**3. Determine qué nivel de pruebas (unidad, sistema, integración y aceptación) es posible crear según las características el software y diseñe los pasos de prueba**

**Tipos de pruebas de software**

**Hay muchos tipos diferentes de pruebas de software, cada una con objetivos y estrategias específicos:**

**Prueba de aceptación: verifica si todo el sistema funciona según lo previsto.**

**Pruebas de integración: asegura que los componentes o funciones del software operen juntos.**

**Pruebas de unidad: valida que cada unidad de software funcione según lo esperado. Una unidad es el componente de prueba más pequeño de una aplicación.**

**Pruebas funcionales: verifica funciones mediante la emulación de escenarios de negocio, en función de los requisitos funcionales. La prueba de caja negra es una forma común de verificar funciones.**

**Pruebas de rendimiento: prueba cómo funciona el software bajo diferentes cargas de trabajo. Las pruebas de carga, por ejemplo, se utilizan para evaluar el rendimiento en condiciones de carga reales.**

**Pruebas de regresión: verifica si las nuevas características rompen o degradan la funcionalidad. Las pruebas de cordura se pueden utilizar para verificar menús, funciones y comandos a nivel superficial, cuando no hay tiempo para una prueba de regresión completa.**

**Pruebas de estrés: prueba cuánta tensión puede soportar el sistema antes de que falle. Considerado como un tipo de prueba no funcional.**

**Pruebas de usabilidad: valida qué tan bien un cliente puede usar un sistema o una aplicación web para completar una tarea.**

**Pruebas unitarias**

La prueba unitaria es el proceso de separar unidades y ejecutar pruebas independientes en cada una. Este artículo incluye una guía completa sobre cuándo simular pruebas unitarias y algunos consejos útiles para las pruebas unitarias de C y C++.

**Ejemplo práctico de cuándo simular pruebas unitarias de código C/C++**

Las interfaces simuladas pueden hacer que el trabajo de prueba sea mucho más fácil. En lugar de que su unidad llame a otros, puede llamar a una interfaz simulada. Su código de prueba puede interponerse en todos los lados de la unidad que desea probar y luego examinar todas las salidas y manejar todas las entradas.

Digamos que en el siguiente ejemplo de código, queremos probar la unidad/función de la barra y hacer que llame a una función foo falsa/simulada. Para hacerlo, necesitamos una forma de reemplazar y controlar la llamada a foo(). Hay varios enfoques de simulación para hacer esto y Parasoft puede automatizar gran parte de esto por usted. En este ejemplo y por motivos de simplicidad, se utiliza una macro (#define FOO) para controlar el colaborador foo().

|  |
| --- |
| #ifdef TEST  #define FOO mock\_foo  #else  #define FOO foo  #endif    int mock\_foo(int x)  {    return x;  }    int bar(int x)  {    int result = 0;    for (int i = 0; i < 10; i++)    {      result += FOO(i + x);    }    return result;  } |

El testing de unidad es una parte esencial del proceso de desarrollo de software para garantizar que cada unidad individual de código funcione correctamente. Aquí te presento un ejemplo simple de pruebas de unidad en lenguaje C utilizando el marco de pruebas Unity. Unity es un marco de pruebas de unidad ligero y fácil de usar para C.

Supongamos que tienes una función simple que suma dos números en un archivo llamado calculator.c:

c

Copy code

// calculator.c

int sum(int a, int b) {

return a + b;

}

Ahora, puedes escribir pruebas de unidad para esta función utilizando Unity. Aquí tienes un ejemplo de archivo de prueba test\_calculator.c:

c

Copy code

// test\_calculator.c

#include "unity.h"

#include "calculator.c"

void test\_sum\_positive\_numbers(void) {

TEST\_ASSERT\_EQUAL\_INT(5, sum(2, 3));

}

void test\_sum\_negative\_numbers(void) {

TEST\_ASSERT\_EQUAL\_INT(-5, sum(-2, -3));

}

void test\_sum\_mixed\_numbers(void) {

TEST\_ASSERT\_EQUAL\_INT(0, sum(-2, 2));

}

int main(void) {

UNITY\_BEGIN();

// Ejecutar las pruebas

RUN\_TEST(test\_sum\_positive\_numbers);

RUN\_TEST(test\_sum\_negative\_numbers);

RUN\_TEST(test\_sum\_mixed\_numbers);

UNITY\_END();

return 0;

}

Asegúrate de tener el marco de pruebas Unity instalado en tu entorno de desarrollo. Puedes encontrar información sobre cómo instalar Unity en el sitio web oficial: https://github.com/ThrowTheSwitch/Unity

Para compilar y ejecutar las pruebas, puedes utilizar un compilador de C (como GCC) junto con los archivos de código fuente y las bibliotecas de Unity. Por ejemplo:

bash

Copy code

gcc -o test\_calculator test\_calculator.c calculator.c -lunity

./test\_calculator

Esto compilará los archivos y ejecutará las pruebas. Si todo está bien, no deberías ver ninguna salida, ya que Unity imprimirá un mensaje si todas las pruebas son exitosas. De lo contrario, te proporcionará información sobre las pruebas que fallaron.

**Prueba sistema**

El testing de unidad es una parte esencial del proceso de desarrollo de software para garantizar que cada unidad individual de código funcione correctamente. Aquí te presento un ejemplo simple de pruebas de unidad en lenguaje C utilizando el marco de pruebas Unity. Unity es un marco de pruebas de unidad ligero y fácil de usar para C.

Supongamos que tienes una función simple que suma dos números en un archivo llamado **calculator.c**:

cCopy code

// calculator.c int sum(int a, int b) { return a + b; }

Ahora, puedes escribir pruebas de unidad para esta función utilizando Unity. Aquí tienes un ejemplo de archivo de prueba **test\_calculator.c**:

cCopy code

// test\_calculator.c #include "unity.h" #include "calculator.c" void test\_sum\_positive\_numbers(void) { TEST\_ASSERT\_EQUAL\_INT(5, sum(2, 3)); } void test\_sum\_negative\_numbers(void) { TEST\_ASSERT\_EQUAL\_INT(-5, sum(-2, -3)); } void test\_sum\_mixed\_numbers(void) { TEST\_ASSERT\_EQUAL\_INT(0, sum(-2, 2)); } int main(void) { UNITY\_BEGIN(); // Ejecutar las pruebas RUN\_TEST(test\_sum\_positive\_numbers); RUN\_TEST(test\_sum\_negative\_numbers); RUN\_TEST(test\_sum\_mixed\_numbers); UNITY\_END(); return 0; }

Asegúrate de tener el marco de pruebas Unity instalado en tu entorno de desarrollo. Puedes encontrar información sobre cómo instalar Unity en el sitio web oficial: <https://github.com/ThrowTheSwitch/Unity>

Para compilar y ejecutar las pruebas, puedes utilizar un compilador de C (como GCC) junto con los archivos de código fuente y las bibliotecas de Unity. Por ejemplo:

bashCopy code

gcc -o test\_calculator test\_calculator.c calculator.c -lunity ./test\_calculator

Esto compilará los archivos y ejecutará las pruebas. Si todo está bien, no deberías ver ninguna salida, ya que Unity imprimirá un mensaje si todas las pruebas son exitosas. De lo contrario, te proporcionará información sobre las pruebas que fallaron.

**pruebas de qa testing sistema en lenguaje c**

Las pruebas de sistema en el contexto de QA (Aseguramiento de Calidad) implican probar todo el sistema como una entidad completa para asegurarse de que cumple con los requisitos especificados. A diferencia de las pruebas de unidad, que se centran en componentes individuales, las pruebas de sistema evalúan la funcionalidad y el rendimiento del sistema en su conjunto. Aquí tienes un ejemplo básico de cómo podrías estructurar pruebas de sistema en C.

Supongamos que tienes un sistema simple que gestiona una lista de tareas. Aquí hay un ejemplo de código en C que implementa una funcionalidad básica:

cCopy code

// task\_manager.c #include <stdio.h> #include <stdlib.h> #include <string.h> // Estructura para representar una tarea typedef struct { char description[100]; int priority; } Task; // Función para agregar una tarea al sistema void add\_task(Task\* task\_list, int\* task\_count, const char\* description, int priority) { if (\*task\_count < 100) { Task new\_task; strncpy(new\_task.description, description, sizeof(new\_task.description) - 1); new\_task.description[sizeof(new\_task.description) - 1] = '\0'; // Asegurar terminación nula new\_task.priority = priority; task\_list[\*task\_count] = new\_task; (\*task\_count)++; } else { printf("Error: la lista de tareas está llena.\n"); } } // Función para imprimir la lista de tareas void print\_tasks(const Task\* task\_list, int task\_count) { printf("Lista de tareas:\n"); for (int i = 0; i < task\_count; ++i) { printf("%d. %s (Prioridad: %d)\n", i + 1, task\_list[i].description, task\_list[i].priority); } }

Ahora, puedes escribir pruebas de sistema para verificar el comportamiento del sistema. A continuación se presenta un ejemplo simple de archivo de prueba **system\_tests.c** utilizando el marco de pruebas Unity:

cCopy code

// system\_tests.c #include "unity.h" #include "task\_manager.c" void test\_add\_task(void) { Task task\_list[100]; int task\_count = 0; add\_task(task\_list, &task\_count, "Hacer la compra", 2); TEST\_ASSERT\_EQUAL\_STRING("Hacer la compra", task\_list[0].description); TEST\_ASSERT\_EQUAL\_INT(2, task\_list[0].priority); TEST\_ASSERT\_EQUAL\_INT(1, task\_count); } void test\_print\_tasks(void) { Task task\_list[100]; int task\_count = 0; add\_task(task\_list, &task\_count, "Estudiar para el examen", 1); // Redirigir la salida estándar para capturarla freopen("output.txt", "w", stdout); print\_tasks(task\_list, task\_count); fclose(stdout); // Leer el archivo de salida y realizar las aserciones FILE\* output\_file = fopen("output.txt", "r"); char output\_buffer[100]; fgets(output\_buffer, sizeof(output\_buffer), output\_file); TEST\_ASSERT\_EQUAL\_STRING("Lista de tareas:", output\_buffer); fclose(output\_file); } int main(void) { UNITY\_BEGIN(); // Ejecutar las pruebas RUN\_TEST(test\_add\_task); RUN\_TEST(test\_print\_tasks); UNITY\_END(); return 0; }

Este es solo un ejemplo simple. Las pruebas de sistema pueden incluir una variedad de escenarios, como entradas y salidas del usuario, manipulación de archivos, interacción con bases de datos, etc. Asegúrate de adaptar las pruebas según las características y requisitos específicos de tu sistema. Además, ten en cuenta que este es un ejemplo básico y en un entorno de desarrollo real, deberías considerar herramientas más avanzadas y técnicas como mocks y stubs para simular partes del sistema que aún no están implementadas o para aislar ciertas funcionalidades durante las pruebas.

**Principio del formulario**

**Pruebas de integración**

Las pruebas de integración en el contexto de QA (Aseguramiento de Calidad) se centran en verificar que los diferentes módulos o componentes de un sistema funcionen correctamente cuando se combinan. Aquí tienes un ejemplo básico de cómo podrías estructurar pruebas de integración en C.

Supongamos que tienes dos módulos: uno para manejar tareas (task\_manager.c) y otro para gestionar usuarios (user\_manager.c). A continuación, se proporcionan los códigos de ejemplo para ambos módulos:

c

Copy code

// task\_manager.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

char description[100];

int priority;

} Task;

void add\_task(Task\* task\_list, int\* task\_count, const char\* description, int priority) {

if (\*task\_count < 100) {

Task new\_task;

strncpy(new\_task.description, description, sizeof(new\_task.description) - 1);

new\_task.description[sizeof(new\_task.description) - 1] = '\0';

new\_task.priority = priority;

task\_list[\*task\_count] = new\_task;

(\*task\_count)++;

} else {

printf("Error: la lista de tareas está llena.\n");

}

}

void print\_tasks(const Task\* task\_list, int task\_count) {

printf("Lista de tareas:\n");

for (int i = 0; i < task\_count; ++i) {

printf("%d. %s (Prioridad: %d)\n", i + 1, task\_list[i].description, task\_list[i].priority);

}

}

c

Copy code

// user\_manager.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

char username[50];

} User;

void add\_user(User\* user\_list, int\* user\_count, const char\* username) {

if (\*user\_count < 50) {

User new\_user;

strncpy(new\_user.username, username, sizeof(new\_user.username) - 1);

new\_user.username[sizeof(new\_user.username) - 1] = '\0';

user\_list[\*user\_count] = new\_user;

(\*user\_count)++;

} else {

printf("Error: la lista de usuarios está llena.\n");

}

}

void print\_users(const User\* user\_list, int user\_count) {

printf("Lista de usuarios:\n");

for (int i = 0; i < user\_count; ++i) {

printf("%d. %s\n", i + 1, user\_list[i].username);

}

}

Ahora, puedes escribir pruebas de integración para verificar que estos dos módulos funcionen correctamente juntos. Aquí hay un ejemplo de archivo de prueba integration\_tests.c:

c

Copy code

// integration\_tests.c

#include "unity.h"

#include "task\_manager.c"

#include "user\_manager.c"

void test\_add\_user\_and\_task(void) {

User user\_list[50];

int user\_count = 0;

Task task\_list[100];

int task\_count = 0;

add\_user(user\_list, &user\_count, "user1");

add\_task(task\_list, &task\_count, "Hacer la compra", 2);

TEST\_ASSERT\_EQUAL\_STRING("user1", user\_list[0].username);

TEST\_ASSERT\_EQUAL\_STRING("Hacer la compra", task\_list[0].description);

TEST\_ASSERT\_EQUAL\_INT(2, task\_list[0].priority);

TEST\_ASSERT\_EQUAL\_INT(1, user\_count);

TEST\_ASSERT\_EQUAL\_INT(1, task\_count);

}

int main(void) {

UNITY\_BEGIN();

// Ejecutar las pruebas

RUN\_TEST(test\_add\_user\_and\_task);

UNITY\_END();

return 0;

}

Este es solo un ejemplo básico, y las pruebas de integración pueden volverse más complejas a medida que el sistema crece. Puedes agregar más pruebas para cubrir diferentes interacciones entre los módulos. Además, es posible que necesites utilizar técnicas como stubs o mocks para simular la interacción con componentes que aún no están completamente implementados o para aislar ciertas funcionalidades durante las pruebas. Asegúrate de ajustar las pruebas según las características y requisitos específicos de tu sistema.

**Pruebas de aceptación**

Las pruebas de aceptación en el contexto de QA (Aseguramiento de Calidad) se centran en verificar que el sistema cumple con los requisitos del cliente y que es aceptable para su uso. Estas pruebas generalmente se realizan en un entorno que simula el entorno de producción. Aquí te doy un ejemplo básico de cómo podrías estructurar pruebas de aceptación en C.

Supongamos que tienes un sistema que gestiona tareas y usuarios. A continuación, se proporcionan los códigos de ejemplo para ambos módulos (**task\_manager.c** y **user\_manager.c**), que se han descrito en respuestas anteriores.

Ahora, puedes escribir pruebas de aceptación para verificar que el sistema funcione correctamente en su conjunto. Aquí hay un ejemplo de archivo de prueba **acceptance\_tests.c**:

cCopy code

// acceptance\_tests.c #include "unity.h" #include "task\_manager.c" #include "user\_manager.c" void test\_add\_user\_and\_task\_integration(void) { User user\_list[50]; int user\_count = 0; Task task\_list[100]; int task\_count = 0; add\_user(user\_list, &user\_count, "user1"); add\_task(task\_list, &task\_count, "Hacer la compra", 2); // Simular la impresión de usuarios y tareas freopen("output.txt", "w", stdout); print\_users(user\_list, user\_count); print\_tasks(task\_list, task\_count); fclose(stdout); // Leer el archivo de salida y realizar las aserciones FILE\* output\_file = fopen("output.txt", "r"); char output\_buffer[100]; fgets(output\_buffer, sizeof(output\_buffer), output\_file); TEST\_ASSERT\_EQUAL\_STRING("Lista de usuarios:", output\_buffer); fgets(output\_buffer, sizeof(output\_buffer), output\_file); TEST\_ASSERT\_EQUAL\_STRING("1. user1", output\_buffer); fgets(output\_buffer, sizeof(output\_buffer), output\_file); TEST\_ASSERT\_EQUAL\_STRING("Lista de tareas:", output\_buffer); fgets(output\_buffer, sizeof(output\_buffer), output\_file); TEST\_ASSERT\_EQUAL\_STRING("1. Hacer la compra (Prioridad: 2)", output\_buffer); fclose(output\_file); } int main(void) { UNITY\_BEGIN(); // Ejecutar las pruebas de aceptación RUN\_TEST(test\_add\_user\_and\_task\_integration); UNITY\_END(); return 0; }

En este ejemplo, la prueba de aceptación simula la adición de usuarios y tareas, y luego verifica que la salida del sistema (impresión de usuarios y tareas) sea la esperada. Ten en cuenta que este es solo un ejemplo básico, y las pruebas de aceptación pueden volverse más complejas según los requisitos del sistema.

Para compilar y ejecutar las pruebas, puedes seguir un proceso similar al que se mencionó en respuestas anteriores utilizando el marco de pruebas Unity y un compilador de C. Asegúrate de tener Unity instalado en tu entorno de desarrollo.

bashCopy code

gcc -o acceptance\_tests acceptance\_tests.c task\_manager.c user\_manager.c -lunity ./acceptance\_tests

Ajusta estas pruebas de acuerdo con los requisitos específicos de tu sistema y las interacciones esperadas entre los diferentes componentes.

……………..

las diferentes pruebas funcionales y cuándo realizarlas. Por ejemplo: *“la****prueba de humo****se realiza en cada compilación entregada a QA porque verifica la funcionalidad a un alto nivel, mientras que la****prueba de regresión****se realiza cuando los errores se corrigen en versiones posteriores”*

**Tipos de pruebas funcionales**

Existen pruebas funcionales y no funcionales. En esta ocasión te explicaremos las funcionales y cómo funcionaría cada una de ellas.

* Pruebas unitarias
* Pruebas de componentes
* Pruebas de humo
* Pruebas de integración
* Pruebas de regresión
* Pruebas de cordura
* Pruebas de aceptación

Pruebas unitarias

Las pruebas unitarias son las que aseguran que cada célula del código desarrollado en un componente brinde los resultados adecuados. En estas pruebas los desarrolladores observan la interfaz y la especificación de un componente, proporcionando la documentación del desarrollo del código se prueba exhaustivamente, claro que de forma independiente antes de pasar a otra unidad.

Las pruebas unitarias admiten pruebas funcionales al ejercer el código que es más probable que se rompa. Por ello, si usas pruebas funcionales sin pruebas unitarias, puedes experimentar algunas dificultades para diagnosticar pruebas fallidas. Así que tenlas muy presente.

Ir arriba

Prueba de componentes

Las pruebas de componentes se ejecutan de forma independiente para comprobar que el resultado sea el requerido. Su objetivo es verificar las funcionalidades y/o usabilidades de los componentes, aunque no solo se limite a eso.

Para ilustrarla mejor, un ejemplo de esta prueba puede ser cualquier elemento que tenga entrada y deba generar alguna salida. Puede ser el módulo de código, página web, pantallas e incluso un sistema dentro de un sistema más grande, en un componente. Aquí algunos usos de los componentes que puedes probar:

Prueba de UI: para usabilidad y accesibilidad

Prueba de carga: para asegurar el rendimiento

Inyección de SQL: a través de componentes de UI para asegurar la seguridad

Prueba de login: con credenciales válidas e inválidas

Ir arriba

Prueba de humo

Las pruebas de humo se realizan para verificar si las funcionalidades más significativas de la aplicación funcionan o no. De forma que lo más básico del software se ejecute de forma correcta con pruebas sencillas y rápidas.

Es una de las pruebas funcionales más importantes y debería ser la primera que se ejecute en una nueva compilación. La prueba de humo es común y aunque a veces no se tiene claro su concepto. No se trata de actuar pruebas exhaustivas, sino de revisar que la funcionalidad crítica del sistema realmente funciona bien.

Si la prueba es exitosa será entonces una compilación estable. El equipo QA actuará pruebas funcionales para las características o funcionalidades recién agregadas posteriormente o pruebas de regresión según la situación. Por otro lado, si esta no es estable y falla la compilación lo usual es que se devuelva al equipo de desarrollo para solucionar los problemas de compilación y crear una nueva.

Ir arriba

Pruebas de integración

La prueba de integración es uno de los tipos de prueba funcional más común y se realiza de forma automatizada. Se realizan para probar componentes individuales con el objetivo de verificar cómo los módulos, que trabajan de forma individual, funcionan cuando estén integrados.

El objetivo de realizar estas pruebas es porque comúnmente los desarrolladores se enfocan en construir diferentes módulos del sistema simultáneamente y no se centran en otros. Las pruebas de integración permiten que los datos y comandos operativos fluyan entre módulos. Hacer que todo actúe como partes de un solo sistema en lugar de aplicativos aislados.

Usualmente nos ayuda a identificar problemas en las operaciones de la interfaz de usuario, formatos de datos, invocar API, acceso a bases datos, entre otras.

Algunas de las verificaciones que se realizan en las pruebas de integración son:

Prueba de interfaz: en la comprobación de las transferencias de datos entre dos componentes. Prueba de interfaces como servicios web, API, entre otros. Se ejecuta para comprobar que los componentes estén sincronizados entre sí. Ayudan a determinar que diferentes funciones, como la transferencia de datos entre los diferentes elementos del sistema, se realizan de acuerdo con la forma en que fueron diseñadas.

Ir arriba

Pruebas de regresión

Es normal que los desarrolladores modifiquen y mejoren las funcionalidades de su desarrollo. Por ello existe una gran posibilidad de que puedan causar ‘efectos’ inesperados en su comportamiento. Estas pruebas de regresión se realizan para asegurar que los cambios o adiciones no hayan alterado ni eliminado las funcionalidades existentes.

El objetivo de las pruebas de regresión es encontrar errores que puedan haber sido introducidos accidentalmente en la compilación existente y así garantizar que los errores eliminados continúen así.

Ir arriba

Prueba de cordura

Si tienes una compilación con modificaciones menores, en vez de ejecutar las pruebas de regresión, realizamos una prueba de cordura.

Con ella podemos determinar que las modificaciones realmente hayan solucionado los problemas. Y que dichas correcciones no hayan generado ningún problema. Usualmente, estas pruebas son sub pruebas de la de ‘Regresión’, ya que están relacionadas con los cambios actuados en el producto. No confundas estas las pruebas de humo con las de cordura por una simples razones:

Las pruebas de humo se inician en la compilación desde el inicio y se inspeccionan las funcionalidades más importantes. Mientras que las de cordura analizan profundamente las compilaciones de software. Es decir, las primeras confirman la estabilidad del producto, mientras que las segundas aseguran la racionalidad del producto.

Ir arriba

Pruebas de aceptación del usuario

Cuando ya hemos seguido e implementado las pruebas que requerimos para nuestro producto, hacemos las pruebas de aceptación. Estas hacen parte de la última fase de este proceso de testing. Aquí los usuarios reales del software lo usan para verificar que cumpla con las tareas requeridas en un ambiente ‘real’. En ocasiones se realiza cuando se hace la entrega del producto “como punto de control final entre todos los tipos de pruebas funcionales”.

Desde el inicio hasta la implementación, el software deberá someterse a varios tipos de pruebas. El objetivo siempre será asegurar la calidad para evitar reprocesos y garantizar las funcionalidades de la aplicación, tanto para el usuario final, como para el cliente.

Sin más, recuerda que estas son las pruebas de aseguramiento de calidad más importantes que puedes implementar para entregar desarrollos productos y/o aplicaciones de otro nivel. Así podrás cumplir con los requerimientos del cliente y entregar soluciones funcionales y de calidad.

……………

**Pasos de casos de pruebas**

Los casos de prueba tienen algunas partes integrales que siempre deben estar presentes en los campos. Sin embargo, cada caso de prueba se puede dividir en 8 pasos básicos.

Paso 1: ID de caso de prueba

Todos los casos de prueba deben llevar ID únicos para representarlos. En la mayoría de los casos, seguir una convención para este ID de nomenclatura ayuda con la organización, la claridad y la comprensión.

Paso 2: Descripción de la prueba

Esta descripción debe detallar qué unidad, característica o función se está probando o qué se está verificando.

Paso 3: Supuestos y condiciones previas

Esto implica que se cumplan las condiciones antes de la ejecución del caso de prueba. Un ejemplo sería requerir una cuenta de Outlook válida para iniciar sesión.

Paso 4: Datos de prueba

Esto se relaciona con las variables y sus valores en el caso de prueba. En el ejemplo de un inicio de sesión por correo electrónico, sería el nombre de usuario y la contraseña de la cuenta.

Paso 5: Pasos a ejecutar

Estos deben ser pasos fácilmente repetibles ejecutados desde la perspectiva del usuario final. Por ejemplo, un caso de prueba para iniciar sesión en un servidor de correo electrónico podría incluir estos pasos:

Abra la página web del servidor de correo electrónico.

Introduzca su nombre de usuario.

Introducir la contraseña.

Haga clic en el botón "Entrar" o "Iniciar sesión".

Paso 6: Resultado Esperado

Esto indica el resultado esperado después de la ejecución del paso del caso de prueba. Al ingresar la información de inicio de sesión correcta, el resultado esperado sería un inicio de sesión exitoso.

Paso 7: Resultado real y condiciones posteriores

En comparación con el resultado esperado, podemos determinar el estado del caso de prueba. En el caso del inicio de sesión por correo electrónico, el usuario iniciará sesión correctamente o no. La condición posterior es lo que sucede como resultado de la ejecución del paso, como ser redirigido a la bandeja de entrada del correo electrónico.

Paso 8: Contraseña errónea

La determinación del estado de aprobado / reprobado depende de cómo se comparan entre sí el resultado esperado y el resultado real.

Mismo resultado = Aprobado

Diferentes resultados = Falla

…………………….

Formato de caso de prueba de unidad estándar

Cada parte de un prueba unitaria bien escrita definirá varios aspectos centrales que incluyen:

Funciones realizadas por la prueba

Datos utilizados en la prueba

Resultado esperado de la ejecución de la prueba

Asegurarse de que la prueba se ejecutó de forma aislada de otras partes del código base

Es importante saber que el formato estándar de las pruebas bien redactadas se compone de las siguientes partes:

Nombre significativo del método de prueba

Datos controlados o simulacros que se utilizarán para realizar pruebas.

Método o unidad bajo prueba (la parte del código que estamos probando)

Aplicar una afirmación

Ejecución de la prueba unitaria de forma aislada

captura de pantalla de código para un caso de prueba unitario bien escrito

¿Existe una plantilla de caso de prueba?

Como se mencionó, existe un formato de caso de prueba estándar. Sin embargo, es probable que la plantilla de caso de prueba varíe de una empresa a otra e incluso de un equipo a otro. En cambio, una plantilla de caso de prueba es el documento con una lista de escenarios de prueba y casos de prueba posteriores.

Ejemplo de caso de prueba de calidad

Aunque los casos de prueba variarán según el tipo de prueba y el campo general de prueba, la construcción de un caso de prueba de calidad se reduce a los pocos elementos confiables anteriores. Recuerde: el nombre del método de prueba debe incluir el método o la unidad bajo prueba y cuál es el resultado esperado.

También debe tenerse en cuenta que cada unidad debe probarse de forma aislada. En este caso, "aislamiento" significa mantener las pruebas enfocadas tanto como sea posible para ejecutar solo la parte de la aplicación que estamos probando.

Este ejemplo proviene de un caso de prueba relacionado con la banca:

Captura de pantalla de código para un caso de prueba relacionado con la banca

Con este nombre de método, sabemos que esta es una prueba unitaria que es:

Probando el método 'isOverDrawn ()'.

El balance utilizado para los datos controlados fue 500.

El resultado esperado es cierto.

Un nombre de método significativo permite que cualquiera que revise los resultados comprenda qué estaba probando la prueba unitaria. Además, indica los datos que se van a probar, el resultado esperado y lo que se probó.

Si la prueba falla, conocer el resultado esperado es fundamental para facilitar la resolución de problemas y asegurando que no se introduzcan regresiones.

Datos del caso de prueba

Los datos utilizados deben ser suficientes para ejecutar la prueba. Para las pruebas unitarias, queremos que sea lo más simple posible probar la unidad más básica de nuestra aplicación. Los datos pueden ser tan simples como crear una cadena o una variable de objeto para la cual puede controlar los datos. O se puede usar un marco simulado para la prueba si una dependencia no está disponible o si necesita que esa dependencia esté en un estado específico.

Tener lo suficiente para probar esa parte si es suficiente. NO es necesario que configure todas las partes de la aplicación para que se ejecute la prueba.

Todo esto afecta cómo se comportará la prueba unitaria, ya que estos son los datos que se utilizan para la ejecución de la prueba unitaria. Como tal, esta parte de las pruebas unitarias es la que consume más tiempo, ya que requiere cierta comprensión del código que está probando para saber qué datos usar para las pruebas.

Manténgalo simple usando solo las partes necesarias para el código que se está probando. Los simulacros son muy útiles en esta fase, ya que le permiten controlar cómo se comportarán los métodos de esos objetos al interactuar con su prueba.

Por ejemplo, dados los siguientes datos:

Captura de pantalla de código que muestra cómo controlar el comportamiento de los objetos.

Evitamos la "clase de cliente real" mediante el uso de una simulación de la "clase de cliente" para probar el aislamiento. No queremos introducir ni configurar otro objeto para esta prueba, ya que agrega otra capa de capacidad de mantenimiento para ese objeto y no afecta el resultado del método bajo prueba.

La siguiente variable a crear es el “saldo inicial”, algo conocido por el conocimiento del código. La siguiente línea muestra el objeto Cuenta que se está creando junto con el simulacro y el Saldo inicial para preparar el método que estamos probando con los datos que acabamos de usar.

Entonces, en este ejemplo, el objeto de la cuenta se configura con el cliente simulado, ya que no nos importan los datos del objeto del cliente y pasamos un saldo inicial que podemos controlar para nuestra prueba.

La siguiente línea define la entrada ya que el método bajo prueba requiere que se use un número. Definimos el "equilibrio" que se utilizará en el método que estamos probando. Luego, el método se ejecuta y el resultado del método se almacena en nuestra variable para que lo usemos más tarde.

Aplicar una afirmación

Una vez que la prueba se puede completar con éxito (ya que se ejecuta de principio a fin sin excepciones ni errores), es el momento de aplicar una afirmación a la prueba unitaria. Sin la afirmación, la prueba unitaria no tiene sentido, ya que no hay nada que imponga para asegurarse de que funciona según lo previsto.

La recopilación de la cobertura de qué líneas se ejecutaron indica qué se ejecutó, pero no proporciona suficientes detalles para determinar lo siguiente:

Si el código se comporta como se esperaba.

Si el código cumple con los objetivos de calidad.

Si los datos devueltos son los datos esperados.

Una afirmación puede ser tan básica como:

Captura de pantalla de código que muestra una afirmación

Siempre que la prueba unitaria contenga una afirmación que verifique el resultado del método bajo prueba, esta es una prueba unitaria significativa.

Captura de pantalla de código para una prueba unitaria significativa con una afirmación

Al aplicar el formato estándar de prueba unitaria, un equipo puede mantener, leer y / o actualizar pruebas fácilmente con más facilidad para ver fácilmente dónde se pueden aplicar más pruebas al resto de la aplicación.

infografía que muestra a un equipo trabajando alrededor de un monitor megagrande. Chico a la izquierda en el teléfono inteligente, chico sentado en la parte superior del monitor trabajando en una computadora portátil, chica en la escalera dibujando un gráfico de barras en el monitor con un lápiz de gran tamaño, chico parado frente al monitor hablando por un megáfono, chica a la derecha del monitor tomando notas.

¿Cuáles son las mejores prácticas para redactar casos de prueba de calidad?

La forma de escribir pruebas y casos de prueba eficaces se puede optimizar con el tiempo. Algunos y las mejores prácticas incluya el uso de títulos sólidos, descripciones sólidas y mantener el lenguaje conciso y claro.

Pero también querrá incluir condiciones previas, suposiciones y los resultados esperados. Toda esta información es relevante para el probador de software, especialmente cuando se determina si el caso de prueba debe ser un "aprobado" o un "error" en su lugar.

Una hoja de trucos para crear casos de prueba que funcionen bien es la siguiente:

Mantenga las cosas simples y transparentes.

Haga que los casos de prueba sean reutilizables.

Mantenga únicos los ID de los casos de prueba.

La revisión por pares es importante.

Los casos de prueba deben tener en cuenta al usuario final o los requisitos definidos.

Especifique los resultados esperados y los supuestos.

Gráfico que muestra un plan de prueba que incluye los conjuntos de pruebas A, B y C

Simple, único, específico, abierto a comentarios y enfocado en la reutilización: esa es la forma de un gran caso de prueba. Para obtener una visión más visual de cómo escribir un caso de prueba de calidad, consulte el seminario web de Parasoft sobre el tema.

Conjunto de pruebas frente a plan de pruebas

El otro aspecto de un caso de prueba implica conjuntos de pruebas y planes de prueba. Estos difieren en formas clave y ambos son vitales para el desarrollo preciso de casos de prueba.

Mejore las pruebas unitarias para Java con automatización

Obtenga el libro electrónico

¿Qué es una suite de pruebas?

Un conjunto de pruebas entra en juego para los casos de prueba en lo que se refiere al código fuente, la colección de dependencias o el conjunto de pruebas que se realizarán en el código. Los conjuntos de pruebas le permiten categorizar los casos de prueba de manera que se alineen con cualquier análisis o necesidad de planificación.

Esto significa que las funciones principales del software pueden tener su propio conjunto de pruebas, mientras que otro conjunto de pruebas es para un tipo de prueba específico, como humo o seguridad. Piense en los conjuntos de pruebas como una estantería para organizar sus casos de prueba.

¿Qué es un plan de prueba?

Por el contrario, un plan de prueba se parece más al paraguas que cubre todas las suites de prueba. Si los casos de prueba son libros y los conjuntos de pruebas son estanterías, los planes de prueba son la sala que contiene la estantería.

Generalmente, los planes de prueba se configuran en términos de pruebas manuales, pruebas automatizadas y un formato general de cómo realizar las pruebas. Probarán el software desde la base utilizando conjuntos de pruebas y casos de prueba antes de implementar cambios o agregar nuevas funciones.

Infografía que muestra a un tipo con camisa naranja y pantalón negro sentado y trabajando en un escritorio con un monitor.

Las mejores herramientas de escritura de casos de prueba

Parasoft generalmente desarrolla sus herramientas y suites con la teoría de “George Jetson” en mente. Es decir que queremos que nuestros clientes puedan “presionar un botón” y tener todo bien cuidado. Si bien esto no es totalmente realista, las herramientas que tienen este enfoque en la automatización son las mejores para usar cuando se trata de escribir casos de prueba.

No solo pueden ayudar con la automatización, sino que también pueden ayudar desde el principio del desarrollo. Después de todo, es muy fácil empantanarse con pequeños detalles o características. Uno podría olvidar que el software solo tiene que funcionar primero. Ahí es donde entra en juego una herramienta de prueba unitaria de Java como Parasoft Jtest.

Esta herramienta permite tanto a los principiantes como a los expertos mejorar sus habilidades de prueba unitaria más rápidamente, así como la experiencia de prueba unitaria. Después de establecer una base, ejecuta las pruebas unitarias y luego guía al usuario para garantizar que las pruebas sean significativas. Cuando puede comprender el tipo de cosas que debe buscar en una prueba, la redacción de casos de prueba se vuelve menos intimidante.

Principio del formulario

4. Determine tipos de pruebas (de toda la gran variedad) deben hacerse según las características del sistema y cree los escenarios.

Las pruebas de calidad (QA, por sus siglas en inglés) en el contexto del lenguaje C, que es un lenguaje de programación de bajo nivel, se centran en asegurar la calidad del software escrito en C. Aquí hay algunos tipos de pruebas comunes que se pueden aplicar al código escrito en C:

Pruebas Unitarias:

Definición: Pruebas que se centran en validar el funcionamiento de unidades individuales de código, como funciones o métodos.

Herramientas: Frameworks de pruebas unitarias como Unity, Ceedling, y herramientas de aserciones como assert.

Pruebas de Integración:

Definición: Pruebas que evalúan la interacción entre módulos o componentes del sistema.

Enfoque en C: Asegurarse de que las funciones y módulos se integren correctamente.

Pruebas Funcionales:

Definición: Pruebas que se centran en verificar que el software cumpla con los requisitos funcionales especificados.

Enfoque en C: Asegurarse de que las funciones y características del programa se comporten como se espera.

Pruebas de Rendimiento:

Definición: Evaluar el rendimiento del software en términos de velocidad, eficiencia y uso de recursos.

Enfoque en C: Evaluar la eficiencia de algoritmos y la gestión de memoria.

Pruebas de Seguridad:

Definición: Identificar y corregir vulnerabilidades de seguridad en el código.

Enfoque en C: Asegurarse de que no haya vulnerabilidades comunes, como desbordamientos de búfer, y usar prácticas seguras de programación.

Pruebas de Estrés:

Definición: Evaluar cómo se comporta el software bajo cargas de trabajo extremas.

Enfoque en C: Evaluar la estabilidad y la capacidad del sistema para manejar grandes volúmenes de datos.

Pruebas de Usabilidad (si hay una interfaz de usuario):

Definición: Evaluar la facilidad de uso y la experiencia del usuario.

Enfoque en C: Si el programa tiene una interfaz de usuario, evaluar la usabilidad de las funciones implementadas en C.

Análisis Estático del Código:

Definición: Revisar el código sin ejecutarlo para encontrar posibles problemas.

Enfoque en C: Utilizar herramientas de análisis estático del código para identificar posibles errores y mejorar la calidad del código.

Pruebas de Aceptación del Usuario (UAT):

Definición: Pruebas realizadas por usuarios finales para validar que el software cumple con sus necesidades.

Enfoque en C: Garantizar que las funcionalidades implementadas en C satisfacen los requisitos del usuario.

Estos son solo algunos ejemplos y la combinación específica de pruebas dependerá del proyecto y sus requisitos. Además, es importante utilizar buenas prácticas de desarrollo, como revisiones de código y pruebas continuas, para mejorar la calidad del software en el proceso de desarrollo.

**Indique en qué lenguaje de programación está el software creado e investigue: - Herramientas que puede usar, un resumen de ella y la justificación dónde usarla o cómo. Es investigación y documentación, no ejecución.**

Herramientas de pruebas unitarias para C/C++:

CPPUnit: Versión del framework para lenguajes C/C++.

QUnit: Librería para pruebas unitarias en Javascript. Creada por la fundación jQuery, ha sido reescrita para ser independiente de la librería jQuery. libunittest: Librería portable para pruebas unitarias en C++ que usa el nuevo estándar C++11.

CUnit: Entorno para escribir, administrar y correr test unitarios en lenguaje C.

Otras herramientas de pruebas:

NUnit: Versión del framework para la plataforma.NET.

FoxUnit: Entorno OpenSource de pruebas unitarias para Microsoft Visual FoxPro

MOQ: Entorno para la creación dinámica de objetos simuladores (mocks). «MOQ».

--------------------------------------------

Lenguaje c

Existen varias herramientas de QA (Aseguramiento de Calidad) que son utilizadas específicamente para el desarrollo en lenguaje C. A continuación, se presentan algunas herramientas populares que puedes considerar:

Cppcheck:

Descripción: Herramienta de análisis estático para C/C++. Detecta una amplia gama de posibles problemas en el código fuente.

Sitio web: Cppcheck

Valgrind:

Descripción: Herramienta para detectar problemas de memoria, como fugas de memoria y acceso no autorizado a la memoria.

Sitio web: Valgrind

Gcov y LCOV:

Descripción: Utilizados para realizar pruebas de cobertura de código en programas en C.

Sitio web Gcov: Gcov

Sitio web LCOV: LCOV

Check:

Descripción: Framework de pruebas unitarias para C. Proporciona un entorno de prueba simple pero potente.

Sitio web: Check

Unity:

Descripción: Otro framework de pruebas unitarias para C. Es ligero y fácil de usar.

Sitio web: Unity

Ceedling:

Descripción: Herramienta que facilita las pruebas unitarias y de integración en proyectos C con CMock y Unity.

Sitio web: Ceedling

Clang Static Analyzer:

Descripción: Un analizador estático de código que forma parte del conjunto de herramientas Clang. Puede ayudar a encontrar bugs y mejoras en el código C.

Sitio web: Clang Static Analyzer

Frama-C:

Descripción: Herramienta de análisis estático que se enfoca en el análisis de software crítico en términos de seguridad y fiabilidad.

Sitio web: Frama-C

Google Test (googletest):

Descripción: Framework de pruebas unitarias para C++. Es utilizado para escribir y ejecutar pruebas automáticas.

Sitio web: Google Test

Splint:

Descripción: Herramienta de verificación estática que ayuda a encontrar problemas en el código C.

Sitio web: Splint

Estas herramientas pueden ser utilizadas de manera individual o en combinación, dependiendo de las necesidades específicas de tu proyecto. Es importante ajustar la selección de herramientas según el tipo de pruebas que desees realizar y los requisitos de tu proyecto en particular.

……………….

**Herramientas de prueba automatizadas**

Las herramientas de prueba de software automatizadas ayudan a ejecutar pruebas funcionales y de regresión en la aplicación. Estas herramientas deben producir resultados consistentes con los datos entrantes proporcionados.

Mermelada

Mermelada es una extensión del navegador que permite a los evaluadores de software capturar y compartir errores con los ingenieros en 2 clics.

Grabe un video de su pantalla, tome una captura de pantalla o capture una reproducción instantánea de un error y envíela a un ingeniero, con todos los detalles de depuración del desarrollador incluidos automáticamente: registros de la consola, solicitudes de red, especificaciones del dispositivo, etc.

Jam es la primera grabadora de pantalla diseñada específicamente para que los probadores de software informen errores y está integrada con las herramientas de seguimiento de problemas más populares (Jira, Linear, Asana y más).

Selenio

Selenio es como un nombre familiar para las pruebas automatizadas. Es una herramienta portátil de código abierto y proporciona pruebas funcionales de varios módulos de aplicaciones web en plataformas y navegadores. Admite la ejecución de texto en paralelo, lo que ahorra mucho tiempo de prueba. Selenium admite muchos lenguajes como Java, Ruby, C #, Python, etc.

selenio

Selenium ejecuta una herramienta de reproducción para registrar pruebas funcionales, por lo que no es necesario que aprenda el lenguaje de programación de pruebas. El selenio es muy fácil de configurar; tienes que agregar el complemento específico del navegador. Automatiza los navegadores web proporcionando una velocidad y un nivel de eficiencia diferentes que no es posible mediante pruebas manuales.

¿Interesado en aprender selenio? Mira esto Curso Udemy.

Kualitee

Una solución de prueba de software automatizada basada en la nube llamada Kualitee permite a las empresas organizar y automatizar sus procedimientos de prueba. Su objetivo es ahorrar tiempo y mejorar la calidad del software desarrollado y probado por desarrolladores y probadores.

Cuenta con la confianza de las principales organizaciones del mundo, como Emirates, T-Mobile, EMAAR, el gobierno de Ajman y muchas otras.

La herramienta automatiza el proceso de prueba de software ejecutando pruebas automáticamente y brindando a los desarrolladores comentarios instantáneos sobre los resultados.

Además, Kualitee tiene muchas herramientas, incluida la gestión de casos de prueba, la ejecución de pruebas y la gestión de defectos, que ayudan a los desarrolladores a crear y gestionar sus pruebas de software.

Los probadores de software pueden ejecutar pruebas impecables con la ayuda de sus capacidades adicionales, que incluyen la gestión integrada de defectos, un perfil personalizable y ajustable, una alternativa a ALM, etc.

También incluye una variedad de informes que se pueden producir y utilizar para monitorear el desarrollo del procedimiento de prueba. Los evaluadores pueden importar y exportar casos de prueba, datos de ejecución e informes de errores con unos pocos clics.

A través de su aplicación para teléfonos inteligentes fácil de usar, incluso los gerentes monitorean la efectividad de sus equipos, el progreso de los proyectos en curso y mucho más.

Selenio, Asana, Jira, Redmine, Linear y muchas otras herramientas conocidas de gestión y desarrollo de flujos de trabajo se encuentran entre las que Kualitee conecta para una mejor trazabilidad y planificación.

Kualitee es una herramienta popular entre los evaluadores de software porque es fácil de usar y ofrece una amplia gama de funciones. Los usuarios pueden usarlo de forma gratuita durante siete días como parte de su enfoque de precios basado en la plataforma local y en la nube.

Girar la cabeza

Girar la cabeza es la primera plataforma de IA de experiencia digital del mundo que combina la infraestructura de dispositivos globales alojados en la nube y en las instalaciones, la automatización de pruebas y el rendimiento basado en ML y el análisis de la calidad de la experiencia para dispositivos móviles, web, audio y video.

HeadSpin empodera a los equipos de ingeniería, control de calidad, operaciones y productos para garantizar experiencias digitales óptimas en todos los canales de entrega a lo largo del ciclo de vida del desarrollo.

Algunas de las características son las siguientes.

Infraestructura global de dispositivos

Permite pruebas y monitoreo de extremo a extremo con miles de dispositivos en cientos de ubicaciones en un operador real y redes WiFi en todo el mundo. No se requiere SDK.

Automatización de pruebas y API

Se integra a la perfección en CI/CD workflows permitiendo pruebas funcionales y de rendimiento antes y después del lanzamiento. Soporte para todos los marcos de prueba.

Rendimiento, QoE y aprendizaje automático

Revela automáticamente los problemas y señala las causas raíz subyacentes en todas las capas de la pila, desde la cuantificación de QoE de audio/video hasta la visibilidad del código y los paquetes.

Perspectivas para la acción

Ofrece información de regresión y agregación a lo largo del ciclo de vida de la experiencia digital, desde la planificación y el desarrollo hasta la emisión de tickets de problemas y las operaciones. Tendencias, alertas y análisis.

Herramientas de prueba de carga

Las herramientas de prueba de carga ayudan a determinar el tiempo necesario para cargar un sitio, una aplicación o páginas. Estas herramientas evalúan la rendimiento del sitio web y ayudar a garantizar un acceso sin problemas de ida y vuelta al sitio web.

Gatling

Descubra una de las mejores herramientas de pruebas de carga: Gatling para probar, tener éxito e iterar de la mejor manera. Esta herramienta de prueba de código abierto está especialmente diseñada para CI/CD y DevOps para pruebas de carga.

Con Gatling, puede evitar bloqueos y tiempos de respuesta lentos. Detecta rápidamente los errores y problemas en el rendimiento de su aplicación durante el desarrollo. Como resultado, puede obtener una imagen precisa de la experiencia del usuario.

Deje que Gatling asegure su negocio y aumente el rendimiento de su aplicación. Se integra con la tubería de desarrollo y prueba continuamente la carga en sus aplicaciones web. También incluye informes coloridos y una grabadora web que le permite comprender los problemas claramente en tiempo real.

Además, Gatling ofrece una interfaz de gestión para sus necesidades de pruebas diarias que incluye funciones y métricas avanzadas para la automatización y la integración. Gatling está disponible en Scala, Kotlin y Java, que la mayoría de los desarrolladores conocen al desarrollar una aplicación.

Descarga esta solución de código abierto y comienza a probar el rendimiento de tu aplicación. Además, también puede elegir un plan perfecto de alojamiento propio y en la nube para comenzar a probar su código. Con el plan, obtendrá inyectores de carga bajo demanda, informes avanzados, funciones de nivel empresarial, fuentes de datos de Grafana, API públicas, enlaces públicos, comentarios, integración continua, modos múltiples, etc.

¿Está interesado en aprender a ejecutar su primera prueba? Revisar la curso en linea de gatling, gratis.

Estudio de prueba

Telerik Test Studio facilita las pruebas automatizadas. Incluye código y capacidades sin código para probar funcional, API y carga.

Telerik tiene muchas herramientas de interfaz de usuario, y estoy seguro de que habrías oído hablar de Fiddler, un software de depuración.

Testim

Testim es una herramienta de prueba de software automatizada inteligente que utiliza el aprendizaje automático para acelerar el diseño, la ejecución y el mantenimiento de casos de prueba automatizados. Los casos de prueba se pueden ejecutar en múltiples plataformas, incluidos dispositivos móviles.

Testim usa anotaciones para encontrar inconsistencias y errores en el sistema. Los errores que se registran se pueden reproducir automáticamente simplemente haciendo clic en la prueba nuevamente. El rastreador de errores Testim se usa para compartir capturas de pantalla anotadas y ver detalles de errores.

LoadRunner

LoadRunner es adecuado para evaluar el rendimiento de un sitio web o una aplicación web en condiciones de alta carga. Puede simular miles de usuarios que utilizan el software para registrar y analizar el rendimiento de la aplicación. Proporciona análisis y seguimiento fáciles de usar.

LoadRunner

Funciona en todos los entornos empresariales más importantes y admite diferentes tipos de aplicaciones y protocolos. LoadRunner puede ejecutarse de forma independiente o agrupada por varios usuarios a través de la versión empresarial. Tiene un panel interactivo a través del cual los usuarios se pueden administrar fácilmente.

carga web

Herramienta muy poderosa con capacidades de scripting, carga web puede probar escenarios de casos de prueba complejos. Se puede utilizar para probar aplicaciones independientes y en la nube, y es compatible con todos los principales protocolos web, aplicaciones empresariales y tecnologías. Con Webload, es fácil crear y cargar escenarios de prueba, ya que la herramienta proporciona más de 80 tipos de gráficos e informes para identificar y trabajar en problemas de rendimiento en el sistema.

carga web

También permite que la instalación ejecute casos de prueba de rendimiento desde la nube utilizando Amazon EC2 para generar carga. Webload tiene un componente llamado PMM, que puede recopilar estadísticas del lado del servidor durante las ejecuciones de prueba, lo que proporciona datos adicionales para realizar análisis de causa raíz.

Medidor de llamas

Medidor de llamas se utiliza para ejecutar casos de prueba de rendimiento en aplicaciones móviles, API o aplicaciones web para evaluar el rendimiento en cualquier etapa del desarrollo de la aplicación. Asegura la entrega de software de alto rendimiento al final. Puede crear una nueva configuración de prueba obteniendo nuevos datos de Google Analytics e integrándolos.

De la misma manera, puede probar múltiples ubicaciones geográficas e incorporarlas en un solo informe. La herramienta es 100% compatible con código abierto y lista para empresas, proporciona automatización de pruebas inteligente y es ideal para un entorno de prueba continuo.

JMeter

JMeter de Apache es una herramienta de prueba de código abierto que se utiliza para probar el rendimiento de sitios web y aplicaciones web dinámicas. Una persona con muy pocos conocimientos técnicos también puede interpretar los resultados proporcionados por JMeter.

Jmetro

Funciona en varios tipos de servidores como SOAP, FTP, LDAP, JDBC, TCP, etc., para pruebas de carga y rendimiento. Los planes de prueba se almacenan en formato XML y se generan mediante un editor de texto. JMeter también se utiliza para pruebas funcionales y automatizadas.

También puedes echarle un vistazo herramientas de prueba de carga basadas en la nube para realizar pruebas de estrés y capacidad.

Herramientas de gestión de pruebas

Las herramientas de gestión de pruebas ayudan a gestionar todo el ciclo de pruebas de un producto. Una herramienta de gestión de pruebas útil debería poder integrarse con otros marcos de automatización y Integración CI.

Radiografía

Radiografía es una de las herramientas de gestión de pruebas preferidas para pruebas manuales y automatizadas. Proporciona la estructura adecuada para organizar y clasificar conjuntos de pruebas y proporciona resultados de prueba eficientes en menos tiempo.

Radiografía

Se integra perfectamente con marcos de prueba como Jira, Selenium, Junit, etc. Puede establecer condiciones previas personalizadas que se pueden reutilizar y asociar con diferentes pruebas. Xray facilita las integraciones de CI con Bamboo, Jenkins y proporciona una trazabilidad perfecta y una correspondencia entre requisitos, pruebas, errores y ejecución.

También tiene disposiciones para establecer entornos de prueba, planes de prueba y la generación de informes integrados.

carril de prueba

carril de prueba es una herramienta de administración de casos de prueba basada en la web que se puede configurar y usar fácilmente con la nube o la configuración local. Es altamente escalable y personalizable. Puede ver información en tiempo real sobre el progreso de las pruebas a través de paneles interactivos, métricas, informes de actividad, etc. Los casos de prueba automatizados y manuales se pueden administrar y documentar fácilmente mediante capturas de pantalla y comparaciones de resultados esperados con los reales.

carril de prueba

Existe una amplia gama de plantillas integradas y los usuarios también pueden crear plantillas personalizadas. Los equipos pueden establecer hitos, objetivos personales, listas de verificación y notificaciones por correo electrónico para mejorar la productividad. TestRail se puede integrar fácilmente con herramientas CI como Jenkins, JIRA, TFS, etc., y es compatible con contenedores Docker.

panel de prueba

panel de prueba utiliza planes de prueba inspirados en listas de verificación para pruebas ágiles, pruebas exploratorias, gestión de casos de prueba tradicional, BDD resaltado por sintaxis y mucho más. Es una herramienta liviana con un editor controlado por teclado y tiene una interfaz de usuario altamente receptiva impulsada por JavaScript.

panel de prueba

Se puede invitar a varias personas a probar la aplicación a través de invitaciones por correo electrónico, y estos probadores invitados no necesitan cuentas ni acceso de inicio de sesión. Puede agregar casos de prueba en cualquier momento durante la prueba. Testpad proporciona una integración perfecta con JIRA.

Herramientas de seguimiento de defectos

Las herramientas de prueba de software de seguimiento de defectos ayudan en el flujo de trabajo de un extremo a otro y en la gestión de errores. Estas herramientas rastrean el progreso del equipo y registran quién está trabajando en qué módulo.

Jira

Jira se usa popularmente para Gestión ágil de proyectos además del seguimiento y registro de defectos. Está integrado con el entorno de desarrollo para fines avanzados de seguimiento, registro y generación de informes. JIRA es fácil de usar y tiene muchas funciones. Por ejemplo, los filtros se pueden crear fácilmente con un clic con el propio lenguaje de consulta de JIRA.

Los evaluadores pueden crear un punto a punto personalizado workflows para construir y probar software. Puede usar una amplia gama de complementos para adaptarse a diferentes tipos de casos de prueba.

Mantis

Mantis es una herramienta de seguimiento de errores de código abierto simple pero potente. Puede colaborar con los miembros del equipo sin problemas y personalizar completamente la herramienta en diferentes niveles, como creación de problemas, notificaciones de flujo de trabajo, etc. Obtiene un panel interactivo que muestra elementos pendientes, elementos asignados a un usuario en particular y filtros para ver solo elementos seleccionados.

mantis2grande

Puede ver varias versiones del proyecto, lo que permite una mejor gestión de versiones. Mantis tiene su propia función de seguimiento del tiempo; por ejemplo, los usuarios pueden registrar el tiempo dedicado a los problemas. La herramienta crea automáticamente un informe del tiempo total dedicado a diferentes problemas. Mantis es compatible con JetBrains y Eclipse y proporciona integración para varios complementos como TestRail, Slack, HipChat, etc.

También tiene una función de correo electrónico, a través de la cual todos los compañeros de equipo pueden conocer los cambios y actualizaciones realizados por otros usuarios.

Herramientas de prueba de seguridad

Estas herramientas prueban la presencia de vulnerabilidades o la posibilidad de ataques de seguridad en la aplicación o el software.

invicti

invicti puede escanear automáticamente sitios web, aplicaciones web y servicios web en busca de lagunas y vulnerabilidades de seguridad. Puede identificar problemas y proporcionar pruebas de explotación sin falsos positivos. Es un escáner en línea totalmente configurable que no requiere mantenimiento, no presenta complicaciones, es escalable y admite la colaboración a nivel empresarial.

Netparker

Puede escanear cualquier tipo de aplicación web, independientemente de la plataforma o el idioma en el que se construyan. Puede integrar Invicti durante el ciclo de vida del desarrollo y rastrear las vulnerabilidades a través de Jira, GitHub, etc.

Herramientas de prueba API

Como su nombre indica, estas herramientas ayudan a probar protocolos como REST, SOAP, etc.

Jabón UI

Jabón UI funciona para servicios SOAP, REST, JMS y AMF y se ha creado en Java con Swing para la interfaz de usuario. Es una herramienta multiplataforma de código abierto que se utiliza tanto para pruebas funcionales como de carga en API. También puede realizar llamadas HTTP y JDBC.

Características como la cobertura y refactorización de WSDL, el inspector de esquemas, el inspector de XML, los asistentes de XPath, el descubrimiento de API y la interfaz gráfica de usuario fácil de usar, hacen SOAPUI la herramienta más popular y ampliamente utilizada. Puede obtener protección contra virus y piratas informáticos a través de las funciones de prueba de seguridad. También obtiene un análisis de prueba exhaustivo utilizando la función de informes.

SoapUI apoya Prueba de inyección SQL, que utiliza métodos SQL para identificar áreas de aplicaciones potencialmente débiles y vulnerables.

Explora más Herramientas de compilación y prueba de API.

Herramientas de prueba entre navegadores

Estas herramientas prueban la funcionalidad o aplicación de un sitio web en diferentes navegadores como IE, Firefox, Chrome, Safari, etc.

LambdaTest

LambdaTest es una herramienta de prueba entre navegadores basada en la nube. Es escalable y se procesa sin problemas en todos los navegadores en dispositivos móviles y de escritorio. Admite la ejecución de casos de prueba en paralelo y realiza pruebas de navegador en vivo de manera eficiente. Por lo tanto, un sitio web se puede probar localmente (antes de la puesta en funcionamiento) utilizando un túnel basado en SSH para conectar la máquina local y los servidores en la nube.

LambdaTest proporciona un cambio fácil de pruebas receptivas a pruebas en varios navegadores en tiempo real. Puede personalizar el tamaño del monitor para tener la misma ventana gráfica incluso cuando cambia el tamaño de la pantalla. La herramienta tiene una función de desplazamiento inteligente para ignorar los desplazamientos en iframes.

SalsaLabs

SalsaLabs admite más de 700 plataformas de navegador, sistemas operativos y combinaciones de dispositivos, lo que brinda una infraestructura completa para pruebas manuales y automatizadas de aplicaciones móviles y de escritorio utilizando Selenium y otros marcos de prueba.

SalsaLabs

Cubre una amplia gama de plataformas y es ideal para pruebas de JavaScript, pruebas entre navegadores, pruebas manuales y pruebas web funcionales. También es compatible con la integración continua y las pruebas.

Más pruebas de diseño visual y entre navegadores aquí.

Herramientas de prueba móviles

Estas herramientas automatizan la prueba de aplicaciones en dispositivos Android e iOS.

Espresso

Una herramienta de prueba de software móvil para Android de primera categoría, desarrollada por Google, Espresso es un marco simple de código abierto, que es fácil de usar e interpretar. Da resultados rápidos, consistentes, confiables y estables.

Espresso

Puede crear fácilmente pruebas de IU y registrarlas sin escribir ningún script o código. La característica más llamativa de Espresso es su API sencilla, que proporciona escalabilidad y flexibilidad. Espresso también cuenta con sincronización automática entre la aplicación y las pruebas, y se pueden automatizar fácilmente las pruebas usando Java y Junit.

También deberías explorar un escáner de aplicaciones móviles para encontrar riesgos de seguridad.

Para Concluir

El trabajo de un probador de software es fundamental incluso después de usar tantas herramientas. Muchos factores como

Compatibilidad entre navegadores y plataformas,

Facilidad de mantenimiento de herramientas,

Requisitos de hardware / software,

Requerimientos del proyecto,

Soporte al cliente y políticas de actualización,

Costo de la licencia, requisitos de capacitación,

Muchos más deben ser evaluados a fondo por el equipo de pruebas de software antes de elegir la herramienta de prueba de software adecuada para sus proyectos. Las herramientas anteriores se ajustan a la mayoría de los requisitos que pueden ayudar a los probadores en gran medida.

Referencia

<https://es.parasoft.com/blog/unit-testing-c-code-when-to-mock/>

<https://unaqaenapuros.wordpress.com/2020/12/09/066-pruebas-unitarias-iii-herramientas/>

<https://es.parasoft.com/blog/how-to-write-test-cases-for-software-examples-tutorial/>

<https://www.ibm.com/es-es/topics/software-testing>

<https://geekflare.com/es/software-testing-tools/>