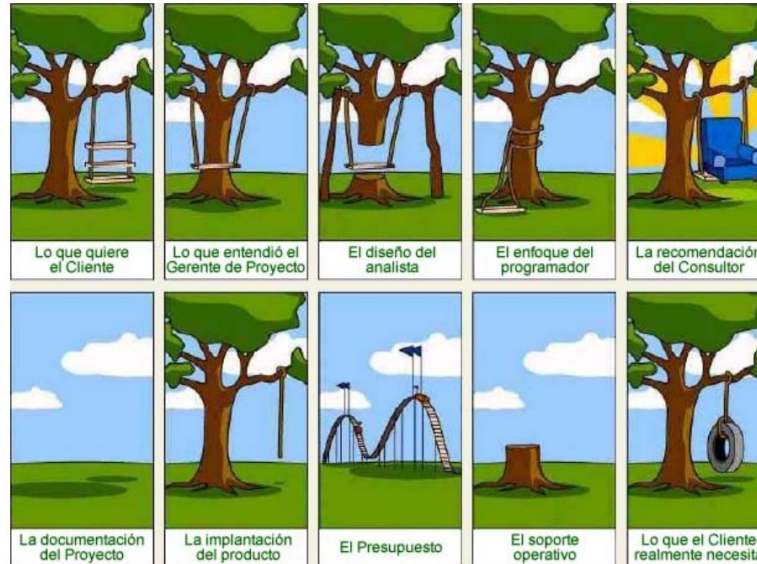


Tema 3

Diseño y realización de pruebas

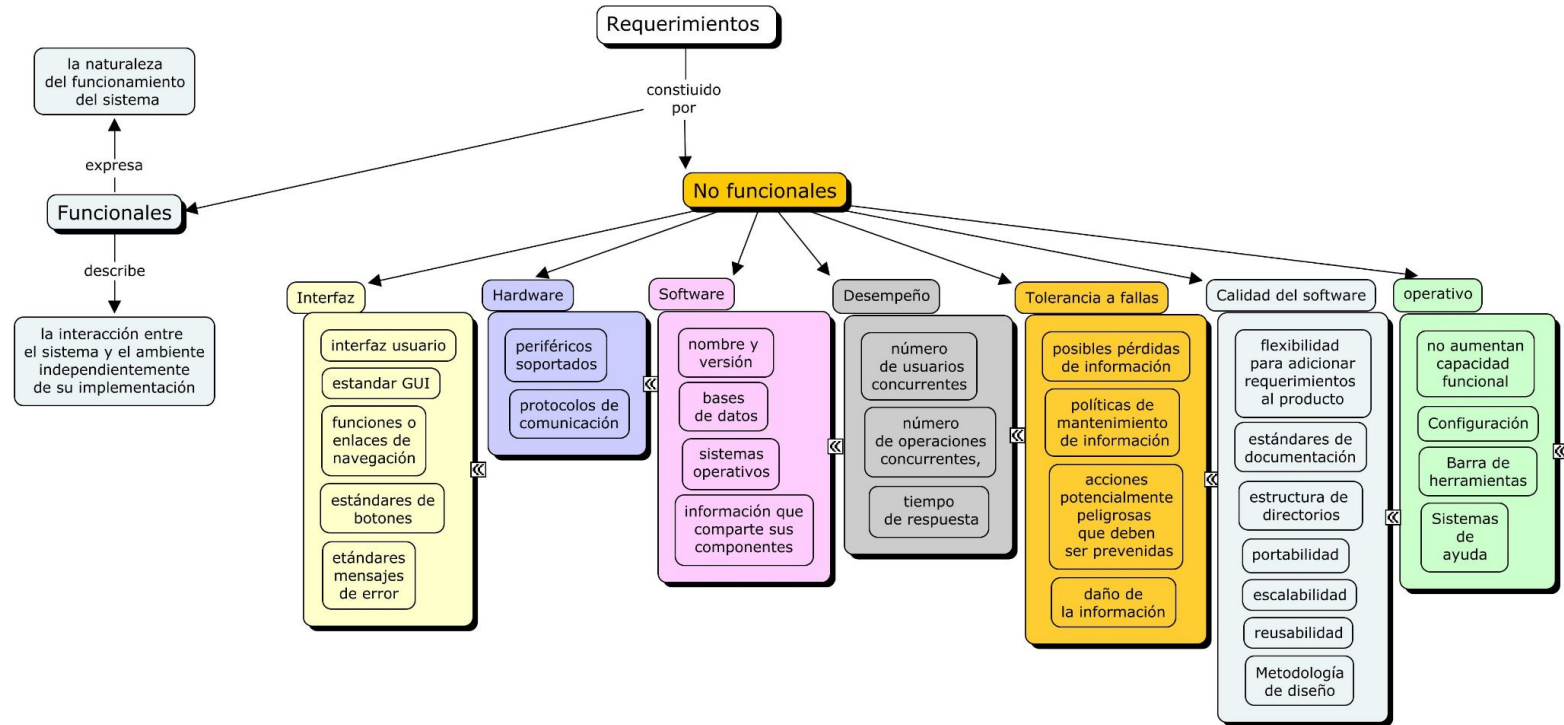
Requisito software

Un requisito de software es una declaración o especificación que describe una característica, función o propiedad que debe tener el software. Estos requisitos son esenciales para definir y comprender las expectativas del cliente o usuario final en relación con el software, además de servir como una guía fundamental para los equipos de desarrollo, asegurando que el software se diseñe, implemente y pruebe de manera coherente y que se cumplan las necesidades del proyecto.



Requisito software

Los requisitos se pueden clasificar en funcionales y no funcionales.



Requisito funcional

Describen las funciones del sistema a diseñar. Es una descripción de lo que será el sistema y cómo funcionará para satisfacer las necesidades del usuario. Proporcionan una descripción clara de cómo se supone que el sistema debe responder a un comando en particular, las características y lo que esperan los usuarios.

Define qué hace el sistema.

Ejemplos:

- El usuario podrá clasificar sus viñetas y cómics según los géneros.
- El Sistema deberá avisar al usuario para la publicación llegada una fecha previamente asignada mediante un aviso en la aplicación y la creación de un evento en el calendario.

Requisito no funcional

Explica las limitaciones y restricciones del sistema a diseñar. Estos requisitos no tienen ningún impacto en la funcionalidad de la aplicación. Además, existe una práctica común de subclasificar los requisitos no funcionales en varias categorías, como:

- **Interfaz de usuario:** La aplicación debe estar en español.
- **Fiabilidad:** La aplicación debe de funcionar adecuadamente aunque se pierda la conexión.
- **Seguridad:** El usuario se tiene que registrar para acceder al panel de control.
- **Rendimiento:** La aplicación no puede tardar más de 1 segundo en responder la petición.
- **Estándares:** Para el almacenamiento de audio se debe emplear el estándar ISO/IEC 11172-3 (mp3).

Define cómo lo hace el sistema.

Prueba software

Las pruebas de software son un proceso controlado y planificado en el que se examina minuciosamente un programa o aplicación de computadora para identificar problemas, errores o defectos. Estos problemas podrían ser fallos en el funcionamiento del software o situaciones inesperadas que podrían hacer que falle. El objetivo principal es encontrar estos problemas antes de que el software se lance públicamente o se implemente en un entorno de producción para que puedan corregirse y el software funcione de manera confiable y cumpla con los requisitos establecidos. Estas pruebas pueden involucrar la ejecución de escenarios específicos, la simulación de situaciones inusuales y la verificación de que todas las partes del software funcionen correctamente en conjunto.

“Las pruebas solo pueden demostrar la presencia de errores, no su ausencia”

Edsger Dijkstra (1930-2002)

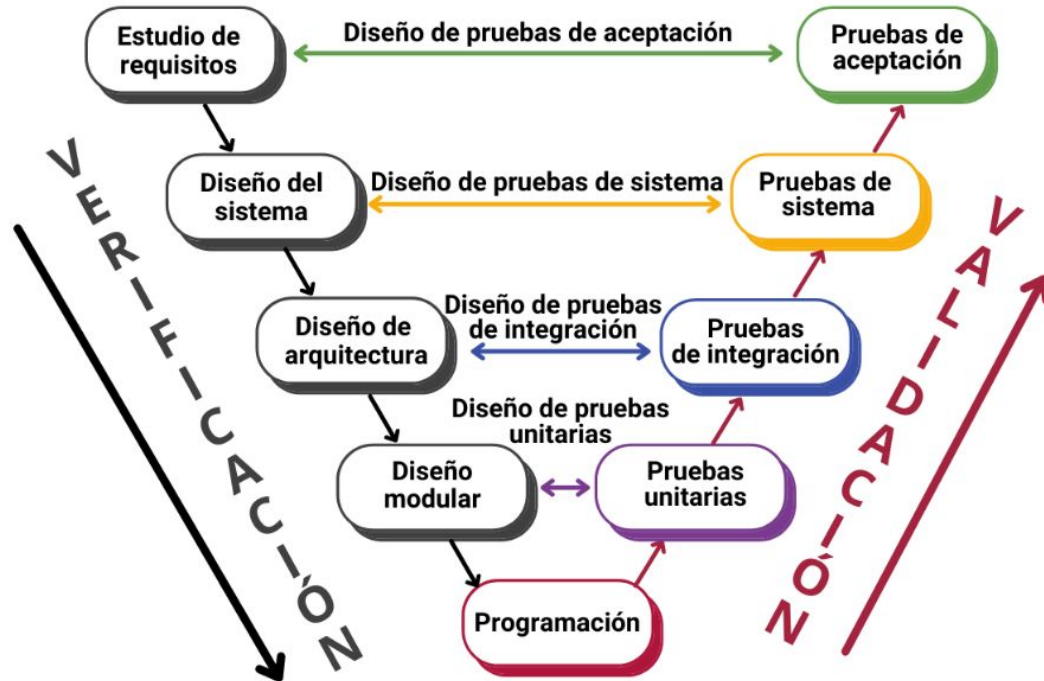
Planificación de las pruebas

Mediante la realización de pruebas de software, se van a realizar las tareas de verificación y validación del software.

- **Verificación:** Se enfoca en confirmar si el software se ha construido correctamente y cumple con las especificaciones y los estándares definidos. Implica revisar el código, realizar pruebas unitarias y de integración, y asegurarse de que el software se haya desarrollado según el diseño y los requisitos previamente establecidos.
- **Validación:** Se concentra en determinar si el software cumple con las necesidades reales del usuario y si satisface los objetivos del negocio. Implica pruebas funcionales, pruebas de aceptación del usuario (UAT) y asegurarse de que el software sea útil y eficaz para sus usuarios finales.

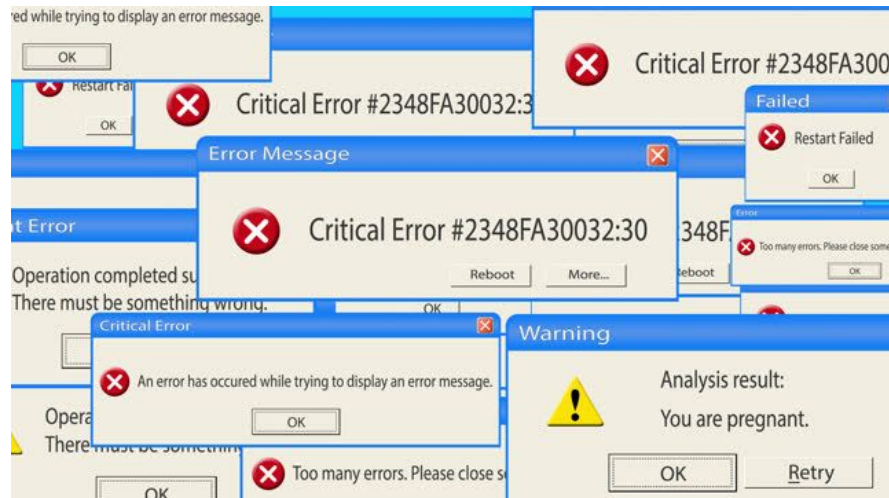
Planificación de las pruebas

El Modelo en V muestra la relación entre cada fase de desarrollo y su correspondiente actividad de prueba. Consiguiendo la calidad de forma interactiva y escalonada, validando y verificando cada uno de los pasos del desarrollo del software.



Planificación de las pruebas

Durante todo el proceso de desarrollo de software, nos vamos a encontrar con un conjunto de actividades, donde es muy fácil que se produzca un error humano. Estos errores humanos pueden ser: una incorrecta especificación de los objetivos, errores producidos durante el proceso de diseño y errores que aparecen en la fase de desarrollo.



Errores de programación

Los errores de programación, también conocidos como bugs, son problemas o defectos en el código de un programa de software. Estos errores pueden causar que el programa no funcione como se esperaba, genere resultados incorrectos o incluso falle por completo. Los desarrolladores de software utilizan técnicas de depuración y pruebas para identificar y corregir estos errores y mejorar la calidad y confiabilidad del software.

Los errores de programación pueden manifestarse en diversas formas, una posible clasificación básica sería:

- Sintáctico (de sintaxis)
- Tiempo de ejecución
- Lógico

Errores de sintaxis

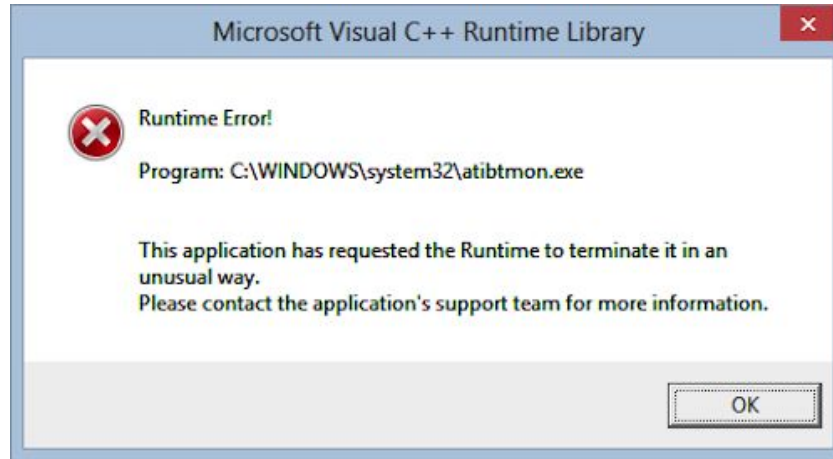
Un error de sintaxis en programación es un fallo que ocurre cuando el código no sigue las reglas gramaticales y estructurales del lenguaje de programación utilizado. Estos errores suelen ser detectados por el compilador o intérprete y se presentan como mensajes de error. Ejemplos comunes incluyen olvidar un punto y coma al final de una línea, utilizar una palabra clave incorrecta o colocar paréntesis incorrectamente. Estos errores son relativamente fáciles de corregir, ya que se refieren a la estructura del código y no a su lógica.

```
int number = "Hello world";
```

Cannot implicitly convert type 'string' to 'int'

Errores en tiempo de ejecución

Un error en tiempo de ejecución en programación ocurre durante la ejecución de un programa y no está relacionado con la sintaxis del código. Estos errores son causados por situaciones inesperadas, como divisiones por cero, acceso a memoria no válida o intentos de conversión de tipos de datos incompatibles. Difieren de los errores de sintaxis porque el código es válido desde el punto de vista gramatical, pero falla debido a condiciones específicas durante la ejecución.



Errores lógicos

Un error lógico en programación se refiere a problemas en la lógica del código que pueden resultar en resultados incorrectos o inesperados. Estos errores no son detectados por el compilador ni generan mensajes de error, ya que el código es sintácticamente correcto. En cambio, afectan la forma en que el programa funciona, lo que puede ser difícil de identificar y corregir. Estos errores suelen surgir debido a una comprensión incorrecta del problema a resolver o una mala implementación de la lógica necesaria para alcanzar el objetivo deseado.

Mi primera calculadora

Numero1:

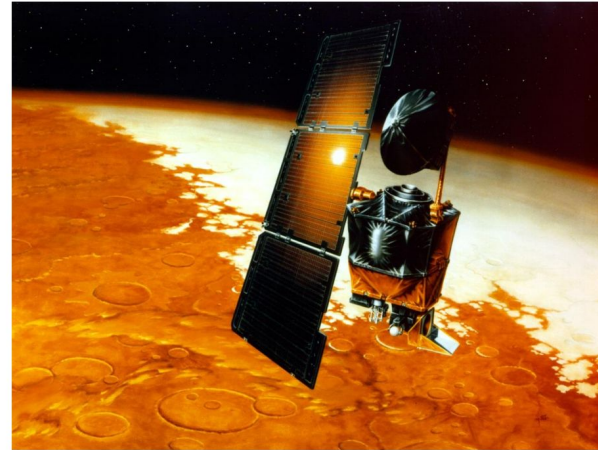
Numero2:

El resultado de $10 + 20 = 1020$

La "Mars Climate" se estrelló en Marte porque la NASA no tradujo kilómetros a millas

Los técnicos olvidaron convertir datos de navegación del sistema métrico decimal al inglés

Éste es el caso de la nave Mars Climate Orbiter, que la pasada semana se estrelló en Marte. Según informó la NASA, el fallo estuvo en una confusión entre millas y kilómetros. Tan simple como eso. **La sonda, construida para navegar según el sistema inglés, recibió antes del despegue las instrucciones de vuelo en el sistema métrico decimal.**



https://elpais.com/diario/1999/10/02/sociedad/938815207_850215.html

Casos de prueba

Los casos de prueba son conjuntos de condiciones y acciones diseñados para evaluar el comportamiento de un programa de software en situaciones específicas. Cada caso de prueba incluye una entrada y se espera que produzca un resultado determinado. Estos casos son esenciales para verificar si el software cumple con los requisitos y las expectativas del usuario. Los casos de prueba pueden ser manuales o automatizados y deben ser detallados y exhaustivos para identificar problemas y garantizar la calidad del software. La planificación y ejecución de casos de prueba son partes cruciales del proceso de aseguramiento de calidad del software.

Casos de prueba

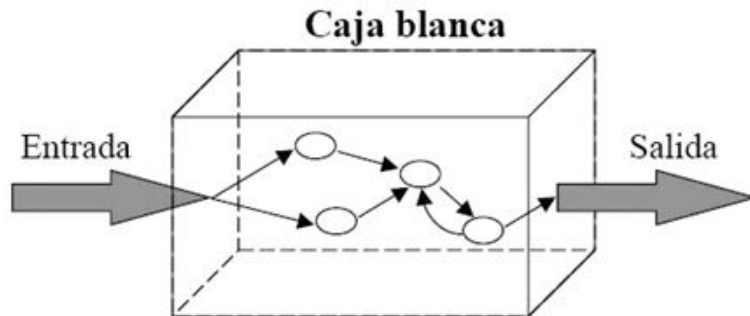
Ejemplo:

ID	E	Nombre CP	Datos de entrada	Resultado esperado
CP1	E1	Registro exitoso	Dni: 14560789. Contraseña: 123123. Datos válidos y existentes en el sistema.	El usuario inicia sesión correctamente en el sistema. El sistema lo redirige hacia la página de inicio, con las funciones disponibles de acuerdo a su rol de usuario.
CP2	E2	Registro cancelado	Dni: N/A. Contraseña: N/A. Click en 'Recuperar contraseña'.	El sistema cierra la pantalla de inicio de sesión y carga la vista necesaria para recuperar una contraseña.
CP3	E3	Registro sin datos obligatorios	Dni: []. Contraseña: [] Botón 'Ingresar'	El sistema muestra un mensaje de error indicando que los campos dni y contraseña son obligatorios. Retorna al paso 3 del flujo normal.
CP4	E3	Usuario inexistente	Dni: 14560789. Contraseña: 321321 Botón 'Ingresar'. La contraseña no es válida pero el usuario existe.	El sistema muestra un mensaje de error indicando que el dni y/o la contraseña son incorrectos. Retorna al paso 3 del flujo normal.

Enfoques en las pruebas

Dos de los enfoques más importantes y ampliamente utilizados son:

- **Pruebas estructurales o de caja blanca:** Se enfocan en la estructura interna del código, requieren acceso al código fuente, y buscan identificar errores de lógica y verificar la cobertura del código.
- **Pruebas funcionales o de caja negra:** Evalúan la funcionalidad del software desde una perspectiva externa, sin acceso al código fuente, centrándose en la entrada y salida para asegurar que el software cumple con los requisitos funcionales.



Tipos de pruebas

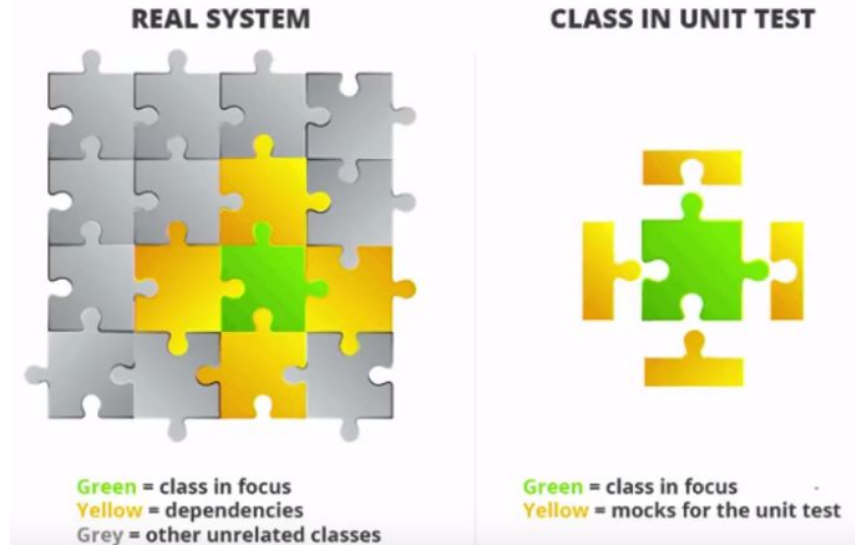
Existen varios tipos populares de pruebas de software que se utilizan ampliamente y de manera conjunta en el desarrollo de aplicaciones para garantizar su calidad y funcionamiento adecuado.

Algunos de los tipos más comunes incluyen:

- Pruebas de unidad (unitarias)
- Pruebas de integración
- Pruebas de aceptación del usuario
- Pruebas de rendimiento
- Pruebas de seguridad
- Pruebas de regresión
- Pruebas de usabilidad
- Pruebas de compatibilidad
- Pruebas de localización e internacionalización
- Pruebas de cobertura

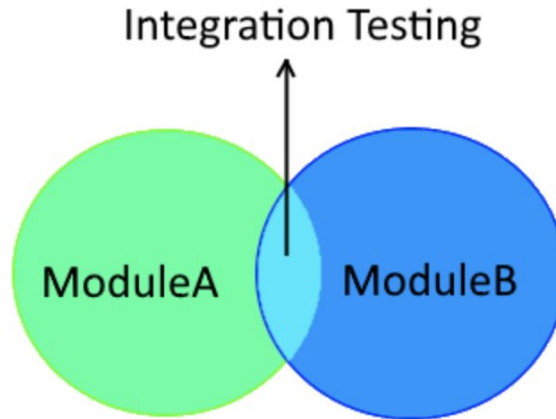
Pruebas unitarias

Las pruebas unitarias son pruebas de software que evalúan unidades individuales de código, como funciones o métodos, de manera aislada. Su objetivo es verificar que cada unidad funcione correctamente y cumpla con sus especificaciones. En este módulo nos centraremos en este tipo de pruebas.



Pruebas de integración

Las pruebas de integración son pruebas en el desarrollo de software que se centran en verificar la interacción adecuada entre diferentes componentes o unidades de código. El objetivo es garantizar que las partes del sistema funcionen correctamente juntas y cumplan con los requisitos preestablecidos. Estas pruebas se realizan después de las pruebas unitarias y antes de las pruebas de sistema para detectar problemas de interacción temprano en el proceso de desarrollo.



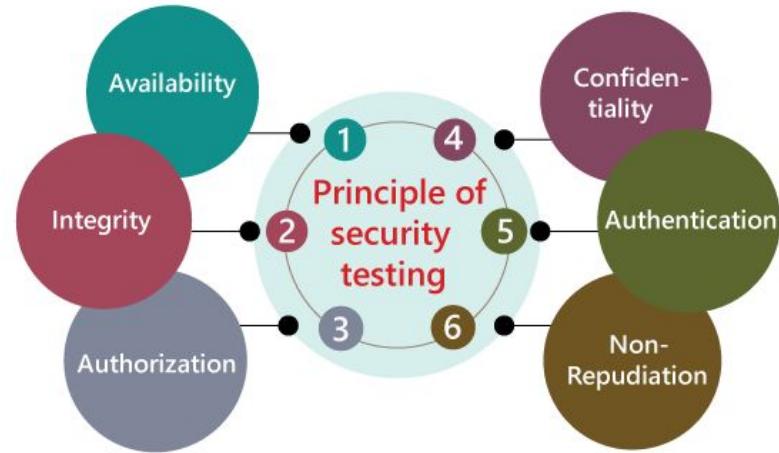
Pruebas de aceptación de usuario

Las pruebas de aceptación de usuario (UAT) son pruebas finales en el desarrollo de software que involucran a los usuarios o representantes del cliente. El propósito principal es validar que el software cumpla con los requisitos y expectativas del usuario final. Estas pruebas evalúan la funcionalidad del software desde la perspectiva del usuario y son cruciales para garantizar que el producto entregado sea satisfactorio para quienes lo utilizarán.



Pruebas de seguridad

Las pruebas de seguridad son evaluaciones del software diseñadas para identificar vulnerabilidades y riesgos de seguridad. Su objetivo principal es garantizar que el software esté protegido contra amenazas y ataques cibernéticos. Estas pruebas incluyen pruebas de penetración y análisis de seguridad para detectar debilidades y fortalecer la seguridad del sistema. Son esenciales para mitigar riesgos y proteger la integridad de los datos y la privacidad de los usuarios.



Más info: <https://www.javatpoint.com/security-testing>

Pruebas de regresión

Las pruebas de regresión son pruebas que se realizan después de realizar cambios en el software para asegurarse de que las modificaciones no hayan introducido nuevos errores o problemas en funcionalidades previamente probadas y funcionales. Su objetivo principal es garantizar que las actualizaciones no afecten negativamente a las partes existentes del software. Estas pruebas son esenciales para mantener la estabilidad del sistema a medida que se realizan cambios y mejoras.

Regression in Software Development

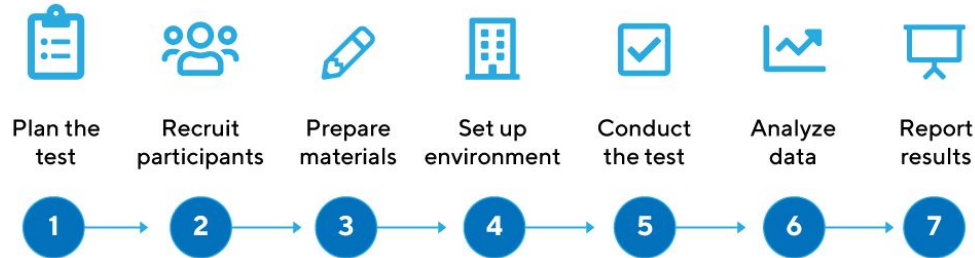
Correcting one bug while simultaneously creating new ones.



Pruebas de usabilidad

Las pruebas de usabilidad son evaluaciones del software centradas en la experiencia del usuario. El propósito principal es garantizar que el software sea intuitivo, eficiente y agradable de usar. Estas pruebas involucran a usuarios reales o representantes del público objetivo para evaluar la interfaz de usuario y la facilidad de navegación. Ayudan a identificar problemas de usabilidad y a mejorar la experiencia del usuario final.

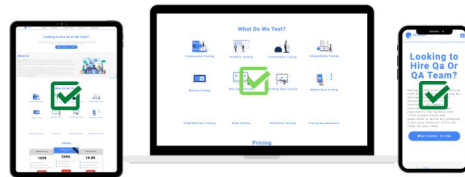
7 Steps to Usability Testing



Pruebas de compatibilidad

Las pruebas de compatibilidad se enfocan en asegurar que el software funcione adecuadamente en diferentes entornos y configuraciones. Se prueban en varios navegadores web, sistemas operativos y dispositivos para verificar su rendimiento y apariencia. Estas pruebas son esenciales para garantizar que la aplicación sea accesible y funcione de manera consistente para todos los usuarios, independientemente de su plataforma de elección.

Compatibility Testing Types



- Hardware
- Operating Systems
- Software
- Network
- Browser
- Devices
- Mobile
- Versions

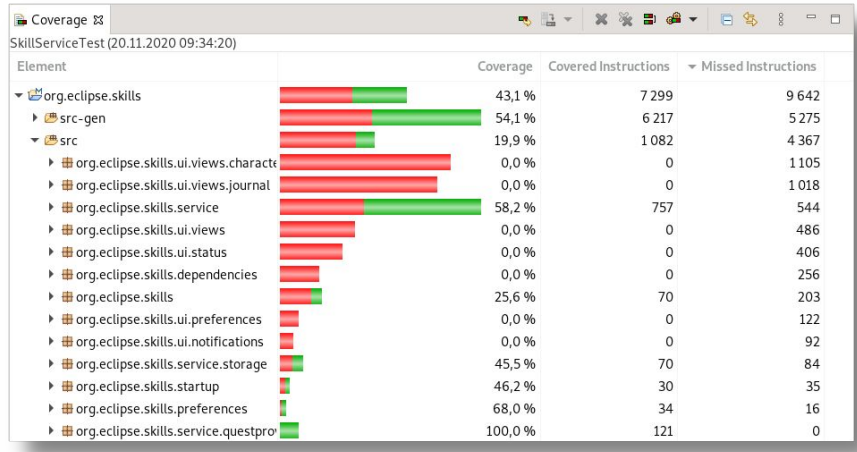
Pruebas de localización e internacionalización

Las pruebas de localización evalúan si el software se adapta correctamente a las especificidades culturales y lingüísticas de una región o mercado específico. Las pruebas de internacionalización aseguran que el software sea diseñado de manera que pueda adaptarse fácilmente a diferentes idiomas y culturas. Ambos tipos de pruebas son esenciales para garantizar que el software sea globalmente accesible y funcione adecuadamente en diversos contextos culturales y lingüísticos.



Pruebas de cobertura

Las pruebas de cobertura son un conjunto de técnicas utilizadas en el desarrollo de software para medir cuánto del código fuente del programa ha sido ejecutado por las pruebas. El objetivo es evaluar la exhaustividad de las pruebas y determinar si se han probado todas las rutas posibles del código. Estas pruebas pueden incluir la cobertura de instrucciones, de ramas y de condiciones. La cobertura de código es útil para identificar áreas no probadas y mejorar la calidad del software al garantizar una mayor cobertura de pruebas. Sin embargo, no garantiza por sí sola la detección de todos los errores en el software.



Not instrumented →

Covered by tests →

Not covered by tests →

```
516 INSTITUTE_TEST_SUITE_P(NativeSequenced,  
517                          ThreadPoolWorkerPoolTest,  
518                          ::testing::Values(PoolExecuti  
519                          test::PoolType::NATIVE,  
520                          test::ExecutionMode::SEQU  
521 #endif  
522  
523 TEST(TaskSchedulerWorkerPoolTest, TestCodeCoverage) {  
524     bool flag = true;  
525     if (!flag) {  
526         int value = 10;  
527         EXPECT_EQ(10, value);  
528     }  
529     EXPECT_TRUE(flag);  
530 }
```

Pruebas de cobertura

Cubrimiento

Considerando que la siguiente función forma parte de un programa mayor, se considera lo siguiente:

- Si durante la ejecución del programa, la función es llamada, al menos una vez, el cubrimiento de la función es satisfecho.
- El cubrimiento de sentencias para esta función, será satisfecho si es invocada, por ejemplo como prueba(1,1), ya que en este caso, cada línea de la función se ejecuta.

```
int prueba(int x, int y)
{
    int z = 0;

    if ((x > 0) && (y > 0))
    {
        z = x;
    }

    return z;
}
```

- Si invocamos a la función con prueba(1,1) y prueba(0,1), se satisfará el cubrimiento de decisión. En el primer caso, la if condición va a ser verdadera, se va a ejecutar z=x, pero en el segundo caso, no.
- El cubrimiento de condición puede satisfacerse si probamos con prueba(1,1), prueba(1,0) y prueba(0,0). En los dos primeros casos (x<0) se evalúa a verdad mientras que en el tercero, se evalúa a falso. Al mismo tiempo, el primer caso hace (y>0) verdad, mientras el tercero lo hace falso.

Pruebas de cobertura

Valores límite

En el código adjunto, aparecen dos funciones que reciben el parámetro x . En la función1, el parámetro es de tipo real y en la función2, el parámetro es de tipo entero. Como se aprecia, el código de las dos funciones es el mismo, sin embargo, los casos de prueba con valores límite van a ser diferente.

Cuando hay que seleccionar una valor para realizar una prueba, se escoge aquellos que están situados justo en el límite de los valores admitidos.

Por ejemplo, supongamos que queremos probar el resultado de la ejecución de una función, que recibe un parámetro x :

- Si el parámetro x de entrada tiene que ser mayor estricto que 5, y el valor es real, los valores límite pueden ser 4,99 y 5,01.
- Si el parámetro de entrada x está comprendido entre -4 y +4, suponiendo que son valores enteros, los valores límite serán -5, -4, -3, 3, 4 y 5.

```
public double funcion1(double x)
{
    if (x > 5)
        return x;
    else
        return -1;
}
```

```
public double funcion2(int x)
{
    if (x > 5)
        return x;
    else
        return -1;
}
```

Pruebas de cobertura

Clases de equivalencia

Con las clases de equivalencia se pretende cubrir el mayor número de entradas posible. Los posibles valores de entrada, se dividen en un número finito de clases de equivalencia. La prueba de un valor representativo de cada clase, permite suponer que el resultado que se obtiene con él, será el mismo que con cualquier otro valor de la clase.

Cada clase de equivalencia debe cumplir:

- Si un parámetro de entrada debe estar comprendido entre un determinado rango, hay tres clases de equivalencia: por debajo, en y por encima.
- Si una entrada requiere un valor entre los de un conjunto, aparecen dos clase de equivalencia: en el conjunto o fuera de él.
- Si una entrada es booleana, hay dos clases: sí o no.

Los mismos criterios se aplican a las salidas esperadas: hay que intentar generar resultados en todas y cada una de las clases

Pruebas de cobertura

Clases de equivalencia

En este ejemplo, las clase de equivalencia serían:

- Por debajo: $x < 0$
- En: $x > 0$ y $x < 100$
- Por encima: $x > 100$

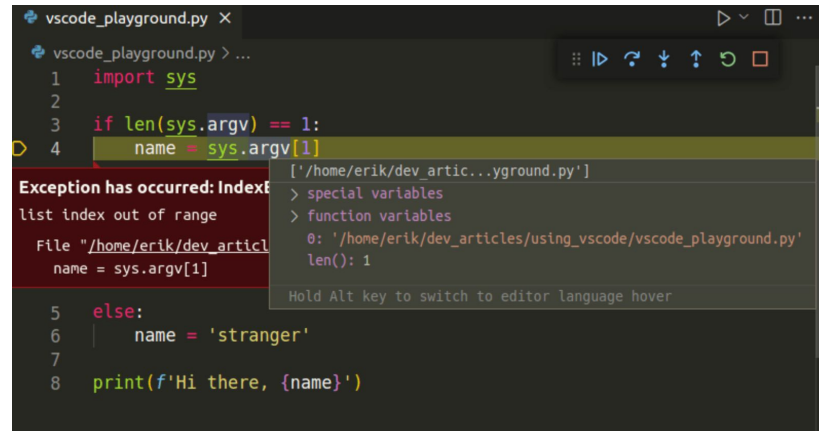
Los respectivos casos de prueba, podrían ser:

- Por debajo: $x = 0$
- En: $x = 50$
- Por encima: $x = 100$

```
public double function(double x)
{
    if (x > 0 && x < 100)
        return x + 2;
    else
        return x - 2;
}
```

Depuración de código

La depuración de código es el proceso de identificar, analizar y corregir errores o fallos en un programa de software. Implica revisar el código en busca de problemas de sintaxis, lógica o comportamiento incorrecto. Los programadores utilizan herramientas de depuración y técnicas para encontrar y solucionar errores, como puntos de interrupción, trazas o inspección de variables. El objetivo es garantizar que el software funcione correctamente, produzca los resultados esperados y se comporte de acuerdo a las especificaciones.



The screenshot shows a VS Code editor window with a file named `vscode_playground.py`. The code is as follows:

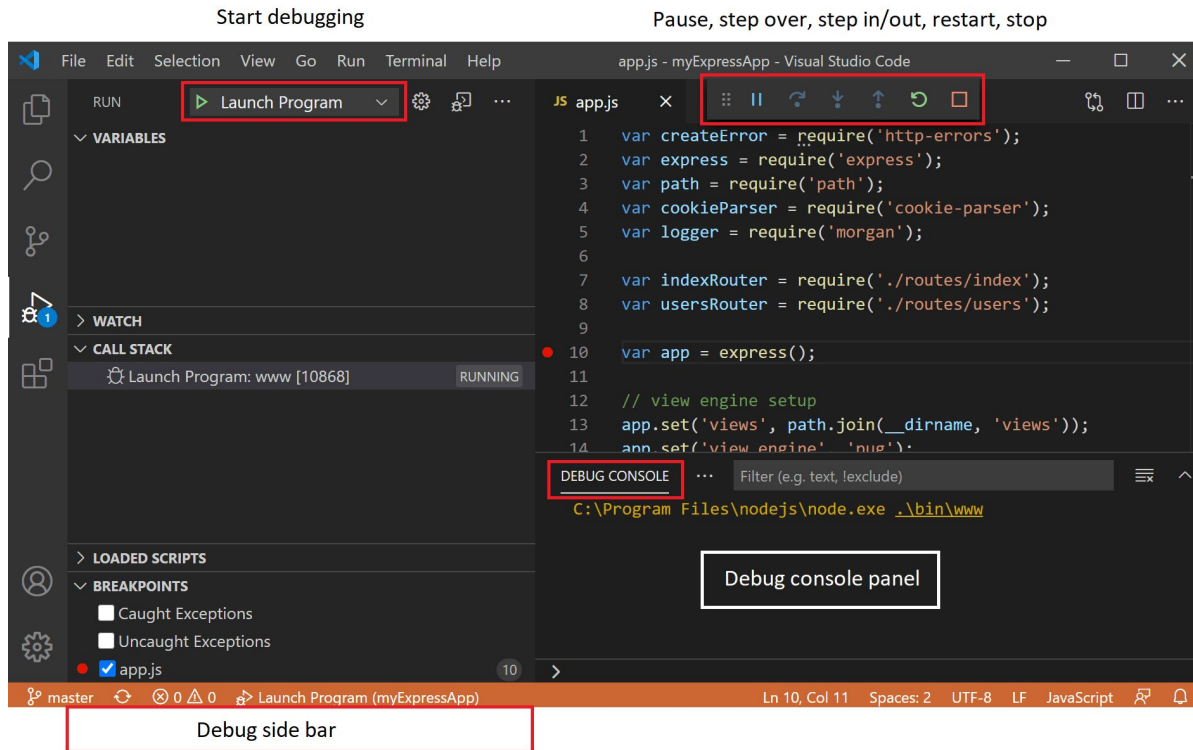
```
1 import sys
2
3 if len(sys.argv) == 1:
4     name = sys.argv[1]
5 else:
6     name = 'stranger'
7
8 print(f'Hi there, {name}')
```

An exception has occurred: `IndexError: list index out of range`. The error message indicates the file `"/home/erik/dev_articles/using_vscode/vscode_playground.py"` and the line `name = sys.argv[1]`. A debug console is open, showing the following output:

```
[ '/home/erik/dev_articles/using_vscode/vscode_playground.py' ]
> special variables
> function variables
0: '/home/erik/dev_articles/using_vscode/vscode_playground.py'
len(): 1
```

Below the console, a message says "Hold Alt key to switch to editor language hover".

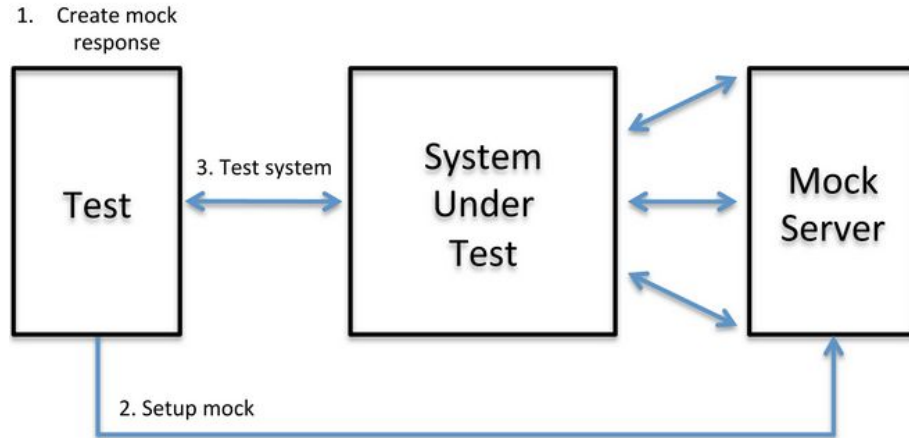
Herramientas de depuración



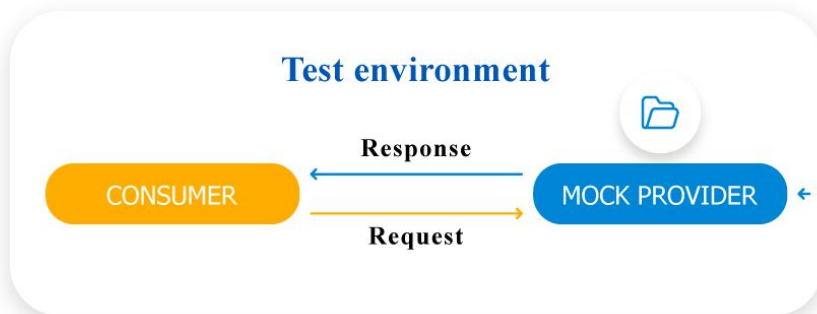
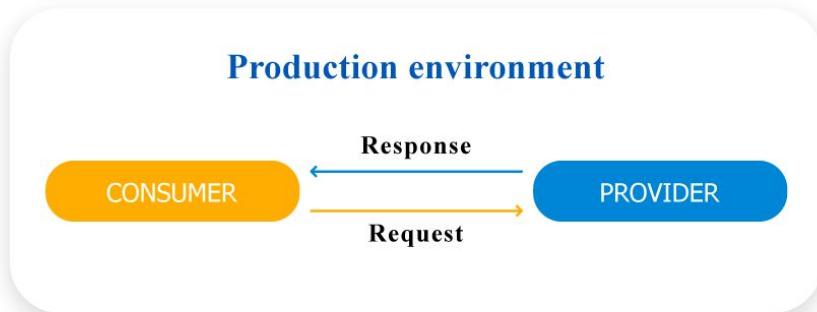
<https://code.visualstudio.com/docs/editor/debugging>

Mock

Un mock es un objeto o componente simulado que se utiliza en pruebas para reemplazar un componente real. Los mocks imitan el comportamiento de un componente real pero se controlan y configuran específicamente para las pruebas. Esto permite aislar la funcionalidad que se está probando y eliminar dependencias externas, lo que hace que las pruebas sean más controladas y repetibles. Los mocks son útiles para verificar la interacción entre componentes y garantizar que una unidad de código se comuniquen correctamente con otros sin necesidad de usar implementaciones reales.



Mock

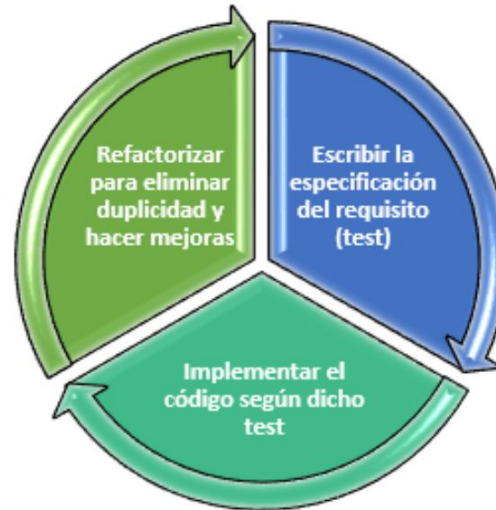


Desarrollo Dirigido por Pruebas (TDD)

El desarrollo dirigido por pruebas (TDD) es una metodología de desarrollo de software que se centra en escribir pruebas antes de escribir el código de la funcionalidad. El ciclo de TDD implica tres pasos:

1. Escribir una prueba que falle.
2. Escribir el código mínimo necesario para que la prueba pase.
3. Refactorizar el código para mejorar su calidad.

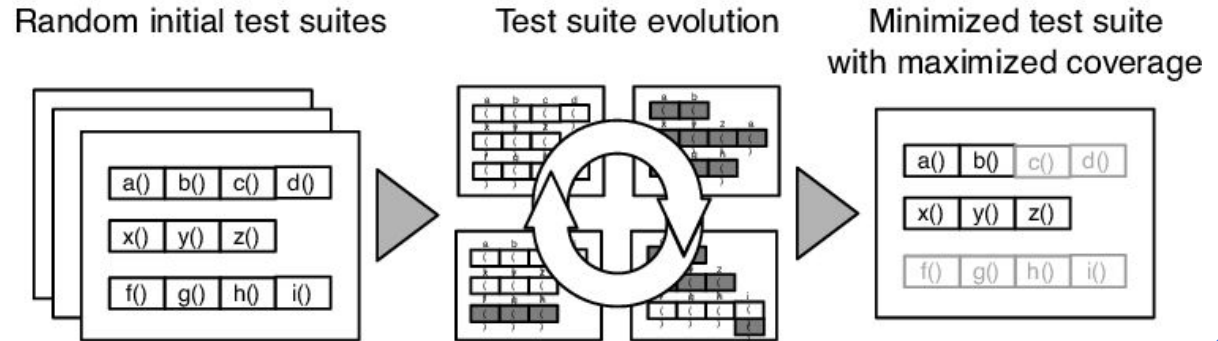
TDD fomenta la creación de código más limpio, confiable y orientado a los requisitos, al tiempo que acelera la detección temprana de errores. Es una práctica clave en la programación ágil y la entrega continua.



Pruebas concólicas (Concolic testing)

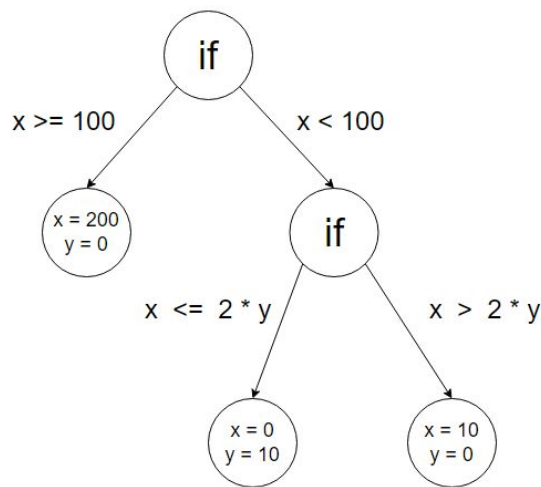
Las pruebas concólicas son una técnica de pruebas de software que combina la ejecución simbólica y la ejecución concreta. Utilizan valores simbólicos para algunas variables del programa y valores reales para otras. Esto permite explorar múltiples caminos de ejecución y encontrar problemas como errores de seguridad o excepciones inesperadas. Las pruebas concolicas son especialmente útiles para analizar programas complejos y verificar su comportamiento en diversas situaciones. Ayudan a identificar vulnerabilidades y mejorar la calidad y seguridad del software.

EvoSuite



Pruebas concólicas (Concolic testing)

```
static void Funcion (int x, int y)
{
    if (x < 100)
    {
        if (x > 2 * y)
        {
            Console.WriteLine("Error");
        }
    }
}
```



Comenzamos con una elección arbitraria para x e y , por ejemplo $x = 200$ y $y = 0$.

En la primera ejecución, falla la primera condición $x \geq 100$ y señala que en la prueba en la línea 2 falló, $x \geq 100$. Esta desigualdad se denomina **condición de ruta** y debe ser verdadera para todas las ejecuciones siguiendo la misma ruta de ejecución que la actual.

Para que el programa siga una ruta de ejecución diferente en la siguiente ejecución, se toma la última condición de ruta encontrada, $x \geq 100$, y se niega, dando $x < 100$.

Luego, se invoca un comprobador de teoremas automatizado para encontrar valores para las variables de entrada x e y (dentro del rango de posibles valores acorde a sus tipos). En este caso, una respuesta válida del demostrador de teoremas podría ser $x = 0$, $y = 10$.

Ejecutar el programa con dicha entrada le permite llegar a la condición interna en la línea 3, lo cual no se toma dado que 0 (x) no es mayor que 20 ($2 * y$). Las condiciones de ruta son $x < 100$, $x \leq 2 * y$. Este último es negado, dando $x > 2 * y$. El probador del teorema busca x e y satisfaciendo $x < 100$, $x > 2 * y$.

Finalmente la respuesta válida sería $x = 10$, $y = 0$, se ejecuta la función y se alcanza el mensaje de error.

Herramientas para pruebas

Lenguaje	Mock	Tipo de prueba					
		Unitarias	Regresión	Integración	Aceptación	Rendimiento	Cobertura
C++	FakeIt	UnitCpp	Boost	Cantata++	ApprovalTests	CPPOCL/test	Gcov
C#	Moq.net	xUnit	Playwright	xUnit	ApprovalTests	BenchmarkDotNet	Coverlet
Java	Mockito	JUnit	Playwright	TestNG	ApprovalTests	JMeter	JaCoCo
PHP	Mockery	PHP Unit	Selenium	Codeception	ApprovalTests	PHPBench	PHP Unit
Python	Unittest	Unittest	Playwright	Pytest	ApprovalTests	Timeit	Coverage.py

Estándares pruebas software

Los estándares que se han venido utilizando en la fase de prueba de software son:

- Estándares British Standard Institution.
 - BS 7925-1, Pruebas de software. Parte 1. Vocabulario.
 - BS 7925-2, Pruebas de software. Parte 2. Pruebas de los componentes software.
- Estándares IEEE de pruebas de software.:
 - IEEE estándar 829, Documentación de la prueba de software.
 - IEEE estándar 1008, Pruebas de unidad
- Otros estándares ISO / IEC 12207, 15289

Problemas en la creación de software

- A finales de los 60 se acuñó el término **crisis del software**:
 - Los proyectos no cumplían los plazos y presupuestos.
- Dificultades inherentes a la naturaleza del software:
 - Complejidad
 - Dificultad de enumerar todos los estados posibles del programa
 - Dominios de aplicación complejos
 - Dificultad de comunicación entre los miembros del equipo
 - Sujeto a continuos cambios

Problemas en la creación de software

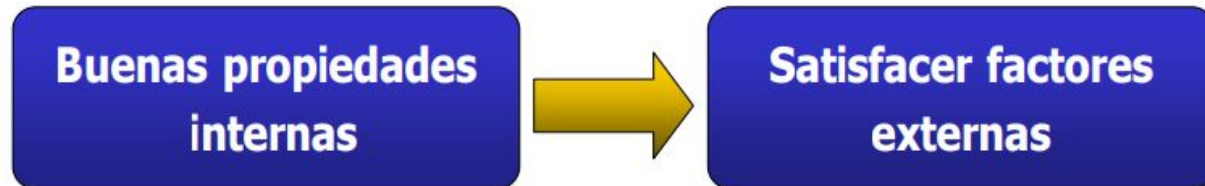
Soluciones:

- Reutilizar código de calidad
- Buenos programadores/diseñadores

Calidad del software

- Factores externos:
 - Pueden ser detectados por los usuarios
 - Calidad externa es la que realmente preocupa
- Factores internos:
 - Sólo lo perciben los diseñadores e implementadores
 - Medio para conseguir la calidad externa

- Objetivo:



Calidad del software

- Factores Externos
 - Robustez
 - Extensibilidad
 - Reutilización
 - Compatibilidad
 - Eficiencia
 - Portabilidad
 - Facilidad de uso
 - Funcionalidad
- Factores Internos
 - Modularidad
 - Legibilidad

Factores de calidad externos

- Robustez:
 - Es la capacidad de los productos software de reaccionar adecuadamente ante situaciones excepcionales.
- Extensibilidad:
 - Es la facilidad de adaptación de los productos software a los cambios en la especificación.
 - La dificultad de adaptación es proporcional al tamaño del sistema.
 - Principios esenciales para facilitar la extensibilidad:
 - Simplicidad de la arquitectura del software
 - Descentralización: módulos autónomos

Factores de calidad externos

- Reutilización:
 - Es la capacidad de un producto software de ser utilizado en la construcción de diferentes aplicaciones
 - Se escribe menos software, luego se puede dedicar más tiempo a mejorar otros factores como la fiabilidad (corrección y robustez)
- Compatibilidad:
 - Es la facilidad de combinar unos elementos software con otros

Factores de calidad externos

- Eficiencia:
 - Es la capacidad de un sistema software de requerir la menor cantidad posible de recursos hardware.
- Portabilidad:
 - Es la facilidad de transferir productos software a diferentes plataformas (entornos hardware y software)

Factores de calidad externos

- Facilidad de uso:
 - Es la facilidad con la que personas con diferentes niveles de experiencia pueden aprender a usar los productos software y aplicarlos a resolver problemas. También incluye la facilidad de instalación, operación y supervisión.
- Funcionalidad:
 - Conjunto de posibilidades ofrecido por un sistema
 - Evitar añadir propiedades de forma incontrolada
 - Mantener constante el nivel de calidad

Otros factores de calidad externos

- Economía:
 - Completarse con el presupuesto asignado
- Integridad:
 - Proteger contra modificaciones y accesos no autorizados
- Facilidad para reparación de errores
- Facilidades de verificación:
 - Datos de prueba y procedimientos para detectar fallos

Factores de calidad internos

- Modularidad
 - Propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados.
 - Alta cohesión: Un módulo con responsabilidades altamente relacionadas y que no hace una gran cantidad de trabajo.
 - Bajo acoplamiento: Un módulo que no depende de demasiados otros módulos.
- Legibilidad:
 - El grado de facilidad con la que una persona puede leer y comprender un fragmento de código fuente, escrito por otra persona.

Consecuencia de los criterios de calidad

- Buena documentación:
 - externa (usuarios) => facilidad de uso
 - interna (desarrolladores) => extensibilidad
 - interfaz del módulo => extensibilidad y reutilización
- Pueden entrar en conflicto. Por ejemplo:
 - Eficiencia y portabilidad
 - Economía y funcionalidad