, se resuelve por el tipo de prueba a realizar: estática durante la programación del componente, o dinámica cuando ya el componente está terminado.

**Creación.** Es un proceso de captura de los pasos específicos para diseñar los casos de pruebas, de acuerdo con los requerimientos del software. Los casos pruebas terminan convirtiéndose en scripts de prueba (manual o automatizada).

**Ejecución.** Consiste en aplicar las pruebas mediante el ensamblaje de secuencias de scripts, en una serie de pruebas.

**Presentación de informes.** Analizan y comunican los distintos resultados del trabajo de comprobación. El informe determina el estado actual de las pruebas, así como el nivel general de la calidad del componente o sistema.

Según el diccionario de la Real Academia Española (2014)**,** la palabra prueba significa:

- Acción y efecto de probar.
- Razón, argumento, instrumento u otro medio con que se pretende mostrar y hacer patente la verdad o falsedad de algo.
- Indicio, señal o muestra que se da de algo.
- Ensayo o experimento que se hace de algo, para saber cómo resultará en su forma definitiva.

La palabra *prueba*, en el ámbito del desarrollo de software, se utiliza para designar un conjunto de casos y procedimientos de prueba (Myers, 2004). Las pruebas de un sistema tienen como objetivo verificar la funcionalidad del mismo, a través de sus interfaces externas, y comprobar que dicha funcionalidad sea la esperada en función de los requisitos del sistema (Abran y Moore, 2004). En el contexto de desarrollo de software, la acción de probar es una actividad en la cual el sistema, o uno de sus componentes, se ejecutan en escenarios específicos. Los resultados que arroje esta ejecución será la prueba que se registrará y evaluará.

Los sistemas de software, tanto los diseñados a la medida como los comerciales, requieren de un plan de pruebas que garantice que han sido desarrollados siguiendo

estándares de calidad internacionales. Dicho plan se realiza mediante la administración del proceso de pruebas de software, es decir, mediante la planificación de las acciones a seguir, así como de los recursos y el tiempo en el cual se elaborarán las pruebas.

Como ya se mencionó, la prueba se aplica una vez que se diseñan los casos. Se entiende como **caso de prueba** al conjunto de pruebas de entrada que se aplican bajo ciertos escenarios desarrollados para un objetivo, tal como probar una ruta particular de un programa o verificar el cumplimiento con un requerimiento específico. **Los escenarios de prueba** son la configuración específica del software que aborda un proceso de negocio en el sistema de software. “Los escenarios son descripciones de ejemplos de las sesiones de interacción. Cada escenario abarca una o más posibles interacciones” (Sommerville, 2011, p. 139).

Hay escenarios por tipos de usuario, por ejemplo: escenario 1, usuario →técnico- , escenario 2, usuario → gerente de contabilidad. También hay escenarios de acuerdo con los tipos de pruebas que se realizarán; por ejemplo, escenario 1, para pruebas funcionales; el escenario 2, para pruebas de integración.

Como parte de la administración del proceso de pruebas de software, en un primer momento se deben definir los roles y las responsabilidades del equipo. Debe tomarse como base la metodología RUP (*rational unified process*, por sus siglas en inglés), que establece los siguientes roles en el proceso de pruebas, responsabilidades y documentos entregables a cargo de cada uno de los miembros del equipo.

| Rol | Responsabilidad | Entregables |
|---|---|---|
| **Administrador de pruebas** | Responsable del éxito o fracaso de la prueba | • Plan de pruebas<br>• Resumen de resultado de pruebas<br>• Lista de problemas y cambios de requerimientos |
| **Analista de pruebas** | Identifica y define las pruebas requeridas | • Casos de pruebas<br>• Modelos de análisis de carga de trabajo<br>• Datos de pruebas<br>• Resultados de pruebas |
| **Diseñador de pruebas** | Define la estrategia de pruebas y asegura su puesta en práctica acertadamente | • Plan de estrategia<br>• Arquitectura de automatización<br>• Configuración del entorno de pruebas |

| | | |
|---|---|---|
| | | • Suite de pruebas |
| **Ejecutor de pruebas** | Responsable de todas las actividades de las pruebas | • Registro de pruebas, script de pruebas |

**Roles del proceso de pruebas. Basado en: Ness y Thomas, 2005.**

| Entregables | Descripción |
|---|---|
| Plan de pruebas | Señala el enfoque, los recursos y el esquema de actividades, así como los elementos a probar, las características, el personal responsable y los riesgos asociados. |
| Resumen de resultado de pruebas | Resume las actividades y resultados. También contiene una evaluación de los elementos de prueba correspondientes contra los criterios de salida. |
| Lista de problemas y cambios de requerimientos | Contiene la problemática inherente y los cambios de requerimientos. |
| Casos de pruebas | Detalla un conjunto de valores de entrada, condiciones previas de ejecución, resultados esperados y las condiciones posteriores de ejecución desarrollado para un objetivo particular o condición de prueba, tales como el ejercicio de una ruta de programa en particular o para verificar el cumplimiento de un requerimiento específico. |
| Modelos de análisis de carga de trabajo | Es la especificación que identifica las variables que afectan al rendimiento del sistema y cómo medir su efecto. |
| Datos de pruebas | Datos necesarios para ejecutar una prueba, así como lo que afecta o es afectado por el componente o sistema bajo prueba. |
| Resultados de pruebas | Reporte de la ejecución de una prueba. Incluye salidas a las pantallas, cambios en los datos, informes y mensajes de comunicación enviados. |
| Plan de estrategia | Contiene la estrategia, el tipo y la técnica de prueba a usar. |
| Arquitectura de automatización | *Testware* utilizado en pruebas automatizadas, tales como los scripts. |
| Configuración del entorno de pruebas | Documento que describe un proceso de pruebas para determinar la portabilidad de un producto de software. |
| Suite de pruebas | Conjunto de varios casos para un componente o sistemas bajo prueba, donde la postcondición de una prueba se utiliza a menudo como una condición previa para la siguiente. |
| Registro de pruebas | Registro cronológico de los datos pertinentes sobre la ejecución de las pruebas. |
| Script de pruebas | Procedimiento de prueba automatizado. |

**Tabla de entregables y su descripción. Basado en: ISTQB, 2012.**

Autores en el área de la ingeniería de software como Pressman (2010) y Sommerville (2011) reconocen que el proceso de ejecución de pruebas debe ser considerado durante

todo el ciclo de vida de un proyecto de un sistema de software, para así obtener un producto de calidad, pues el éxito depende de la aplicación de la estrategia adecuada.

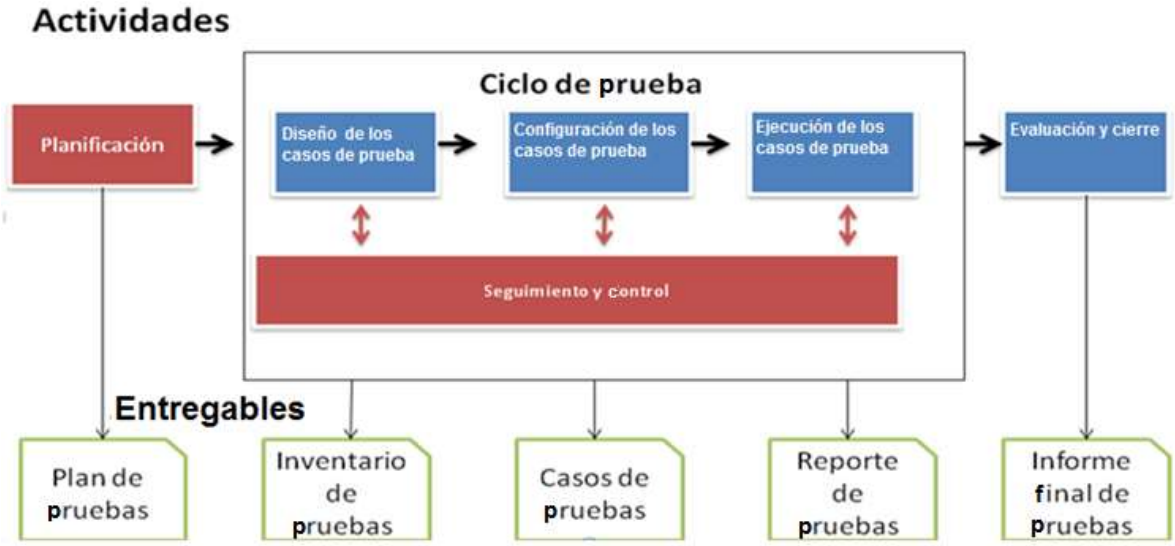
Por *estrategia de prueba* se entiende un esquema que describe el enfoque de prueba del ciclo de desarrollo de software. Está creada para informar al líder de proyecto, evaluadores y desarrolladores acerca de todo el proceso de pruebas. Esto incluye el objetivo, control, seguimiento, tiempo total y los recursos necesarios que conlleva el proceso y el entorno de prueba.

Las estrategias describen qué tipos de prueba y qué criterios de entrada o salida se aplicarán. Las estrategias de prueba se desarrollan sobre la documentación del diseño del software, ya que describen su funcionalidad. Para cada etapa del diseño del desarrollo del software, una estrategia de prueba correspondiente se debe crear para probar la funcionalidad del componente o sistema.

La información acerca de lo que es una prueba, un proceso y los escenarios, conduce a la necesidad de gestionar el proceso a través de la administración de pruebas del software, lo que involucra la selección de una estrategia a seguir donde se resalta la importancia de seleccionar la más adecuada, de acuerdo con las características del sistema, del entorno de aplicación, la definición de roles, responsabilidades del equipo de pruebas y la definición de los entregables. Con estos elementos definidos se cuenta con una base para la elaboración del plan de pruebas y su respectivo seguimiento, que verifique que las personas indicadas aplican las pruebas al software en las fases acordadas, para analizar posteriormente los resultados y definir la línea de acciones de mejora.

### 2.1.1. Plan de pruebas de software

El plan de pruebas de software, como parte de la administración de procesos de pruebas, consiste en la aplicación de la planificación de una serie de actividades y su correspondiente seguimiento, que garanticen que tanto el componente como el sistema cumplen con las expectativas establecidas en el acta del proyecto de software.




**Ciclo del plan de pruebas. Fuente: Ministerio de Comunicaciones, 2008, p. 13.**

Los pasos que se observan en el esquema se desarrollarán en el presente subtema y en el 2.1.2. Seguimiento del plan de pruebas.

**Paso 1. Planificación de pruebas**
Consiste en determinar qué, cómo, cuándo y con qué se va a evaluar. Depende del nivel de software (tema a desarrollar en el 2.3. Niveles de prueba). Se determina la técnica de prueba de software a utilizar, ya sea de caja blanca o negra (2.2. Técnicas de pruebas). Depende de lo que se evaluará, el código o la interfaz del componente o sistema de software. Toca entonces diseñar los casos de pruebas. Con todo lo anterior, se establece una estrategia de plan de pruebas a aplicar.

**Paso 2. Diseño de los casos de prueba**
Es un documento que refleja las **expectativas** de los usuarios. Permite **verificar** tales expectativas y **validarlas.** Describe detalladamente el cómo se llevará a cabo la prueba.

Los casos de pruebas se **delimitan** con base en las necesidades que el sistema debe satisfacer. Los siguientes, son elementos del diseño que todo caso de pruebas debe cubrir:

- **Casos de uso. "**Son una técnica que se basa en escenarios para la obtención de requerimientos […] identifica el tipo de interacción y los actores involucrados" (Summerville, 2005, p.140).

- **Requerimientos suplementarios.** Son requerimientos no funcionales[1] de alto nivel a implementarse en proyecto en términos de necesidades de usuarios finales. Algunos de éstos son:

    - Seguridad, usabilidad, confiabilidad.

- **Requerimientos de configuración**. **Controlan** y **documentan** el cambio en el desarrollo de un sistema. A medida que crece el esfuerzo, crece también la necesidad de implementar la gestión de configuración que no es más que el proceso de **controlar** y **documentar** el cambio en el desarrollo de un sistema. Los requerimientos de configuración tienen tres propósitos principales:

    - **Identificar** la **configuración** del producto en diferentes puntos del tiempo.
    - **Controlar cambios** en la configuración.
    - **Mantener** la **integridad** y el **seguimiento** de la configuración a través del ciclo de vida del software.

- **Requerimientos de configuración.** Consisten en el **contexto** de aplicación de los casos de pruebas.
- **Requerimientos de seguridad**. Se refiere a las **características** que determinan el **esquema** y el **nivel** de **seguridad,** para **minimizar** los riesgos que amenazan al sistema de software.

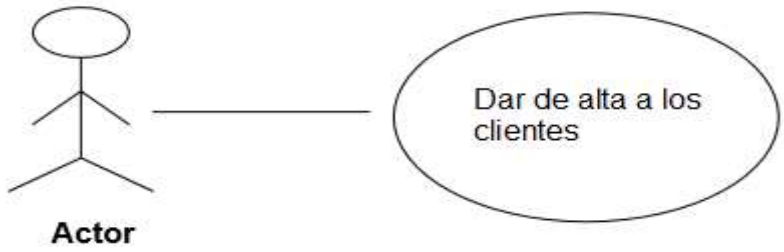
- **Requerimientos de instalación**. Se refiere a las características **técnicas** a cumplir para que el sistema se **instale** y **funcione** correctamente, entre ellas se incluye:

    - La versión y sistema operativo.
    - La memoria RAM.
    - La capacidad en disco duro.
    - La capacidad del procesador.

[1]Requerimientos no funcionales. Son restricciones de los servicios o funciones ofrecidos por el sistema.

Es muy importante tomar en cuenta que los casos de prueba no cubren la verificación de todos los requisitos, por lo que se seleccionan los más críticos para el alcance del trabajo. Esto implica la combinación de los escenarios posibles de los casos de uso del sistema.

El diseño de los casos de prueba, basado en los casos de uso, es básico. Según Fowler (2000), un *caso de uso* es un conjunto de escenarios unidos por una meta de un usuario, y describe la forma en que interactúa con los escenarios. Siguiendo la metodología de Heumman (2001), para el diseño de los casos de prueba se tiene, en primer lugar, el siguiente caso de uso:




**Ejemplo gráfico caso de uso alta del cliente. Basado en Fowler, 2000, p. 42.**

El caso de uso *alta del cliente* describe la interacción del usuario llamado actor de dar de alta a clientes en el módulo de clientes.

Para establecer el nombre de los casos de pruebas, se toma el nombre del caso de uso para relacionarlos.

Siguiendo con el ejemplo, se desarrolla un caso de prueba basado en un caso de uso. Para ello, se siguen los pasos que se describen a continuación:

**a) Identificar los flujos básicos de un caso de uso,** por ejemplo**:**

1. **Seleccionar** un cliente de la cartera de clientes.
2. **Capturar** el nombre del cliente.
3. **Verifica** si el cliente no ha sido registrado.

**b) Identificar flujos alternos,** por ejemplo**:**

1. **Verificar** si el cliente ha sido registrado. Si ya lo fue, envía el mensaje correspondiente.

**c) Verificar** que el caso de uso se ha implementado con todos los flujos**,** por ejemplo**:**

- **Flujo básico.** Dar de alta a un cliente nuevo de la cartera de clientes.
- **Fuljo alterno 1.** El cliente ya fue dado de alta.

**d) Verificar** los escenarios, por ejemplo**:**

**Escenario 1**. Cliente dado de alta correctamente/flujo básico.

**e) Diseñar** los casos de prueba**,** por ejemplo**:**

Caso de uso **1. Alta cliente.** El caso de prueba asociado sería el siguiente: **1-01 Cliente ya registrado.**

El siguiente formato se utiliza para documentar el caso de prueba y contiene diversa información, como la versión de la prueba, nombre, autor, fecha de creación del caso, quién va aplicarla. Se describe cuál es la interacción entre el actor[2] y el componente de software. Las precondiciones para que se dé el caso, las condiciones que se generan después de llevarse a cabo el proceso que describe el caso de uso y, por último, en los puntos de extensión se especifica el criterio de aceptación, qué es lo que se espera después de aplicar la prueba. Si este criterio de aceptación no se cumple, hay que hacer las correcciones hasta que se cumplan. El punto de documentos externos hace referencia cualquier documento necesario para llevar a cabo la prueba.

| | |
|---|---|
| **Plan de pruebas** | Alta cliente 1.0 |
| **Nombre**: | 1-01 Cliente ya registrado |
| **Autor**: | Carlos Rojas Peréz |
| **Fecha**: | 05/03/13 |
| **Realizador de prueba**: María Lopéz Garcia | |
| **Caso de uso:** 1_Caso de uso Alta cliente | |
| **Descripción**: Da de alta al cliente y nombre existe, el sistema debe mandar error | |
| **Precondiciones**: Seleccionar nombre del cliente | |
| **Post-condiciones**: El sistema registra al nuevo cliente | |
| **Criterio de aceptación:** El sistema debe mandar mensaje "cliente ya registrado" | |
| **Documentos externos:** Cartera de clientes | |

Ejemplo documento de caso de prueba. Fuente: Heumman, 2001.

[2] Usuario

Para realizar el inventario de las pruebas del plan de pruebas, es necesario llevar un registro de las mismas. Contiene el nombre de los casos de prueba y la versión del plan de pruebas con que se está trabajando.

El inventario de los casos de prueba se realiza a través de un formato en donde se registra el número de caso, el escenario donde se aplicará, las condiciones de entrada y el resultado esperado. El formato se puede realizar con ayuda de alguna herramienta de control, tal como un sistema de software para gestión de proyectos como el Microsoft Project, una hoja de cálculo como Excel u Open Office Excel, en una herramienta especializada como el Test Manager de Microsoft, el Rational de IBM, o en la herramienta libre de Bugzilia.

El siguiente formato describe el documento que se obtiene del paso de diseño de casos de pruebas, que es un inventario de los casos generados con el número del caso de prueba y el escenario donde se va a desarrollar el proceso, así como las condicionantes necesarias para que se lleve a cabo el proceso y el resultado esperado.

| Número de caso de prueba | Escenario | Condición | Resultado esperado |
|---|---|---|---|
| 1-01 | Escenario 1 | El cliente no está dado de alta | No se ejecuta flujo alterno 1, continúa con el flujo básico |

**Formato de control de inventario de casos de pruebas. Fuente: Heumman, 2001.**

Se recomienda revisar el documento llamado ***Plantilla RUP*** en los materiales de apoyo. Es una plantilla de plan de pruebas basado en el proceso RUP para documentar los casos de pruebas, fecha, versión, y el número de instancias que fueron necesarias.

El *Rational Unified Process* (RUP) es una herramienta **CASE** que proporciona un marco de proceso de desarrollo de software **iterativo**, **adaptable**, que permite seleccionar los elementos del proceso apropiados, de acuerdo con las necesidades del proyecto. RUP es una implementación específica del proceso unificado (PU).

A continuación, se expone la fase de configuración.

**Paso 3. Configuración.** Para llevar a cabo las pruebas se debe configurar, ya sea de forma manual o automatizada a través de un script, un archivo de órdenes o de procesamiento por lotes.

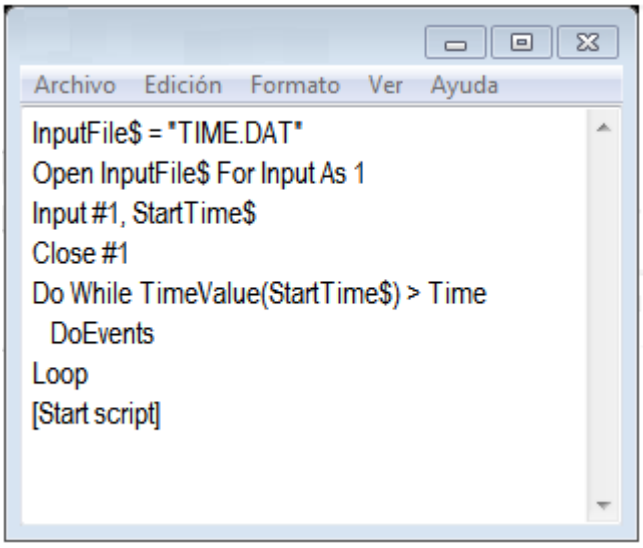
- **Scripts manuales.** Describen, en un documento, los pasos que el ejecutor de pruebas debe realizar para llevar a cabo el script, por ejemplo:

| |
|---|
| 1. Solicitar archivo con valores iniciales |
| 2. La aplicación debe leerlos |
| 3. Si son o no valores iniciales correctos |
| 4. Generar reporte de comportamiento |

**Ejemplo de script manual**

- **Scripts automatizados**. Se pueden crear o generar automáticamente con herramientas de automatización de pruebas, o con un lenguaje de programación. Por ejemplo, el script siguiente está grabado en un archivo de texto, que se puede ejecutar desde la línea de comando de Linux de la siguiente forma:

  **$ ./prueba.sh** cuando da inicio abre el archivo **Time** para registrar la hora de aplicación, mientras el tiempo de inicio sea **menor** al de realización de todos los eventos que siga ejecutando el script:




**Ejemplo de script automatizado**

Los documentos generados en este paso son los **casos** de **pruebas** implementadas y listas para su ejecución, ya sean manuales o automatizadas.

**Paso 4. Ejecución.** La ejecución de los casos de pruebas se llevará a cabo por el personal designado para ello y en las fechas que se haya acordado, según la planificación, y con las herramientas seleccionadas, ya sea de forma manual o automatizada.

La persona que evalúe el código forzosamente debe conocer cómo está la programación del componente. Si se evalúa la interfaz sólo se requiere conocer los datos de entrada del componente, y lo que se espera que haga el componente con tales datos. Todo caso de prueba ejecutado debe estar documentado con versión y fecha de aplicación.

Para evaluar correctamente las pruebas, sin dejar lugar a dudas o interpretaciones personales, se establecen criterios de evaluación que variarán de acuerdo con el proyecto de software y sus requerimientos funcionales.

La tabla siguiente muestra un ejemplo de los criterios para evaluar como aceptables las diversas pruebas que se aplican a los componentes y al sistema.

| Tipos de pruebas | Criterios de evaluación |
|---|---|
| Pruebas unitarias | • Detectar errores en la ejecución de las pruebas.<br>• El 90% de las pruebas realizadas deben ser exitosas. |
| Pruebas de sistema | • Detectar errores en la ejecución de las pruebas.<br>• Que los reportes generados por las herramientas de automatización de las pruebas contengan las mínimas variables que permitan un análisis acertado de cada una de las pruebas realizadas.<br>• Tener en cuenta todos los escenarios posibles.<br>• El 90% de las pruebas realizadas deben ser exitosas. |
| Pruebas de integración | • La totalidad de los puntos de control probadas debe ser mayor al 75% de los componentes que integran la solución.<br>• Detectar errores en la ejecución de las pruebas.<br>• El 90% de las pruebas realizadas deben ser exitosas. |
| Pruebas de regresión | • Se debe tomar como base los criterios de aceptación de las pruebas que se volverán a realizar. |
| Pruebas funcionales | • El resultado de cada caso de prueba debe ser igual al resultado de salida esperado.<br>• Encontrar fallas al ejecutar los diferentes casos de pruebas. |

| | |
|---|---|
| | • La aplicación cumple con los requerimientos funcionales especificados en la fase de análisis.<br>• La aplicación cumple con los requerimientos mínimos para el funcionamiento. |
| Pruebas de usabilidad | • El resultado de cada caso de prueba debe ser igual al de salida esperado.<br>• Se deben incluir tanto los datos de entrada válidos y esperados como los no validos e inesperados.<br>• Encontrar los errores al ejecutar los diferentes casos de pruebas.<br>• La aplicación debe cumplir con los requerimientos funcionales especificados en la fase de análisis.<br>• La aplicación debe cumplir con los requerimientos mínimos para el funcionamiento. |
| Pruebas de seguridad | • El resultado de cada caso de prueba debe ser igual al de salida esperado.<br>• La aplicación debe cumplir con los requerimientos mínimos de seguridad. |
| Pruebas de configuración | • Considerar todos los escenarios posibles.<br>• Que el sistema funcione bien en el ambiente de pruebas. |
| Pruebas de aceptación | • Tomar como base los criterios de aceptación de las pruebas que se volverán a realizar. |

**Ejemplo de criterios de evaluación de pruebas. Fuente: Ministerio de Comunicaciones, 2008, p. 44.**

El documento que se genera en este paso es el *Reporte de pruebas,* cuyos elementos básicos son los siguientes:

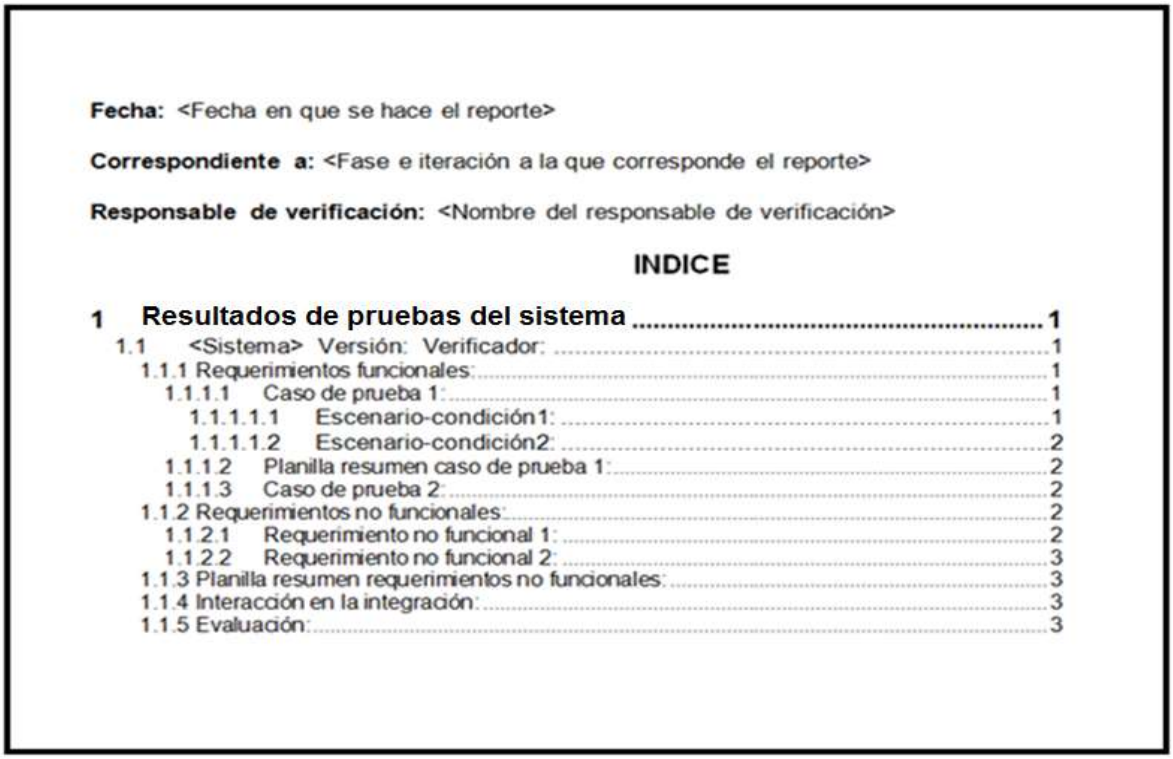
**Encabezado** con los siguientes datos:

- **Fecha** de realización del reporte
- **Correspondiente a:** la fase e iteración del reporte, la fase del desarrollo del software correspondiente
- **Responsable de verificación:** nombre del encargado de la aplicación de las pruebas, revisar los requerimientos funcionales y los casos de uso

**Índice**

**Resultados de pruebas del sistema.** El apartado se conforma de los siguientes elementos:

- Sistema. Versión: verificador
  - Requerimientos funcionales
    - Caso de prueba 1
      - Escenario-condición 1
      - Escenario-condición 2
    - Planilla resumen caso de prueba 1
    - Caso de prueba 2
  - Requerimientos no funcionales
    - Requerimiento no funcional 1
    - Requerimiento no funcional 2
  - Planilla resumen requerimientos no funcionales
  - Interacción en la integración
  - Evaluación



**Fecha:** <Fecha en que se hace el reporte>

**Correspondiente a:** <Fase e iteración a la que corresponde el reporte>

**Responsable de verificación:** <Nombre del responsable de verificación>

**INDICE**

**1 Resultados de pruebas del sistema ..... 1**
1.1 <Sistema> Versión: Verificador: ..... 1
1.1.1 Requerimientos funcionales: ..... 1
1.1.1.1 Caso de prueba 1: ..... 1
1.1.1.1.1 Escenario-condición1: ..... 1
1.1.1.1.2 Escenario-condición2: ..... 2
1.1.1.2 Planilla resumen caso de prueba 1: ..... 2
1.1.1.3 Caso de prueba 2: ..... 2
1.1.2 Requerimientos no funcionales: ..... 2
1.1.2.1 Requerimiento no funcional 1: ..... 2
1.1.2.2 Requerimiento no funcional 2: ..... 3
1.1.3 Planilla resumen requerimientos no funcionales: ..... 3
1.1.4 Interacción en la integración: ..... 3
1.1.5 Evaluación: ..... 3

**Ejemplo de reporte de pruebas del sistema. Basado en: Ministerio de Comunicaciones, 2008.**

**Paso 5. Evaluación y cierre.** Se centra en el reporte final de pruebas de software. Este reporte es generado después de que las pruebas de regresión se han aplicado. Verifican y validan que los errores encontrados en las diferentes pruebas aplicadas en las diferentes fases de desarrollo del software se hayan corregido, cerrando efectivamente el proceso de prueba, y liberando al sistema de software para su implantación. Éste último reporte tiene que estar firmado por el cliente, lo que significa que da su visto bueno de que el sistema se entregó en condiciones óptimas, y con un grado de aceptación de todas las pruebas aplicadas al software.

También se genera un documento evaluación de las pruebas en general, tiempos, aplicaciones, resultados y lecciones aprendidas.

| Evaluación del proceso de pruebas | |
|---|---|
| Actividad | Roles involucrados |
| Evaluacion de la satisfacción del cliente | Cliente<br>Líder de proyecto |
| Ajustes y mejoras del proceso de pruebas | Líder de proyecto<br>Diseñador<br>Tester |
| Reporte final del proyecto de pruebas | Líder de proyecto |

Tabla de actividad de evaluación y cierre del proceso de pruebas. Basado en: Ministerio de Comunicaciones, 2008.

El proceso y los resultados se ven reflejados en el documento *Plan de pruebas*, que presenta los siguientes elementos.

| 1. Introducción | Descripción |
|---|---|
| 1.1. Propósito del plan | Breve descripción de lo que abarca el documento |
| 1.2. Alcance | Cuál es el alcance a lograr |

| | |
|---|---|
| 1.3. Versión | Qué versión del plan se aplica |
| 1.4. Referencias | Tipos de requisitos considerados (funcionales y no funcionales) |
| 1.5. Visión general del plan | Elementos que conforman el plan de pruebas |
| 2. Requerimientos de pruebas | Requisitos necesarios para aplicar las pruebas |
| 2.1. Introducción | |
| 2.2. Filosofía de la prueba | Lo que se pretende evaluar y validar |
| 2.2.1. Generalidades | |
| 2.2.2. Áreas funcionales | Según los requerimientos funcionales |
| 2.2.3. Categorías de resultados de prueba | |
| 2.3. Entorno de la prueba | Hardware, software y datos de entrada para la ejecución del componente, así como la aplicación del caso de prueba |
| 2.3.1. Generalidades | |
| 2.3.2. Hardware | |
| 2.3.3. Software | |
| 2.3.3. Datos de prueba | |
| 2.4. Roles y responsabilidades del equipo de pruebas | Descripción del equipo de pruebas |
| 2.5. Identificación de la prueba | Justificación de la prueba |
| 2.5.1. Scripts de prueba | Automatización de las pruebas |
| 2.5.2. Reporte de resultados | |
| 2.5.3. Criterios de aceptación | Criterios basados en los requisitos funcionales |
| 2.5.4. Errores de prueba | Tipos de errores encontrados |
| 2.5.5. Documentación de la prueba | |
| 2.6. Pruebas funcionales | Basados en los requisitos funcionales |
| 3. Estrategia de pruebas | Niveles de pruebas a aplicar |
| 3.1. Pruebas por caso de uso | |

| | |
|---|---|
| 3.1.1. Pruebas de integración | |
| 3.1.2. Pruebas de aceptación | |
| 4. Casos de prueba | Descripción de los casos de pruebas |
| 4.1. Caso de prueba a | |
| 4.2. Caso de prueba b | |
| 4.3. Caso de prueba c | |
| 5. Resultados | |

Elementos de un plan de pruebas. Basado en: Ministerio de Comunicaciones, 2008.

El plan de pruebas se hace por cada nivel (se explicará a profundidad en el tema 2.3. Niveles de pruebas):

- Plan de pruebas de unitarias
- Plan de pruebas de integración
- Plan de pruebas de implantación
- Plan de pruebas de aceptación
- Plan de pruebas de regresión

El plan de pruebas es un proceso que involucra las actividades de **planeación**, **diseño**, **configuración**, **ejecución** de los casos de pruebas, con su respectivo seguimiento y control a través de la **documentación** generada en cada fase, lo anterior con el fin de garantizar un software con un alto grado de aceptación, que se ve traducido en el informe final y **cierre** del proceso de pruebas.

Se sabe de la importancia de los entregables como elementos indispensables para el debido seguimiento y control del proceso de pruebas, con el fin de que se realicen en tiempo y estén ajustados al presupuesto original. Esto es el resultado de una correcta administración del proceso de pruebas.

En el siguiente tema, se abordará más a detalle el proceso de seguimiento del plan de pruebas como elemento determinante en la administración del proceso de software.

### 2.1.2. Seguimiento del plan de pruebas

En el tema anterior se expusieron los pasos requeridos para la elaboración de un plan de pruebas con sus respectivos casos, que se aplican según fechas y escenarios previstos para obtener los resultados, mismo que hay que evaluar y corregir si es necesario, volver a aplicar la prueba y verificar que el error efectivamente se haya corregido, como parte de la administración de proceso de pruebas de software.

El seguimiento del plan de pruebas tiene como fin asegurarse que éstas se aplican en tiempo y forma, además de que se genere la respectiva documentación que respalde el resultado obtenido, hasta alcanzar el grado de satisfacción.

Por ejemplo, para realizar un seguimiento a un conjunto de pruebas después de ejecutarlas en la herramienta del Microsoft Test Manager. Se puede observar su progreso en la opción **ejecutar** pruebas, actividad que se realiza para cada conjunto de pruebas de forma individual.

Lo que muestra la imagen siguiente es cómo, a través de un código de color, se señalan las pruebas activas, las que no han sido superadas y las que sí se superaron. Las pruebas no superadas se pueden volver a aplicar después de su respectivo correctivo.

**Ejemplo en Microsoft Test Manager de seguimientos de pruebas. Fuente: MSDN, 2014b.**

El seguimiento se realiza a través de la documentación generada en cada etapa, verificando que los responsables de las actividades entreguen, en fecha y forma, los resultados de las pruebas dependiendo de la etapa de desarrollo del sistema de software.

Los informes que se generan en el proceso de pruebas son:

**Resultado de las pruebas de integración**. En este informe se entregan los resultados de las pruebas realizadas por los desarrolladores y el líder técnico. Se expone la validación de la integración entre los diferentes módulos que componen el sistema de software, con el fin de garantizar que su operación integrada sea correcta, y se documente cuáles fueron los resultados de las pruebas del sistema integrado.

**Resultado de las pruebas de implantación.** En este informe se entregan los resultados de las pruebas realizadas por el analista y el líder técnico de la validación de los requerimientos no funcionales del proyecto (requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema sino a la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento).

**Resultado de las pruebas de regresión.** En este informe se entregan los resultados de las pruebas realizadas por los desarrolladores, el analista de pruebas y el líder técnico de la validación. Se realiza al sistema después de que las correcciones son aplicadas.
En este informe se entregan los resultados de las pruebas unitarias realizadas por el analista de la validación del código de cada componente del sistema de software.

**Resultado de las pruebas de aceptación.** En este informe se entregan los resultados de las pruebas realizadas por los usuarios de empresa, validando los requerimientos funcionales establecidos.

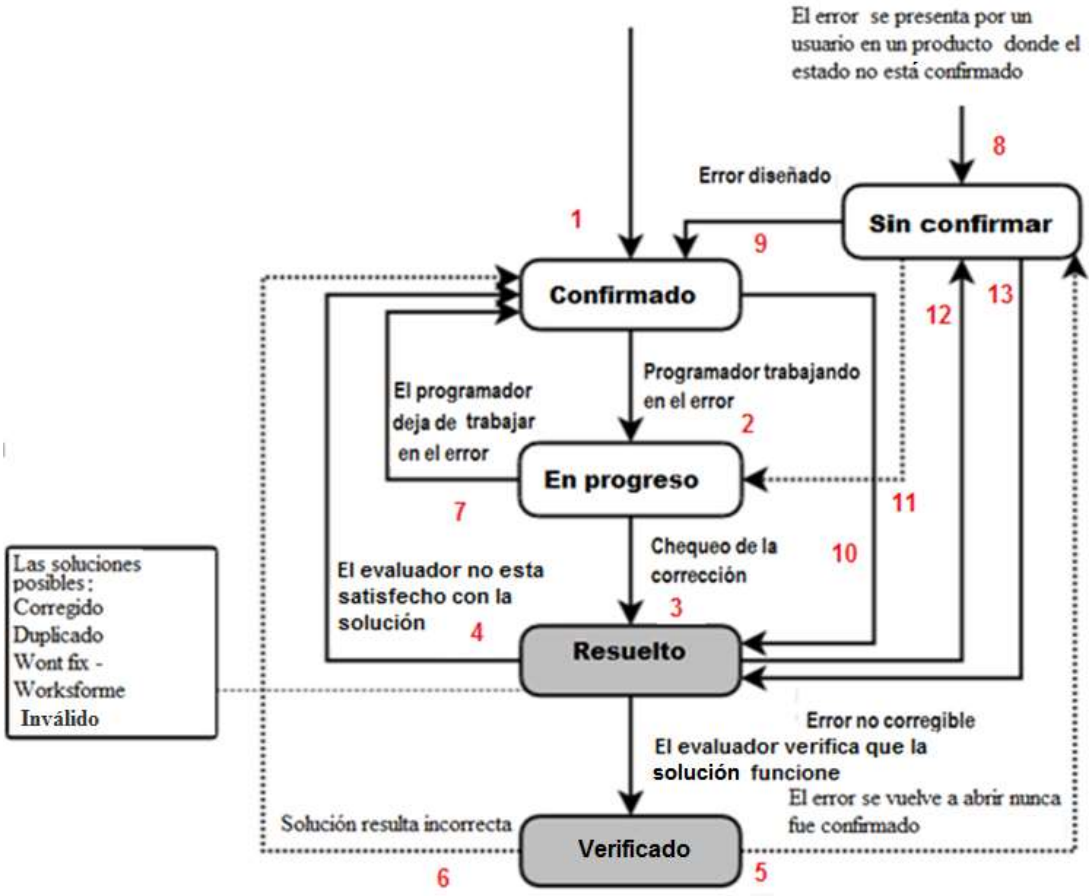
Los reportes por si mismos deben indicar el resultado de las pruebas. En estos resultados pueden ser que no se hayan detectado errores (lo cual es muy difícil), pero lo más probable es que sí existan, según los criterios de evaluación establecidos. Los reportes

deben ser enviados al líder del proyecto de desarrollo de software para que se realicen las mejoras, y así el equipo vuelva a aplicar las pruebas para validar que los cambios han sido realizados, y no hayan generado otros errores o salten otros ocultos.

Se hace necesario entonces distinguir los posibles errores que las pruebas pueden arrojar. Para repasar los conceptos de error, bug, falla y nota, revisa el *Glosario de términos frecuentes*.

La siguiente imagen muestra el **ciclo de vida de un error**, se inicia en el paso 1 donde se **confirma**; en el 2, el programador **trabaja** en el error; en el 3, se presenta una **solución** al evaluador que vuelve a ejecutar el caso de prueba. Si el evaluador no da su visto bueno el proceso se **repite**; El 4 sólo se da hasta que haya **conformidad** o se determine que no tiene solución o no se confirme el error.

Los caminos alternativos suceden cuando se deja de trabajar en la solución, no se está satisfecho con ésta o resulta incorrecta.



**Ciclo de vida de un error en Bugzilla. Fuente: The Bugzilla Guide, 2006.**
Las posibles soluciones a un error son:

- Corrección del error
- Detectado como duplicado del error o *wont fix*
- No se arregla porque no es considerado un error, es un workforme
- Si el error descrito no lo detectan otras personas, significa que es de la configuración del generador de reportes

Una herramienta útil para la gestión de pruebas, ya que realiza el reporte, el seguimiento y por ende el control de los errores es **Bugzilla**; además, **organiza** los defectos de software de acuerdo con la prioridad y el **grado** de **criticidad**; también asigna versiones a la solución de los mismos.

La verificación de que las pruebas se apliquen, así como la validación de que los posibles errores se resuelvan, se observan en las versiones del plan de prueba. Si son automatizadas, se pueden usar herramientas de apoyo tales como el Test Manager de Microsoft o Bugzilla.

Como se ha estudiado, la administración del proceso de pruebas de software es toda una metodología de trabajo, donde todo el equipo de pruebas de software debe estar consciente de su papel y responsabilidad en el proceso, para realizar las actividades según lo planeado con respecto a tiempo y recursos.

## 2.2. Técnicas de pruebas

Dentro de la administración del proceso de pruebas se establece un plan. Para ello es necesario realizar el diseño mediante dos tipos de evaluaciones, de acuerdo con la fase del desarrollo de software en que se encuentre el producto.
Los dos tipos de evaluación son los siguientes:

**Evaluación estática.** Es la **aplicación** de **pruebas sin ejecutar** el código del componente, según el glosario ISTQB de la Junta Internacional de Calificaciones de Pruebas de Software (ISTQB, 2012, p. 41). Se deben evaluar los distintos documentos o

productos que se generan en el desarrollo de software (especificación de requisitos, modelos conceptuales, diseño, código, etcétera), con el fin de comprobar la calidad del código.

**Evaluación dinámica.** Es la **aplicación** de pruebas mediante **la ejecución del componente** de software, según el glosario ISTQB (2012, p. 19). Denominada también prueba del software, proporciona distintos criterios para generar casos de prueba que provoquen fallos en los programas.

**Relación entre las técnicas de prueba y la fase del desarrollo de software. Basado en: SoftQaNetwork, 2009.**

Como se observa en la imagen anterior, la evaluación estática se lleva a cabo en el mismo orden en que se van generando los componentes del sistema de software, así como las dinámicas en la fase de codificación termina.

Mediante las pruebas estáticas, se realizan diferentes revisiones del análisis, diseño del software, y del código del sistema de software que evalúa los documentos o productos del desarrollo de software. Se busca detectar posibles errores en la documentación de los requisitos, diseño o código, mediante una lectura detallada.

Algunos tipos de revisiones más comunes se mencionan a continuación:

- **Revisiones informales.** Se realiza mediante un **intercambio** de opiniones entre los participantes del proyecto.

- **Revisiones formales o inspecciones.** Los participantes son responsables de la **fiabilidad** de la evaluación, y generan un informe que refleja el acto de la revisión.

Los tipos de revisiones formales son:

  - **Recorrido (*walkthrough*).** Consiste en simular la ejecución de casos de prueba para el programa que se está evaluando, semejante a realizar la corrida de un algoritmo.
  - **Auditorías**. Se comparan los documentos generados durante el desarrollo de un software, con estándares generales o de la organización. Se busca que toda la información necesaria se encuentre documentada y sea congruente.

Las pruebas dinámicas utilizan técnicas que se agrupan en:

**Pruebas de caja blanca o estructural.** Realizan un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa (algunos ejemplos se exponen en los siguientes subtemas).

**Pruebas de caja negra o funcionales.** Evalúan la interfaz del programa a probar. Se entiende por interfaz las entradas y salidas de dicho programa (algunos ejemplos se exponen en los siguientes subtemas).

Estas técnicas dinámicas de pruebas son las que se estudiarán con más detalle en los siguientes subtemas.

### 2.2.1. Pruebas de caja blanca

Las pruebas de caja blanca, como parte de la técnica dinámica de pruebas, se aplican después de la codificación del sistema de software, y se basan en el conocimiento de la lógica interna del código del sistema.

**Las pruebas de caja blanca o caja de cristal**. Conforman un método de casos de prueba que usa la estructura de control del diseño procedimental para obtenerlos (Pressman, 2010).

**Técnica de diseño de pruebas de caja blanca.** Es el procedimiento para generar casos de prueba que examinan que el código trabaje como se espera.

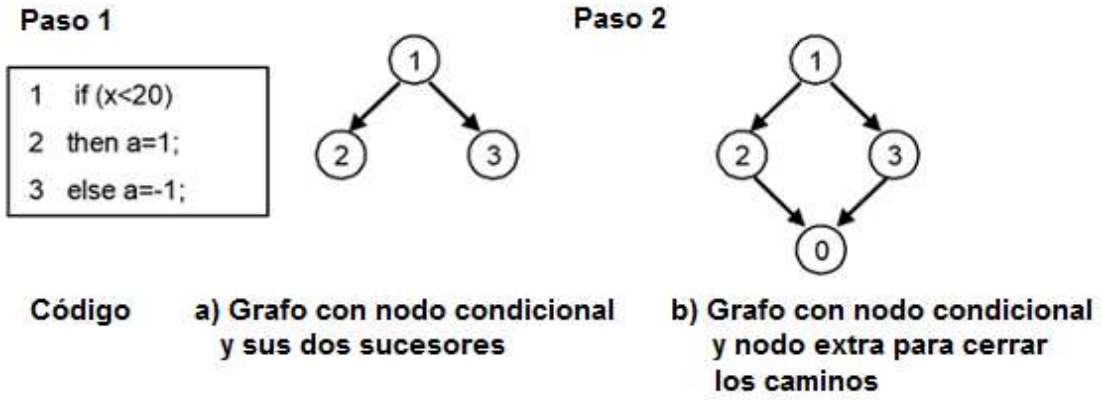
Las pruebas de caja blanca se centran en el análisis de la cobertura de código. Se refiere al grado, expresado en porcentaje, al que un elemento (código) ha sido ejercitado por un conjunto de pruebas, mediante las cuales se identifican los vacíos en una suite de caso de prueba. Esto permite identificar qué parte del código no se ejecutó por un determinado conjunto de casos. Al identificar lagunas en la suite, se agrega el caso respectivo. Por lo tanto, ayuda a mejorar la calidad de la aplicación de los casos de pruebas. Las pruebas de caja blanca se aplican en tres niveles: unidad, integración y regresión (se estudiarán en el tema 2.3. Niveles de pruebas)

Las pruebas contemplan los distintos caminos que se pueden generar gracias a las estructuras condicionales y a los distintos estados. Las técnicas de pruebas de caja blanca son las siguientes:

**Técnica de caja blanca cobertura del camino básico.** Fue propuesta en 1976 por Tom McCabe. El objetivo es definir un conjunto básico de caminos de ejecución o escenarios de las pruebas.
Los pasos a seguir para aplicar esta técnica son los siguientes:

1. Se identifica el código fuente a probar con base en la importancia del proceso a evaluar.
2. Se dibuja el grafo de flujo asociado.
3. Se calcula la complejidad ciclomática del grafo.
4. Se determina un conjunto básico de caminos independientes.
5. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

El siguiente ejemplo muestra la aplicación de estos pasos, la imagen muestra una fracción de código condicional (paso 1), así como los grafos que se generan (paso 2).




**Ejemplo de los pasos 1 y 2 de cobertura del camino básico**

**Paso 1.** Se identificó el código fuente a probar con base en la importancia del proceso a evaluar, en este caso la evaluación de una condición.

**Paso 2.** Se dibujan los grafos de flujo asociado, con las posibles variantes.

**Paso 3. Complejidad ciclomática del grafo.** La complejidad ciclomática es el cálculo del número de caminos independientes dentro de un fragmento de código, y determina el número máximo de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez. La forma clásica de calcular la complejidad ciclomática es mediante la siguiente fórmula:

**v(G)= a-n +2**
**Donde:**

**a:** número de arcos

**n:** número de nodos

**v:** vértices

**(G):** grafo

Aplicando la fórmula al ejemplo de prueba del camino básico se tiene que:

Para el caso a: **v(G)= 2-3+2= 1**

Para el caso b: **v(G)= 4-4+2= 2**

**Paso 4. Se determina un conjunto básico de caminos independientes.** De acuerdo

con los resultados anteriores para el caso **a,** el cálculo de la complejidad arrojó uno. Por lo tanto, el número de caminos independientes es uno; es decir, es uno el camino a probar.

1.1, 2 (el recorrido del nodo 1 al 2)

Los caminos independientes para el caso **b** son dos caminos a probar:

1.- 1, 2,0 (el recorrido del nodo 1 al 2 y del 2 al 0)

2.-1, 3,0 (el recorrido del nodo 1 al 3 y del 3 al 0)

**Paso 5. Se diseña un caso de prueba** para el caso **a** y dos para el caso **b**.

| Número de caso de prueba | Escenario | Condición | Resultado esperado |
|---|---|---|---|
| **a.-01** | Escenario 1 | X es menor que 20 | a = 1 |
| **b-.01** | Escenario 1 | X es menor que 20 | a = 1 |
| **b.-02** | Escenario 1 | X es mayor que 20 | a = -1 |

**Tabla de casos de pruebas para caso a y b**

**Técnica de prueba de caja blanca cobertura de sentencias.** Es el porcentaje de sentencias ejecutables que se han ejercido por una serie de pruebas (ISTQB, 2012). También es conocida como **cobertura de línea** o **de segmento**. Sus principales características son las siguientes:

- La cobertura de sentencia cubre sólo las **condiciones** cuyo **resultado** sea **verdadero**.
- A través de la cobertura de sentencias se **identifican** las **ejecutadas,** así como las que **no se ejecutan** debido a un **bloqueo** del código.
- En este proceso de la cobertura de sentencias cada línea de código tiene que ser **revisado** y **ejecutado**.

La fórmula de cobertura de sentencias se determina por deducción, si hay **n** sentencias por evaluar y en un caso de prueba se ejercitan **x** sentencias, el porcentaje de cobertura es de x/n, lo cual se observa en la siguiente fórmula:

$$Cobertura\ de\ sentencias = \frac{Número\ de\ sentencias\ ejercitadas}{Total\ de\ número\ de\ sentencias} * 100\%$$

**Fórmula de cobertura de sentencias. Basada en el concepto de cobertura de sentencia de ISTQB, 2012.**

La herramienta para utilizar esta técnica de caja blanca es Cantata ++. Es una categoría de herramientas de software usada para pruebas dinámicas, brinda casos **compilados** para ejecutarse con el código a probar, programado en C o C++.

A continuación, se ejemplifica la aplicación de la cobertura de sentencias manualmente. Sigue estos lineamientos:

- Cada línea se considera una sentencia.
- Una sentencia puede ser una sola línea o extenderse en varias.
- Una línea puede contener más de una sentencia, sólo una, o una parte.
- Las sentencias pueden contener otras sentencias dentro de ellas.

En el siguiente código se cuenta con dos sentencias de lectura, una de asignación y luego una instrucción condicional **SI** en dos líneas, pero la instrucción **SI** contiene otra sentencia (impresión) como parte de ella.

Paso 1. Se **enumera** cada línea del código y se considera a cada una como una sentencia:

**1** LEER X
**2** LEER Y
**3** z = X + 2 * Y
**4** SI z > 50 ENTONCES
**5** IMPRIME z grande
**6** FIN SI

Paso 2. Se **plantean** el conjunto de pruebas 1, **asignándoles** valores arbitrarios a las variables X y Y, con el fin considerar los tres posibles estados de la variable z, que sea menor que 50, que sea cercano a 50 e igual a 50.

Conjunto de prueba 1

Prueba 1_1: X = 2, Y = 3

Prueba 1_2: X = 0, Y = 25

Prueba 1_3: X = 47, Y = 1

Paso 3. Se **identifican** las sentencias cubiertas. Cada línea numerada es una sentencia:

- En la prueba 1_1, el valor de Z será de 12, por lo que cubrirá las sentencias en las líneas 1 a 4 y la 6 (cinco sentencias de seis).
- En la prueba 1_2, el valor de Z será 50, por lo que cubrirá exactamente las mismas sentencias de la prueba 1_1.
- En prueba 1_3, el valor de Z será de 49, así que de nuevo se cubrirán las mismas sentencias.

Se cubren cinco de las seis sentencias, por lo que la cobertura es del 83% (con tres pruebas **5/6= .83333*100= 83%**).

Paso 4. Falta la prueba para la sentencia 5, para ello, se **seleccionan** valores arbitrarios de tal forma que el valor de z sea mayor que 50:

Prueba 1_4: X = 20, Y = 25

Paso 5. El valor para z es de 550, y se **imprime** “z grande” con lo que se ejercitan las seis sentencias, por lo que ahora la cobertura es del 100 % (6/6=1*100= 100%).

**Técnica de prueba de caja blanca cobertura de decisión**. Es el porcentaje de los resultados de las decisiones que se han ejercido por una serie de pruebas, según el Glosario ISRQB. La cobertura de decisión 100% implica la cobertura de sentencia al mismo porcentaje, también conocida como cobertura de rama o de todos los límites. Sus características principales son:

- Cubre tanto las condiciones verdaderas y falsas como las sentencias.
- Una rama es el resultado de una decisión, es un camino de acuerdo con el resultado de una condicionante, por lo que la cobertura de rama simplemente mide los resultados de las decisiones. Se tiene una evaluación más profunda que la cobertura de sentencia.
- Una decisión es una instrucción condicional SI, una declaración de control de bucle (por ejemplo, DO-WHILE o REPEAT-UNTIL), o una instrucción **CASE**, donde hay dos

o más resultados a partir de la declaración. Con una instrucción **SI**, la salida puede ser verdadera o falsa, según el valor de la condición lógica que viene después del **SI**.

La fórmula de cobertura de decisión se determina por deducción. Si hay dos posibles caminos o ramas por evaluar, y en un caso de prueba se ejercita una rama, el porcentaje de cobertura de decisión es de 1/2, lo cual se observa en la siguiente fórmula.

$$Cobertura\ de\ decisión = \frac{Número\ de\ salidas\ de\ decisiones\ ejercitadas}{Número\ total\ de\ decisiones\ de\ salida} * 100\%$$

Formula de cobertura de decisión. Basada en el concepto de cobertura de decisión de ISTQB, 2012.

Ejemplo de aplicación de la cobertura de decisión:
La cobertura de decisión requiere que en las ramas se contemple verdadera y falsa. Se requiere lograr una cobertura de decisión del 100 %.
En el siguiente código se cuenta con dos sentencias de lectura, una de asignación y otra de instrucción condicional SI, en dos líneas, y contiene una sentencia de impresión como parte de ella.
Paso 1. Se enumera cada línea del código y se considera cada una como una sentencia:

**1** LEER A
**2** LEER B
**3** C = A – (2 * B)
**4** SI C < 0 ENTONCES
**5** IMPRIME”C negativo"
**6** FIN SI

Paso 2. Se plantean el conjunto de pruebas 1, asignándoles valores arbitrarios a las variables A y B, con el fin de considerar que el estado de la variable C sea menor que cero.

Conjunto de pruebas 1

1_1: A = 20, B = 15

Paso 3. Se identifica si la condición es verdadera o falsa:

- En la prueba 1_1, el valor de C será de -10. La condición C< 0 es verdadera y se imprimirá “C negativo”. El resultado de la sentencia de decisión es verdadero.

Cubre las sentencias de las líneas uno a la seis.

Paso 4. Probar el resultado falso de la sentencia de decisión. Para ello, se analizará el diagrama de flujo del fragmento del código para comprobar que, efectivamente, el caso de prueba con los valores de entrada determina que la condición sea verdadera.

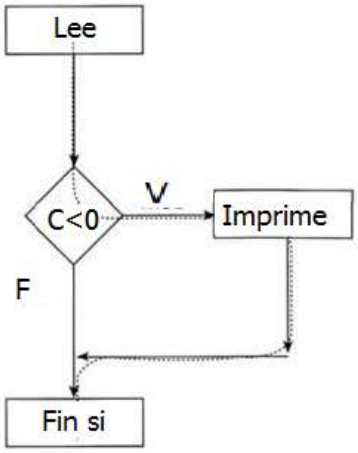



Diagrama de flujo del código de prueba

La línea de puntos encierra la prueba 1_1. Claramente muestra que no se ha probado que la condición sea falsa.

La prueba 1_1 cubre al 100% de cobertura de sentencia, así como el resultado de la decisión –verdadero- al 50% de la cobertura de decisión.

Según la fórmula de la cobertura de decisión:

Cobertura de decisión = ½ *100= 50%

Paso 5. Se agrega otra prueba al conjunto de pruebas 1.

Conjunto de pruebas 1:

Prueba 1_1: A = 20, B = 15

Prueba 1_2: A = 10, B = 2 (se le dan valores arbitrarios para determinar que la condición sea falsa)

El valor de C ahora es seis, por lo tanto, no es menor que cero y el resultado de la condición es falsa.

Y se obtiene según la fórmula de cobertura de decisión = 2/2*100= 100% de cobertura para la prueba de decisión.

**Técnica de prueba de caja blanca cobertura de condición**. Es el **porcentaje** de los **resultados** de condiciones que se han **ejercido** por una serie de pruebas (ISTQB, 2012). La cobertura de condición al 100% requiere que cada estado individual de cada sentencia de decisión se ha **probado** como **verdadero** y **falso**. También se conoce como **cobertura** de **predicado.** Esta prueba **evalúa** cada una de las **expresiones booleanas** en el **código** a **probar**.

El siguiente ejemplo tiene tres expresiones booleanas A, B y C, las cuales se deben probar para tener un 100% de cobertura.

```
SI ((A || B) && C)
 Entonces
        {
         << sentencias >>
        }
 Si no
        {
          << sentencias >>
        }
```

**Procedimiento**

Con el fin de **garantizar** los **criterios** de **cobertura** de condición para el ejemplo anterior, A, B y C deben ser evaluados al menos una vez.

La condición indica que si A o B y C son verdaderos, la condición es verdadera.

**En primer lugar,** se **asignan** valores de verdadero, falso y **no evaluado** a las variables A, B y C, para que sean **evaluados** al menos una vez.

El conjunto de pruebas cubren el 100% de cobertura de condición al evaluarse, al menos, una vez cada expresión booleana.

| Número de caso de prueba | A | B | C | Condición |
|---|---|---|---|---|
| **1_1** | Verdadero | No evaluado | Falso | - |
| **1_2** | Falso | Verdadero | Verdadero | Verdadero |
| **1_3** | No evaluado | Falso | Verdadero | - |
| **1_4** | Falso | Falso | No evaluado | - |

**Casos de pruebas para la cobertura de decisión.**

**Cobertura decisión/condición**. Es un tipo de prueba mixta que trata de cubrir el 100% de las decisiones/condiciones del código, para que se ejecuten las decisiones/condiciones posibles al menos una vez.

Si se toma como base el ejemplo de la cobertura de condición, se observa que se cubre el 100%. Para la decisión implícita **SI**, se cubre sólo el 50% de cobertura con la prueba 1_2, donde la condición es verdadera.

```
SI ((A || B) && C)
 Entonces
        {
         << sentencias >>
        }
Si no
        {
          << sentencias >>
        }
```

Para probar que la condición sea falsa, se agrega un caso al conjunto de pruebas.

| Número de caso de prueba | A | B | C | Condición |
|---|---|---|---|---|
| **1_1** | Verdadero | No evaluado | Falso | - |
| **1_2** | Falso | Verdadero | Verdadero | Verdadera |
| **1_3** | No evaluado | Falso | Verdadero | - |
| **1_4** | Falso | Falso | No evaluado | - |
| **1_5** | Falso | Verdadero | Falso | Falsa |

Con lo que se Logra un 100% de cobertura de decisión.

**Técnica de prueba blanca cobertura de condición múltiple.** También es conocida como cobertura combinada de condición. Aquí, para cada decisión se deben **evaluar** todas las **combinaciones** de las expresiones booleanas.

El siguiente ejemplo presenta una decisión **SI** con dos expresiones booleanas, donde se deben **evaluar** todas las combinaciones posibles (a diferencia de la cobertura de decisión en el que, al menos una vez, cada expresión booleana debe ser probada):

```
SI(A | | B)
 Entonces
  Imprimir C
FIN SI
```

Hay dos expresiones booleanas, A y B, por lo que el criterio para la cobertura de condición múltiple es:

| Número de caso de prueba | A | B | Condición |
|---|---|---|---|
| **1_1** | Verdadero | Verdadero | Verdadero |
| **1_2** | Verdadero | Falso | Verdadero |
| **1_3** | Falso | Verdadero | Verdadero |
| **1_4** | Falso | Falso | Falso |

Como se puede ver, hay **cuatro** casos de prueba para **dos** condiciones. El número de combinaciones de los dos estados de la variable A por los dos estados de la variable B, resulta en los cuatro casos de pruebas. Si existiera una tercera condición, se plantearían ocho casos de prueba.

Si hay **n** condiciones habrá 2 ^ n casos de pruebas.

Normalmente, se alcanza un porcentaje de cobertura del 80 al 90%. Es aceptable. Se recomienda a los encargados de las pruebas que usen la combinación de las técnicas para aumentar el porcentaje de cobertura.

Las diferentes técnicas de pruebas de caja blanca trabajan sobre el código del componente, asegurándose de que al menos una vez el código, condiciones y decisiones, han sido ejecutados. Los resultados, aún si hubiera incongruencias con los esperados, se verán reflejados en el debido reporte para que las personas correspondientes (los programadores) hagan las mejoras pertinentes.

Continúa ahora probar la interfaz del componte o sistema, qué tan funcional es. Para esto, no se tiene que conocer cómo está programado el componente, pero sí que tipos de datos requieren las entradas, y qué es lo que se espera que haga después. Esto se abordará en el siguiente subtema.

### 2.2.2. Pruebas de caja negra

Una vez que las pruebas de caja blanca se han aplicado para **evaluar** los **procedimientos**, las **rutas lógicas** del software y la colaboración entre los componentes con casos de prueba, se ejecutan conjuntos específicos de condiciones, bucles o ambos.

Es necesaria la aplicación de las pruebas de caja negra como complemento de las de caja blanca, para conocer si un componente realiza la función específica para la que se programó.

**Las pruebas de caja negra. Evalúan elementos funcionales y no funcionales** sin hacer referencia a la estructura interna del componente o sistema (ISTQB, 2012).

**Técnica de diseño de pruebas de caja negra**. Es el procedimiento para **generar** casos de prueba seleccionados con base en un **análisis** de **requerimientos**, ya sea funcional o no funcional, de un componente o sistema sin hacer referencia a su estructura interna (ISTQB, 2012).

Las pruebas de caja negra se aplican a todos los **niveles** de **prueba**: de unidad, integración, implantación, aceptación y regresión (niveles que se estudiarán en el tema 2.3. niveles de pruebas).

Estas pruebas proporcionan un conjunto de condiciones de entrada que ejercitan todos los requisitos funcionales de un componente del sistema, y descubre errores tales como:

- **Funciones incorrectas** o ausentes.
- **Errores de interfaz**. Por ejemplo, que un botón no realice la función para la que fue programado.
- **Errores en estructuras de datos o accesos a base de datos externas**. Por ejemplo, que no esté definida la estructura de datos o la conexión a base de datos de otros sistemas.
- **Errores de rendimiento.** Por ejemplo, que el sistema consuma más recursos de los que ofrece el equipo computacional.
- **Errores de inicialización y finalización.** Por ejemplo, que no se carguen las variables iniciales del sistema, o que no guarde los cambios en el momento de cerrar el componente o sistema.

Para probar una página web mediante la técnica de caja negra, el procedimiento es el siguiente:

Contexto del ejemplo. La persona que **aplica** las pruebas **desconoce** cómo está programada la página y **realiza** las siguientes acciones:

- Carga la página en un navegador en particular.
- Hace las entradas al hipertexto dando clic en botones, ligas, imágenes, etcétera. Verifica las salidas con el resultado esperado.
- Hace el reporte de los resultados.

Las pruebas de la técnica de caja negra se clasifican a continuación:
**Técnica de caja negra equivalencia de partición.** Consiste en dividir los valores de entrada en particiones válidas y no válidas. Selecciona los valores representativos de cada partición como datos de prueba.

**Ejemplo de aplicación de equivalencia de partición.** Se requiere probar un módulo de software que tiene la finalidad de aceptar el nombre de un producto (capturado) de una

tienda de abarrotes. Existe una lista de los diferentes pesos del producto, especificados en onzas. Las especificaciones de cada producto se conforman de la siguiente manera:

- El nombre del elemento debe conformarse de 2 a 15 caracteres alfabéticos.
- El dato del peso deberá corresponder a un valor en el intervalo de 1 a 48.
- Identifica sólo números enteros.
- Los datos del peso se deben introducir en orden ascendente (tamaños pequeños primero).
- Se puede introducir un máximo de cinco tamaños para cada producto.
- El nombre del elemento se debe introducir en primer lugar, seguido de una coma, y de una lista de tamaños.
- Una coma se utiliza para separar cada tamaño, los espacios se ignoran en la entrada del dato.

Las clases de equivalencia son las siguientes:

1. Nombre del artículo es alfabético (válido)
2. Nombre del artículo no es alfabético (no válido)
3. Nombre del elemento es menor de 2 caracteres de longitud (no válido)
4. Nombre del artículo es de 2 a 15 caracteres de longitud (válido)
5. Nombre del artículo es superior a 15 caracteres de longitud (no válido)
6. Valor de tamaño es inferior a 1 (no válido)
7. Tamaño valor está en el rango de 1 a 48 (válida)
8. Valor de tamaño es mayor que 48 (no válido)
9. Valor de tamaño es un número entero (válido)
10. Tamaño valor es un decimal (no válido)
11. El tamaño es el valor numérico (válido)
12. Valor del dato del peso del producto incluye caracteres no numéricos (no válido)
13. Los valores del dato del peso del producto introducidos en orden ascendente (válido)
14. Los valores del dato del peso del producto introducidos en orden no ascendente (no válido)
15. No hay valores del peso (no válido)
16. De uno a cinco valores del peso entraron (válido)
17. Más de cinco datos del peso entraron (no válido)
18. Nombre del artículo es el primero (válido)

19. Nombre del artículo no es la primera (no válido)
20. Una sola coma separa cada entrada en la lista (válido)
21. Una coma no separa dos o más entradas de la lista (no válido)
22. La entrada no contiene espacios en blanco (indiferente)
23. La entrada contiene espacios en blanco (indiferente)

Una vez enlistadas las clases de equivalencia, se agrupan en una tabla como la siguiente:

| Número de caso de prueba | Datos de prueba | Salida esperada | Clases cubiertas |
|---|---|---|---|
| 1 | xy,1 | T | 1,4,7,9,11,13,16,18,20,22 |
| 2 | AbcDefghijklmno,1,2,3 ,4,48 | T | 1,4,7,9,11,13,16,18,20,23 |
| 3 | a2x,1 | F | 2 |
| 4 | A,1 | F | 3 |
| 5 | Abcdefghijklmnop | F | 5 |
| 6 | Xy,0 | F | 6 |
| 7 | XY,49 | F | 8 |
| 8 | Xy,2.5 | F | 10 |
| 9 | xy,2,1,3,4,5 | F | 14 |
| 10 | Xy | F | 15 |
| 11 | XY,1,2,3,4,5,6 | F | 17 |
| 12 | 1,Xy,2,3,4,5 | F | 19 |
| 13 | XY2,3,4,5,6 | F | 21 |
| 14 | AB,2#7 | F | 12 |

**Tabla de casos de prueba de caja negra de la tienda de abarrotes basada en las clases de equivalencia detectadas.**

La tabla anterior representa los casos de pruebas generados a partir de las clases de equivalencias, en donde la columna de la izquierda contiene el número de caso de prueba; la columna datos de pruebas son las entradas de datos; la columna salida de datos contiene el resultado de la entrada de datos a la prueba que puede ser verdadero o falso; y la columna de clases cubiertas indica qué clases son cubiertas por el número de caso de prueba. El formato de la tabla es invariable.

**Técnica de caja negra análisis del valor límite**. Es una técnica de diseño de casos de prueba para probar el valor límite entre particiones (válidas e inválidas). Un *valor límite* es un valor de entrada o salida en la frontera de una partición de equivalencia, incluye los valores mínimos y máximos en el interior y fuera de las fronteras. El análisis del valor límite parte de las pruebas de tensión.

Por ejemplo, un caso de prueba en el que el cuadro de texto **dirección** permite un máximo de 500 caracteres. Se hace imposible probar casos para cada caracter, así que se elige la prueba del valor límite.

| Partición inválida | Partición válida | Partición inválida |
|---|---|---|
| Menos de 8 caracteres | De 8 a 12 caracteres | Más de 12 caracteres |

**Tabla de particiones válidas e inválidas.**

A continuación, se enlistan los casos de prueba para el valor de particiones válidas, no válidas o inválidas.

**Válidas**

• Casos de prueba 2: probar la contraseña de longitud exactamente 8
• Casos de prueba 3: probar la contraseña de longitud entre 9 y 11
• Casos de prueba 4: probar la contraseña de longitud exactamente 12

**Inválidas**

• Casos de Prueba 5: probar la contraseña de longitud de más de 12
• Casos de Prueba 1: probar longitud de la contraseña de menos de 8

**Técnica de caja negra gráfico causa-efecto.** Muestra gráficamente la relación entre un resultado dado y todas las cuestiones que lo manipulan. Se conoce también como diagrama de Ishikawa o espina de pescado, debido a su estructura, inventado por Koirala, Sheikh y Ishikawa (Koirala y Sheikh, 2008).

Adaptado para pruebas de software, se aplica al comportamiento externo del sistema. Se apoya en la selección de casos de prueba que se relacionan lógicamente: causas (insumos) y efectos (resultados) para producir casos.

Una *causa* es una condición de entrada separada que se obtiene sobre un cambio interno en el sistema. Un efecto representa una condición de salida, un sistema de transformación, o un estado resultante de una combinación de causas (Koirala y Sheikh, 2008).

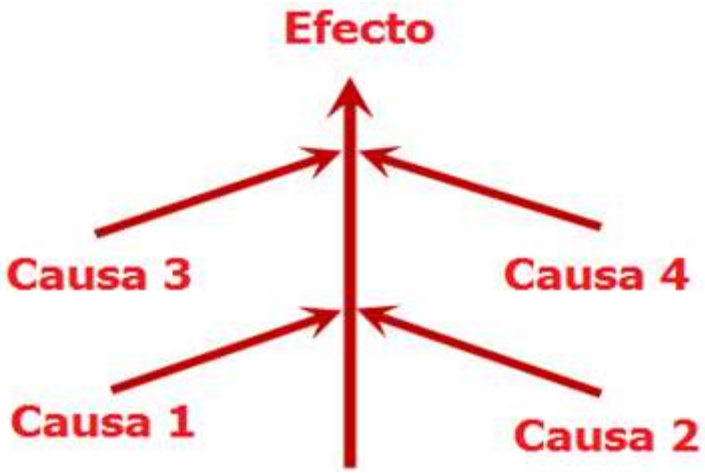



**Diagrama causa-efecto. Basado en: Tutorialspoint, 2014.**

Ejemplo de aplicación del diagrama causa-efecto para pruebas de software.

Se realizan los siguientes pasos:

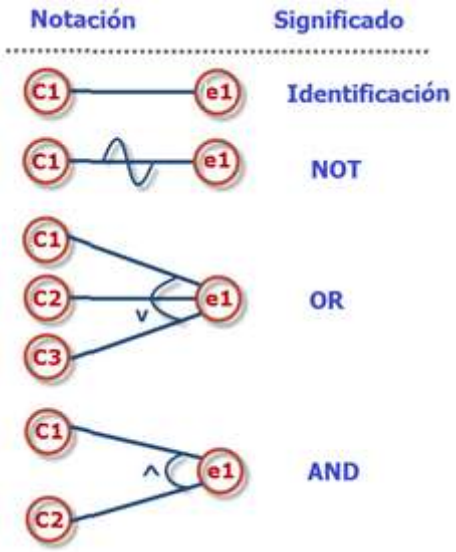
1. Reconocer y describir las condiciones de entrada (causas) y las acciones (efecto)
2. Construir un gráfico de causa-efecto
3. Convertir el gráfico de causa-efecto en una tabla de decisión
4. Convertir reglas de la tabla de decisión para probar los casos. Cada columna de la tabla de decisión representa un caso de prueba

Antes de empezar, hay que identificar la notación y el significado del gráfico causa y efecto.

La identificación considera que el efecto 1 está relacionado con la causa 1.
La negación (NOT) considera que el efecto 1 no está relacionado con la causa 1.
La alternativa (OR) considera que el efecto 1 puede tener varias causas.

La conjunción (AND, considera que el efecto tiene dos causas.
Lo anterior es la base para la construcción del diagrama causa-efecto.

Según la imagen siguiente:




**Significado de la notación del gráfico causa-efecto. Basado en Kong, 2009.**

Se asume que cada nodo puede tener el valor de cero o uno, donde 0 representa el **sin estado** y 1 muestra el **estado actual**.

Los **estados** de la **función** de **identidad** se observan cuando c1 = 1 e1 = 1 o c0 = 0 y e0 = 0.

La función NOT establece que si c1 = 1, e1 = 0 y viceversa. La función OR establece que, si c1 o c2 o c3 = 1, e1 = 1 otra cosa e1 = 0. La función AND afirma que, si ambos c1 y c2 = 1, e1 = 1, de lo contrario e1 = 0. A las funciones AND y OR se les permite tener cualquier número de entradas.

Los casos de prueba pueden ser diseñados, por ejemplo, para determinar si una figura es un triángulo con base en sus medidas de las siguientes maneras:

1. Reconocer y describir las condiciones de entrada (causas) y acciones (efecto).

Las causas asignadas por la letra “c” son las siguientes:

c1: Lado “x” es menor que la suma de la “y” y “z"
c2: Lado “y” es menor que la suma de “x” y “z"
c3: Lado “z” es inferior a la suma de “x” e “y"
c4: Lado “x” es igual a otro “y"

c5: Lado “x” es igual a otro “z"
c6: Lado “y” es igual a otro “z"
Los efectos con la letra “e” son los siguientes:

e1: no es un triángulo
e2: triángulo escaleno
e3: triángulo isósceles
e4: triángulo equilátero
e5: imposible

Después de reconocer y describir las condiciones de entrada (causas) y acciones (efecto) se construye el gráfico de causa-efecto:

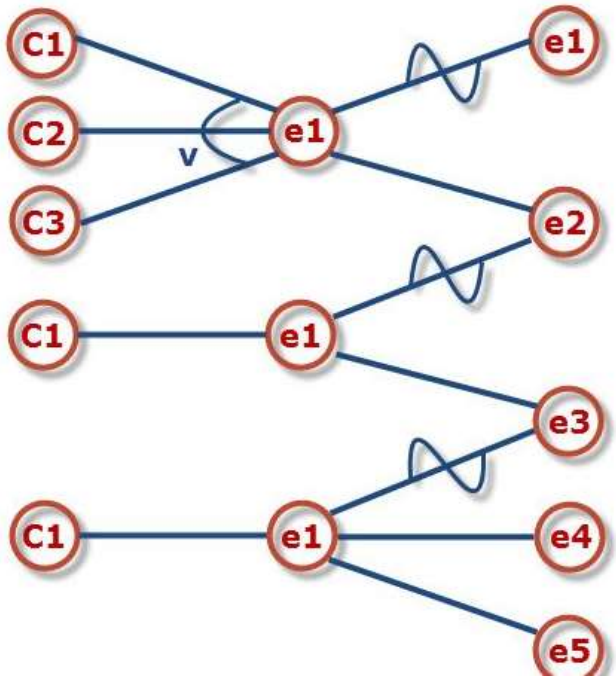


**Gráfico causa-efecto.**

Se convierte el gráfico de causa-efecto en una tabla de decisión, donde la **R** representa las **reglas** que se cumplen.

| Condiciones | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 | R10 | R11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C1: X < Y+Z? | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| C2: X < Y+Z? | X | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| C3: X < Y+Z? | X | X | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| C3: X=Y? | X | X | X | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| C4: X=Y? | X | X | X | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| C5: X=Y? | X | X | X | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| C6: X=Y? | X | X | X | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| e1: no es un triángulo | 1 | 1 | 1 | | | | | | | | |
| e2: es un triángulo escaleno | | | | | | | | | | | 1 |
| e3: es un triángulo isósceles | | | | | | | 1 | | 1 | 1 | |
| e4: es un triángulo equilátero | | | | 1 | | | | | | | |
| e5: imposible | | | | | 1 | 1 | | 1 | | | |

**Tabla de decisión del gráfico causa-efecto**

Se obtienen 11 casos de pruebas derivadas de las reglas enlistadas en la tabla anterior, y se integran en una tabla como la que se expone a continuación:

| Casos de pruebas | X | Y | Z | Resultados esperados |
|---|---|---|---|---|
| 1 | 4 | 1 | 2 | No es un triángulo |
| 2 | 1 | 4 | 2 | No es un triángulo |
| 3 | 1 | 2 | 4 | No es un triángulo |
| 4 | 5 | 5 | 5 | Equilátero |
| 5 | ? | ? | ? | Imposible |
| 6 | ? | ? | ? | Imposible |
| 7 | 2 | 2 | 3 | Imposible |
| 8 | ? | ? | ? | Imposible |
| 9 | 2 | 3 | 2 | Isósceles |
| 10 | 3 | 2 | 2 | Isósceles |
| 11 | 3 | 4 | 5 | Escaleno |

Tabla de casos de prueba

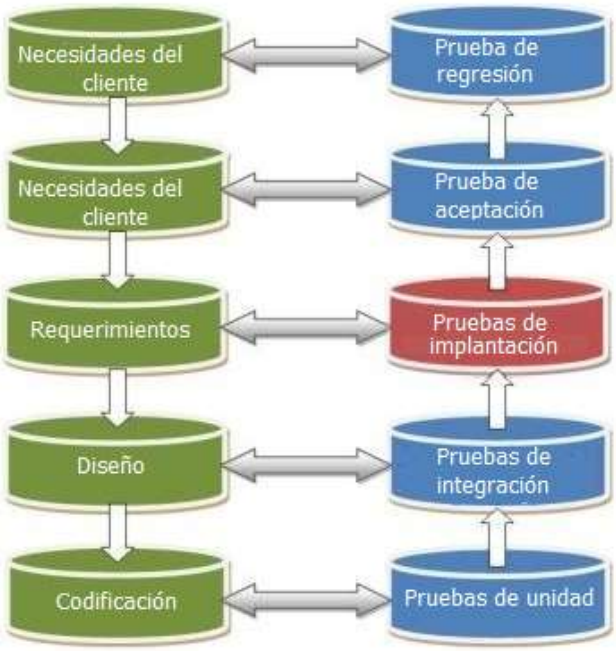
En este tema se expuso la **clasificación** de las pruebas estáticas y dinámicas. Se determinaron los **tipos** de pruebas que cada técnica engloba, dando un **panorama** general para el **diseño** de casos como parte de un **plan** dentro de la **administración** de procesos de pruebas.

Ahora es necesario identificar los niveles de pruebas a los cuales se deben **aplicar** los casos ya generados.

## 2.3. Niveles de pruebas

Los niveles de pruebas son un grupo de actividades se organizan y gestionan conjuntamente. Un nivel de prueba está relacionado con las responsabilidades en un proyecto (ISTQB, 2012).

En el presente tema se debe determinar la aplicación de las técnicas de prueba, de acuerdo con ciertos niveles que responden a ciertas necesidades, como se puede observar en el siguiente esquema:




**Niveles de pruebas. Basado en: UnADM, 2014.**

Como primer nivel, se observan las **pruebas unitarias** o de **unidad** que verifican la **codificación**; las de **integración,** el **diseño** del software; las **de implantación,** que se cubran los **requerimientos** especificados en la **planeación** del desarrollo del sistema.

Las **pruebas de aceptación** están enfocadas en el **cliente**, es quien da el **visto bueno** al sistema de software, en caso de que se detecte la necesidad de realizar **mejoras** mediante las pruebas anteriores y, finalmente, se **comprueba** si efectivamente se atendieron las fallas a través de las **pruebas de regresión**, con base en las necesidades del cliente, por supuesto.

En los subtemas siguientes se expondrá a detalle la explicación de las características de los tipos de niveles de pruebas.

### 2.3.1. Pruebas unitarias

Son las pruebas de los componentes individuales del software (ISTQB, 2012). Se realizan durante el proceso de desarrollo del software. La prueba, en este nivel, consiste en comprobar que el componente esté correcto y funcione como se espera. Normalmente la lleva a cabo el programador.

**Método:** esta prueba implica la técnica de **caja blanca**, que consiste en la **verificación** de que el código este correcto.

**Tareas:** las actividades que se realizan en esta prueba son las siguientes.

1. Plan de pruebas unitarias
    - Preparar
    - Comentar
    - Revisión
    - Validación de la especificación de los requisitos

2. Diseño de los casos prueba
    - Revisión
    - Validación

3. Prueba unitarias
   - Aplicación

4. Resultados

Es la primera prueba que se aplica, según la imagen de niveles de prueba.

El siguiente ejemplo es la verificación de código en Java, con la herramienta JUnit, que es un marco sencillo para escribir pruebas unitarias repetitivas.

1. Se **elabora** el plan de pruebas con los elementos propuestos en el subtema 2.1.1. Plan de pruebas
2. Se **diseñan los** casos de pruebas unitarias.
3. Aplicación
4. Resultados

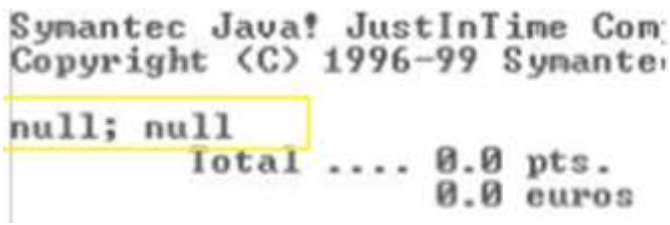
Los casos de pruebas **verifican** que los constructores de la clase **Factura** efectivamente funcionen cuando se crea una **instancia**.

**Caso de prueba 211**

| | |
|---|---|
| Factura f=new Factura (null, null); | Se crea un objeto de la clase factura, con parámetros nulos |
| f.show(); | Muestra el objeto f |

**Código del caso de prueba 211.**

El resultado es el esperado, con los parámetros **null**



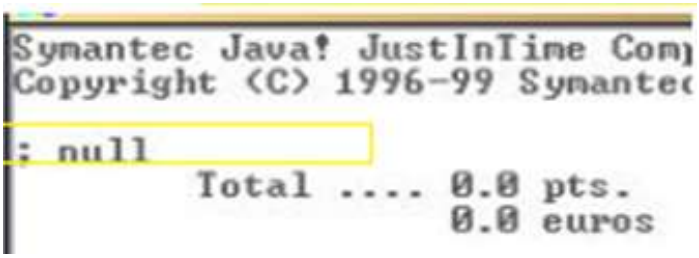

Resultado del caso de prueba 211

Caso de prueba 212

| | |
|---|---|
| Factura f=new Factura (new String(), null); | Se crea un objeto de la clase factura con un parámetro de cadena de caracter y otro nulo |
| f.show(); | Muestra el objeto |

Código del caso de prueba 212

El resultado es el esperado, con un parámetro de cadena de carácter y otro nulo.




**Resultado del caso de prueba 212.**

La verificación que el código funcione según lo que se espera es primordial, así como la detección de errores de sintaxis y de lógica, de tal forma que el componente sea totalmente funcional.

### 2.3.2. Pruebas de integración

Las pruebas de integración se utilizan para **exponer** los **defectos** en las **interfaces** y la **interacción** entre los componentes integrados (ISTQB, 2012).

Se inician después de que los módulos de software individuales se integran como un grupo. Como ya es de tu conocimiento, un proyecto de software consta de varios módulos y el desarrollo de cada módulo es asignado a diversos miembros del equipo, de tal manera que las pruebas de integración se enfocan en la **comprobación** de la relación entre los módulos después de integrarlos.

La mayoría de los problemas se observan en este tipo de pruebas pues, al integrarse los componentes desarrollados en forma individual, se deben sincronizar con el resto que conforman todo el sistema.

**Método:** esta prueba implica la técnica de caja blanca, que consiste en la verificación de que el código sea correcto.

**Tareas**

**1. Plan de pruebas de integración**

- Preparar
- Comentar
- Revisión
- Validar la especificación de los requisitos

**2. Diseño de los casos prueba**

- Revisión
- Validación

**3. Prueba de integración**

- Aplicación

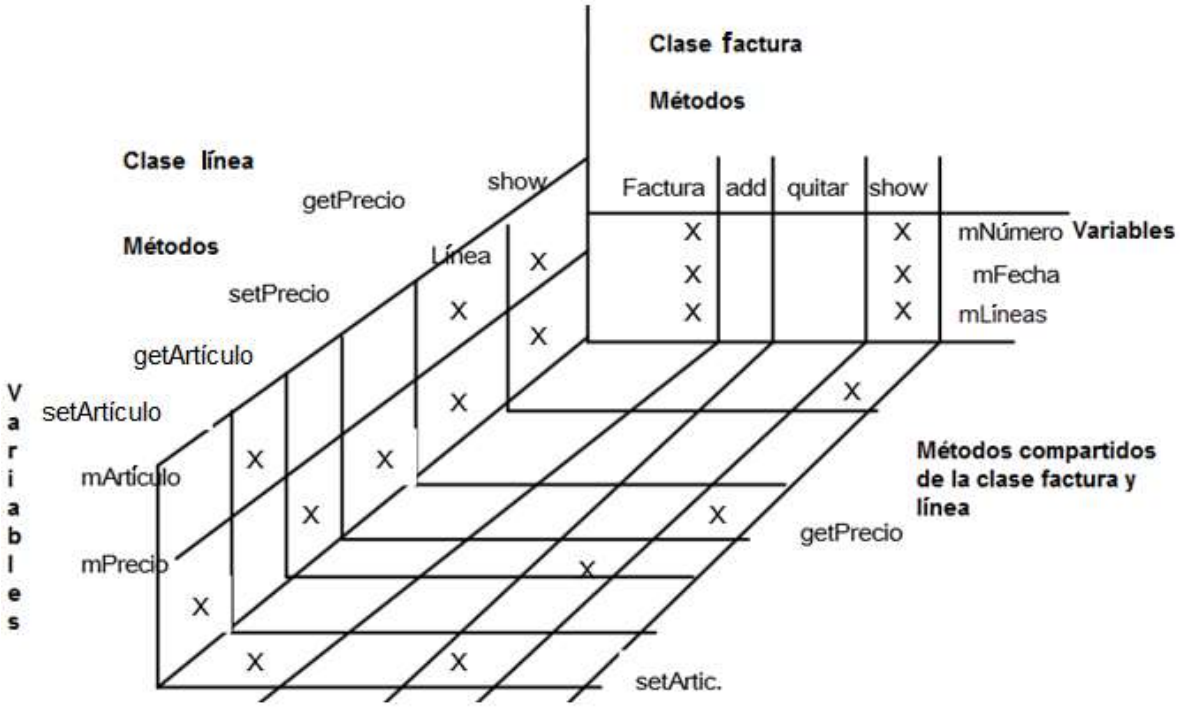
**4. Resultados**

Es la segunda prueba que se aplica **después** de que cada componente ha sido evaluado.

En el siguiente ejemplo, cada componente ya ha superado las pruebas unitarias. Observa cómo realizar la prueba de integración entre los dos componentes.

Se tienen dos clases con sus respectivos métodos y variables (atributos), La clase *factura* y la clase *línea*, estas dos se integran. La primera proporciona el formato de una factura, mientras que la segunda agrega líneas, según los artículos que se agreguen a la factura.

1. Se **elabora** el plan de pruebas con los elementos propuestos en el subtema 2.1.1. Plan de pruebas
2. Se **diseñan** los casos de pruebas integración.
3. Aplicación
4. Resultados




**Matriz de integración la clase factura y línea.**

**Se realizan las siguientes acciones:**

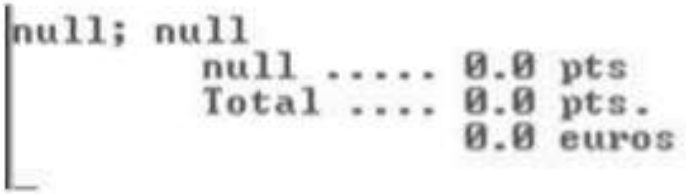
Primero, dar valores de prueba al atributo líneas, añadiendo objetos de clase Línea mediante el método add (Línea):

Caso de prueba I-211-1

| | |
|---|---|
| Factura f=new Factura(null, null); | Se crea una instancia de la clase Factura |
| Línea l=new Línea(null, 0); | Se crea una instancia de la clase Línea |
| f.add(l); | Agrega el objeto l |
| f.show(); | Muestra el objeto l |

**Código del caso de prueba I-211-1**

El resultado es el esperado, se crea el objeto factura, se crea el objeto línea y muestra la línea agregada a la factura.



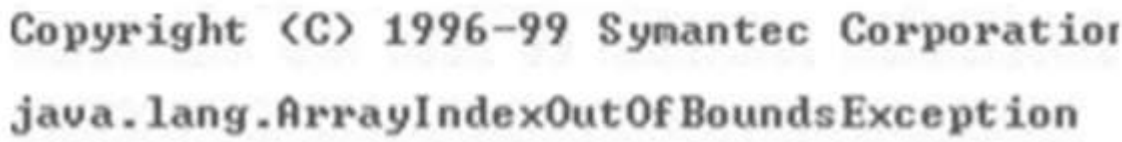

Resultado del caso de prueba I-211-1

Caso de prueba I-211-2

| | |
|---|---|
| f.add(l); | Se agrega objeto línea |
| f.add(12); | Número de elementos 12 del arreglo |
| f.show(); | Muestra la línea |
| Línea líneas[]={ | Se crea un arreglo |
| new Línea("Agua", 25), new Línea("Leche", 85), | Se agregan líneas con productos |
| new Línea("Pan", 50), new Línea("Mistol", 240), | |
| new Línea("Gel", 195), new Línea("Tomates", 70), | |
| new Línea("Agua 2", 25), new Línea("Leche 2", 85), | |
| new Línea("Pan 2", 50), new Línea("Mistol 2", 240), | |
| new Línea("Gel 2", 195), new Línea("Tomates 2", 70) }; | |
| Factura f=new Factura(null, null); | Se crea el objeto f de la clase factura |
| try { for (int i=0; i<lineas.length; i++) | Se establece un controlador de excepciones, mientras la variable de control sea menor que el número del tamaño del arreglo que agregue las líneas y se muestren |
| f.add(lineas[i]); | Si se quieren agregar artículos después de llegar al límite que genere una excepción |
| f.show(); } | |
| catch (Exception e) {System.out.println(e);} | |
| | |

**Código del caso de prueba I-211-2.**

El resultado es el esperado. Se genera una excepción porque la condición establece que i<12, y se pretende agregar 12 productos, por lo que al llegar a i=12 y evaluar la condición 12<12, no se cumple.




Resultado del caso de prueba I-211-2

De esta forma se aplican las pruebas de **integración**. Debe aplicarse a todo el sistema de software, y **verificar** que la comunicación entre los componentes sea la que se estableció en el diseño del software.

### 2.3.3. Pruebas de implantación

Se aplican a un sistema integrado para verificar que cumple con los requisitos no funcionales (ISTQB, 2012), en un entorno de preproducción de calidad, instalado ya en su ambiente.

Ejemplos de requisitos no funcionales son: escalabilidad, rendimiento, confiabilidad, carga, que responden a cuestiones de calidad del sistema.

| Tipo de prueba | Parámetros | Ejemplo de criterio de entrada | Ejemplo de criterios de salida |
|---|---|---|---|
| Escalabilidad | Límites máximos | El producto debe escalar hasta un millón de registros o mil usuarios | El producto debe escalar hasta 10 millones de registros o 5000 usuarios |
| Rendimiento | Respuesta en tiempo Rendimiento Latencia | La consulta para mil registros debería tener una respuesta en menos de tres segundos | La consulta para 10 mil registros deberían tener una respuesta en menos de tres segundos |
| Confiabilidad | Fallas por interacción Fallas por duración dela prueba | Las fallas deben ser menor que 2% cuando las consultas estén corriendo en mil registros por 24 horas | Las fallas deben ser menor que .1% cuando las consultas estén corriendo en mil registros por 48 horas |
| Carga | Sistema estresado más allá de los limites | El producto debe de soportar a 25 usuarios entrando simultáneamente por cinco horas en una | El producto debe de soportar a 100 usuarios entrando simultáneamente por cinco horas en una configuración de 100 usuarios |

| | | | |
|---|---|---|---|
| | | configuración de 20 usuarios | |

**Criterios de entradas y salidas típicos de pruebas no funcionales. Fuente: Srinivasan, 2006.**

Se definen los umbrales de las métricas cualitativas y cuantitativas. Si el sistema de software no los superara, no se pasa a la producción, aunque funcionalmente esté correcto. El resultado de las pruebas de aceptación puede ser **base** y **retroalimentación** para las de implantación.

**Método:** esta prueba implica la técnica de caja negra que consiste en **verificar** que lo que se espera que realice el sistema de software, lo haga efectivamente.

**Tareas**

**1. Plan de pruebas implantación**

- Preparar
- Comentar
- Revisión
- Validación de la especificación de los requisitos

**2. Diseño de los casos prueba**

- Revisión
- Validación

**3. Prueba de implantación**

- Aplicación

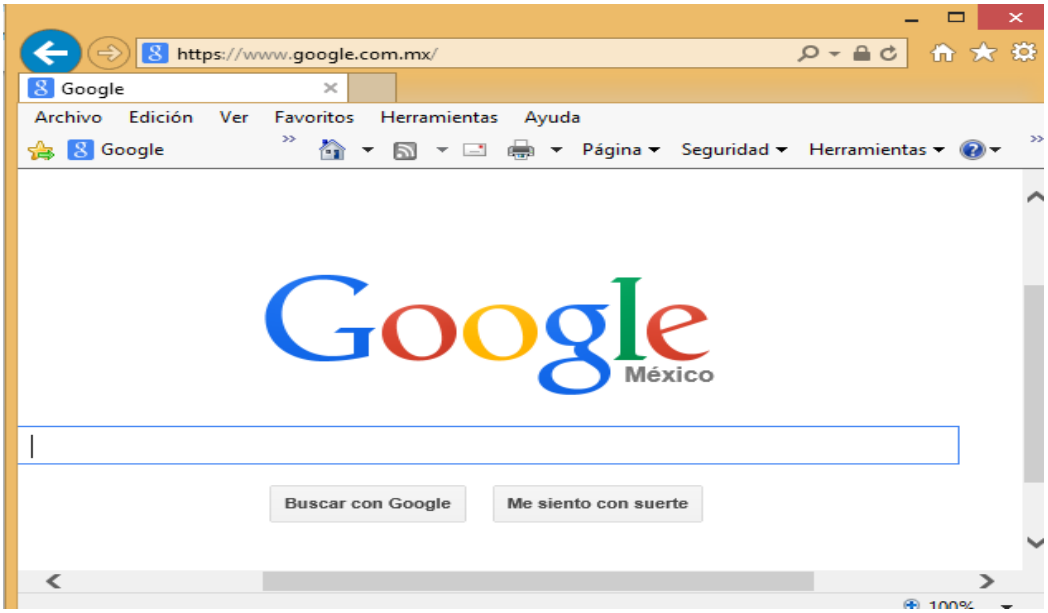
**4. Resultados**

Es la tercera prueba que se aplica después de las de integración al sistema.

El ejemplo que se describe a continuación es de una prueba de carga (stress) a Internet Explorer de Microsoft, navegador de Internet que ha estado en producción desde 1995. De acuerdo con el W3 (numerónimo que se refiere a World Wide Web), Internet Explorer

es el navegador más utilizado, con puntuaciones globales de uso que indica que en junio de 2012 compone el 72,8% de los navegadores de Internet que se utilizan.

**1. Se elabora el plan de pruebas** con los elementos propuestos en el tema 2.1.1. Plan de pruebas.




**Internet Explorer 10**

**2. Diseño de casos de pruebas implantación-carga.** Se busca evaluar la capacidad de Internet Explorer para el manejo de no respuesta.

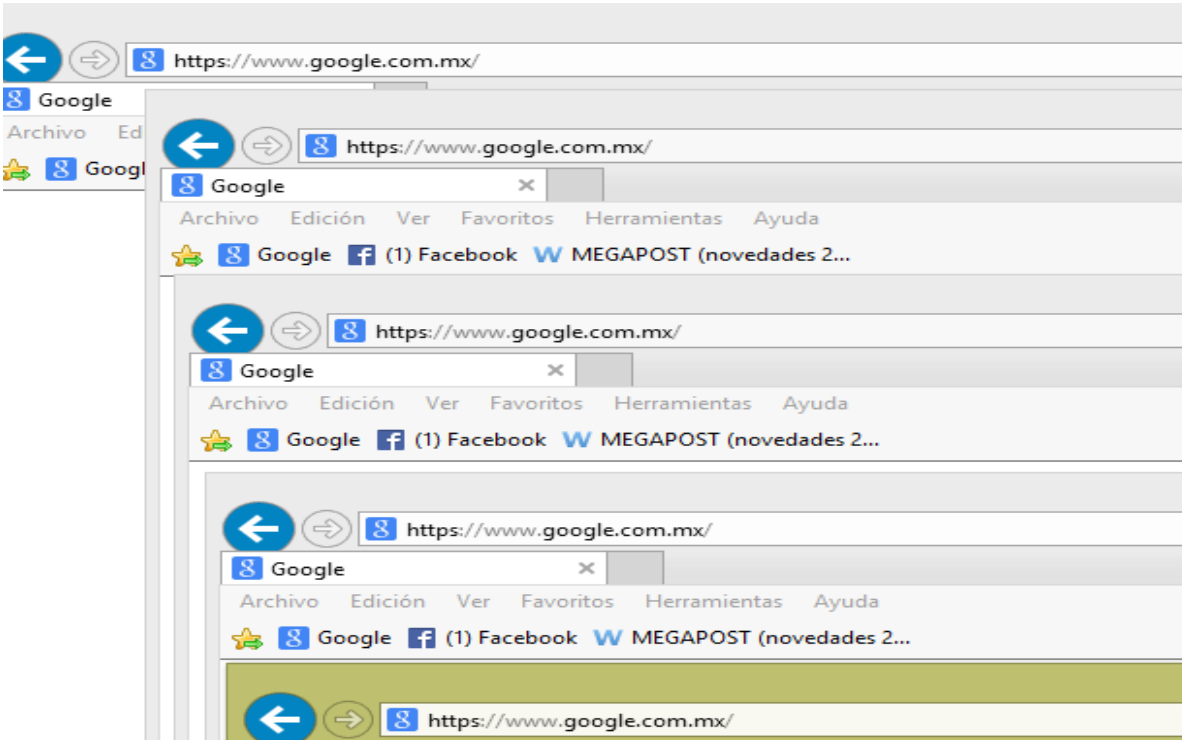
- Abrir muchas instancias de Internet Explorer
- Hacer que una de las instancias se bloquee
- Asegurarse que las otras instancias del Internet Explorer funcionan normalmente

Las instancias abiertas del navegador se ven afectadas por una que está bloqueada.

**3. Aplicación y realización de la prueba**

a. Abrir una instancia de Microsoft Internet Explorer.

b. Abrir más instancias del navegador usando Archivo-> Nuevo-> Ventana, o pulsando Ctrl+N en el teclado:




c. Para bloquear Internet Explorer es necesario que un comando de la hoja de estilo sea incorrecto:

- Crear esa hoja de estilo y abrirlo en sólo uno de los navegadores de Internet Explorer.
- Dar clic derecho en el escritorio y seleccionar Nuevo-> Documento de texto.
- Abrir el documento de texto y pegarlo en el código siguiente: <STYLE> </ style> @;/ *
- Guardar el documento como test.htm.
- Seleccionar una de las ventanas de Internet Explorer y navegar hasta el escritorio. Escribir “escritorio” en la barra de direcciones y presionar **Enter** en el teclado.
- Dar doble clic sobre test.htm

**4. Resultados**

Cuando la ventana de Internet Explorer trata de abrir la página que se creó, se visualiza inmediatamente este mensaje:

**Mensaje de problema**

Después de seleccionar el botón **Enviar reporte** o **No enviar**, la ventana de Internet Explorer se cierra junto con todas las abiertas. Esto se debe a que los navegadores de Internet Explorer son dependientes de la primera ventana que se ha abierto, por lo que, si un error ocurre en una, se van a cerrar todas las demás de Internet Explorer.

Esto es un error crítico porque representa una posible pérdida de datos. Si en algún momento se tiene un error en una de las ventanas del navegador, se va a perder el trabajo, así como toda la información de búsqueda de todas las ventanas.

**La prueba de carga** permitió encontrar rápida y fácilmente el error porque simplemente se **generó** al alterar una función que dependía de una determinada condición ambiental y, al quitar esa condición, se bloqueó el navegador.

Características de hardware y software necesarias para desarrollar la prueba:

Prueba de Microsoft Internet Explorer v6sp2

Equipo:

Dell OptiPlex GX240

256 MB SDRAM

Intel Pentium IV, 1.69GHz

Software:

Microsoft WindowsXP, SP2rc2

Ya cubiertas las pruebas para los requisitos no funcionales, se probarán a continuación los requisitos **funcionales,** que son los del cliente, a través de las pruebas de aceptación.

### 2.3.4. Pruebas de aceptación

Las pruebas de aceptación son pruebas formales con respecto a las necesidades del usuario. Consiste en requisitos y procesos de negocio llevados a cabo para **determinar** si un sistema cumple con los **criterios** de **aceptación,** y para permitir que el usuario, los clientes u otra entidad autorizada **determinen** si se acepta o no el sistema (ISTQB, 2012). El principal propósito de esta prueba es **validar,** de extremo a extremo, el flujo de negocio en el sistema de software.

Los tipos de pruebas de aceptación son las siguientes:

**Pruebas de aceptación del usuario.** Mediante estas pruebas los usuarios verifican que el sistema de software sea funcional según sus requerimientos.

**Pruebas de aceptación operacional.** Son las pruebas mediante las que se comprueba que los procesos y procedimientos están en su lugar, para permitir que el sistema sea usado y permita su mantenimiento. Esto incluye el control realizado a la copia de seguridad, los procedimientos para la recuperación del sistema, la capacitación para los usuarios finales, los procedimientos de mantenimiento y de seguridad.

**Pruebas de contrato y regulación.** El sistema se prueba con los criterios de aceptación según se documenta en un contrato, antes de ser aceptado, así como para asegurar que cumple con los estándares gubernamentales, legales y de seguridad.

**Pruebas alfa y beta.** Se llevan a cabo en los sitios de los desarrolladores, y consiste en **pruebas del sistema operativo** por parte del personal interno, antes de que sea entregado a los clientes externos. Las pruebas beta se llevan a cabo en los sitios de los clientes, y consiste en que un grupo de clientes que utilizan el sistema proporcionen

información antes de que se libere a otros clientes. A menudo es llamada *prueba de campo*.

**Método:** esta prueba implica la técnica de caja negra que consiste en verificar que lo que se espera que realice el sistema de software, lo haga efectivamente.

**Tareas**

1. **Plan de pruebas de aceptación**
   - Preparar
   - Comentar
   - Revisión
   - Validación de la especificación de los requisitos

**2. Diseño de los casos prueba**
   - Revisión
   - Validación

**3. Prueba de implantación**

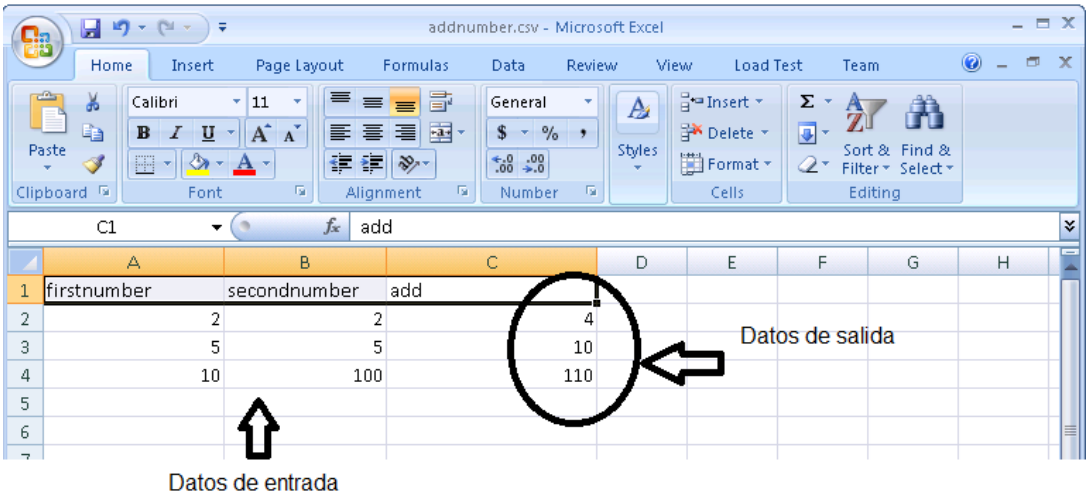
   - Aplicación

**4. Resultados**

Se realiza después de haber aplicado las pruebas de implantación al sistema. Ahora es el turno de probar los requisitos funcionales del sistema de software.

El ejemplo siguiente es una prueba de aceptación operacional apoyado en la interfaz de usuario (UI, *user interface,* por sus siglas en inglés), mediante Visual Studio 2010 Microsoft Test Manager y las tablas de ajuste de estilo a la aplicación calculadora.

Las tablas de ajuste consisten en coordinar las partes interesadas y los analistas del sistema, para generar datos de entrada esperados y obtener la salida adecuada en una hoja de cálculo.



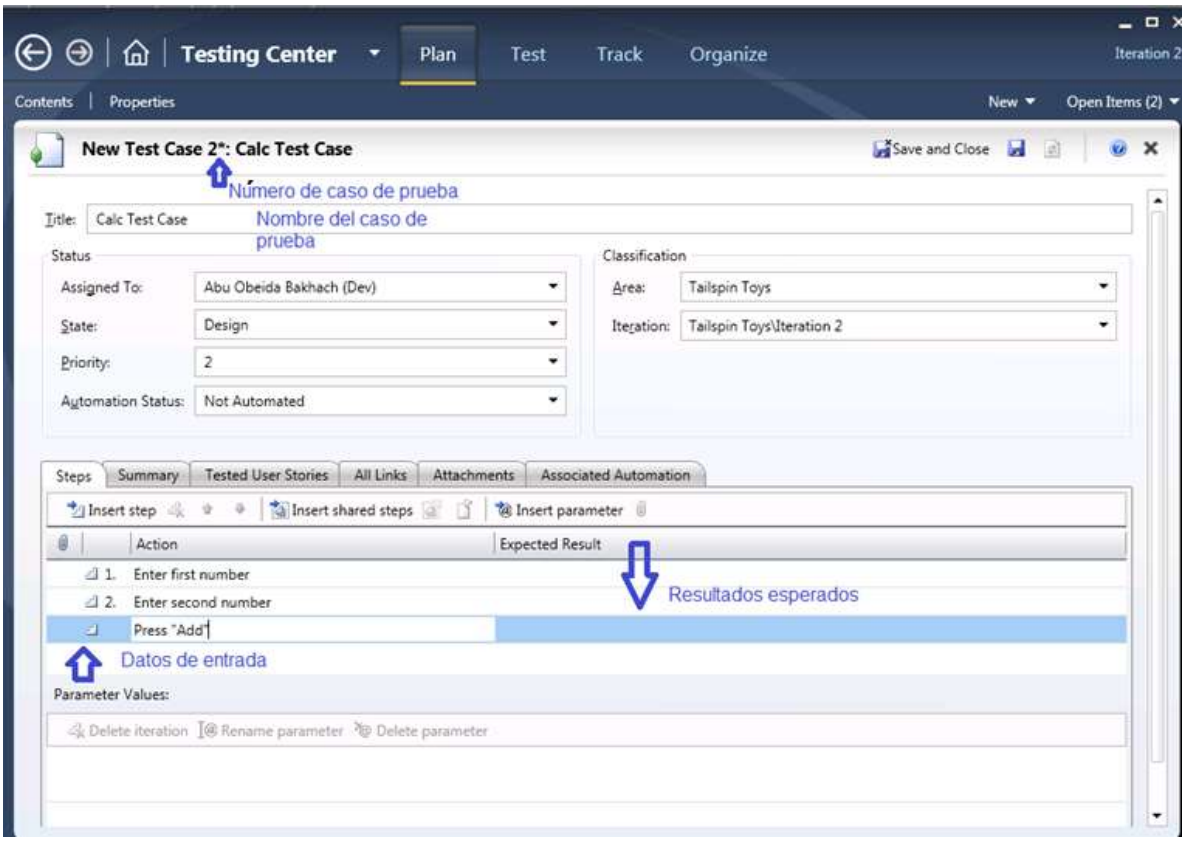

**Datos de prueba de entrada de la UI.**

El desarrollador puede hacer uso de los datos de entrada a través estas tablas utilizando la automatización de pruebas, para verificar que el sistema de software haga lo se esperaba.

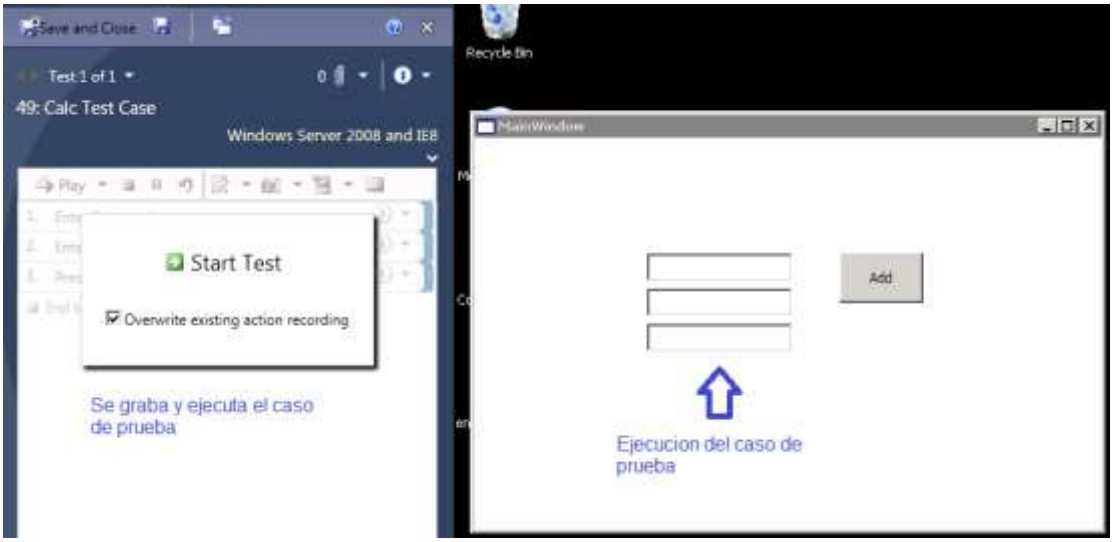
Los datos de prueba de entrada de la UI son obligatorios.

**1. Se elabora el plan de pruebas** con los elementos propuestos en el subtema 2.1.1 Plan de pruebas.

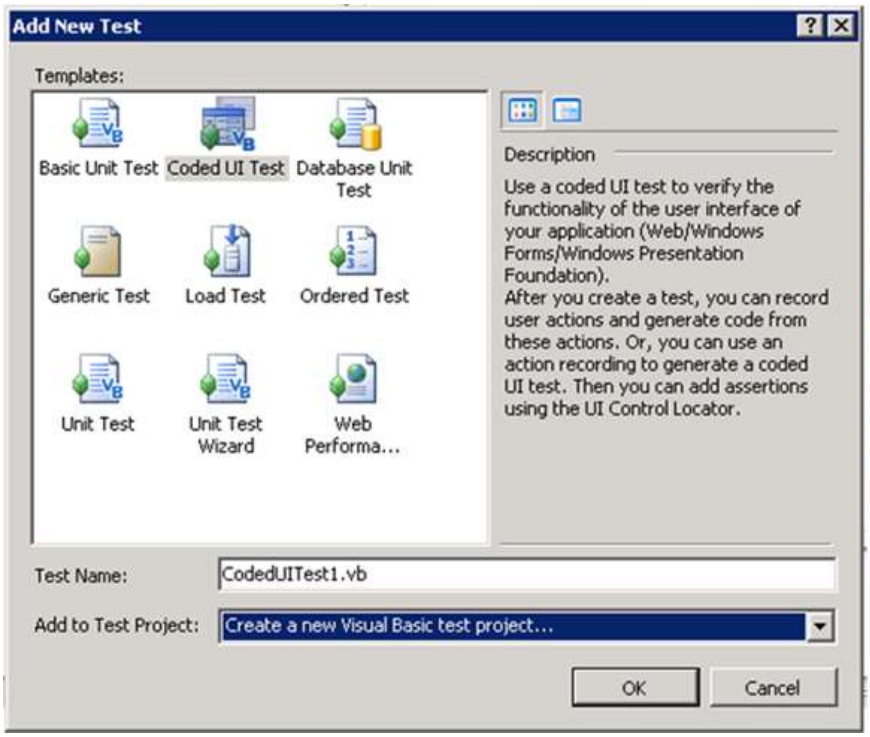
**2. Diseño de los casos prueba.** Se crea un caso de prueba en el Test Manager. Se introducen los pasos en orden para determinar que la prueba sea efectiva.




**Se guarda las acciones del caso de prueba.**

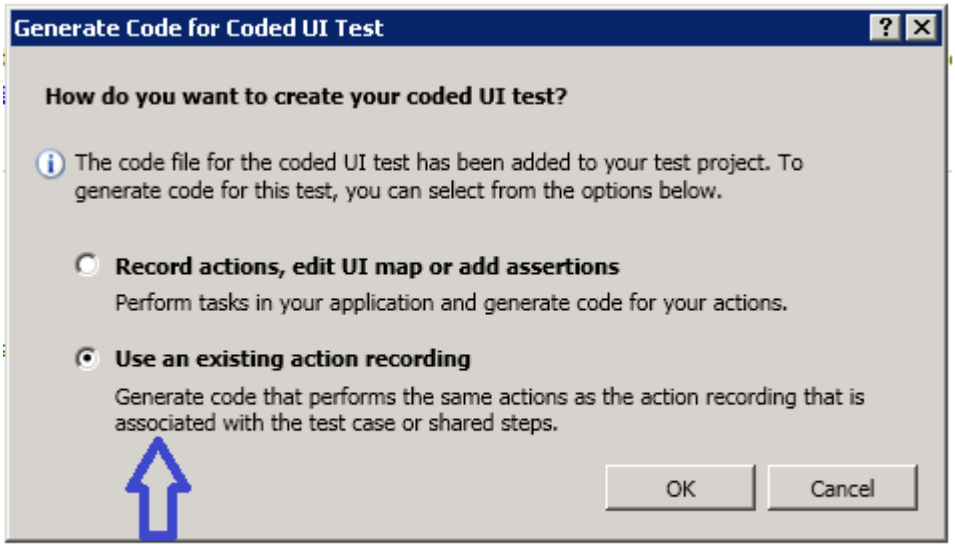



**Creación de la UI de prueba (Menú Test\New Test).**




**Descripción:**
**Usa una prueba de UI codificada para verificar la funcionalidad de la interfaz del usuario de la aplicación (Web/Windows...). Después de crear la prueba, puede grabar las acciones y generar el código, o usar las acciones grabadas para generar una prueba UI codificada, y agregar las afirmaciones usando el control de la UI.**

Se genera el código.




Se crea la prueba de UI codificada usando una acción grabada existente

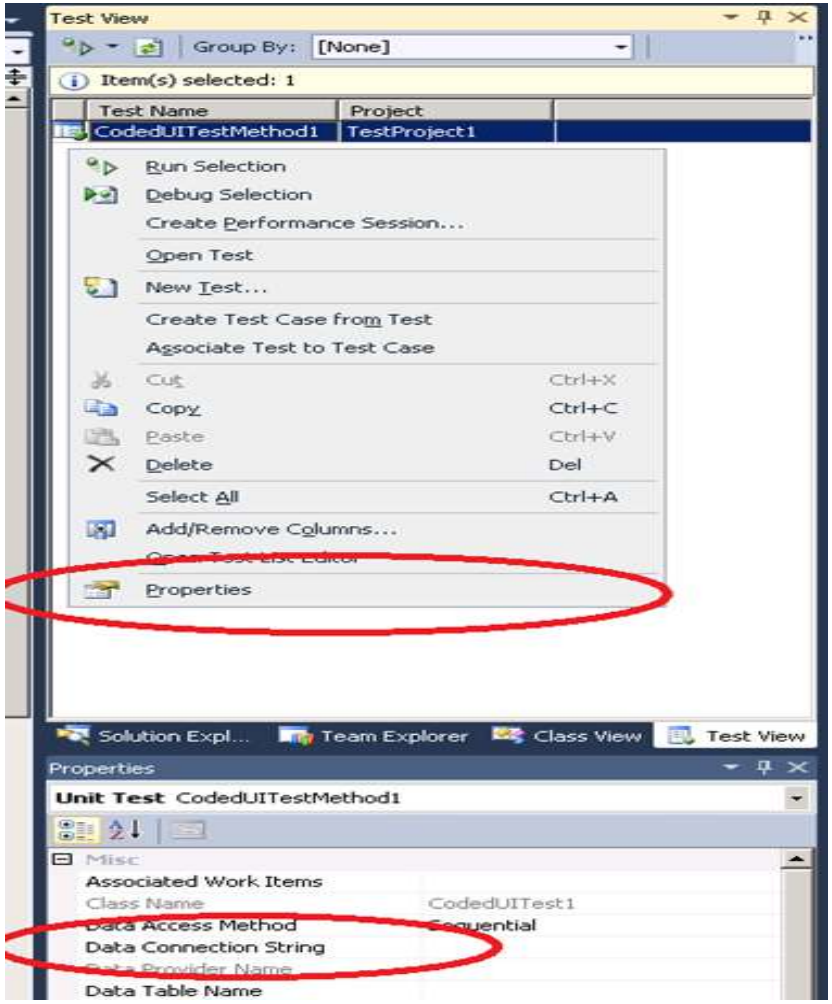
A continuación, se muestra el código:

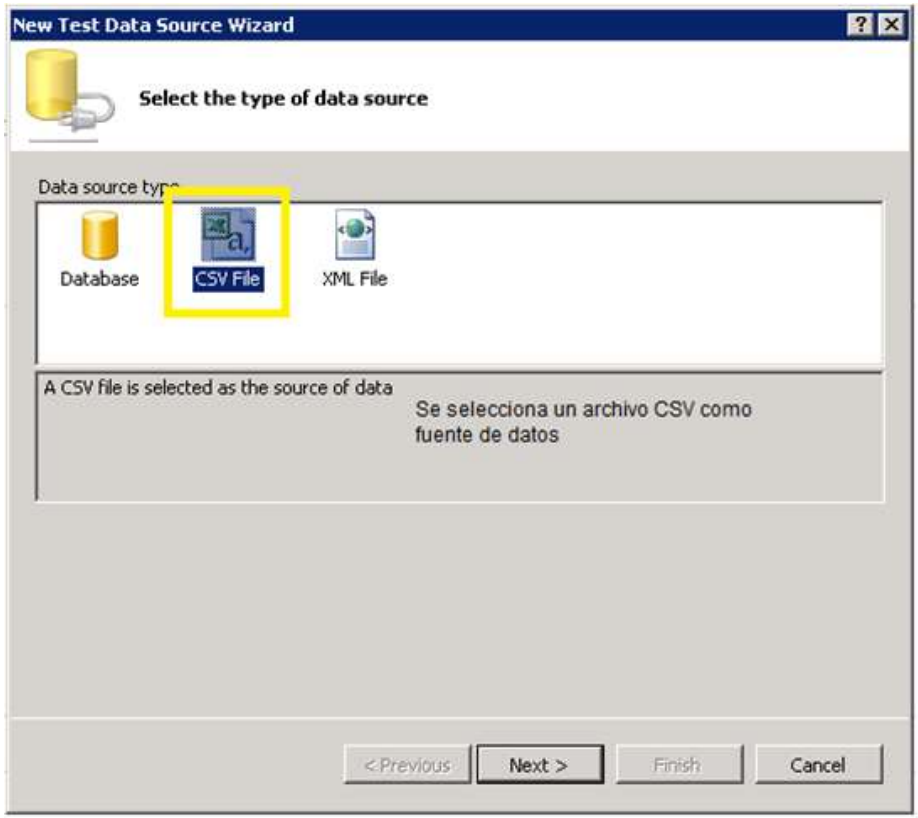
<CodedUITest()>

```
Public Class CodedUITest1  // Clase de prueba
   Public Sub CodedUITestMethod1() //Método
    Me.UIMap.Enterfirstnumber() // Obtiene primer número
    Me.UIMap.Entersecondnumber() // Obtiene segundo número
    Me.UIMap.PressAdd() //Operación a realizar
 End Sub
```

Para obtener más información sobre el código generado consulta el sitio MSDN (2014a)

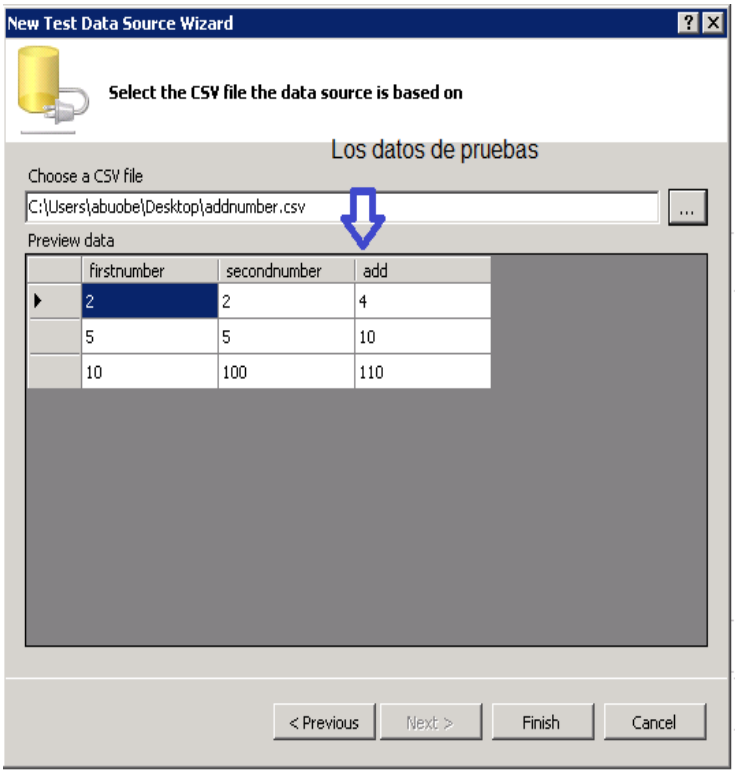
**Se abre las propiedades de IU codificada.** Test\ Windows\Test\ View. Se selecciona propiedades y elige la cadena de conexión, para ligar los datos a la prueba de UI codificada.




**Ventana de datos del Test Manager.**

**Ventana de propiedades del Test Manager. Ventana para seleccionar la fuente.**




**Ventana vista de los datos de prueba en el Test Manager.**

Lo anterior completa la definición del método de IU. Codificado con una fuente de datos se puede acceder desde el objeto de contexto de prueba.

| | |
|---|---|
| Public Class CodedUITest1 | Clase |
| <DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV","\|DataDirectory\|\addnumber.csv","addnumber#csv",DataAccessMethod.Sequential)> <DeploymentItem("TestProject1\addnumber.csv")> <TestMethod()> | Fuente de datos Acceso secuencial |
| Public Sub CodedUITestMethod1() | Método de prueba |

**Código del caso de prueba de aceptación.**

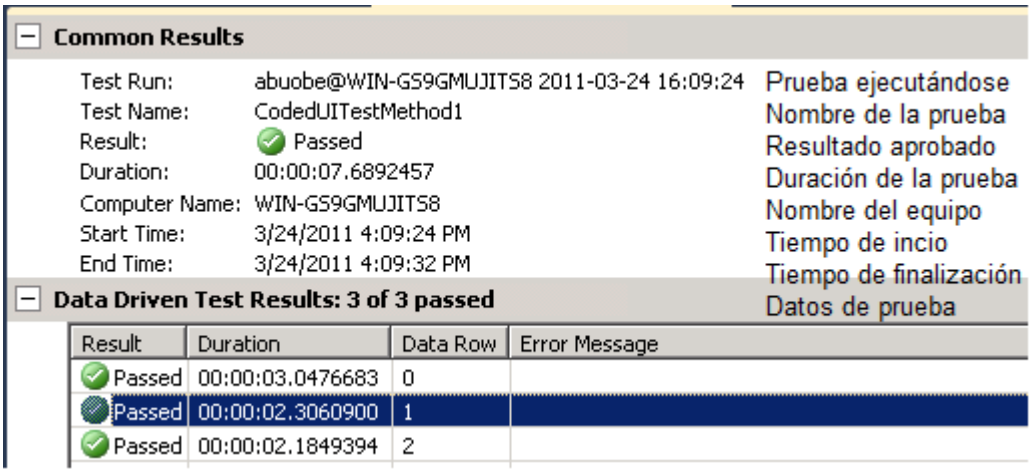
En la prueba de IU codificada, se usa el objeto **param** para establecer el valor de entrada de la fuente de datos.

**Para este caso de prueba se tiene:**

Me.UIMap.EnterfirstnumberParams.UITextBox1EditText =
TestContext.DataRow("firstnumber")

Para este caso de prueba específico, el objeto **param** toma el mismo nombre del caso de prueba.

**3. Prueba de aceptación.** Se ejecuta y se observa la ejecución secuencial de acuerdo con los datos de pruebas. Es posible cambiar de secuencial al azar en la lista de propiedades del conjunto de datos.




Resultados de la prueba de aceptación en el Test Manager

**4. Resultados.** Se documenta que en la prueba de aceptación operacional el resultado fue negativo. No se observaron errores.

Como se observa, las pruebas de aceptación verifican que los requerimientos definidos desde el diseño del software están presentes en el producto final. Si el requisito no lo tuviera el sistema, se está a tiempo de agregarlo y lograr un grado de aceptación por parte del usuario final.

### 2.3.5. Pruebas de regresión

Son pruebas aplicadas a un sistema de software previamente probado, después de aplicar los correctivos necesarios, para asegurar que los defectos efectivamente se hayan corregido, o generado otros como resultado de los cambios realizados. Se realizan cuando el software o su entorno cambia (ISTQB, 2012).

Este tipo de pruebas suele llevarse a cabo por los especialistas. Se recomiendan las automatizadas como la mejor opción para llevar a cabo las de regresión. En el mercado hay herramientas libres (*open source*), disponibles para la aplicación de las pruebas de regresión.

Las siguientes herramientas pueden ser usadas para la automatización de pruebas a todos los niveles, incluyendo las de regresión.

• QTP

• Regresión Tester

• Selenio

• Rational Functional Tester

• Watir

• WinRunner

• ActiWate

• AdventNet QEngine

• SilkTest

• Vtest

**Método:** esta prueba implica la técnica de caja negra, que consiste en la verificación de que la interfaz acepte los datos de entrada, y los procese como se espera.

**Tareas**

**1. Plan de pruebas de regresión**

- Preparar
- Comentar
- Revisión
- Validación de la especificación de los requisitos

**2. Diseño de los casos prueba**

- Revisión
- Validación

**3. Prueba de regresión**

- Aplicación

**4. Resultados**

El siguiente ejemplo se trata del reproductor multimedia WinAmp de Nullsoft, diseñado principalmente para la reproducción de archivos de música mp3.

Para que los usuarios puedan compartir una lista de las canciones que escuchan, WinAmp crea una función que genera versiones HTML de las listas de reproducción de canciones. Sin embargo, la función de lista de reproducción Generate HTML presenta una falla que permite, a un usuario malicioso, insertar códigos peligrosos en el interior de un archivo de música WinAmp etiqueta ID3 (etiqueta de información de la canción), que se ejecuta con la generación de la lista de reproducción de html. El error apareció originalmente en la versión 2.76, se corrigió en 2.77 y 2.78, pero reapareció en la versión 2.79. Después de ésta, Nullsoft tuvo que liberar una nueva versión fija del error en WinAmp 2.80.

El usuario hace uso de las etiquetas. Se utiliza principalmente en archivos sonoros como MP3, que permiten registrar información sobre la canción, tal como el artista, título, número de pista, etcétera.

**Interfaz de WinAmp v2.80**

Se aplican las tareas de la prueba de regresión:

**1. Plan de pruebas de regresión**

**2. Diseño de los casos de prueba**

Dado que el Generate HTMLPlaylist usa el título de la canción y el artista, se realizarán las clases de equivalencia en función de los tipos de entrada que tendrán los campos. Además, dado que toman los mismos tipos de entrada, sólo hay que probar uno de ellos.

Así, para el título y el artista, se tiene una clase básica de equivalencia:

Título/artista = { null, average, max lenght }

Se modifica la clase de equivalencia para incluir código HTML, ya que se está generando una lista de canciones en HTML.

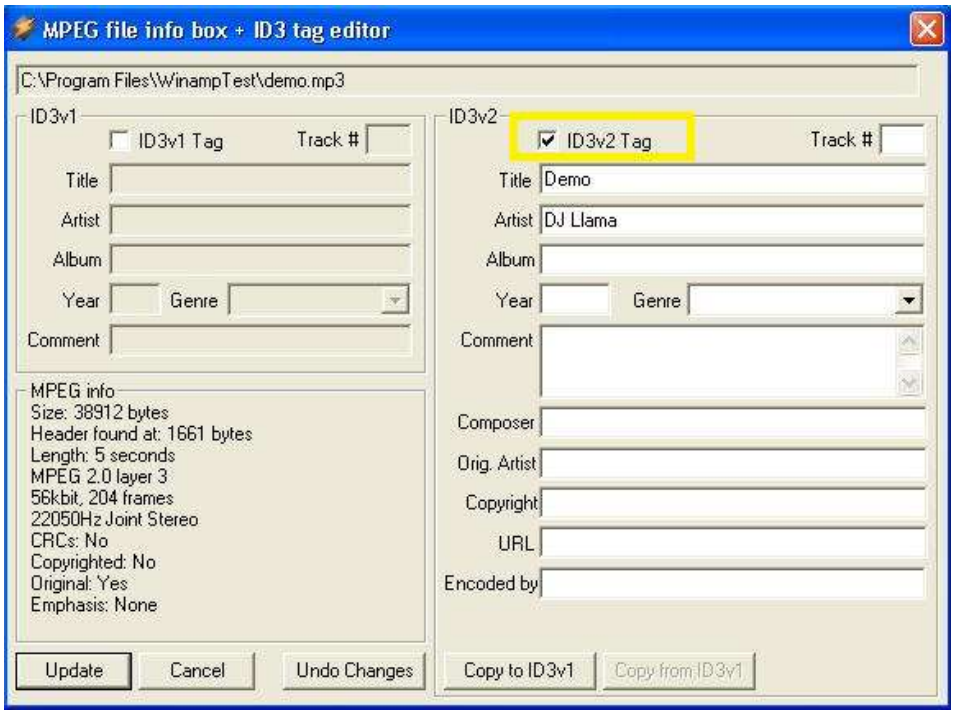
Esta es la misma prueba que se realizó en la versión 2.76 de WinAmp, y en dos versiones posteriores. Nullsoft cambió el fragmento de código que permite que el error aparezca.

**3. Ejecución del caso de prueba**

a. Se inicia WinAmp v2.76 y se abre un archivo mp3.

b. Se abre el editor de etiquetas ID3, con alt+3 o botón derecho sobre la canción en la ventana de lista de reproducción, y se selecciona la información del archivo

c. En la ventana de la etiqueta ID3, se selecciona la etiqueta ID3v2 de la casilla.




**Selección de etiqueta ID3v2.**

En el campo 'Title', pega lo siguiente:

<script language="JavaScript"> var message =“This is some HTML Code"; alert(message); </script> // Indica que es un código html

Da clic en **Update** para regresar a la ventana de WinAmp.

Se genera una lista en HTML.

**Generación de la lista de reproducción**

Se repite el mismo caso de prueba de las versiones 2.77 y 2.78 (donde este error no debería presentarse).

Se repite el mismo caso de prueba en la versión 2.79 (donde este error podría presentarse).

**4. Resultados**

**Código ejecutado correctamente.**

Se muestra el mensaje de la imagen anterior. Significa que el código HTML se ejecutó correctamente. Indica que el error no fue corregido, ya que igualmente se pudo haber insertado el código siguiente:

<script language="JavaScript"> while (true) { var message =“Tu internet está bloqueado"; alert(message); }</script>// Muestra un letreo infinitamente porque mientras lenguaje sea JavaScript que muestre el mensaje”Tu internet está bloqueado”.

Este error es crítico, ya que abre una puerta de entrada que permite a otros alterar o dañar la información de otros usuarios.

La prueba de regresión permitió reusar las mismas pruebas aplicadas anteriormente para localizar viejos errores.

Características de hardware y software para desarrollar la prueba:

Objeto de prueba, NullSoft's WinAmp v2.79

Equipo:

Dell OptiPlex GX240
Microsoft Windows XP
256 MB SDRAM
Procesador Intel Pentium IV, 1.69 GHz

Requerimiento de software adicional:

NullSoft WinAmp 2.76
NullSoft WinAmp 2.77 o 2.78
Explorador Mozilla FireFox Internet 0.8
Explorador Microsoft Internet Explorer 6.0

Las pruebas de regresión son determinantes. Si no son superadas, se retrasa la puesta en marcha del sistema de software y se tendrá que aplicar las veces que sean necesarias hasta lograr un grado de satisfacción por parte del cliente, para que firme el documento de aceptación. Si se arrastran estos errores se hace al sistema de software o aplicación vulnerable y no seguro.

Es importante la aplicación de pruebas a todos los niveles porque se va garantizando, en cada nivel, un grado de aceptación que da como resultado que el sistema de software cumpla con los requerimientos funcionales y no funcionales, establecidos al inicio del proyecto.

Las pruebas de unidad, de integración, implantación, aceptación y, las de regresión, se utilizan para garantizar que el sistema de software sea funcional y cumpla con los requerimientos funcionales y no funcionales, no hay una garantía del 100% que esté libre de errores, pero sí un porcentaje aceptable de fiabilidad[3].

## Cierre de la unidad

Durante esta unidad se expusieron las fases de la administración del proceso de pruebas, la importancia de las mismas y la serie de actividades que están involucradas desde la planeación hasta el cierre del proyecto de pruebas. Dentro de la administración del proceso se encuentra el plan de pruebas que, a su vez, implica una serie de pasos que van desde la creación de casos de prueba, hasta la aplicación y evaluación de las mismas. Por supuesto, se incluye el cierre del proyecto de pruebas cuyo fin es el liberar el producto de software para su implantación.

También se identificaron los tipos de evaluación y las técnicas de pruebas disponibles, de acuerdo con lo que se desea evaluar, con su respectiva batería de pruebas:

- Caja blanca
- Caja negra

Se identificaron los niveles de pruebas que se evalúan, según las necesidades del proyecto de software:

- Pruebas unitarias
- Pruebas de integración
- Pruebas de implantación
- Pruebas de aceptación
- Pruebas de regresión

Cada nivel tiene un propósito definido que marca un precedente para el siguiente. La estrategia de pruebas está dada por el nivel de pruebas y la técnica a aplicar, así como los casos de pruebas y el personal capacitado y adecuado. El proceso de pruebas debe

[3] Fiabilidad es la probabilidad de que un sistema de software realice adecuadamente su función prevista a lo largo del tiempo, cuando opera en el entorno para el que ha sido diseñado.

arrojar resultados base para las mejoras y correcciones necesarias que garanticen un sistema tolerante a fallos.

Después de haber superado las pruebas, el sistema está listo para su implantación y puesto en marcha. Lo que sigue es una fase importante en todo desarrollo de software: el mantenimiento, ya que es muy probable que un sistema en funcionamiento evolucione según las necesidades de la empresa. Para esto, es necesario que el mantenimiento de sistemas de software esté definido y de respuesta a las necesidades de crecimiento de una organización.

Recuerda que la mejor manera de aprender y comprender estos temas es practicar, por ello, es recomendable que repases los contenidos, así como que realices prácticas de elaboración de casos y planes de prueba. Considera todos los elementos, desde la selección de la estrategia hasta el diseño de los casos y la ejecución de las mismas. No olvides el análisis de los resultados y las acciones a realizar a partir de éste.

Antes de concluir con la unidad, se integran algunas reflexiones sobre las pruebas de software. Hace ocho años, el profesor Luis Vinicio León Carrillo (2007)[4] afirmó que hasta el año 2006 aún no se contaba con una industria de prueba de software suficientemente grande y especializada en el país, con una oferta integral de servicios de alto valor agregado bien diferenciados.

Hoy se pueden observar en el mercado varias empresas que se dedican exclusivamente a la prueba de software. Las ofertas de trabajo también se han incrementado, ya sea para las empresas especializadas que realizan las pruebas de software a terceros o para aquellas que diseñan y prueban sus sistemas. También hay empresas que ofrecen la certificación de pruebas de software, alineada al programa de certificación del ISTQB, reconocido como un estándar internacional para probadores de software. La certificación del ISTQB consta de tres niveles: fundamentos, avanzado y experto. La actividad de pruebas de software sigue siendo un área de oportunidad importante a considerar como futuros profesionistas.

[4] Profesor-investigador del Departamento de Electrónica, Sistemas e Informática del ITESO, y director general de e-Quallity S.A. de C.V., empresa especializada en pruebas de software, tomado de su artículo “La Industria de prueba de software en México, buenas oportunidades si nos especializamos y crecemos.”

### Para saber más

Es importante que investigues sobre la empresa Moprosof (Modelo de Procesos para la Industria del Software). Ofrece un modelo para la mejora y evaluación de los procesos de desarrollo y mantenimiento de sistemas y productos de software. Fue desarrollado por la Asociación Mexicana para la Calidad en Ingeniería de Software, a través de la Facultad de Ciencias de la Universidad Nacional Autónoma de México (UNAM), y a solicitud de la Secretaría de Economía para obtener una norma mexicana que resulte apropiada a las características de tamaño de la gran mayoría de empresas mexicanas de desarrollo y mantenimiento de software. Es un área de oportunidad no explotada al 100%, por lo que cada día se requieren profesionales especialistas con formación en el concepto calidad. Las pruebas son una forma de garantizar un grado calidad en un sistema tolerante a fallos.

Para conocer y profundizar más en el subtema 2.3.1. Pruebas unitarias, accede al siguiente enlace:

- MSDN Microsoft Developer Network. (2014). *Información general de pruebas unitarias*. Recuperado de http://msdn.microsoft.com/es-es/library/ms182516(v=vs.90).aspx

En el siguiente enlace, encontrarás un tutorial donde para ejecutar pruebas unitarias con JUnit:

- ExpertoJ2EE. (2013). *Tutorial sobre JUnit*. Recuperado de: http://expertoj2ee.blogspot.com/2013/06/tutorial-sobre-junit.html
- En el siguiente artículo se expone un caso de pruebas que se recomienda revisar como ejemplo de elaboración de pruebas de software.
- Garzás, J. (2013) *Caso de estudio: Cómo hace Google sus planes de pruebas (y los hace en menos de 10 minutos*. Recuperado de http://www.javiergarzas.com/2012/12/google-pruebas.html

## Fuentes de consulta

### Bibliografía Básica


- Abran, C. y Moore J. (2004). *Guide to the Software Engineering Body of Knowledge. EU. IEEE Computer Society Software TestinG*. Disponible en https://cs.fit.edu/~kgallagher/Schtick/Serious/SWEBOKv3.pdf

- Bohem, B., Brown J., Kaspar, H., Lipow M., McLeod G. y Merritt, M. (1978). *Characteristics of Software Quality* (tomo 1). Cleveland: TRW Systems and Energy.

- Bourque, P. y Fairley, R. (2014). *Guide to the Software Engineering Body of Knowledge Version 3.0. Disponible en* https://cs.fit.edu/~kgallagher/Schtick/Serious/SWEBOKv3.pdf

- Calero, C. *et. ál*. (2010). *Calidad del producto y proceso software*. Madrid: RA-MA.

- Carrol, S. *et. ál*. (2007). *Fundamental Concepts for the Software Quality Engineer,* 2a. ed. Milwaukee: American Society for Quality.

- Davis, C. (2006). *Test management best practices*.

- *DRAE. (s. f.). [Entrada: Prueba]. Disponible en https://dle.rae.es/prueba?m=form*

- Fowler, M. (2000). *UML Distilled,* 2a. ed. Boston, MA: Addison Wesley Longman.

- Gutiérrez, R. Escalona, C., Mejías, R. y Torres, J. (2008). *Generation of Test Cases from Functional Requirements: a Survey*. Disponible en https://www.researchgate.net/publication/228979324_Generation_of_test_cases_from_functional_requirements_A_survey

- Heumman, J. (2001). Generating Test Cases from Use Cases. The Rational Edge. Disponible en https://www.researchgate.net/publication/220724437_Generating_Test_Cases_from_Sequences_of_Use_Cases

- IBM Rational Unified Process. RUP. (1998). *Rational Unified Process. Best Practices for Software. Development Teams Rational Software*. Disponible https://www.academia.edu/4962602/Rational_Unified_Process_Best_Practices_for_Software_Development_Teams

- IEEE (2017). 12207-2017 - ISO/IEC/IEEE International Standard - Systems and software engineering -- Software life cycle processes. Disponible en: https://standards.ieee.org/ieee/12207/5672/

- IEEE Standard. (2014). 610.12-1990 *IEEE Standard Glossary of Software Engineering Terminology*. Disponible en https://n9.cl/3q2u3

- ISTQB. (2023). Standard Glossary of Terms Used in Software Testing Disponible en https://glossary.istqb.org/es_ES/search

- Koirala, S. y Sheikh, S. (2008). *Software Testing*. Burlington, MA: Jones & Bartlet Publishers.

- León, L. (2007). *La industria de prueba de software en México. Buenas oportunidades si nos especializamos y crecemos.* Disponible en https://sg.com.mx/content/view/369

- Llorens, J. (2005) *Gerencia de proyectos de tecnología de información.* Caracas: CEC.

- Ministerio de Comunicaciones (2008). *Plan de pruebas detallado Sistema de notificación en línea*. Bogotá, Colombia. Disponible en https://www.mintic.gov.co/portal/715/articles-125156_recurso_00.pdf

- Myers, G. Wiley, J. y Sons (2004). *The Art of Software Testing,* 2a. ed. New Jersey: John Wiley & Sons.

- MSDN Microsoft Developer Network (2014a). *How to: Generate a Coded UI Test by Recording the Application Under Test*. Disponible en https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2010/dd286608(v=vs.100)?redirectedfrom=MSDN

- MSDN Microsoft Developer Network. (2022b). *Informes del progreso de las pruebas para los planes de pruebas.* Disponible en *https://docs.microsoft.com/es-es/azure/devops/test/progress-report?view=azure-devops*

- Ness, p. y Thomas, L. (2005). *The Rational Unified Process for testers. IBM.*

- Périssé, M. y Pepe, L. (2007). *Casos de prueba.*

- Reynolds, G. (1995). *Information Systems for Managers*, 3a. Ed. New York: West Publishing Company.

- Sommerville, I. (2011). *Ingeniería del software,* 9a. ed. México: Pearson Educación.

- Srinivasan, D. (2006). *Software Testing*. Delhi: Dorling Kindersley.

- Testing Standards. (2014). *BS 7925-1 Glossary of Terms Used in Software Testing.* Disponible en http://www.testingstandards.co.uk/bs_7925-1_online.htm

- Thakare, S. y Chavan, S. (2012). Software Testing Strategies and Technique. *International Journal of Emerging Technology and Advanced Engineering*, (2).

- ISO. (2001). *ISO/IEC TR 9126-1. Software Engineering: Product Quality. Part 1. Quality Model*. Switzerland: ISO.

- Pressman, R. (2010). Software Engineering: A Practitioner's Approach. Disponible en https://n9.cl/jiwrpx

**Imágenes**


- Kong, W. (2009). *Towards a Formal and Scalable Approach for Quantifying Software Reliability at Early Development Stages*. College Park: University of Maryland.

- SoftQaNetwork. (2009). *El Famosíssim V-Model.*

- The Bugzilla Guide. (2006). *6.4. Life Cycle of a Bug*. Disponible en https://www.bugzilla.org/docs/2.18/html/lifecycle.html

- Tutorialspoint. (2014). *Cause-Effect Graph. Disponible en* http://www.tutorialspoint.com/software_testing_dictionary/cause_effect_graph.htm

- UnADM. Universidad Nacional Abierta y a Distancia. (2014). *Lección 41. Enfoque de la Prueba del Software.*