Las pruebas de software

Breve descripción:

Las pruebas de software evalúan la calidad del código a través de diversos métodos. Las pruebas funcionales, unitarias y de integración se enfocan en la funcionalidad y la interacción de componentes. Por otro lado, las pruebas no funcionales y de rendimiento miden aspectos como la escalabilidad y la eficiencia. Cada tipo tiene ventajas, como la precisión, y desventajas, como el tiempo requerido.

Junio 2024

Tabla de contenido

Introducción 3

1. Generalidades de las pruebas de software 4

2. Tipos de pruebas y sus características 6

2.1. Pruebas funcionales 7

2.2. Pruebas unitarias 8

2.3. Pruebas de integración 9

2.4. Pruebas no funcionales 11

2.5. Pruebas de rendimiento 12

2.6. Ventajas y desventajas de los tipos de pruebas 13

3. Los casos de pruebas 15

4. Desarrollo guiado por pruebas 17

Ciclo del desarrollo guiado por pruebas 20

Síntesis 21

Material complementario 22

Glosario 23

Referencias bibliográficas 24

Créditos 25

Introducción

El intercambio y manejo de información, es esencial en todos los aspectos, porque involucra los niveles operativos, económicos, culturales e intelectuales de la humanidad.

Como parte fundamental, son los sistemas de información, que se desarrollan para optimizar el cambio y almacenamiento de esta información.

Sanz (2005), afirma que las pruebas de software son parte fundamental en el desarrollo de los sistemas de información, debido a que son el conjunto de procesos encaminados a encontrar posibles fallos de funcionamiento, usabilidad o configuración de un programa o aplicación.

Las pruebas de software, también conocidas como software testing, son la actividad de control de calidad más realizada en los proyectos de desarrollo o mantenimiento de software. En ellas se encuentran errores como los bugs, que a simple vista son insignificantes, pero pueden ser trágicos en los procesos que el programa desarrolla.

Bienvenido a este proceso de aprendizaje.

1. Generalidades de las pruebas de software

Las pruebas software son parte fundamental del proceso de calidad y adopción de buenas prácticas, estas deben realizarse en todo el proceso de desarrollo de software.

"Este proceso debe incluir revisión de los requerimientos, realización de análisis documentales, identificación de defectos, pruebas funcionales y no funcionales, pruebas dinámicas y estáticas, pruebas de integración, informes de confianza en el nivel de calidad, información para la toma de decisiones, planes de mejora continua". Mera (2016)

El mismo autor, afirma que dentro de las actividades de un proceso de prueba, se deben incluir las siguientes fases:

¿Qué debemos tener en cuenta al momento de elaborar una lista de chequeo?

- El control.
- Comprobación de resultados.
- Elaboración de informes de proceso.
- Aplicación objeto de las pruebas.
- Bitácoras de experiencias.

Es de suma importancia realizar una bitácora y una matriz, con el fin de ejecutar los casos de prueba y así concluir si el caso funciona como se esperaba, para agregarlo como una conformidad, o en su defecto, pasaría a ser una no conformidad.

Según la organización ISO (2020), estas pruebas se regulan por la norma ISO/IEC 25000 Sistemas y Requisitos de Calidad y Evaluación de software (SQuaRE).

Las pruebas de software son aplicables a todos aquellos que deseen crear especificaciones de prueba, marcos correspondientes, o crear automatización de pruebas.

2. Tipos de pruebas y sus características

En los últimos años, se han propuesto algunos principios, con el fin de establecer pautas universales para que los programadores de software las adapten a sus procesos de pruebas. Mera (2016), lista los siguientes principios:

- **Las pruebas muestran la presencia de defectos**

  Las pruebas son herramientas que facilitan detectar defectos, no obstante, no garantizan que no haya defectos ocultos y no es una evidencia de que el software esté óptimo.

- **Pruebas tempranas**

  Encontrar los errores en etapas tempranas, permite un importante ahorro de recursos.

- **No es posible realizar pruebas exhaustivas**

  Hay que realizar un análisis de riesgos, para identificar prioridades y tener una óptima toma de decisiones, centralizando esfuerzos y utilizando el talento humano y recursos de manera correcta.

- **Las pruebas dependen del contexto**

  Estas dependen del contexto en el cual se ejecutan, por lo cual, se debe dar más énfasis en las que sean para un sistema crítico, como un software financiero, para el cual se requiere realizar más pruebas, en comparación con otras aplicaciones con niveles críticos bajos.

- **Agrupación de defectos**

  Las pruebas deben agruparse, la mayoría de fallos operativos se enfocan en un número reducido de módulos.

- **La paradoja de los pesticidas**

  No hay que repetir la misma prueba varias veces, por lo cual es importante que los casos de prueba se revisen constantemente y escribir nuevos, con el fin de encontrar más defectos.

- **Ausencia de error**

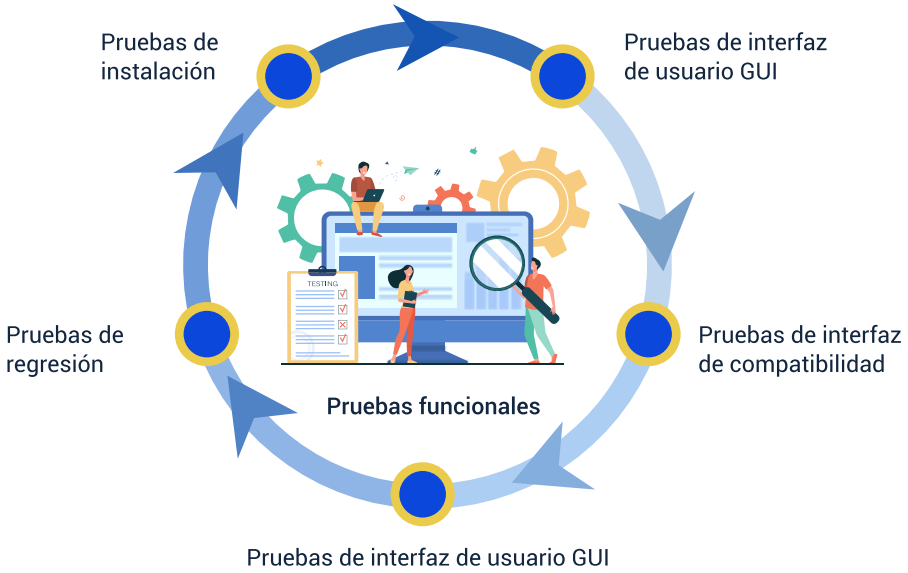
  De nada sirve corregir en el sistema si este no cumple con los requerimientos o necesidades del usuario.

Conozcamos cuáles son los tipos de pruebas que se pueden realizar.

2.1. Pruebas funcionales

IBM (2021), plantea que estas pruebas se basan en la ejecución y revisión de las funciones y en su interoperabilidad con sistemas específicos, se llevan a cabo en todos los niveles de prueba. Buscan evaluar cada una de las opciones con las que cuenta el paquete informático, principalmente en el comportamiento externo del producto o aplicativo, también se las identifica como de tipo caja negra.

Figura 1. Pruebas funcionales



2.2. Pruebas unitarias

Las pruebas unitarias o como también se las conocen, Unit testing, son pequeños test, en los cuales se revisa que el comportamiento de un objeto y su lógica funcionen adecuadamente. Por lo general, se realiza durante la fase de desarrollo de las aplicaciones o software, y es ejecutada por los programadores, aunque también pueden realizarlas los aseguradores de calidad. Las razones por las cuales se desarrollan este tipo de pruebas, son:

- **Prueba uno**

  Son rápidas de realizar, por lo que se pueden realizar en gran cantidad.

- **Prueba dos**

  Demuestran que la lógica del código funciona en todos los casos.

- **Prueba tres**

  Permite a los programadores entender bien el código base, lo que facilitará realizar cambios oportunos.

- **Prueba cuatro**

  Se obtendrá un código de calidad.

En algunos casos, las pruebas pueden hacerse de manera manual, pero lo mejor es usar herramientas que permitan ejecutar el servicio de la mejor manera posible. Existen muchas herramientas en el mercado y estas varían en función del lenguaje de programación que se esté utilizando; a continuación, se mencionarán algunas de las más conocidas:

- **xUnit**

  Es una herramienta de pruebas unitarias para el framework.NET.

- **Junit**

  Es una herramienta de pruebas de aplicaciones Java.

- **Nunit**

  Es una herramienta de pruebas de plataformas .NET.

- **PHPUnit**

  Es una herramienta de pruebas de programación PHP.

Para realizar las pruebas unitarias se deben tener en cuenta los siguientes criterios:

a) Probar un solo código a la vez.
b) Hay que realizar pruebas frecuentemente mientras se programa.
c) Corregir los bugs identificados en las pruebas antes de continuar.
d) Cualquier cambio que se realice también debe pasar la prueba.

2.3. Pruebas de integración

De acuerdo con Mera (2016), estas pruebas son las encargadas de verificar el óptimo acople entre los componentes y probar las interfaces entre los módulos, como el sistema de archivos en integración con el hardware y el componente de validación de usuario con el sistema operativo.

Se deben tener en cuenta los objetos típicos de prueba, los cuales son:

a) Bases de datos de subsistemas.
b) Datos de configuración.
c) Configuración del sistema.
d) Infraestructura.
e) Interfaces.

Ahora, las principales funcionalidades de las pruebas de integración, son: verificar que un gran conjunto de partes de la aplicación funcionen juntas; verificar el correcto ensamble entre componentes, para que funcionen adecuadamente; no necesita resguardo, lo cual es un gran beneficio; se realiza en cada uno de los módulos del sistema y se inicia desde los módulos de abajo hacia arriba.

El mismo autor argumenta que en las pruebas de integración se revisan las interfaces entre componentes o subsistemas, para cerciorarse de que son llamados cuando se necesite y que los datos o mensajes que se transmiten son los adecuados. Existen los siguientes tipos de pruebas de integración:

- **Integración incremental**

  Se prueba el nuevo componente en conjunto con los que ya se probaron y se incrementa progresivamente con los nuevos componentes que se probarán.

- **Descendente**

  También conocidas como top down, se realizan siguiendo el flujo de control, ejemplo, desde el GUI o menú principal y los componentes o sistemas se sustituyen por stubs.

- **Ascendente**

  También conocidas como bottom-up, las pruebas se ejecutan desde la parte inferior del flujo hacia arriba y los componentes o sistemas se sustituyen por controladores.

- **Integración no incremental**

  También conocida como Big-bang, se prueba cada componente por separado y al final, se integran todos de una vez, realizando las pruebas pertinentes.

- **Combinadas**

  Se desarrolla combinando el enfoque ascendente y descendente, pero hay que realizar una planificación, para que los componentes individuales se encuentren en el medio.

2.4. Pruebas no funcionales

En este tipo de pruebas se comprueban los requisitos basados en la operación de un software, no en su funcionalidad. Estas pruebas pueden ayudar a establecer la carga que resiste el producto, si su rendimiento es el adecuado o si es estable a nivel de contacto con el servidor, en otras palabras, nos dice si tienen un bajo desempeño o rendimiento en la producción. Aquí se encuentran:

- **Pruebas de carga**

  Simulan la demanda esperada que puede tener una aplicación y mide el resultado.

- **Pruebas de rendimiento**

  Se calcula la respuesta del programa con diferentes medidas de peticiones del usuario.

- **Pruebas de volumen**

  Se prueba el funcionamiento del programa con ciertos volúmenes de datos.

- **Pruebas de estrés**

  Cuántos usuarios o peticiones puede soportar el programa, con esta prueba se determina si el sistema es estable.

- **Pruebas de configuración**

  Aquí se valida qué efectos en el desempeño tienen los cambios de configuración que se pueden aplicar, como en diferentes hardware, software, versiones y navegadores.

- **Prueba de resistencia**

  Aquí se prueba cómo se comporta una aplicación luego de un uso prolongado.

- **Pruebas de evaluación**

  Se evalúa si la aplicación cumple con los objetivos para la cual fue diseñada.

- **Pruebas de usabilidad**

  Se centran en validar qué tan fácil se utiliza el programa en cuanto a facilidad de aprendizaje, eficiencia, memorización, errores y satisfacción.

2.5. Pruebas de rendimiento

Según IBM (2021), estas pruebas sirven para evaluar lo rápido que el programa realiza una tarea en condiciones específicas de trabajo, también evalúa la escalabilidad, fiabilidad y uso de los recursos. Los objetivos de estas pruebas son:

- **Identificar y localizar problemas de rendimiento**

  Es permitir encontrar un fallo en el rendimiento, y también ayudar a localizar en qué parte está ese problema de rendimiento, para poder solucionarlo.

- **Verificar el cumplimiento de los SLA**

  SLA - Acuerdos de Nivel de Servicio. Se utiliza para medir los tiempos de respuesta del software en condiciones y tiempos específicos.

- **Localizar cuellos de botella**

  Ayudar a detallar en qué lugar se generan estos cuellos de botella, si se debe a problemas de hardware como CPU, memoria, o es el ancho de banda.

2.6. Ventajas y desventajas de los tipos de pruebas

Todo proceso tiene sus ventajas y desventajas, a continuación, se presentan cada una de ellas.

a) Pruebas funcionales

- Optimiza las funcionalidades y la calidad del producto.
- Verifica que la seguridad y la arquitectura sean las adecuadas.
- Asegura que el comportamiento del software y de todas sus funciones sea el esperado.
- Se asegura que el software o aplicación esté libre de errores.

b) Pruebas unitarias

- Simplifica la integración y permite una buena documentación.
- Funciona con el proceso de desarrollo ágil.
- Detección temprana de errores en sus funcionalidades, lo que minimiza tiempo y costos.
- Optimiza la efectividad del código.

c) Pruebas de integración

- Detalla los errores y conflictos en cada módulo y en sus conexiones.

- Mejora la calidad del código.
- Garantiza que las funciones del software serán confiables y estables entre sus módulos.
- Garantiza que todos los módulos del software estén correctamente integrados y funcionen en conjunto como se necesita.

d) Pruebas de rendimiento

- Mejora las características del software o aplicación, perfeccionando el sistema.
- Identifica errores que no se detectan en las pruebas funcionales.
- Localiza los cuellos de botella para solucionarlos y optimizar su rendimiento.
- Valora la escalabilidad y velocidad de la aplicación o website.

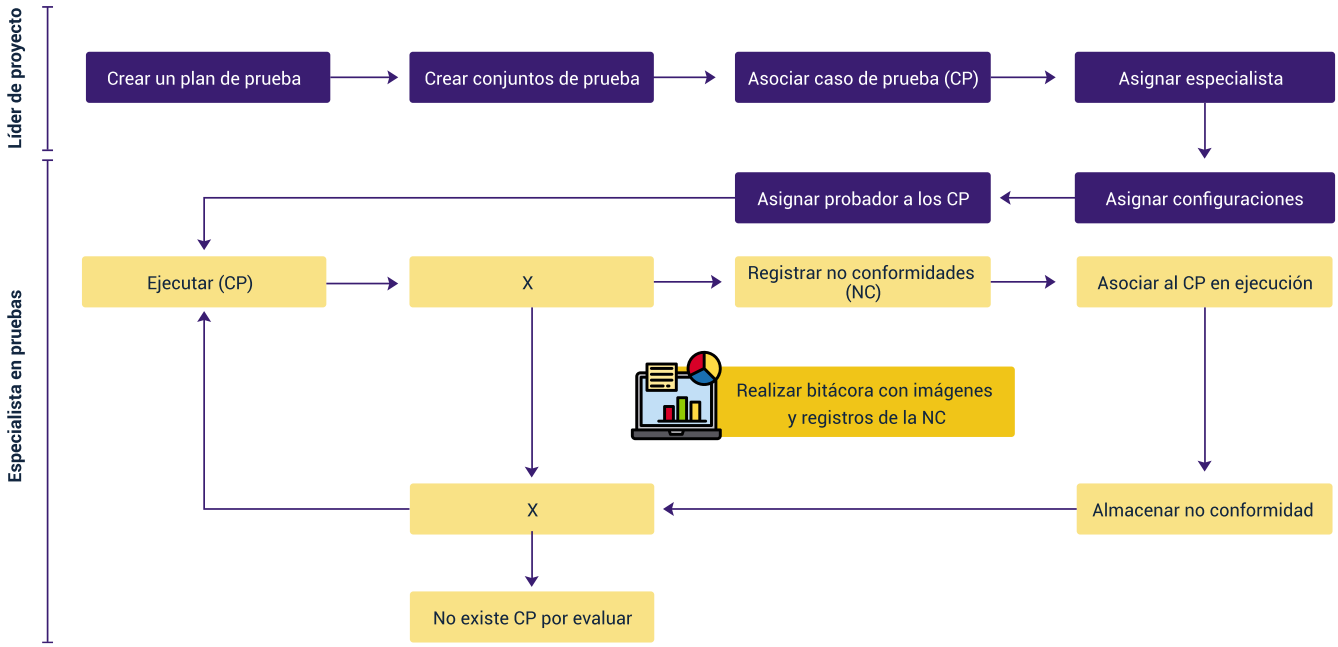
Conclusión de los factores que afectan o determinan la calidad del software

Por lo general, todas las pruebas de software traen beneficios al desarrollador, dependiendo del tipo de la prueba los beneficios pueden ser más altos y enfocados en un fin o propósito específico, dando como resultado una mejor calidad de producto, mejorando su funcionalidad, reduciendo costos, tiempos, recursos humanos y mejorando la experiencia del usuario. Se podría considerar como una desventaja, si se comparan los tipos de prueba, porque puede variar en tiempo, costo, esfuerzo, recursos humanos y exactitud de las pruebas. Sin embargo, la única desventaja de las pruebas de software es no hacerlas, porque el omitir este paso, desencadena una nefasta serie de errores como un producto no funcional y colocarían en tela de juicio la profesionalidad del programador y su equipo de trabajo.

3. Los casos de pruebas

Son los documentos de los productos que se generan al momento de realizar una prueba de software, en los cuales se plasman las precondiciones, entradas y resultados esperados, incluyendo la implementación correcta, la identificación de errores, el chequeo de la calidad y otras informaciones consideradas importantes. En general, no hay una plantilla específica, solo la bitácora, listas de chequeo e imágenes en las que se registran las no conformidades que se encuentran en las pruebas. A continuación se presenta un esquema de casos de prueba.

Figura 2. Esquema de casos de prueba



En el diseño de las pruebas de software, se deben identificar y describir casos de prueba, por lo tanto, para documentar un caso de prueba se debe comprender que este contiene un conjunto de variables o condiciones, en el cual un tester determina si un sistema funciona según lo esperado; así mismo, cabe resaltar que la documentación de un caso de prueba se convierte en una actividad sin dificultad, si se tiene la

información necesaria para su proceso de elaboración, teniendo en cuenta que al momento de probar un software, es de mucho aporte, puesto que se transforma en una herramienta esencial en el proceso de registro, seguimiento y control. A continuación, se nombran los elementos principales que debe contener un caso de prueba.

Tabla 1. Elementos de un caso de prueba

Elemento	Descripción
Identificador	Puede ser alfanumérico o numérico.
Nombre	Nombre del caso de prueba de manera concisa.
Descripción	Objetivo del caso de prueba, también describe qué probará, en ciertas ocasiones se incluye el ambiente de pruebas.
Número de orden de ejecución	Orden en la cual se ejecuta el caso de prueba, en la situación donde se tengan múltiples casos de prueba.
Requerimiento asociado	Si se plantea un caso de prueba, se debe saber a qué requerimiento va asociado, para mantener la trazabilidad.
Precondición	Estado en el cual se debe encontrar el sistema antes de comenzar la prueba.
Postcondición	El estado en que debe encontrarse el sistema luego de ejecutar la prueba.
Resultado esperado	Objetivo que debe ser alcanzado, posterior a la ejecución de la prueba.

4. Desarrollo guiado por pruebas

El desarrollo de software dirigido por pruebas, es una técnica de programación en la cual las pruebas que normalmente son las unitarias, se escriben primero y luego se escribe el código que pase las pruebas. Con esta práctica se consigue, entre otras cosas, un código más robusto, más seguro, más mantenible y una mayor rapidez en el desarrollo.

Según Herranz (2021), cuando el TDD (Desarrollo Dirigido por Pruebas), se combina con el desarrollo ágil, tiene como consecuencia un espacio más amplio, el cual cubre el desarrollo de software, hasta el diseño de software, el cual propone el proceso denominado:

- **Rojo**. Hacer que las pruebas fallen.
- **Verde**. Hacer que las pruebas pasen.
- **Refactory**. Corregir errores lógicos.

"Para usar TDD, el desarrollador de software tiene que cambiar su método de trabajo. Ahora, en vez de crear los casos de uso y después implementarlos, va a intentar convertir un caso de uso en un ejemplo. Si se realiza esto con todos los casos de uso, llegará un momento en el que los ejemplos describirán por sí solos la próxima tarea a implementar". Sánchez (2017)

El proceso de diseño de software, combinando TDD con metodologías ágiles, sería el siguiente:

- **Historia del usuario**

  El cliente escribe su historia de usuario.

- **Aceptación de la historia de usuario**

  Se definen, en conjunto con el cliente, los criterios de aceptación de la historia de usuario, descomponiéndola todo lo que se pueda, para simplificar lo más posible.

- **Punto de aceptación**

  Se selecciona el punto de aceptación más sencillo y se transforma en la prueba unitaria.

- **Verifica**

  Se verifica que esta prueba falla.

- **Prueba correcta**

  Se escriben las líneas de código para hacer que la prueba sea correcta.

- **Automático**

  Se llevan a cabo todas las pruebas de carácter automático.

- **Vuelve**

  Se vuelve a escribir el código para que quede limpio.

- **Volvamos al punto 3**

  Se debe regresar al paso número 3 en cuanto se detecten criterios de aceptación que hagan falta, y se debe repetir el ciclo una y otra vez hasta lograr la aplicación completa.

Es de comentar que este es un proceso de desarrollo relativamente nuevo, el cual está comenzando a ganar terreno y ser conocido en el ámbito del desarrollo de software en las organizaciones a nivel mundial, el cual se basa en evidencias anecdóticas y una serie de evaluaciones empíricas. Hablando del desarrollo tradicional,

las pruebas, se usan para validar y verificar la calidad del software y su creación se realiza después de que existe la funcionalidad.

Nota:

En el TDD, las pruebas adicionales al tema de validación y verificación se usan con fines de especificar, donde primero se desarrolla la prueba y posteriormente la funcionalidad. Esto permite que la calidad del software aumente y la productividad del desarrollador mejore, dado que se lleva un mejor esfuerzo en depurar y se avanza rápidamente con el progreso del producto software.

A continuación, se pueden observar los beneficios del desarrollo guiado por pruebas:

- **TDD**

  Permite una verificación de las funcionalidades desarrolladas en ciclos más pequeños, lo que permite la detección de errores y así mismo, su solución temprana.

- **Implementación**

  La implementación de esta forma de trabajo permitirá elevar la fiabilidad del producto software, no solo desde el punto de vista de errores funcionales, sino avanzar con las necesidades requeridas por el cliente.

- **Optimización**

  Otro de los beneficios es la optimización y mejora en el rendimiento del código, dado que se hace énfasis en escribir solamente el código necesario para la resolución de la necesidad.

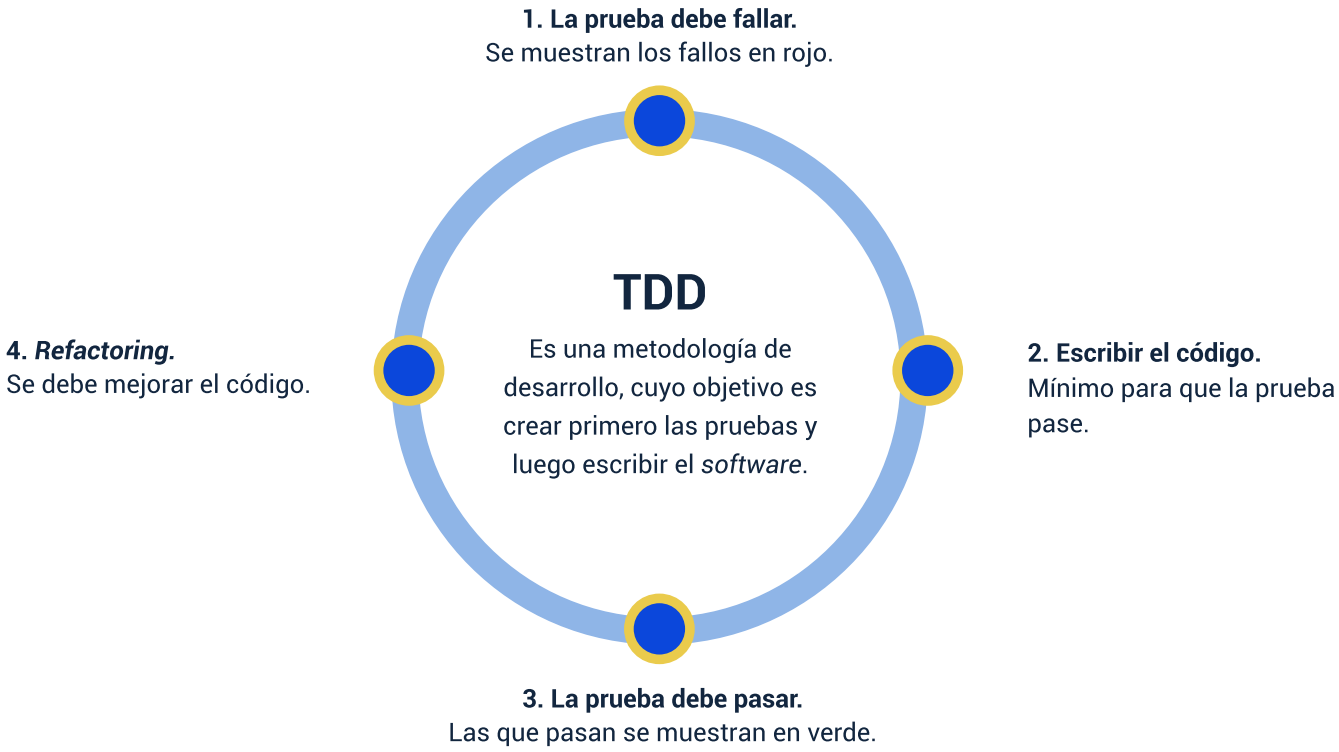
- **Proceso**

  El proceso de realizar pruebas por cada nueva funcionalidad o requisito, facilita el proceso de desarrollo de forma modular, lo que permite la escalabilidad y el futuro mantenimiento del producto software.

Ciclo del desarrollo guiado por pruebas

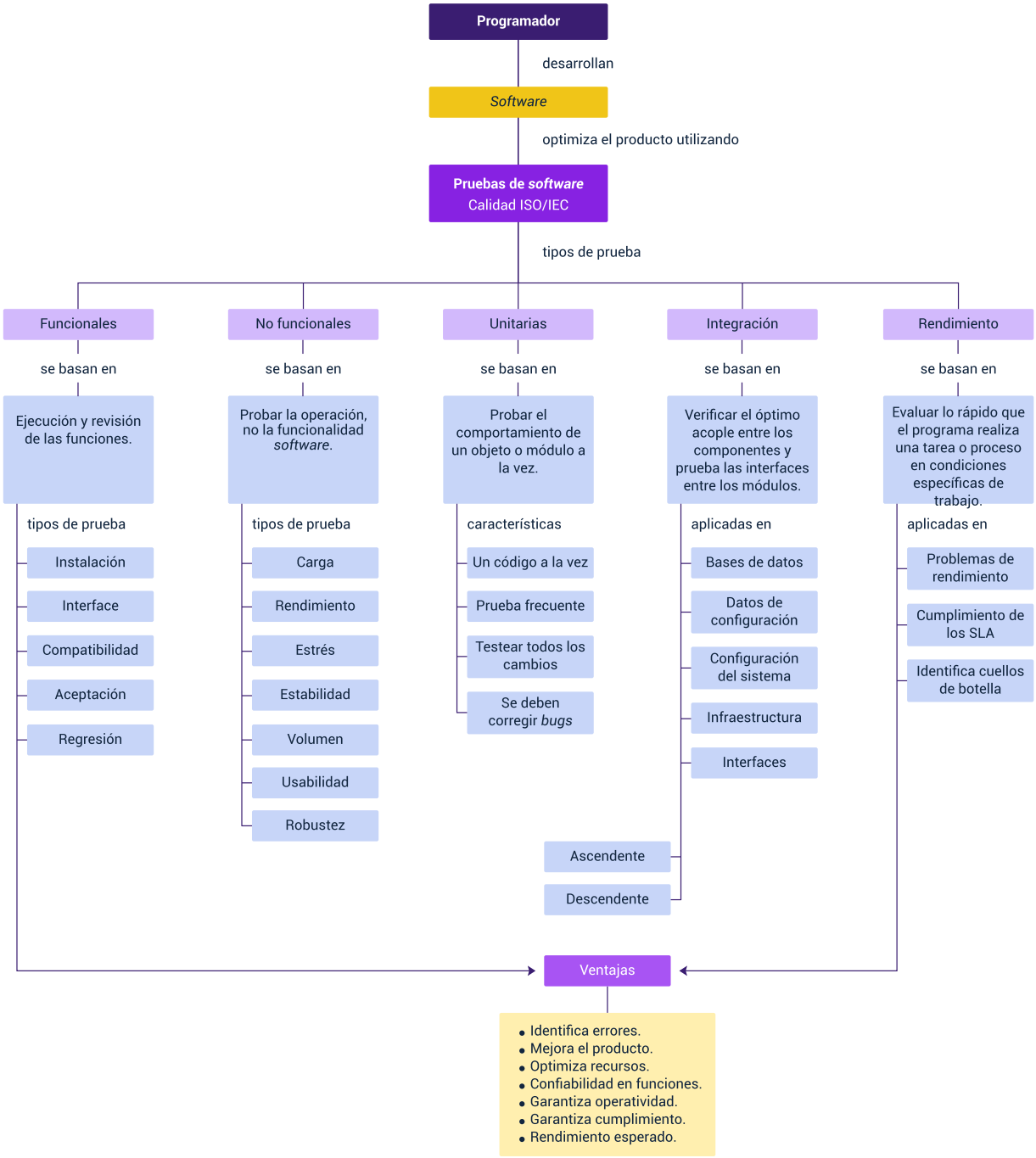
Ahora que se conocen los beneficios que implica aplicar TDD, es posible ver el ciclo completo del proceso, revisando cada uno de sus pasos. En la siguiente figura se presenta el flujo de manera gráfica.

Figura 3. Ciclo TDD



Síntesis

A continuación, se presenta a manera de síntesis, un esquema que articula los elementos principales abordados en el desarrollo del componente formativo, con relación a las pruebas de software.



Material complementario

Tema	Referencia	Tipo de material	Enlace del recurso
Tipos de pruebas y sus características	IBM. (2021). Pruebas funcionales.	Página web	https://www.ibm.com/docs/es/rtw/9.1.0?topic=SSBLQQ_9.1.0/com.ibm.rational.test.ft.doc/topics/Getting_Started_With_Ivory.html
Tipos de pruebas y sus características	IBM. (2021). Pruebas de rendimiento.	Página web	https://www.ibm.com/docs/es/rtw/9.0.0?topic=phases-performance-testing

Glosario

Bottom-up: pruebas ascendentes.

QA (Quality Assurance): especialistas en pruebas de software, verifican que un software no contenga fallos.

Refactory: acción de limpiar o reconstruir el código de una aplicación.

TDD: desarrollo dirigido por pruebas.

Top-down: pruebas descendentes.

Referencias bibliográficas

Herranz, J.I. (2021). TDD como metodología de diseño de software. https://www.paradigmadigital.com/dev/tdd-como-metodologia-de-diseno-de-software/

IBM (2021). Pruebas de rendimiento. https://www.ibm.com/docs/es/rtw/9.0.0?topic=phases-performance-testing

IBM (2021). Pruebas funcionales. https://www.ibm.com/docs/es/rtw/9.1.0?topic=SSBLQQ_9.1.0/com.ibm.rational.test.ft.doc/topics/Getting_Started_With_Ivory.html

Mera Paz, J. A. (2016). Análisis del proceso de pruebas de calidad de software. Ingeniería solidaria, 12 (20).

Organización ISO. (2020). Ingeniería de sistemas y software - Requisitos de calidad y evaluación de sistemas y software (SQuaRE) - Guía de SQuaRE. https://www.iso.org/standard/64764.html

Sánchez, A.F (2017). Agile Testing. Estado del arte. Su aplicación en empresas TIC de Extremadura. Universidad De Extremadura.

Sanz, L. F. (2005). Un sondeo sobre la práctica actual de pruebas de software en España. REICIS. Revista Española de Innovación, Calidad e Ingeniería del Software,1(2), pp. 43-54.

Créditos

Nombre	Cargo	Centro de Formación y Regional
Milady Tatiana Villamil Castellanos	Responsable del Ecosistema	Dirección General
Olga Constanza Bermúdez Jaimes	Responsable de Línea de Producción	Centro de Servicios de Salud - Regional Antioquia
Mario Fernando Meneses Calvache	Experto Temático	Centro de Teleinformática y Producción Industrial - Regional Cauca
Carlos Hernán Muñoz	Experto Temático	Centro de Teleinformática y Producción Industrial - Regional Cauca
Ana Catalina Córdoba Sus	Evaluadora Instruccional	Centro de Servicios de Salud - Regional Antioquia
Yerson Fabián Zárate Saavedra	Diseñador de Contenidos Digitales	Centro de Servicios de Salud - Regional Antioquia
Edward Leonardo Pico Cabra	Desarrollador Fullstack	Centro de Servicios de Salud - Regional Antioquia
Edgar Mauricio Cortés García	Actividad Didáctica	Centro de Servicios de Salud - Regional Antioquia
Jaime Hernán Tejada Llano	Validador de Recursos Educativos Digitales	Centro de Servicios de Salud - Regional Antioquia
Margarita Marcela Medrano Gómez	Evaluador para Contenidos Inclusivos y Accesibles	Centro de Servicios de Salud - Regional Antioquia
Daniel Ricardo Mutis Gómez	Evaluador para Contenidos Inclusivos y Accesibles	Centro de Servicios de Salud - Regional Antioquia