**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | Análisis y desarrollo de *software* |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501096. Desarrollar la solución de software de acuerdo con el diseño y metodologías de desarrollo. | RESULTADOS DE APRENDIZAJE | 220501096-05. Realizar pruebas al software para verificar su funcionalidad. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 26 |
| NOMBRE DEL COMPONENTE FORMATIVO | 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. |
| PALABRAS CLAVE | Actividades, pruebas, fallos, *software.* |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | 2 - CIENCIAS NATURALES, APLICADAS Y RELACIONADAS |
| IDIOMA | Idioma principal utilizado en el contenido del recurso |

1. **TABLA DE CONTENIDOS:**

# **1. Generalidades de las pruebas de *software***

# **2. Tipos de pruebas y sus características**

2.1 Pruebas funcionales

2.2 Pruebas unitarias

2.3 Pruebas de integración

2.4 Pruebas no funcionales

2.5 Pruebas de rendimiento

2.6 Ventajas y desventajas de los tipos de pruebas

# **3. Los casos de pruebas**

# **4. Desarrollo guiado por pruebas**

1. **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 en 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. **DESARROLLO DE CONTENIDOS:**

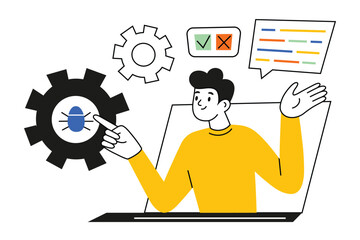
# **1. Generalidades de las pruebas de *software***

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*.

De acuerdo con Mera (2016):

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.

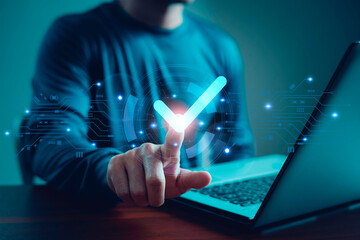
El mismo autor, afirma que dentro de las actividades de un proceso de prueba, se deben incluir las siguientes fases:



* 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.

La matriz ayuda a establecer métricas que ayudan a definir tiempos, esfuerzos, asignaciones, y riesgos logrando optimizar recursos y talento humano.

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:

|  |
| --- |
| CF26\_2\_Principios |

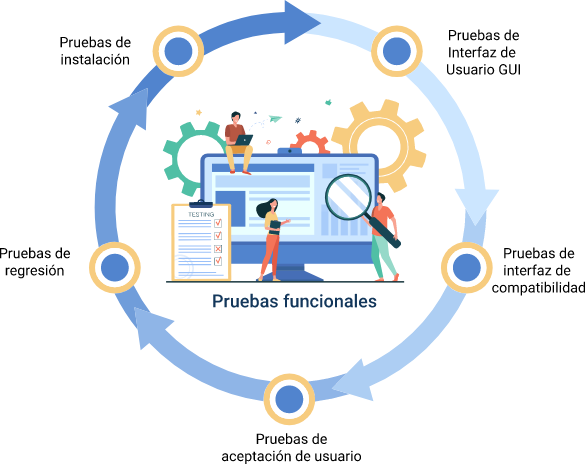
Conozcamos cuáles son los tipos de pruebas que se pueden realizar.

**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:

|  |
| --- |
| CF26\_2\_2\_Razones |

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:

|  |
| --- |
| CF26\_2\_2\_Herramientas |

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

* Probar un solo código a la vez.
* Hay que realizar pruebas frecuentemente mientras se programa.
* Corregir los bugs identificados en las pruebas antes de continuar.
* 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:

* Bases de datos de subsistemas.
* Datos de configuración.
* Configuración del sistema.
* Infraestructura.
* Configuración del sistema.
* Infraestructura.
* 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 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:

|  |
| --- |
| CF26\_2\_3\_Tipos |

**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:

|  |
| --- |
| CF26\_2\_4\_Pruebas |

**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:

|  |
| --- |
| CF26\_2\_5\_Objetivos |

**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.

**Ventajas**

|  |
| --- |
| CF26\_2\_6\_Ventajas |

**Desventajas de estos tipos de pruebas**

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 de programador y su equipo de trabajo.

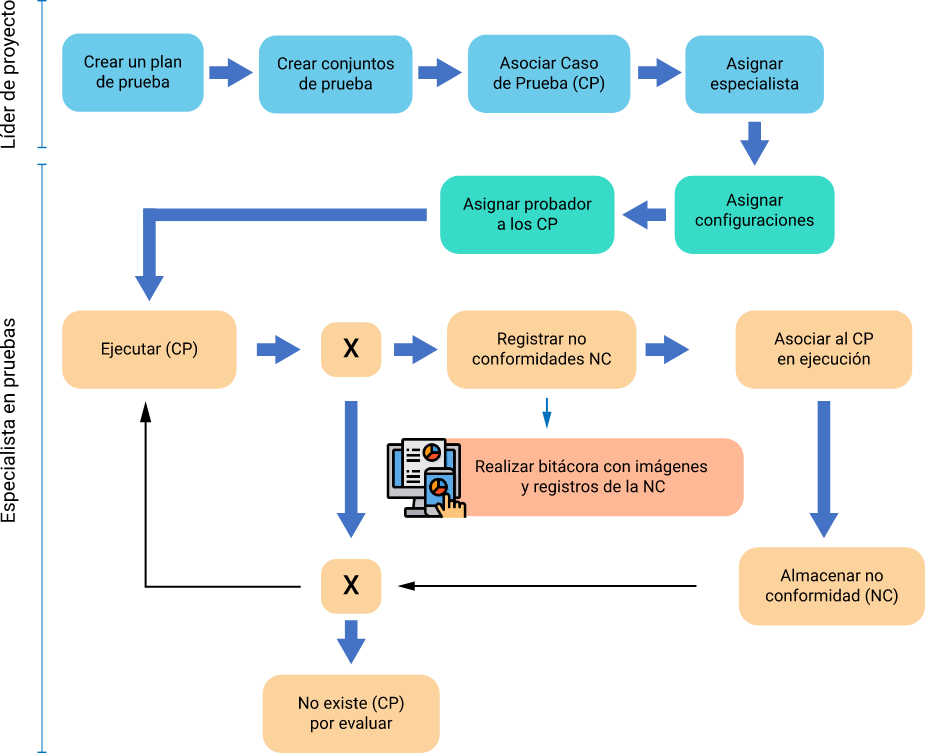
# 

# **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.

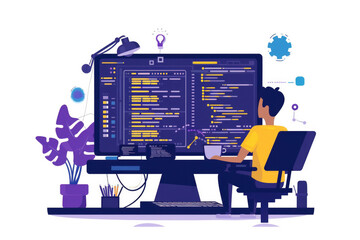
**Tabla 1**

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 que 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 la 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 ejecutarse 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.

Según Sánchez (2017):

“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”.

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

|  |
| --- |
| CF26\_4\_Proceso |



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 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:

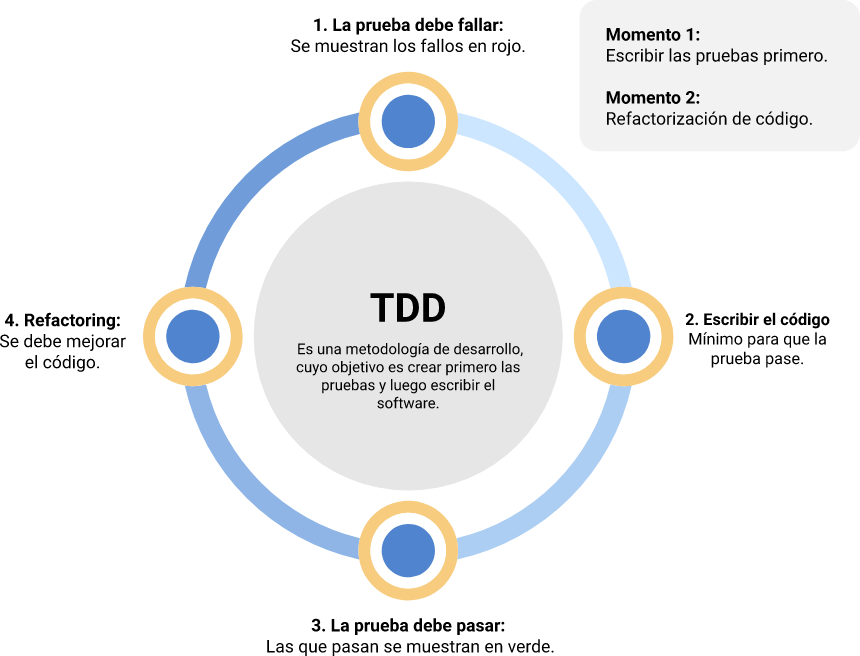
|  |
| --- |
| CF26\_4\_Beneficios |

**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.

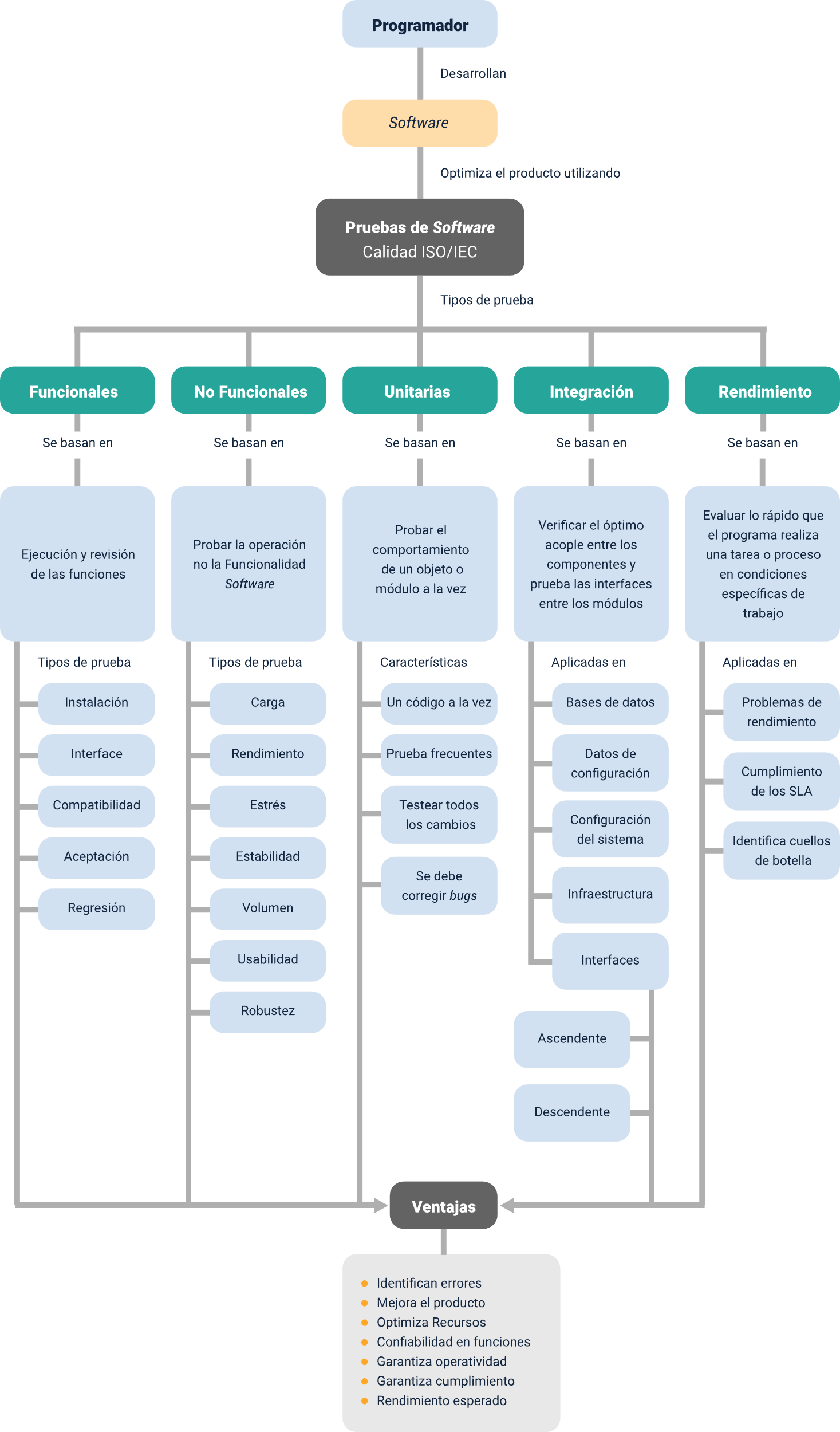
**Figura 3**

Ciclo TDD



1. **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.



1. **ACTIVIDADES DIDÁCTICAS (Se debe incorporar mínimo 1, máximo 2)**

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la Actividad | Conceptos básicos de pruebas de *software* |
| Objetivo de la actividad | Afianzar algunos conceptos básicos de pruebas de *software.* |
| Tipo de actividad sugerida | Arrastrar y soltar |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | CF26\_Actividad\_didactica.docx |

1. **MATERIAL COMPLEMENTARIO:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| 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> |

1. **GLOSARIO:**

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| *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. |

1. **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.

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha |
| Autor(es) | Mario Fernando Meneses Calvache | Experto Temático | Regional Cauca - Centro de Teleinformática y Producción Industrial |  |
| Carlos Hernán Muñoz | Experto Temático | Regional Cauca - Centro de Teleinformática y Producción Industrial |  |
| Ana Catalina Córdoba Sus | Evaluadora instruccional | Regional Antioquia - Centro de Servicios de Salud | Abril 2024 |
| Olga Constanza Bermúdez Jaimes | Responsable Línea de Producción Antioquia | Regional Antioquia - Centro de Servicios de Salud | Abril 2024 |

1. **CONTROL DE CAMBIOS**

**(Diligenciar únicamente si realiza ajustes a la Unidad Temática)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) |  |  |  |  |  |